Average-case analysis of algorithms and data structures

Philippe Flajolet, J.S. Vitter

INRIA

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France

Tél.(1) 39 63 55 11

Rapports de Recherche

N° 718

AVERAGE-CASE ANALYSIS OF ALGORITHMS AND DATA STRUCTURES

Philippe FLAJOLET
J. S. VITTER

AOUT 1987

Average-Case Analysis of Algorithms and Data Structures

J. S. Vitter[1] and *Ph. Flajolet*[2]

Abstract. This report is a draft of a contributed chapter to the *Handbook of Theoretical Computer Science* (North-Holland, 1988, to appear). Its aim is to describe the main mathematical methods and applications in the average-case analysis of algorithms and data structures. It comprises two parts: First, we present basic combinatorial enumerations based on symbolic methods and asymptotic methods with emphasis on complex analysis techniques (such as singularity analysis, saddle point, Mellin transforms). Next, we show how to apply these general methods to the analysis of sorting, searching, tree data structures, hashing, and dynamic algorithms. The emphasis is on algorithms for which exact "analytic models" can be derived.

L'analyse en moyenne des algorithmes et des structures de données

Résumé. Ce rapport est une version préliminaire d'un chapitre à paraître dans le *Handbook of Theoretical Computer Science* (North-Holland, 1988). Son but est de décrire les principales méthodes et applications de l'analyse de complexité en moyenne des algorithmes. Il comprend deux parties. Tout d'abord, nous donnons une présentation des méthodes de dénombrements combinatoires qui repose sur l'utilisation de méthodes symboliques, ainsi que des techniques asymptotiques fondées sur l'analyse complexe (analyse de singularités, méthode du col, transformation de Mellin). Ensuite, nous décrivons l'application de ces méthodes génerales à l'analyse du tri, de la recherche, de la manipulation d'arbres, du hachage et des algorithmes dynamiques. L'accent est mis dans cette présentation sur les algorithmes pour lesquels existent des "modèles analytiques" exacts.

---

[1] Dept. of Computer Science, Brown University, Providence, R. I. 02912, USA. Research was also done while the author was on sabbatical at INRIA in Rocquencourt, France, and at Ecole Normale Supérieure in Paris, France.

[2] INRIA, Domaine de Voluceau, Rocquencourt, B. P. 105, 78153 Le Chesnay Cedex, France

Average-Case Analysis of Algorithms and Data Structures

J. S. Vitter and Ph. Flajolet

Analyzing an algorithm means, in its broadest sense, characterizing the amount of computational resources that an execution of the algorithm will require when applied to data of a certain type. Many algorithms in classical mathematics, primarily in number theory and analysis, were analyzed by eighteenth and nineteenth century mathematicians. For instance, Lamé in 1845 showed that Euclid's GCD algorithm requires at most $\approx \log_\phi n$ division steps (where ϕ is the golden ratio $(1+\sqrt{5})/2$) when applied to numbers that are $\leq n$. Similarly, the well-known quadratic convergence of Newton's method is a way of describing its complexity/accuracy tradeoff.

This chapter presents methods for and examples of *average case analysis* of some of the main algorithms used for processing non-numerical data, with emphasis on searching and sorting algorithms and on "analytic models". This leads to a precise classification of algorithmic solutions to a number of essential problems like sorting; designing data structures for searching; retrieval of multidimensional data; efficient access to large files stored on secondary memory, etc. An analysis requires defining an *input data model*, and a *complexity measure*.

1. Assume algorithm $\mathcal{A}$ takes as inputs data of type $\mathbf{I}$. Each commonly used data type carries a natural notion of size: the size of an array is the number of its elements (its dimension); the size of a file is the number of its records; the size of a character string is its length etc. If $\mathbf{I}_n$ is the subset of inputs of size n, an input model is specified by a probability distribution over $\mathbf{I}_n$ for each n. For instance, in comparison-based sorting algorithms, a classical model is the independence model over $\mathbf{I}_n = [0,1]^n$ (all components independently uniformly $[0,1]$-distributed). An equivalent model considers all permutations over $\{1,2,\ldots,n\}$ to be equally likely.
2. Complexity measures for algorithms executed on sequential machines are usually *time* (τ) and *space utilization* (σ). These may be either "raw" measures (viz. the time in nanoseconds on a MIX machine; the number of bits necessary for storing temporary variables) or "abstract" measures (number of comparisons for a sorting algorithm; number of disk pages).

Let us consider an algorithm $\mathcal{A}$ with complexity measure μ_n. The worst-case and best-case complexity of algorithm $\mathcal{A}$ over $\mathbf{I}_n$ are defined in an obvious way. Characterizing the worst-case complexity requires constructing extremal configurations that force the algorithm to perform a large number of operations. Once a probabilistic input model has been specified, the *average-case* complexity is defined by

$$\overline{\mu_n}[\mathcal{A}] = \mathbf{E}\{\mu[\mathcal{A}](e) \mid e \in \mathbf{I}_n\},$$

with $\mathbf{E}\{.\}$ denoting expectations. By the definition of expectation, this is

$$\overline{\mu_n}[\mathcal{A}] = \sum_k k \Pr\{\mu[\mathcal{A}](e) = k \mid e \in \mathbf{I}_n\},$$

with $\Pr\{.\}$ denoting a probability. Frequently, $\mathbf{I}_n$ is a finite set and the probabilistic model is equivalent to a uniform probability distribution over $\mathbf{I}_n$. In that case, $\overline{\mu_n}[\mathcal{A}]$ takes the form

$$\overline{\mu_n}[\mathcal{A}] = \frac{1}{I_n} \sum_k k J_{nk},$$

where $I_n = \mathrm{card}(\mathbf{I}_n)$ and J_{nk} is the number of structures of size n with cost k for algorithm $\mathcal{A}$.

Analyzing an algorithm $\mathcal{A}$ thus reduces to *combinatorial enumeration* (counting) problems. We next try to place the complexity of the algorithm inside standard asymptotic scales composed of standard functions like $n^\alpha (\log n)^\beta (\log\log n)^\gamma$, so that analysis results become easy to interpret. A second phase thus consists of obtaining *asymptotic estimates*. The most elementary route, whenever feasible, for analyzing an algorithm is as follows (see Figure 1):

1. *RECUR:* To determine the probabilities or the expectations in exact form, start by setting up recurrences that relate the behaviour of algorithm $\mathcal{A}$ on inputs of size n to its behaviour on smaller (and similar) inputs.
2. *SOLVE:* Solve previous recurrences explicitly using classical algebra.
3. *ASYMPT:* Use standard real asymptotics to estimate those explicit expressions.

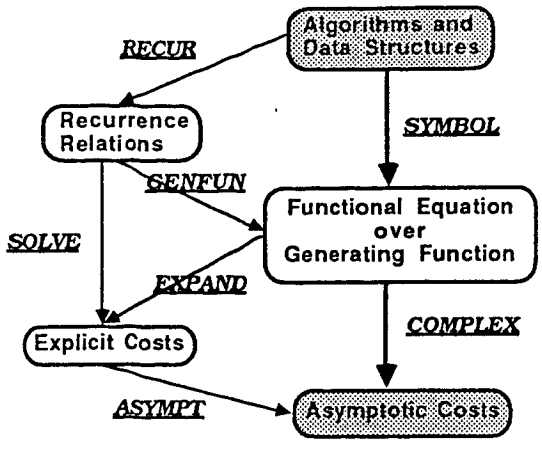

Figure 1. Methods used in the average-case analysis of algorithms.

An important way to solve recurrences is via the use of *generating functions*:

4. *GENFUN:* Translate recurrences into equations over generating functions. In general we obtain a set of *functional equations.*
5. *EXPAND:* Solve those functional equations using classical tools from algebra and analysis, then expand the solutions to get the coefficients in explicit form.

The above methods are the basis of many elementary analyses. They can, in most cases, be bypassed by the following more powerful methods:

6. *SYMBOL:* Instead of going through recurrences, translate directly set-theoretic definitions of data structures or underlying combinatorial structures into functional equations over generating functions.
7. *COMPLEX:* Use complex analysis to translate directly information available from functional equations into asymptotic forms of coefficients.

The symbolic method (*SYMBOL*) is usually fairly direct; it has the advantage of characterizing the class of *special functions* that occur when analyzing a natural class of related algorithms. The *COMPLEX* method provides powerful tools for direct asymptotics from generating functions. It has the intrinsic advantage of allowing in many cases asymptotic expansion of coefficients of functions known *only implicitly* from their functional equations.

References. General references for the analysis of algorithms are [Knuth 1973a], [Knuth 1981], [Knuth 1973b], [Sedgewick 1983a], [Flajolet 1981], [Greene and Knuth 1983], [Kemp 1984], [Flajolet 1985], and for an elementary introduction [Purdom and Brown 1985]. Our presentation of combinatorial enumerations is based on symbolic methods that are related to recent research in combinatorial analysis: See [Goulden and Jackson 1983] for a complete exposition, [Comtet 1974] for an encyclopedic treatment of combinatorial analysis, [Stanley 1975] for a survey, [Stanley 1986], and for applications to the analysis of algorithms, [Flajolet 1981] or [Greene 1983]. Complex methods come from classical analysis: see [Henrici 1977], [De Bruijn 1981], [Bender 1974], [Bender and Orszag 1983], and [Flajolet 1985] for a summary of applications to the analysis of algorithms. Apart from Knuth's books, good references for the description of algorithms analyzed in this text are [Sedgewick 1983b] and [Gonnet 1984].

Part I. Methods

In Part I, we develop general techniques for the mathematical analysis of algorithms. Applications to particular algorithms appear in Part II. In this part, we study two major topics:

1. Combinatorial Enumerations. We present the main methods used to obtain counting results for the analysis of algorithms, with emphasis on symbolic methods (*SYMBOL*). Our main mathematical tool is the generating function associated to a class of structures. A rich set of combinatorial constructions translate directly into functional relations over generating functions.
2. Asymptotic Analysis. We briefly review elementary real analysis methods and then concentrate on complex analysis techniques (*COMPLEX*). There, we use analytic properties of generating functions to recover information on coefficients representing enumeration results or expectations. An important fact is that the methods are also applicable to functions only known indirectly via functional equations, a situation that naturally presents itself when counting recursively defined structures.

1. Combinatorial Enumerations

Our main objective in this section is to introduce combinatorial constructions that have a direct translation into generating functions. Such constructions are called admissible. We examine in Section 1.2 admissible constructions for ordinary generating functions. Exponential generating functions are related to the enumeration of labeled structures and they form the subject of Section 1.3.

1.1. Overview

The most elementary structures may be enumerated using sum/product rules†

THEOREM 0 [Sum-Product Rule]. *Let $\mathcal{A}, \mathcal{B}, \mathcal{C}$ be sets with cardinalities a, b, c. Then*

$$\begin{aligned} \mathcal{C} = \mathcal{A} \cup \mathcal{B}, \quad \text{with } \mathcal{A} \cap \mathcal{B} = \emptyset \quad &\Longrightarrow \quad c = a + b, \\ \mathcal{C} = \mathcal{A} \times \mathcal{B} \quad &\Longrightarrow \quad c = a \cdot b. \end{aligned}$$

Thus, the number of binary strings of length n is 2^n; the number of permutations of $\{1, 2, \ldots, n\}$ is $n!$, the number of ways of choosing n elements amongst m (combinations) is $\binom{m}{n}$, and so on.

In the next order of difficulty, explicit forms are replaced by recurrences when structures are defined in terms of themselves. For example, let F_n be the number of coverings of an interval of length n with segments of length either 1 or 2. By considering the two possibilities for the last segment used, we get

$$F_n = F_{n-1} + F_{n-2}, \qquad \text{for } n \geq 2 \tag{1a}$$

with initial conditions $F_0 = F_1 = 1$. Thus, from the classical theory of linear recurrences, we find the Fibonacci numbers expressed in terms of the "golden ratio" ϕ:

$$F_n = \frac{1}{\sqrt{5}}(\phi^{n+1} - \bar{\phi}^{n+1}), \qquad \text{with } \phi, \bar{\phi} = \frac{1 \pm \sqrt{5}}{2}. \tag{1b}$$

This example illustrates recurrence methods (*RECUR*) in (1*a*) and derivation of explicit solutions (*SOLVE*) in (1*b*).

Another example, which we shall discuss in more detail in Section 4.1, is the number B_n of plane binary trees with n internal nodes [Knuth 1973a]. By considering all possibilities for left and right subtrees, we get:

$$B_n = \sum_{k=0}^{n-1} B_k B_{n-k-1}, \qquad \text{for } n \geq 1, \tag{2a}$$

† We also use the sum notation $\mathcal{A} = \mathcal{B} + \mathcal{C}$ to represent the union of $\mathcal{A}$ and $\mathcal{B}$ when $\mathcal{A} \cap \mathcal{B} = \emptyset$.

with the initial conditions $B_0 = B_1 = 1$. To solve (2*a*), we introduce a *generating function* (GF): let $B(z) = \sum_{n\geq 0} B_n z^n$. A simple computation from Eq.(2a) leads to

$$B(z) = 1 + zB^2(z), \tag{2b}$$

and solving the quadratic equation for $B(z)$, we get

$$B(z) = \frac{1-\sqrt{1-4z}}{2z}. \tag{2c}$$

Finally, the Taylor expansion of $(1+x)^{1/2}$ gives us

$$B_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{n!(n+1)!}. \tag{2d}$$

In this case, we started with recurrences (*RECUR*) in (2*a*), introduced generating functions (*GENFUN*) leading to (2*b*); solving and expanding (*EXPAND*) gave the explicit solutions (2*c*) and (2*d*). (The example goes back to Euler and the B_n are called Catalan numbers).

The symbolic method (*SYMBOL*) that we are going to present can be applied to this last example as follows: The class $\mathcal{B}$ of binary trees is defined recursively by the equation

$$\mathcal{B} = \{\blacksquare\} \cup (\{\circ\} \times \mathcal{B} \times \mathcal{B}) \tag{3a}$$

A standard lemma asserts that disjoint unions and cartesian products of structures correspond respectively to sums and products of corresponding generating functions. Therefore, specification (3a) translates term by term directly into the generating function equation (cf. (2*b*)):

$$B(z) = 1 + z \cdot B(z) \cdot B(z). \tag{3b}$$

DEFINITION. A *class of combinatorial structures* $\mathcal{C}$ is a finite or denumerable set together with an integer valued function $|.| \equiv |.|_{\mathcal{C}}$, called the *size* function, such that for each $n \in \mathbf{N}$ the number of structures of size n in $\mathcal{C}$ is finite. The *counting sequence* for class $\mathcal{C}$ is the integer sequence $\{C_n\}$ such that C_n is the number of structures of size n in $\mathcal{C}$. The *ordinary generating function* (OGF) $C(z)$ and the *exponential generating function* (EGF) $\widehat{C}(z)$ of a class $\mathcal{C}$ are defined respectively by:

$$C(z) = \sum_{n\geq 0} C_n z^n, \qquad \widehat{C}(z) = \sum_{n\geq 0} C_n \frac{z^n}{n!}. \tag{4}$$

The coefficient of z^n in a function $f(z)$ is written $[z^n]f(z)$. By definition $[\frac{z^n}{n!}]f(z) = n![z^n]f(z)$.

The generating functions $C(z)$ and $\widehat{C}(z)$ can also be expressed as

$$C(z) = \sum_{\gamma\in\mathcal{C}} z^{|\gamma|} \qquad \text{and} \qquad \widehat{C}(z) = \sum_{\gamma\in\mathcal{C}} \frac{z^{|\gamma|}}{|\gamma|!}, \tag{5}$$

which can be checked by counting the number of occurrences of z^n in the sums.

We shall adopt the notational convention that a class ($\mathcal{C}$), its counting sequence (C_n or c_n), and the associated ordinary and exponential generating functions ($C(z)$ or $c(z)$, and respectively $\widehat{C}(z)$ or $\widehat{c}(z)$) are represented by the same group of letters.

The basic notion for the symbolic method is that of an *admissible construction* in which the counting sequence of the construction depends only upon the counting sequences of its components (see [Goulden and Jackson 1983], [Flajolet 1981], [Greene 1983]); such a construction thus "translates" over generating functions. It induces an operator of a more or less simple form over formal power series. For instance, let $\mathcal{U}$ and $\mathcal{V}$ be two classes of structures, and let

$$\mathcal{W} = \mathcal{U} \times \mathcal{V} \tag{6a}$$

be their cartesian product. If the size of an ordered pair $w = (u, v) \in \mathcal{W}$ is defined as $|w| = |u| + |v|$, then by counting possibities, we get

$$W_n = \sum_{k\geq 0} U_k V_{n-k}, \tag{6b}$$

so that (6a) has the corresponding (ordinary) generating function equation

$$W(z) = U(z)V(z). \tag{6c}$$

Such a combinatorial (set-theoretic) construction that translates in the manner of (6a)–(6c) is called *admissible*.

1.2. Ordinary Generating Functions

In this section we present a catalog of admissible constructions for ordinary generating functions (OGFs). We assume that the size of an element of a disjoint union $\mathcal{W} = \mathcal{U} \cup \mathcal{V}$ is inherited from the size of its original domain; the size of a composite object (product, sequence, subset, etc.) is the sum of the sizes of its components.

THEOREM 1 [Fundamental Sum/Product Theorem]. *The disjoint union and cartesian product constructions are admisssible:*

$$\begin{aligned} &\mathcal{W} = \mathcal{U} \cup \mathcal{V}, \quad \textit{with } \mathcal{U} \cap \mathcal{V} = \emptyset \quad &\Longrightarrow \quad &W(z) = U(z) + V(z) \\ &\mathcal{W} = \mathcal{U} \times \mathcal{V} &\Longrightarrow \quad &W(z) = U(z)V(z). \end{aligned}$$

PROOF. Use recurrences $W_n = U_n + V_n$ and $W_n = \sum_{0\leq k\leq n} U_k V_{n-k}$. Alternatively, use Eq. (5) for GFs, which yields for cartesian products

$$\sum_{w\in\mathcal{W}} z^{|w|} = \sum_{(u,v)\in\mathcal{U}\times\mathcal{V}} z^{|u|+|v|} = \sum_{u\in\mathcal{U}} z^{|u|} \cdot \sum_{v\in\mathcal{V}} z^{|v|}. \quad \blacksquare$$

Class $\mathcal{W}$ is called the *sequence class* of class U, denoted $\mathcal{W} = \mathcal{U}^*$, if $\mathcal{W}$ is composed of all sequences $(v_1, v_2, \ldots, v_k)$ with $v_j \in \mathcal{U}$. Class $\mathcal{W}$ is the (finite) *powerset* of class $\mathcal{U}$, denoted $\mathcal{W} = 2^{\mathcal{U}}$, if $\mathcal{W}$ consists of all finite subsets $\{v_1, v_2, \ldots, v_k\}$ of $\mathcal{U}$ (the v_j are distinct).

THEOREM 2. *The sequence and powerset constructs are admissible:*

$$\begin{aligned} &\mathcal{W} = \mathcal{U}^* \quad &\Longrightarrow \quad &W(z) = \frac{1}{1 - U(z)} \\ &\mathcal{W} = 2^{\mathcal{U}} \quad &\Longrightarrow \quad &W(z) = e^{\Phi(U(z))}, \quad \textit{where } \Phi(f) = \frac{f(z)}{1} - \frac{f(z^2)}{2} + \frac{f(z^3)}{3} - \cdots. \end{aligned}$$

PROOF. Let ϵ denote the empty sequence. Then, for the sequence class of $\mathcal{U}$, we have

$$\begin{aligned} \mathcal{W} = \mathcal{U}^* &\equiv \{\epsilon\} + \mathcal{U} + (\mathcal{U} \times \mathcal{U}) + (\mathcal{U} \times \mathcal{U} \times \mathcal{U}) + \cdots \\ W(z) &= 1 + U(z) + U^2(z) + U^3(z) + \cdots = (1 - U(z))^{-1}. \end{aligned} \tag{7}$$

The powerset class $\mathcal{W} = 2^{\mathcal{U}}$ is equivalent to an infinite product:

$$\begin{aligned} \mathcal{W} = 2^{\mathcal{U}} &= \prod_{v\in\mathcal{U}} (\{\epsilon\} + \{v\}) \\ W(z) &= \prod_{v\in\mathcal{U}} (1 + z^{|v|}) = \prod_n (1 + z^n)^{U_n} \end{aligned} \tag{8}$$

Computing logarithms, and expanding, we obtain

$$\log W(z) = \sum_n U_n \log(1 + z^n) = \sum_n U_n z^n - \frac{1}{2}\sum_n U_n z^{2n} + \cdots. \quad \blacksquare$$

Other constructions can be shown to be admissible:

1. Diagonals and subsets with repetitions. The *diagonal* $\mathcal{W}$ of $\mathcal{U} \times \mathcal{U}$, written $\mathcal{W} = \Delta(\mathcal{U} \times \mathcal{U})$, satisfies $W(z) = U(z^2)$. The class of subsets with repetitions of class $\mathcal{U}$ is denoted $\mathcal{W} = \mathbf{R}\{\mathcal{U}\}$. It is isomorphic to $\prod_{v \in \mathcal{U}} \{v\}^*$, so that its OGF satisfies:
$$W(z) = e^{\Psi(U(z))} \qquad \text{where } \Psi(f) = \frac{f(z)}{1} + \frac{f(z^2)}{2} + \frac{f(z^3)}{3} + \cdots. \tag{9}$$
2. Marking and composition. If $\mathcal{U}$ is formed with "atomic" elements (nodes, letters, etc.) that determine its size, then we define the *marking* of $\mathcal{U}$, denoted $\mathcal{W} = \mu\{\mathcal{U}\}$, to consist of elements of $\mathcal{U}$ with one individual atom marked. Since $W_n = nU_n$, it follows that $W(z) = zU'(z)$. Similarly, the composition of $\mathcal{U}$ and $\mathcal{V}$, denoted $\mathcal{W} = \mathcal{U}[\mathcal{V}]$, is defined as the class of all structures resulting from substitutions of atoms of $\mathcal{U}$ by elements of V; then $W(z) = U\big(V(z)\big)$.

EXAMPLES. 1. *Combinations.* Let m be a fixed integer and $\mathcal{J}^{(m)} = \{1, 2, \ldots, m\}$, each element of $\mathcal{J}^{(m)}$ having size 1. The class $\mathcal{C}^{(m)} = 2^{\mathcal{J}^{(m)}}$ is the set of all combinations of $\mathcal{J}^{(m)}$. Therefore, the number $C_n^{(m)}$ of n-combinations of a set with m elements is obtained along the lines of the proof of Theorem 2:
$$C^{(m)}(z) = (1+z)^m \qquad \Longrightarrow \qquad C_n^{(m)} = \binom{m}{n} = \frac{m!}{n!(m-n)!}.$$
Similarly, for $R^{(m)}$ the class of combinations with repetitions, we have
$$R^{(m)}(z) = (1-z)^{-m} \qquad \Longrightarrow \qquad R_n^{(m)} = \binom{m+n-1}{m-1}.$$

2. *Compositions and partitions.* Let $\mathcal{N} = \{1, 2, 3, \ldots\}$, each $i \in \mathcal{N}$ having size i. The class $\mathcal{C} = \mathcal{N}^*$ is called the set of integer compositions. Since $N(z) = z/(1-z)$ and $C(z) = \big(1 - N(z)\big)^{-1}$, we have
$$C(z) = \frac{1-z}{1-2z} \qquad \Longrightarrow \qquad C_n = 2^{n-1}.$$
The class $\mathcal{P} = \mathbf{R}\{\mathcal{N}\}$ is the set of integer partitions, and
$$P(z) = \prod_{n \geq 1} \frac{1}{1-z^n}. \tag{10}$$

3. *Formal languages.* Combinatorial processes can often be naturally encoded as strings over some finite alphabet $\mathcal{A}$. Regular languages are defined by regular expressions or equivalently by deterministic/non-deterministic finite automata. This is illustrated by the following two theorems, based on the work of Chomsky and Schützenberger (1963).

THEOREM 3A [Regular languages and rational functions]. *If $\mathcal{L}$ is a regular language, then its OGF is a rational function $L(z) = P(z)/Q(z)$, where $P(z)$ and $Q(z)$ are polynomials. The counting sequence L_n satisfies a linear recurrence with constant coefficients, and we have when $n \geq n_0$*
$$L_n = \sum_j \pi_j(n) \omega_j^n,$$
for a finite set of constants ω_j and polynomials $\pi_j(z)$.

PROOF. Let D be a deterministic automaton that recognizes $\mathcal{L}$, and let S_j be the set of words accepted by D when D is started in state j. The S_j satisfy a set of linear equations (involving unions and concatenation with letters) constructed from the transition table of the automaton. For generating functions, this translates into a set of linear equations with polynomial coefficients that can be solved by Cramer's rule. ∎

THEOREM 3B [Context-Free languages and algebraic functions]. *If $\mathcal{L}$ is an unambiguous context-free language, then its OGF is an algebraic function. The counting sequence L_n satisfies a linear recurrence with polynomial coefficients: for a family $q_j(z)$ of polynomials and $n \geq n_0$, we have*
$$L_n = \sum_{1 \leq j \leq m} q_j(n) L_{n-j}.$$

PROOF. Since the language is unambiguous, its counting problem is equivalent to counting derivation trees. A production in the grammar like $S \to aTbU + bUUa + abba$ translates into $S(z) = z^2T(z)U(z) + z^2U^2(z) + z^4$, $S(z)$ being the generating function associated with non-terminal S. We obtain a set of polynomial equations that reduces to a single equation $P(z, L(z)) = 0$ through elimination. To obtain the recurrence, use Comtet's theorem [Comtet 1969] (see also [Flajolet 1987] for corresponding asymptotic estimates). ∎

4. *Trees.* We shall study trees in great detail in Section 4.1. All trees here are rooted; in plane trees, subtrees under a node are ordered; in non plane trees, they are unordered. If $\mathcal{G}$ is the class of general plane trees with all nodes degrees allowed, then $\mathcal{G}$ satisfies an equation $\mathcal{G} = \{\circ\} \times \mathcal{G}^*$ (a tree is a root followed by a sequence of trees). Thus, we have

$$G(z) = \frac{z}{1 - G(z)} \quad \Longrightarrow \quad G(z) = \frac{1 - \sqrt{1-4z}}{2} \quad \text{and} \quad G_n = \frac{1}{n}\binom{2n-2}{n-1}.$$

If $\mathcal{H}$ is the class of general non-plane trees, then $\mathcal{H} = \{\circ\} \times \mathbf{R}\{\mathcal{H}\}$, so that $H(z)$ satifies the functional equation:

$$H(z) = ze^{H(z)+H(z^2)/2+H(z^3)/2+\cdots}. \tag{11}$$

There are no closed form expressions for $H_n = [z^n]H(z)$. Howevever, complex analysis methods make it possible to determine H_n asymptotically [Polya 1937]. ∎

1.3. Exponential Generating Functions

Exponential generating functions are essentially used for counting *well-labeled structures.* Such structures are composed of "atoms" (the size of a structure being the number of its atoms) and each atom is labeled with a distinct integer. For instance, a labeled graph of size n is just a graph over the set of nodes $\{1, 2, \ldots, n\}$. A permutation (respectively, circular permutation) can be viewed as a linear (respectively, cyclic) directed graph whose nodes are labeled by distinct integers.

The basic operation over labeled structures is the *partitional product* [Foata 1971], [Goulden and Jackson 1983], [Flajolet 1981, Chap. I], [Greene 1983]. The partitional product of $\mathcal{U}$ and $\mathcal{V}$ consists in forming ordered pairs (u, v) from $\mathcal{U} \times \mathcal{V}$ and relabeling them in all possible ways that preserve the order of the labels in u and v. More precisely, let $w \in \mathcal{W}$ be a labeled structure of size q. An injective function θ from $\{1, 2, \ldots, q\}$ to $\{1, 2, \ldots, r\}$, where $r \geq q$, defines a relabeling, denoted $w' = \theta(w)$, by substituting label j in w by $\theta(j)$. Let u and w be two labeled structures of respective sizes m and n, then the partitional product of u and v is denoted by $u * v$, and it consists of the set of all relabeled (ordered) pairs $(u', v') = \{(\theta_1(u), \theta_2(v))\}$ of (u, v) where $\theta_1 : \{1, 2, \ldots, m\} \rightarrow \{1, 2, \ldots, m+n\}$, $\theta_2 : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, m+n\}$ satisfy

1. θ_1 and θ_2 are monotone functions. (Thus the order structure of u and v is preserved).
2. The ranges of θ_1 and θ_2 are disjoint and cover the set $\{1, 2, \ldots, m+n\}$.

The partitional product of two classes $\mathcal{U}$ and $\mathcal{V}$ is denoted $\mathcal{W} = \mathcal{U} * \mathcal{V}$ and is the union of all $u * v$, for $u \in \mathcal{U}$ and $v \in \mathcal{V}$.

THEOREM 4 [Sum/Product Theorem for Labeled Structures]. *The disjoint union and partitional product over labeled structures are admissible for exponential generating functions:*

$$\begin{aligned} &\mathcal{W} = \mathcal{U} \cup \mathcal{V}, \qquad \textit{with } \mathcal{U} \cap \mathcal{V} = \emptyset \quad &\Longrightarrow \quad &\widehat{W}(z) = \widehat{U}(z) + \widehat{V}(z) \\ &\mathcal{W} = \mathcal{U} * \mathcal{V} &\Longrightarrow \quad &\widehat{W}(z) = \widehat{U}(z)\widehat{V}(z). \end{aligned}$$

PROOF. Obvious for unions. For products, observe that

$$W_q = \sum_{m=0}^{q} \binom{q}{m} U_m V_{q-m}, \tag{12}$$

since the binomial coefficient counts the number of bipartitions of $\{1, 2, \ldots, q\}$ into two sets of cardinality m and $q - m$. Dividing in Eq.(7) by $q!$, we get:

$$\frac{W_q}{q!} = \sum_{m=0}^{q} \frac{U_m}{m!} \frac{V_{q-m}}{(q-m)!}. \quad ∎$$

The *partitional complex* of $\mathcal{U}$ is denoted $\mathcal{U}^{(*)}$. It is analogous to the sequence class construction and is defined by

$$\mathcal{W} = \mathcal{U}^{(*)} \equiv \{\epsilon\} + \mathcal{U} + (\mathcal{U} * \mathcal{U}) + (\mathcal{U} * \mathcal{U} * \mathcal{U}) + \cdots$$

so that $\widehat{W}(z) = (1-\widehat{U}(z))^{-1}$. The kth partitional power of $\mathcal{U}$ is noted $\mathcal{U}^{(k)}$. The *Abelian partional power*, noted $\mathcal{U}^{[k]}$, is the collection of all *sets* $\{\upsilon_1, \upsilon_2, \ldots, \upsilon_k\}$ such that $(\upsilon_1, \upsilon_2, \ldots, \upsilon_k) \in \mathcal{U}^{(k)}$. In other words, the order of components is not taken into account. We can write symbolically $\mathcal{U}^{[k]} = \frac{1}{k!}\mathcal{U}^{(k)}$ so that the EGF of $\mathcal{W} = \mathcal{U}^{[k]}$ is $\frac{1}{k!}\widehat{U}^k(z)$. The *abelian partitional complex* of $\mathcal{U}$ is defined analogously to the powerset construction:

$$\mathcal{W} = \mathcal{U}^{[*]} \equiv \{\epsilon\} + \mathcal{U} + \mathcal{U}^{[2]} + \mathcal{U}^{[3]} + \ldots.$$

THEOREM 5. *The partitional complex and abelian partitional complex are EGF admissible:*

$$\begin{aligned} \mathcal{W} = \mathcal{U}^{(*)} \quad &\Longrightarrow \quad \widehat{W}(z) = \frac{1}{1-\widehat{U}(z)} \\ \mathcal{W} = \mathcal{U}^{[*]} \quad &\Longrightarrow \quad \widehat{W}(z) = e^{\widehat{U}(z)}. \end{aligned} \tag{13}$$

EXAMPLES. 1. *Permutations and Cycles.* Let $\mathcal{P}$ be the class of all permutations, $\mathcal{C}$ the class of circular permutations. Since $P_n = n!$, we have $\widehat{P}(z) = (1-z)^{-1}$. Since any permutation decomposes into (an undordered set of) cycles, $\mathcal{P} = \mathcal{C}^{[*]}$, so that $\widehat{C}(z) = \log(1/(1-z))$ and $C_n = (n-1)!$. This construction also shows that the EGF for permutations having k cycles is $\log^k(1/(1-z))$ whose coefficient of $\frac{z^n}{n!}$ is a Stirling number of the first kind, denoted $s_{n,k}$.

Let $\mathcal{Q}$ be the class of permutations without cycles of size 1 (that is, without fixed points, also called derangements). Let $\mathcal{D}$ be the class of cycles of size at least 2. We have $\widehat{D}(z) = \log(1-z)^{-1} - z$, so that

$$\widehat{Q}(z) = e^{\widehat{D}(z)} = \frac{e^{-z}}{1-z}. \tag{14}$$

Similarly, the generating function for involutions (permutations with cycles of length 1 or 2 only) is

$$\widehat{I}(z) = e^{z+z^2/2}. \tag{15}$$

2. *Labeled graphs.* Let $\mathcal{G}$ be the class of all labeled graphs, $\mathcal{K}$ the class of connected graphs. Then $G_n = 2^{n(n-1)/2}$, and $\widehat{K}(z) = \log \widehat{G}(z)$, from which we can prove that $K_n/G_n \to 1$ as $n \to \infty$.

3. *Occupancies and Set Partitions.* Define the *urn* of size n to be the structure formed from the unordered collection of the integers 1 to n, and $\mathcal{U}$ be the class of all urns. Then $\widehat{U}(z) = e^z$. The class $U^{(k)}$ represents all possible ways of throwing distinguishable balls into k distinguishable urns, and its EGF is e^{kz} so that, as anticipated: $U_n^{(k)} = k^n$. Similarly, the generating function for the number of ways of throwing n balls into k urns, no urn being empty, is the coefficient $[\frac{z^n}{n!}](e^z-1)^k$; this quantity equals $k!S_{n,k}$, with $S_{n,k}$ a Stirling number of the second kind.

Finally, if $\mathcal{S} = \mathcal{V}^{[*]}$, with $\mathcal{V}$ the class of non-empty urns, then an element of $\mathcal{S}$ of size n corresponds to a *partition* of the set $\{1, 2, \ldots, n\}$ into equivalence classes. Thus

$$\beta_n = \left[\frac{z^n}{n!}\right] \exp(e^z - 1) \tag{16}$$

is the number of such partitions, also known as a Bell number. In the same vein, the EGF of "surjections" (mappings from $\{1, 2, \ldots, n\}$ to an initial segment $\{1, 2, \ldots, m\}$ of the integers, for some $m \geq 1$), corresponding to $\mathcal{V}^{(*)}$ is

$$\widehat{S}(z) = \frac{1}{1-(e^z-1)} = \frac{1}{2-e^z}. \quad \blacksquare \tag{17}$$

1.4. From Generating Functions to Counting

In the previous section we saw how generating function equations can be written directly from structural definitions of combinatorial objects. We discuss here how to go from the functional equations to exact counting results, and then indicate some extensions of the symbolic method to multivariate generating functions.

Direct Expansions from generating functions. Using classical rules for Taylor expansions and sums, products and compositions of generating functions often lead to *explicit forms* for coefficients when a GF is given explicitly. Examples related to previous calculations are the Catalan numbers (2), derangement numbers (14) and Bell numbers (16):

$$[z^n]\frac{1}{\sqrt{1-4z}} = \binom{2n}{n}; \qquad \left[\frac{z^n}{n!}\right]\frac{e^{-z}}{1-z} = n!\sum_{0\le k\le n}\frac{(-1)^k}{k!}; \qquad \left[\frac{z^n}{n!}\right]\exp(e^z-1) = n!e^{-1}\sum_{k\ge 0}\frac{k^n}{k!}.$$

Another method for obtaining coefficients of implicitly defined GFs is the method of indeterminate coefficients. If coefficients of $f(z)$ are sought, we translate over coefficients the functional relation for $f(z)$. An important subcase is that of a first-order linear recurrence $f_{n+1} = a_n + b_n f_n$, whose solution is found by iteration:

$$f_n = a_n + b_n a_{n-1} + b_n b_{n-1} a_{n-2} + b_n b_{n-1} b_{n-2} a_{n-3} + \cdots. \tag{18}$$

Solution Methods for Functional Equations. Algebraic equations over GFs may be solved explicitly if of low degree, and the solutions can then be expanded (see the Catalan numbers in Section 1.1). For equations of higher degrees and some transcendental equations, the *Lagrange-Bürmann* inversion theorem is useful:

THEOREM 6 [Lagrange Bürmann]. *Let $f(z)$ be defined implicitly by the equation $f(z) = z\Phi(f(z))$, where $\Phi(u)$ is a series with $\Phi(0) \neq 0$. Then the coefficients of $f(z)$, its powers $f^k(z)$, and an arbitrary composition $g(f(z))$ are related to the coefficients of the powers of $\Phi(u)$ by:*

$$[z^n]f(z) = \frac{1}{n}[u^{n-1}]\Phi^n(u); \qquad [z^n]f^k(z) = \frac{k}{n}[u^{n-k}]\Phi^n(u); \qquad [z^n]g(f(z)) = \frac{1}{n}[u^{n-1}]\Phi^n(u)g'(u). \tag{19}$$

Thus $f(z) = \sum_{n\ge 1} n^{n-1}\frac{z^n}{n!}$ is the solution to $f(z) = ze^{f(z)}$, and taking coefficients of $e^{\alpha y}e^{\beta y} = e^{(\alpha+\beta)y}$ yields the *Abel identities*:

$$(\alpha+\beta)(n+\alpha+\beta)^{n-1} = \alpha\beta\sum_k \binom{n}{k}(k+\alpha)^{k-1}(n-k+\beta)^{n-k-1}.$$

Letting $b(z) = z + zb^2(z)$ (which is related to $B(z)$ defined in (2b) by $b(z) = zB(z^2)$, see also Section 4.1) and $\Phi(u) = 1+u^2$, we find that $[z^n]B^k(z) = \frac{k}{2n+k}\binom{2n+k}{n}$ (these are the *ballot numbers*).

Differential equations occur especially in relation to binary search trees. For the first-order linear *differential equation*, we have, by the variation-of-parameter method

$$\frac{df(z)}{dz} = a(z) + b(z)f(z) \quad\Longrightarrow\quad f(z) = e^{B(z)}\int_{z_0}^{z} a(t)e^{-B(t)}\,dt, \qquad \text{with } B(z) = \int_0^z b(u)\,du. \tag{20}$$

The lower bound z_0 is chosen to satisfy the initial conditions on $f(z)$.

For other functional equations, *iteration* (or *bootstrapping*) may be useful. For instance, under suitable (formal or analytic) convergence conditions (compare with Eq. 18), with $\gamma^{(k)}(z)$ the kth iterate $\gamma(\gamma(\cdots(\gamma(z))\cdots))$ of $\gamma(z)$, we have

$$f(z) = a(z) + b(z)f(\gamma(z)) \quad\Longrightarrow\quad f(z) = \sum_{k\ge 0}\left(a(\gamma^{(k)}(z))\prod_{0\le j\le k-1} b(\gamma^{(j)}(z))\right). \tag{21}$$

In general, the whole arsenal of algebra can be used on generating functions, and above methods only represent the most commonly used techniques. Many equations still escape exact solution methods but asymptotic methods based on complex analysis can often be used to extract asymptotic information on coefficients.

Multivariate generating functions. If we need to count structures of size n with a certain combinatorial characteristic having value k, we can try to treat k as a parameter (see examples above with

Stirling numbers). Let $g_{n,k}$ be the corresponding counting sequence. We may also consider bivariate generating functions, such as

$$G(u,z) = \sum_{n,k\geq 0} g_{n,k} u^k z^n \qquad \text{or} \qquad G(u,z) = \sum_{n,k\geq 0} g_{n,k} u^k \frac{z^n}{n!}.$$

Extensions of previous translation schemes exist (see [Goulden 1983]). For instance, for the Stirling numbers $s_{n,k}$ and $S_{n,k}$, we have

$$\sum_{n,k\geq 0} s_{n,k} u^k \frac{z^n}{n!} = \exp\bigl(u \log(1-z)^{-1}\bigr) = (1-z)^{-u}; \qquad \sum_{n,k\geq 0} S_{n,k}\, k! u^k \frac{z^n}{n!} = \frac{1}{1-u(e^z-1)}. \tag{22}$$

Multisets. Another extension of the symbolic method to multisets is carried out in [Flajolet 1981]. Consider a class $\mathcal{S}$ of structures, and for each $\sigma \in \mathcal{S}$ a "multiplicity" $\mu(\sigma)$. The pair $(\mathcal{S},\mu)$ is called a *multiset*, and its generating function is by definition $S(z) = \sum_{\sigma\in\mathcal{S}} \mu(\sigma) z^{|\sigma|}$ so that $S_n = [z^n]S(z)$ is the cumulated value of μ over all structures of size n. That extension is useful for obtaining directly generating functions of expected (or cumulated) values of parameters over combinatorial structures, since translation schemes based on admissible constructions also exist for multisets.

We shall encounter such extensions when analyzing Shellsort (Section 3.3), Trees (Section 4) and Hashing (Section 5 1).

2. Asymptotic Methods

In this section, we start with elementary asymptotic methods. Next we present complex asymptotic methods, based on singularity analysis or saddle point integrals, which allow in most cases a direct derivation of asymptotic results for coefficients of generating functions. Then we introduce Mellin tranform techniques that permit asymptotic estimations of a large class of combinatorial sums. We conclude by a discussion of (asymptotic) limit theorems for probability distributions.

2.1. Generalities

We briefly recall in this subsection standard *real analysis* techniques, and next introduce *complex analysis* methods.

Real Analysis. Asymptotic evaluation of the most elementary counting expressions may be effected directly, and a useful formula is *Stirling's formula*:

$$n! \sim n^n e^{-n}\sqrt{2\pi n}\left(1 + \frac{1}{12n} + \frac{23}{288n^2} + \cdots\right). \tag{1}$$

For instance, the central binomial coefficient satisfies $\binom{2n}{n} \sim 4^n/\sqrt{\pi n}$.

The *Euler–Maclaurin* summation formula applies when an expression involves a sum at regularly spaced points (a Riemann sum) of a continuous function: such a sum is approximated by the corresponding integral, and the formula provides a full expansion. The basic form is the following:

THEOREM 1 [Euler Maclaurin summation formula]. *If $g(x)$ is C^∞ over $[0,1]$, then for any integer m,*

$$\frac{g(0)+g(1)}{2} - \int_0^1 g(x)\,dx = \sum_{1\leq j\leq m-1} \frac{B_{2j}}{(2j)!}\bigl(g^{(2j-1)}(1) - g^{(2j-1)}(0)\bigr) - \int_0^1 g^{(2m)}(x)\frac{B_{2m}(x)}{(2m)!}\,dx, \tag{2a}$$

where $B_j(x) \equiv [\frac{z^j}{j!}]ze^{xz}/(e^z-1)$ is a Bernoulli polynomial, and $B_j = B_j(1)$ is a Bernoulli number.

We can derive several formulæ by summing (2a). If $\{x\}$ denotes the fractional part of real x, we find:

$$\sum_{0\leq j\leq n} g(j) - \int_0^n g(x)\,dx = \sum_{1\leq j\leq m-1} \frac{B_{2j}}{(2j)!}\bigl(g^{(2j-1)}(n) - g^{(2j-1)}(0)\bigr) - \int_0^n g^{(2m)}(x)\frac{B_{2m}(\{x\})}{(2m)!}\,dx, \tag{2b}$$

which expresses the difference between a discrete sum and an integral. By a change of scale, for h small, setting $g(x) = f(hx)$, we obtain the asymptotic expansion of a Riemann sum, when the step h tends to 0:

$$\sum_{0\le jh\le 1} f(jh) \sim \frac{1}{h}\int_0^1 f(x)\,dx + \frac{f(0)+f(1)}{2} + \sum_{j=1}^{\infty} \frac{B_{2j}h^{2j-1}}{(2j)!}\left[f^{(2j-1)}(1) - f^{(2j-1)}(0)\right] \qquad (2c)$$

EXAMPLES. 1. The *harmonic numbers* are given by $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$, and they satisfy $H_n = \log n + \gamma + \frac{1}{2n} + \cdots$.

2. The binomial coefficient $\binom{2n}{n-k}$ is, for $k < n^{2/3}$, asymptotically equal to the central coefficient $\binom{2n}{n}$ times $\exp(-k^2/n)$, as follows from estimating its logarithm. This *Gaussian approximation* is a special case of the central limit theorem of probability theory. ∎

Laplace's method for sums is a classical approach for evaluating sums $S_n = \sum_k f(k,n)$ that have a dominant term. First we locate the dominant term. If k_0 is the rank of that term, we can often show for "smooth" functions $f(k,n)$ that $f(n,k) \approx f(n,k_0)\phi\big((k-k_0)h\big)$, with $h = h(n)$ small (like $1/\sqrt{n}$ or $1/n$). We conclude by applying the Euler–Maclaurin summation to $\phi(k)$, and get

$$S_n \sim \frac{f(n,k_0(n))}{h(n)}\int_{-\infty}^{+\infty}\phi(u)\,du.$$

An example is the asymptotics of the Bell numbers defined in (1.16) [De Bruijn 1981, page 108] or the number of involutions (1.15) [Knuth 1973, page 65]. There are extensions to multiple sums involving multivariate Euler-Maclaurin summations.

Complex Analysis. The most powerful method perhaps (and the one that is often computationally simplest), uses *complex analysis* to go directly from a generating function to the asymptotic form of its coefficients. For instance, the EGF for the number of 2–regular graphs [Comtet 1974, page 273] is

$$f(z) = \frac{e^{-z/2-z^2/4}}{\sqrt{1-z}}, \qquad (3)$$

and $[z^n]f(z)$ is sought. A bivariate Laplace method is feasible. It is however simpler to notice that $f(z)$ is regular for complex z, except when $z = 1$. There, a "singular expansion" holds:

$$f(z) \sim \frac{e^{-3/4}}{\sqrt{1-z}}, \qquad \text{as } z \to 1. \qquad (4a)$$

General theorems that we are going to discuss in the next section let us "transfer" an approximation of the function to an approximation of the coefficients:

$$[z^n]f(z) \sim [z^n]\frac{e^{-3/4}}{\sqrt{1-z}}. \qquad (4b)$$

Thus, $[z^n]f(z) \sim e^{-3/4}4^{-n}\binom{2n}{n} \sim e^{-3/4}/\sqrt{\pi n}$.

2.2. Singularity Analysis

A *singularity* is a point at which a function ceases to be analytic. A dominant singularity is one of smallest modulus. It is known that a function with positive coefficients that is not entire always has a dominant positive real singularity. In most cases, the asymptotic behaviour of the coefficients of the function is determined by that singularity.

Location of singularities. The classical exponential-order formula relates the location of singularities of a function to the exponential growth of its coefficients.

THEOREM 2 [Exponential Growth Formula]. *If $f(z)$ is analytic at the origin and has non-negative coefficients, and if ρ is its smallest positive real singularity, then its coefficients $f_n = [z^n]f(z)$ satisfy*

$$(1-\epsilon)^n \rho^{-n} <_{i.o.} f_n <_{a.e.} (1+\epsilon)^n \tag{5}$$

for any $\epsilon > 0$. There, "i.o." means infinitely often (for infinitely many values) and "a.e." means "almost everywhere" (except for finitely many values).

EXAMPLES. 1. Let $f(z) = 1/\cos(z)$ (EGF for "alternating permutations") and $g(z) = 1/(2-e^z)$ (EGF for "surjections"). Then bounds (5) apply with respectively $\rho = \pi/2$ and $\rho = \log 2$.

2. The solution $f(z)$ of the functional equation $f(z) = z + f(z^2 + z^3)$ is the OGF of 2-3 trees [Odlyzko 81]. Setting $\sigma(z) = z^2 + z^3$, that functional equation has the formal solution (obtained by iteration, see Eq. (1.21))

$$f(z) = \sum_{m \geq 0} \sigma^{(m)}(z), \tag{6a}$$

and it can be checked that the sum in (6a) converges geometrically when $|z|$ is less than the smallest positive root ρ of $\rho = \sigma(\rho)$, and it becomes infinite at $z = \rho$. We have $\rho = 1/\phi$ (ϕ the golden ratio); therefore,

$$\left(\frac{1+\sqrt{5}}{2}\right)^n (1-\epsilon)^n <_{i.o.} [z^n]f(z) <_{a.e.} \left(\frac{1+\sqrt{5}}{2}\right)^n (1+\epsilon)^n. \tag{6b}$$

The bound (6b), and even an asymptotic expansion [Odlyzko 1982], is obtainable without an explicit expression for the coefficients. ∎

Nature of singularities. Another way of expressing Theorem 2 is $f_n \sim \theta(n)\rho^{-n}$ where the *subexponential factor* $\theta(n)$ is i.o. larger than any decreasing exponential and a.e. smaller than any increasing exponential. Common forms for $\theta(n)$ are $n^\alpha \log^\beta n$. The subexponential factors are usually related to the growth of the function around its singularity (the singularity may be taken equal to 1 by normalization).

METHOD [Singularity analysis]. *Assume that $f(z)$ has around its dominant singularity 1 an asymptotic expansion of the form*

$$f(z) = \sigma(z) + R(z), \qquad \text{with } R(z) \ll \sigma(z), \qquad \text{as } z \to 1, \tag{7a}$$

where $\sigma(z)$ is amongst a standard set of functions including $(1-z)^a \log^b(1-z)$. Then under general conditions Eq. (7a) leads to

$$[z^n]f(z) = [z^n]\sigma(z) + [z^n]R(z), \qquad \text{with } [z^n]R(z) \ll [z^n]\sigma(z), \qquad \text{as } n \to \infty. \tag{7b}$$

Applications of this principle are based on a variety of conditions on function $f(z)$ or $R(z)$, giving rise to several methods:

1. *Transfer methods* only require growth information on the remainder term $R(z)$ but the approximation has to be established for $z \to 1$ in some region of the complex plane. They largely originate in [Odlyzko 1982]. They are further developed in [Flajolet and Odlyzko 1982] and exposed systematically in [Flajolet and Odlyzko 1987].
2. *Tauberian theorems* only assume Eq. (7a) to hold when z is real < 1, that is as $z \to 1^-$, but require *a priori* Tauberian side conditions (positivity, monotonicity) to be satisfied by coefficients f_n, and are restricted to less general types of growth for $R(z)$. (See [Feller 1966, Vol. 2, page 447] and [Greene and Knuth 1983, page 52] for a combinatorial application).
3. *Darboux's method* assumes smoothness conditions (differentiability) on the remainder term $R(z)$ [Henrici 1977, page 447].

Our transfer method approach is the one that is easiest to apply and the most flexible for combinatorial enumerations. First, we need the asymptotic growth of coefficients of standard singular functions. For $\sigma(z) = (1-z)^{-s}$, by Newton's expansion, the nth Taylor coefficient is $\binom{n+s-1}{s-1}$, which is $\sim n^{s-1}/\Gamma(s)$. For many standard singular functions, like $(1-z)^{-1/2}\log^2(1-z)^{-1}$, we may use either Euler Maclaurin summation on the explicit form of the coefficients or contour integration to find $\sigma_n \sim (\pi n)^{-1/2}\log^2 n$. Next we need to "transfer" coefficients of remainder terms.

THEOREM 3 [Transfer Lemma]. *If $R(z)$ is analytic for $|z| < 1+\delta$ for some $\delta > 0$ (with the possible exception of a sector around $z = 1$, where $|\operatorname{Arg}(z-1)| < \epsilon$ for some $\epsilon < \frac{\pi}{2}$), and $R(z) = O((1-z)^r)$ as $z \to 1$ for some real r, then*

$$[z^n]R(z) = O(n^{-r-1}). \tag{8b}$$

The proof proceeds by choosing a contour of integration made of part of the circle $|z| = 1+\delta$ and the boundary of the sector, except for a small notch of diameter $\frac{1}{n}$ around $z = 1$. Furthermore, when $r \leq -1$, we need only assume the function to be analytic for $|z| \leq 1$, $z \neq 1$.

EXAMPLES. 1. The EGF of 2-regular graphs defined in Eq. (3). We can expand the exponential around $z = 1$ and get

$$f(z) \equiv \frac{e^{z/2+z^2/4}}{\sqrt{1-z}} = e^{-3/4}(1-z)^{-1/2} + O((1-z)^{1/2}), \qquad \text{as } z \to 1. \tag{9a}$$

Function $f(z)$ is analytic in the complex plane slit along $z > 1$, and Eq. (9a) holds there in the vicinity of $z = 1$. Thus, by the transfer lemma with $r = \frac{1}{2}$, we have

$$[z^n]f(z) = e^{-3/4}\binom{n-\frac{1}{2}}{-\frac{1}{2}} + O(n^{-3/2}) = \frac{e^{-3/4}}{\sqrt{\pi}}n^{-1/2} + O(n^{-3/2}). \tag{9b}$$

2. The EGF of surjections is $f(z) = (2-e^z)^{-1}$. It is analytic for $|z| \leq 3$ except for a simple pole at $z = \log 2$, where local expansions show that

$$f(z) = \frac{1}{2\log 2} \cdot \frac{1}{1-z/\log 2} + O(1), \qquad \text{as } z \to \log 2, \tag{10a}$$

so that

$$[z^n]f(z) = \frac{1}{2}\left(\frac{1}{\log 2}\right)^{n+1}\left(1 + O\left(\frac{1}{n}\right)\right). \tag{10b}$$

3. A functional equation. The OGF of certain trees [Polya 1937] $f(z) = 1 + z + z^2 + 2z^3 + \cdots$ is only known via the functional equation

$$f(z) = \frac{1}{1-zf(z^2)}.$$

It can be checked that $f(z)$ is analytic at the origin. Its dominant singularity is a simple pole $\rho < 1$ determined by cancellation of the denominator, $\rho f(\rho^2) = 1$. Around $z = \rho = 0.59475\ldots$, we have

$$f(z) \equiv \frac{1}{\rho f(\rho^2) - zf(z^2)} = \frac{1}{c(\rho - z)} + O(1), \qquad \text{with } c = \frac{d}{dz}zf(z^2)\Big|_{z=\rho}. \tag{11a}$$

Thus, with $K = (c\rho)^{-1} = 0.36071$, we find that

$$[z^n]f(z) = K\rho^{-n}\left(1 + O\left(\frac{1}{n}\right)\right). \tag{11b}$$

More precise expansions exist for coefficients of meromorphic functions (functions with poles only), like the ones in the last two examples (see e.g. [Henrici 1977] and [Flajolet 1985]). For instance, the error of approximation (11*b*) is less than 10^{-15} when $n = 100$. Finally, the OGF of 2-3 trees (6a) is amenable to transfer methods, though extraction of singular expansions is appreciably more difficult [Odlyzko 1982]. ∎

We conclude this subsection by citing the lemma at the heart of Darboux's method [Henrici 1977, page 447] and a classical Tauberian Theorem [Feller 1966, page 447].

THEOREM 4 [Darboux's Method]. *If $R(z)$ is analytic for $|z| < 1$, continuous for $|z| \leq 1$ and d times continuously differentiable over $|z| = 1$, then*

$$[z^n]R(z) = O\left(\frac{1}{n^d}\right). \tag{12}$$

For instance, if $R(z) = (1-z)^{5/2}H(z)$, where $H(z)$ is analytic for $|z| < 1+\delta$, then we can use $d = 2$ for Theorem 4 and obtain $[z^n]R(z) = O(1/n^2)$. The theorem is usually applied to derive expansions of coefficients of functions of the form $f(z) = (1-z)^r H(z)$, with $H(z)$ analytic in a larger domain than $f(z)$. Such functions can however be treated directly by transfer methods (Theorem 3).

THEOREM 5 [Tauberian Theorem of Hardy–Littlewood–Karamata]. *Assume that the function* $f(z) = \sum_{n\geq 0} f_n z^n$ *has radius of convergence* 1 *and satisfies for real* z, $0 \leq z < 1$,

$$f(z) \sim \frac{1}{(1-z)^s} L\left(\frac{1}{1-z}\right), \qquad \text{as } z \to 1^-, \tag{13a}$$

where $s > 0$ *and* $L(u)$ *is a function varying slowly at infinity like* $\log^b(u)$. *If the two "side conditions"*

$$C1:\ f_n \geq 0; \qquad \text{and} \qquad C2:\ \{f_n\} \text{ monotonic;} \tag{13b}$$

are satisfied, then

$$f_n \sim \frac{n^{s-1}}{\Gamma(s)} L(n). \tag{13c}$$

An application to the function $f(z) = \prod_k (1 + \frac{z^k}{k})$ is given in [Greene and Knuth 1983, page 52]; the function represents the EGF of permutations with distinct cycle lengths. That function has a natural boundary at $|z| = 1$ and hence is not amenable to Darboux or transfer methods.

In this chapter, singularity analysis is used extensively for plane tree statistics, partial match, hashing with linear probing.

2.3. Saddle Point Methods

Saddle point methods are used for extracting coefficients of entire functions (such functions having no singularity at a finite distance), and of functions that "grow fast" around their dominant singularities, like $\exp(1/(1-z))$. They are also an important component in obtaining limiting distribution results and exponential tails for discrete probability distributions.

A Simple Bound. Assume that $f(z) = \sum_n f_n z^n$ is entire and has positive coefficients. Then by Cauchy's formula, we have

$$f_n = \frac{1}{2i\pi} \int_\Gamma f(z) \frac{dz}{z^{n+1}}. \tag{14}$$

Take as contour Γ the circle $|z| = R$. Then, a trivial majorization is

$$f_n \leq f(R) R^{-n}, \tag{15}$$

which is valid for *any* $R > 0$. In particular, we have $f_n \leq \min_R \{f(R)R^{-n}\}$. This proves the following:

THEOREM 6 [Saddle point bound]. *If* $f(z)$ *has positive coefficients, then for all* n, *we have*

$$[z^n] f(z) \leq \frac{f(R)}{R^n} \tag{16}$$

where $R = R(n)$ *is the smallest positive root of*

$$\frac{Rf'(R)}{f(R)} = n. \tag{17}$$

Complete Saddle Point Analysis. The saddle point method is a refinement of the previous bounds. It applies in general to integrals depending upon a large parameter of the form $I = \frac{1}{2i\pi} \int_\Gamma e^{h(z)}\, dz$, where for Cauchy integrals (14), $h(z) = h_n(z) = \log f(z) - (n+1)\log z$. A point σ such that $h'(\sigma) = 0$ is called a *saddle point* owing to the topography of $|e^{h(z)}|$ around σ: There are two perpendicular directions at σ, one along which the integrand $|e^{h(z)}|$ has a local minimum at $z = \sigma$; the other (called the *axis* of the saddle point) along which the integrand has a local maximum. The principle of the saddle point method is as follows:

1. Show that the contribution of the integral is asymptotically localized to a fraction Γ_ϵ of the contour around σ traversed along its axis. (This forces ϵ to be not too small).
2. Show that over this subcontour, $h(z)$ is suitably approximated by $h(R) + \frac{(z-R)^2}{2} h''(R)$. (This imposes a conflicting constraint that ϵ should not be too large).

If points 1 and 2 can be established, then I can be approximated by

$$I \approx \frac{1}{2i\pi}\int_{\Gamma_\epsilon} \exp\left(h(\sigma) + \frac{(z-\sigma)^2}{2}h''(\sigma)\right) dz \approx \frac{1}{2i\pi}\frac{e^{h(\sigma)}}{\sigma\sqrt{2\pi h''(\sigma)}}. \tag{18}$$

Classes of functions such that the saddle point applies to Cauchy integrals (14) are called admissible, and have been described by several authors [Hayman 1956], [Harris 1968], [Odlyzko and Richmond 1985].

THEOREM 7 [Saddle point method for Cauchy integrals]. *If $f(z)$ has positive coefficients and is in a class of admissible functions, then*

$$f_n \sim \frac{f(R)}{\sqrt{2\pi C(n)}R^n}, \qquad \text{with } C(n) = \frac{d^2}{dz^2}\log f(z)\Big|_{z=R} + (n+1)R^{-2}, \tag{19}$$

where R is the smallest positive real root of

$$\frac{Rf'(R)}{f(R)} = n+1. \tag{20}$$

EXAMPLES. 1. The number of *involutions* is given by

$$I_n = \left[\frac{z^n}{n!}\right] e^{z+z^2/2} = \frac{n!}{2i\pi}\int_\Gamma e^{z+z^2/2}\frac{dz}{z^{n+1}},$$

and the saddle point is $R = \sqrt{n} + \frac{1}{2} + \frac{5}{8\sqrt{n}} + \cdots$. We choose $\epsilon = n^{-2/5}$, so that for $z = Re^{i\epsilon}$, we have $(z-R)^2h''(R) \to \infty$ while $(z-R)^3h'''(R) \to 0$. Thus,

$$\frac{I_n}{n!} \sim \frac{e^{3/4}}{2\sqrt{\pi}}n^{-n/2}e^{n/8}.$$

The asymptotics of the Bell numbers can be done in the same way [De Bruijn 1981, page 104].

Applications of saddle point methods appear here in the analysis of extendible hashing, coalesced hashing and longest probe sequence problems (Section 5).

2. A function with a finite singularity. For $f(z) = \exp(\frac{z}{1-z})$, we find

$$f_n \equiv [z^n]\exp\left(\frac{z}{1-z}\right) \sim C\frac{e^{d\sqrt{n}}}{n^\alpha}. \tag{21}$$

A similar method can be applied to the integer partition function $p(z) = \prod_{n\geq 1}(1-z^n)^{-1}$ though it has a natural boundary, and estimates (21) are characteristic of functions whose logarithm has a pole–like singularity. ∎

Specializing some of Hayman's results, we can define inductively a class $\mathcal{H}$ of admissible functions as follows: (i) If $p(z)$ denotes an arbitrary polynomial with positive coefficients then $e^{p(z)} \in \mathcal{H}$. (ii) If $f(z)$ and $g(z)$ are arbitrary functions of $\mathcal{H}$, then $e^{f(z)}$, $f(z)\cdot g(z)$, $f(z)+p(z)$ and $p(f(z))$ are also in $\mathcal{H}$.

2.4. Mellin Transforms

The Mellin transform, a tool originally developed for analytic number theory, is useful for analyzing sums where arithmetic functions appear or non-trivial periodicity phenomena occur. Such sums often present themselves as expectations of combinatorial parameters or generating functions.

Basic Properties. Let $f(x)$ be a function defined for real $x \geq 0$. Then its *Mellin transform* is a function $f^*(s)$ of the complex variable s defined by

$$f^*(s) = \int_0^\infty f(x)x^{s-1}\,dx. \tag{22}$$

If $f(x)$ is continuous and is $O(x^\alpha)$ as $x \to 0$ and $O(x^\beta)$ as $x \to \infty$, then its Mellin transform is defined in the "fundamental strip" $-\alpha < \Re(s) < -\beta$, which we denote by $\langle -\alpha; -\beta \rangle$. For instance the Mellin transform of e^{-x} is the well-known *Gamma function* $\Gamma(s)$, with fundamental strip $\langle 0; +\infty \rangle$ and the transform of $\sum_{n \geq k} (-x)^n/n!$ is $\Gamma(s)$ with fundamental strip $\langle -k; -k+1 \rangle$. There is also an inversion theorem à la Fourier:

$$f(x) = \frac{1}{2i\pi} \int_{c-i\infty}^{c+i\infty} f^*(s) x^{-s}\, ds, \tag{23}$$

where c is taken arbitrarily in the fundamental strip.

The important principle for asymptotic analysis is that *under the Mellin transform, there is a correspondence between terms of asymptotic expansions of $f(x)$ at 0 (respectively, $+\infty$) and singularities of $f^*(s)$ in a left (respectively, right) half-plane.* To see why this is so, assume that $f^*(s)$ is small at $\pm i\infty$ and has only polar singularities. Then, we can close the contour of integration in (23) to the left (for $x \to 0$) or to the right for ($x \to \infty$) and derive by Cauchy's residue formula

$$\begin{aligned} f(x) &= +\sum_\sigma \operatorname{Res}\left(f^*(s)x^{-s}; s=\sigma\right) + O(x^{-d}), \qquad \text{as } x \to 0. \\ f(x) &= -\sum_\sigma \operatorname{Res}\left(f^*(s)x^{-s}; s=\sigma\right) + O(x^{-d}), \qquad \text{as } x \to \infty. \end{aligned} \tag{24}$$

The sum in the first equation is extended to all poles σ where $d \leq \Re(\sigma) \leq -\alpha$; the sum in the second equation is extended to all poles σ with $-\beta \leq \Re(\sigma) \leq d$. Those relations have the character of asymptotic expansions of $f(x)$ at 0 and $+\infty$: We observe that if $f^*(s)$ has a kth-order pole at σ, then a residue in (24) is of the form $Q_{k-1}(\log x)x^{-\sigma}$, where $Q_{k-1}(u)$ is a polynomial of degree $k-1$.

There is finally an important functional property of the Mellin transform: If $g(x) = f(\mu x)$, then $g^*(s) = \mu^{-s} f^*(s)$. Hence, transforms of sums (called "harmonic sums") decompose into the product of a generalized Dirichlet series ($\sum \lambda_k \mu_k^s$) and transform of the basis function ($f^*(s)$):

$$F(x) = \sum_k \lambda_k f(\mu_k x) \qquad \Longrightarrow \qquad F^*(s) = \left(\sum_k \lambda_k \mu_k^{-s}\right) f^*(s). \tag{25}$$

Asymptotics of Sums. The standard usage of Mellin transforms devolves from a combination of Eqs. (24) and (25):

THEOREM 8 [Mellin asymptotic summation formula]. *Assume that in (25), the transform $f^*(s)$ of $f(x)$ is small towards $\pm i\infty$ with only polar singularities and that the Dirichlet series is meromorphic of finite order. Then, the asymptotic behaviour of a harmonic sum $F(x) = \sum_k \lambda_k f(\mu_k x)$, as $x \to 0$ (respectively, $x \to \infty$), is given by*

$$\sum_k \lambda_k f(\mu_k x) \sim \pm \sum_\sigma \operatorname{Res}\left(\left(\sum_k \lambda_k \mu_k^{-s}\right) f^*(s)\,;\, s=\sigma\right). \tag{26}$$

For an asymptotic expansion of the sum at 0 (respectively, ∞), the sign in (26) is "+" (respectively, "−"), and the sum is taken over poles to the left (respectively to the right) of the critical strip.

EXAMPLES. 1. An arithmetical sum. Let $F(x)$ be the harmonic sum $\sum_{k \geq 1} d(k) e^{-k^2 x^2}$, where $d(k)$ is the number of divisors of k. Making use of (25) and the fact that the transform of e^{-x} is $\Gamma(s)$, we have

$$F^*(s) = \frac{1}{2}\Gamma\left(\frac{s}{2}\right) \sum_{k \geq 1} d(k) k^{-s} = \frac{1}{2}\Gamma\left(\frac{s}{2}\right) \zeta^2(s), \tag{27a}$$

where $\zeta(s) = \sum_{n \geq 1} n^{-s}$. $F^*(s)$ is defined in the fundamental strip $\langle 1; +\infty \rangle$. To the left of this strip, it has a simple pole at $s = 0$ and a double pole at $s = 1$. By expanding $\Gamma(s)$ and $\zeta(s)$ around $s = 0$ and $s = 1$, we get for any $d > 0$

$$F(x) = -\frac{1}{2}\frac{\log x}{x} + \frac{1}{2}\frac{\gamma}{x} + \frac{1}{4} + O(x^d), \qquad \text{as } x \to 0. \tag{27b}$$

2. A sum with hidden periodicities. Let $F(x)$ be the harmonic sum $\sum_{k\geq 0}(1-e^{-x/2^k})$. The transform $F^*(s)$ is defined in the fundamental strip $\langle -1;0\rangle$, and by (25) we find

$$F^*(s) = -\Gamma(s)\sum_{k\geq 0} 2^{ks} = -\frac{\Gamma(s)}{1-2^s}. \tag{28a}$$

The expansion of $F(x)$ as $x \to \infty$ is determined by the poles of $F^*(s)$ to the right of the fundamental strip. There is a double pole at $s=0$ and the denominator of (28*a*) gives simple poles at $s = \chi_k = 2ik\pi/\log 2$ for $k \neq 0$. Each simple pole χ_k contributes a fluctuating term $x^{-\chi_k} = \exp(2ik\pi \log_2 x)$ to the asymptotic expansion of $F(x)$ toward ∞. Therefore collecting fluctuations, we have

$$F(x) = \log_2 x + P(\log_2 x) + O(x^{-d}) \qquad \text{as } x \to \infty, \tag{28b}$$

where $P(u)$ is a periodic function with period 1 and a convergent Fourier expansion. ∎

Mellin transforms are the primary tool to study tries and radix exchange sort (Section 4.3). They are also useful in the study of certain plane tree algorithms (Section 4.1), bubble sort (Section 3.4), interpolation search and extendible hashing (Section 5.1).

2.5. Limit Probability Distributions

General refernces for this section are [Feller 1965] and [Billingsley 1986]. We recall that if X is a real-valued *random variable* (RV), then its *distribution function* is $F(x) = \Pr\{X \leq x\}$, and its *mean* and *variance* are $\mu \equiv \mu_X = \mathbf{E}\{X\}$ and $\sigma^2 \equiv \sigma_X^2 = \mathbf{E}\{X^2\} - (\mathbf{E}\{X\})^2$. The kth moment of X is $M_k = \mathbf{E}\{X^k\}$. For an integer-valued (or discrete) RV, its *probability generating function* (PGF) is defined by $p(x) = \sum_{k\geq 0} p_k x^k$, where $p_k = \Pr\{X = k\}$; the mean and variance are respectively $\mu = p'(1)$ and $\sigma^2 = p''(1) + p'(1) - (p'(1))^2$. It is well known that the PGF of a sum of independent RVs is the product of their PGFs, and conversely, a product of PGFs corresponds to a sum of independent RVs.

The problem that naturally presents itself in the analysis of algorithms is as follows: Given a class $\mathcal{C}$ of combinatorial structures (such as trees, permutations, etc.), with X_n^* a "parameter" over structures of size n (path length, number of inversions etc.), determine the *limit* (asymptotic) distribution of the normalized variable $X_n = (X_n^* - \mu_n)/\sigma_n$. Such a limit should exist in most cases, as simulations reveal; if available, it usually provides more information than a plain average-case analysis. Main concepts in the area of limit distributions are:

1. Characteristic functions (or Fourier transforms), defined by

$$\phi(t) = \mathbf{E}\{e^{itX}\} = \int_{-\infty}^{+\infty} e^{itx}\, dF(x). \tag{29}$$

   For a discrete RV, we have $\phi(t) = p(e^{it})$.

2. Laplace transforms, defined for positive RVs by

$$g(t) = \mathbf{E}\{e^{-tX}\} = \int_{0}^{+\infty} e^{-tx}\, dF(x). \tag{30}$$

   The Laplace transform is also sometimes called "moment generating function" since it is essentially the EGF of moments. For a discrete RV, we have $g(t) = p(e^{-t})$.

Limit Theorems. Such theorems provide appropriate conditions under which the distribution function $F_n(x)$ of a sequence of RVs X_n converges pointwise to a limit $F(x)$ at each point of continuity of $F(x)$. This phenomenon is known as "weak convergence" or "convergence in distribution" and we write in this case $F = \lim F_n$ [Billingsley 1986].

THEOREM 9 [Continuity Theorem for characteristic functions]. *Let X_n be a sequence of RVs with characteristic functions $\phi_n(t)$ and distribution functions $F_n(x)$. If there exists a function $\phi(t)$*

continuous at the origin such that $\lim \phi_n(t) = \phi(t)$, *then there exists a distribution function* $F(x)$ *such that* $F = \lim F_n$. *Function* $F(x)$ *is the distribution function of the RV with characteristic function* $\phi(t)$.

THEOREM 10 [Continuity Theorem for Laplace transforms]. *Let* X_n *be a sequence of RVs with Laplace transform* $g_n(t)$ *and distribution functions* $F_n(x)$. *If for some* a *and all* t *such that* $|t| \leq a$ *there exists a limit* $g(t) = \lim g_n(t)$, *then there exists a distribution function* $F(x)$ *such that* $F = \lim F_n$. *Function* $F(x)$ *is the distribution function of the RV with Laplace transform* $g(t)$.

Similar limit conditions exist when the moments of the X_n converge to the moments of a RV X, provided the "moment problem" for X has a unique solution. A sufficient condition for this is $\sum_{j\geq 0} \mathbf{E}\{X^{2j}\}^{-1/(2j)} = +\infty$.

Generating functions. If the X_n are discrete RVs, then they define a sequence $p_{n,k} = \Pr\{X_n = k\}$, and the problem is to determine the asymptotic behaviour of the distributions $\pi_n = \{p_{n,k}\}_{k\geq 0}$ (or the associated cumulative distribution functions F_n) as $n \to \infty$. In simple cases, such as binomial distributions, explicit expressions are available and can be treated using the real asymptotic techniques of Section 2.1.

In several cases, either the "horizontal" GFs $p_n(u)$ or "vertical" GFs $q_k(z)$

$$p_n(u) = \sum_{k=0}^{\infty} p_{n,k} u^k, \qquad q_k(z) = \sum_{n=0}^{\infty} p_{n,k} z^n, \tag{31}$$

have explicit expressions, and complex analysis methods can be used to extract their coefficients asymptotically.

Sometimes, only the bivariate generating function

$$P(u,z) = \sum_{n,k\geq 0} p_{n,k} u^k z^n \tag{32}$$

has an explicit form, and a two stage method must be employed.

Univariate Problems. The most well known application of univariate techniques is the *central limit theorem*. If $X_n = A_1 + \cdots + A_n$ is the sum of independent identically distributed RVs with mean 0 and variance 1, then $X_n/\sqrt{n}$ tends to a normal distribution with unit variance. The classical proof [Feller 1966, page 515] uses characteristic functions: the characteristic function of $X_n/\sqrt{n}$, $\phi_n(t) = \phi^n(t/\sqrt{n})$, where $\phi(t)$ is the characteristic function of each A_j, converges to $e^{-t^2/2}$, the characteristic function of the normal distribution.

Another proof [Greene and Knuth 1983] that provides information on the rate of convergence and on densities when the A_j are integer-valued, uses the saddle point method applied to

$$\Pr\{X_n = k\} = \frac{1}{2i\pi} \int_{O+} p^n(z) \frac{dz}{z^k} \tag{33}$$

where $p(z)$ is the PGF of the A_j.

A general principle is that univariate problems can be often solved using either continuity theorems or complex asymptotics (singularity analysis or saddle point) applied to vertical or horizontal generating functions.

EXAMPLES. 1. A horizontal generating function. The probability that a random permutation of n elements has k cycles is $[u^k]p_n(u)$, where

$$p_n(u) = \frac{1}{n!} u(u+1)(u+2)\ldots(u+n-1).$$

Like for the basic central limit theorem above, either characteristic functions or saddle point methods can be used to establish normality of the limiting distribution as $n \to \infty$ (Goncharov's theorem). The same normality result holds for the distribution of inversions in permutations, for which

$$p_n(u) = \frac{1}{n!} \prod_{j=1}^{n} \frac{1-u^j}{1-u}.$$

Inversions will be studied in Section 3.1 in connection with sorting.

2. A vertical generating function. The probability that a random binary string with length n has no "1-run" of length k is $[z^n]q_k(z)$, where

$$q_k(z) = \frac{1-z^k}{1-2z+z^{k+1}}.$$

A singularity analysis (Knuth) can be used: the dominant singularity of $q_k(z)$ is at $z = \varsigma_k \approx 1-2^{-k-1}$, and we have $[z^n]q_k(z) \approx e^{-n/2^{k+1}}$. ∎

Bivariate Problems. For bivariate problems with explicit bivariate generating functions (32), the following two stage approach may be useful. First, treating u as a parameter, and applying singularity analysis or saddle point to the Cauchy integrals, we get

$$p_n(u) = \frac{1}{2i\pi}\int_{O+} P(z,u)\,\frac{dz}{z^{n+1}},$$

whch is often a good approximation to $p_n(u)$.

Second, if u is real and close to 1 ($u = e^{-t}$, with t close to 0), we may be able to conclude using the continuity theorem for Laplace transforms. If u is complex, $|u| = 1$ (that is $u = e^{it}$), we try to conclude using the continuity theorem for characteristic functions. For instance, [Bender 1973] and [Canfield 1977] have obtained general normality results for distributions corresponding to bivariate generating functions of the form

$$\frac{1}{1-ug(z)} \qquad \text{and} \qquad e^{ug(z)}.$$

These results are useful since they correspond to the distribution of the number of components in a sequence (or partitional complex) construct and an abelian partitional complex construct, respectively.

Little is known about bivariate GFs defined only implicitly via non-linear functional equations, a notable exception being [Jacquet and Régnier 1986, 1987]. Finally, other multivariate (but less analytical) techniques are used in the analysis of random graph models [Bollobás 1986].

Part II. Applications

In this part we apply the methods for combinatorial enumeration and asymptotics developed in Part I to the analysis of a large variety of algorithms and data structures. In Section 3, we look at elementary sorting algorithms and related issues involving inversion tables. In Section 4, we extend our admissibility constructions of Sections 1.2 and 1.3 to consider valuations on combinatorial structures, and we apply that to the analysis of trees and structures with a tree-like recursive decomposition. This includes plane trees, binary and multidimensional search trees, digital search trees, quicksort, radix-exchange sort, and algorithms for register allocation, pattern matching, and tree compaction. In Section 5, we present a unified approach to hashing, address calculation techniques, and occupancy problems. Section 6 is devoted to performance measures that span a period of time, such as the expected amortized time and expected maximum space used by an algorithm.

3. Elementary Sorting Algorithms

A permutation of a set of n elements is a 1–1 mapping from the set onto itself. Typically we represent the set of n elements by $\{1, 2, \ldots, n\}$. We use the notation $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ to denote the permutation that assigns i to σ_i, for $1 \le i \le n$. For purposes of average-case analysis, we assume that the input array (or input file) $x[1]$, $x[2]$, ..., $x[n]$ forms a random permutation of the n elements. This is often justified in practice, such as when the key values are generated independently from a common continuous distribution.

3.1. Inversions

The common thread running through most of the analyses of elementary sorting algorithms is the connection between the running time of the algorithm and the fundamental notion of *inversion*. An inversion in permutation σ is an "out of order" pair (σ_k, σ_j) of elements, in which $k < j$ but $\sigma_k > \sigma_j$. The number of inversions is thus a measure of the amount of disorder in a permutation. Let us define the RV I_n to be the number of inversions; the number of inversions in a particular permutation σ is denoted $I_n[\sigma]$. This concept was introduced two centuries ago as a means of computing the determinant of a matrix $A = (A_{i,j})$:

$$\det A = \sum_{\sigma \in S_n} (-1)^{I_n[\sigma]} A_{1,\sigma_1} A_{2,\sigma_2} \ldots A_{n,\sigma_n}, \tag{1}$$

where S_n denotes the set of $n!$ possible permutations.

DEFINITION 1. The *inversion table* of the permutation $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ is the ordered sequence

$$b_1, b_2, \ldots, b_n, \qquad \text{where } b_k = \left|\{1 \le j < \sigma_k^{-1} \mid \sigma_j > k\}\right|. \tag{2a}$$

(Here σ^{-1} denotes the inverse permutation to σ; that is, σ_k^{-1} denotes the index of k in σ.) In other words, b_k is equal to the number of elements in the permutation σ that precede k but have value $> k$.

The number of inversions can be expressed in terms of inversion tables as follows:

$$I_n[\sigma] = \sum_{1 \le k \le n} b_k.$$

Another important RV is the number of *left-to-right minima*, denoted by L_n. For a permutation $\sigma \in S_n$, $L_n[\sigma]$ is the number of elements in p that are less than all the preceding elements in p. In terms of (2a), we have

$$L_n[\sigma] = \left|\{1 \le k \le n \mid b_k = \sigma_k^{-1} - 1\}\right|.$$

It is easy see that there is a 1–1 correspondance between permutations and inversion tables. We can view inversion tables as a cross product

$$\prod_{1 \le k \le n} \{0, 1, \ldots, n-k\} \tag{2b}$$

We associate each inversion table $b_1, b_2, \ldots, b_n$ with the monomial $x_{b_1}x_{b_2}\ldots x_{b_n}$, and we define the generating function

$$F(x_0, x_1, \ldots, x_{n-1}) = \sum_{\sigma\in S_n} x_{b_1}x_{b_2}\ldots x_{b_n}, \tag{3}$$

which is the sum of the monomials among all $n!$ permutations. The following fundamental formula, which follows directly from (2*b*), plays a central rôle in our analyses:

THEOREM 1. *The generating function defined in* (3) *satisfies*

$$F(x_0, x_1, \ldots, x_{n-1}) = x_0(x_0 + x_1)\ldots(x_0 + x_1 + \cdots + x_{n-1}).$$

Theorem 1 is a powerful tool for obtaining statistics related to inversion tables. For example, let us define $I_{n,k}$ to be the number of permutations of n elements having k inversions. By Theorem 1, the OGF $I(z) = \sum_k I_{n,k}z^k$ is given by

$$I_n(z) = z^0(z^0 + z^1)(z^0 + z^1 + z^2)\ldots(z^0 + z^1 + z^{n-1}), \tag{4a}$$

since each monomial $x_{b_1}x_{b_2}\ldots x_{b_n}$ in (3) contributes $z^{b_1+b_2+\cdots+b_n}$ to $I_n(z)$. We can convert (4*a*) into the PGF $\Phi_n(z) = \sum_k \text{Prob}\{I_n = k\}z^k$ by dividing by $|S_n| = n!$:

$$\Phi_n(z) = \sum_k \frac{I_{n,k}}{n!}z^k = \prod_{1\le k\le n} b_k(z), \qquad \text{where } b_k(z) = \frac{z^0 + z^1 + \cdots + z^{k-1}}{k}. \tag{4b}$$

The expected number of inversions $\overline{I_n}$ and the variance $\text{var}(I_n)$ are thus equal to

$$\overline{I_n} = \Phi_n'(1) = \frac{n(n-1)}{4}, \qquad \text{var}(I_n) = \Phi_n''(1) + \Phi_n'(1) - \big(\Phi_n'(1)\big)^2 = \frac{n(2n+5)(n-1)}{72}. \tag{4c}$$

The mean $\overline{I_n}$ is equal to half the worst-case value of I_n. Note from (4*b*) that $\Phi_n(z)$ is a product of individual PGFs $b_k(z)$, which indicates by a remark at the beginning of Section 2.5 that I_n can be expressed as the sum of independent RVs. This suggests another way of looking at the derivation: The decomposition of I_n in question is the obvious one based upon the inversion table (2a); we have $I_n = b_1 + b_2 + \cdots + b_n$, and the PGF of b_k is $b_k(z)$ given above in (4*b*). The formula for $\overline{I_n}$ and $\text{var}(I_n)$ follows by summing $\overline{b_k}$ and $\text{var}(b_k)$, for $1 \le k \le n$. By a generalization of the central limit theorem to sums of independent but nonidentical RVs, it follows that $(I_n - \overline{I_n})/\,\text{st dev}(I_n)$ converges to the normal distribution, as $n \to \infty$.

Similarly, let us define $L_{n,k}$ to be the number of permutations of n elements having k left-to-right minima. By Theorem 1, the OGF $L_n(z) = \sum_k L_{n,k}z^k$ is given by

$$L_n(z) = z(z+1)(z+2)\ldots(z+n-1), \tag{5a}$$

since the contribution to $L_n(z)$ from the x_j term in the kth factor in (3) is z if $j = k-1$ and 1 otherwise. The PGF $\Lambda_n(z) = \sum_k \text{Prob}\{L_n = k\}z^k$ is thus

$$\Lambda_n(z) = \sum_k \frac{L_{n,k}}{n!}z^k = \prod_{1\le k\le n} \ell_k(z), \qquad \text{where } \ell_k(z) = \frac{z+k-1}{k}. \tag{5b}$$

Taking derivatives as above, we get

$$\overline{L_n} = \Lambda_n'(1) = H_n, \qquad \text{var}(I_n) = \Lambda_n''(1) + \Lambda_n'(1) - \big(\Lambda_n'(1)\big)^2 = H_n - H_n^{(2)}, \tag{5c}$$

where H_n is the nth harmonic number $\sum_{1\le k\le n} 1/k$, and $H_n^{(2)} = \sum_{1\le k\le n} 1/k^2$. The mean $\overline{L_n}$ is much less than the worst-case value of L_n, which is n. As above, we can look at this derivation in the way suggested by the product decomposition of $\Lambda_n(z)$ in (5*b*): We can decompose L_n into a sum of independent RVs $\ell_1 + \ell_2 + \cdots + \ell_n$, where $\ell_k[\sigma]$ is 1 if σ_k is a left-to-right minimum, and 0 otherwise. The PGF for ℓ_k is $\ell_k(z)$ given in (5*b*), and summing $\overline{\ell_k}$ and $\text{var}(\ell_k)$ for $1 \le k \le n$ gives (5*c*). The central limit theorem shows that L_n, when normalized, converges to the normal distribution.

The above information about I_n and L_n suffices for our purposes of analyzing sorting algorithms, but it is interesting to point out that Theorem 1 has further applications. For example, let $T_{n,i,j,k}$ be the number of permutations $\sigma \in S_n$ such that $I_n[\sigma] = i$, $L_n[\sigma] = j$, and there are k *left-to-right maxima*. By Theorem 1, the OGF of $T_{n,i,j,k}$ is

$$T_n(x,y,z) = \sum_{i,j,k} T_{n,i,j,k} x^i y^j z^k = yz(y+xz)(y+x+x^2z)\dots(y+x+x^2+\cdots+x^{n-1}z). \quad (6)$$

3.2. Insertion Sort

Insertion sort is the method card players typically use to sort card hands. In the kth loop, for $1 \le k \le n-1$. The first k elements $x[1], \dots, x[k]$ are already in sorted order, and the $(k+1)$st element $x[k+1]$ is inserted into its proper place with respect to the preceding elements.

In the simplest variant, called *straight insertion*, the correct position for $x[k+1]$ is found by successively comparing $x[k+1]$ with $x[k]$, $x[k-1]$,$\dots$ until an element $\le x[k+1]$ is found. The intervening elements are then bumped one position to the right to make room. (For simplicity, we assume that there is a dummy element $x[0]$ with value $-\infty$ so that an element $\le x[k+1]$ is always found.) When the values in the input file are distinct, the number of comparisons in the kth loop is equal to 1 plus the number of elements $> x[k+1]$ that precede $x[k+1]$ in the input. In terms of the inversion table (2a), this is equal to $1 + b_{x[k+1]}$. By summing on k, we find that the total number of comparisons used by straight insertion to sort a permutation σ is $I_n[\sigma] + n - 1$. The following theorem follows directly from (4c):

THEOREM 2. *The mean and the variance of the number of comparisons performed by straight insertion when sorting a random permutation are* $n^2/4 + 3n/4 - 1$ *and* $n(2n+5)(n-1)/72$, *respectively.*

An alternative to straight insertion is to store the already-sorted elements in a binary search tree; the kth loop consists of inserting element $x[k+1]$ into the tree. After all n elements are inserted, the sorted order can be obtained via a preorder traversal. A balanced binary search tree can be used to insure $O(n \log n)$ worst-case time performance, but the overhead of the balancing operations slows down the algorithm in practice. When the tree is unbalanced, there is no overhead, and the average running time is faster. We defer the analysis until our discussion of binary search trees in Section 4.2.

3.3. Shell Sort

The main reason why straight insertion is relatively slow is that the items are inserted sequentially; each comparison reduces the number of inversions (which is $\Theta(n^2)$ on the average) by at most 1. Thus, the average running time is $\Theta(n^2)$. D. L. Shell [1959] proposed an efficient variant (now appropriately called *shellsort*) in which the insertion process is done in several passes of successive refinements. For a given input size n, the passes are determined by an "increment sequence" $(h_t, h_{t-1}, \dots, h_1)$, where $h_t > h_{t-1} > \dots > h_1 = 1$. The h_i pass consists of straight insertion sorts of each of the h_i subfiles

$$\begin{aligned} &\text{subfile } 1: \quad x[1], x[1+h_i], x[1+2h_i], \dots \\ &\text{subfile } 2: \quad x[2], x[2+h_i], x[2+2h_i], \dots \\ &\quad\dots \qquad\qquad \dots \\ &\text{subfile } h_i: \quad x[h_i], x[2h_i], x[3h_i], \dots \end{aligned} \quad (7)$$

In the early passes (when the increments are large), elements can be displaced far from their previous positions with only a few comparisons; the later passes "fine tune" the placement of elements. The last pass, when $h_1 = 1$, consists of a single straight insertion sort of the entire array; we know from Section 3.2 that this is fast when the number of remaining inversions is small.

Two-Ordered Permutations. A good introduction to the average-case analysis of shellsort is the two-pass version with increment sequence $(2, 1)$. We assume that the input is a random permutation. Our measure of complexity is the total number of inversions encountered in the subfiles (7) during the course of the algorithm. For simplicity, we restrict ourselves to the case when n is even.

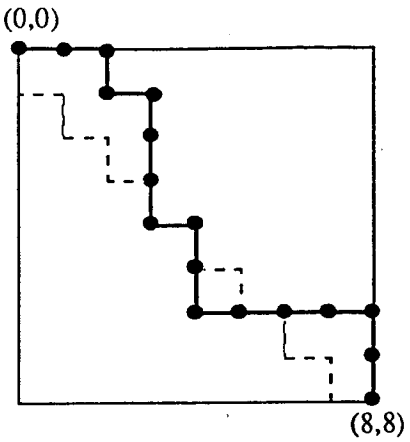

Figure 1. Correspondence between 2-ordered permutations of $2n$ elements and monotone paths from $(0,0)$ to (n,n), for $n = 8$. The dotted path represents the sorted permutation. The dark path corresponds to the permutation $\sigma = (3,1,5,2,6,4,7,8,9,11,10,12,15,13,16,14)$. The number of inversions in σ (namely, 9) is equal to the shaded area between σ's path and the staircase path.

The first pass is easy to analyze, since it consists of two independent straight insertion sorts, each of size $n/2$. We call a permutation *k-ordered* if $x[i] < x[i+k]$, for all $1 \le i \le n-k$. At the end of the first pass, the permutation is 2-ordered, and by our randomness assumption it is easy to see that each of the $\binom{n}{n/2}$ possible 2-ordered permutations is equally likely. The analysis of the last pass consists in determining the average number of inversions in a random 2-ordered permutation.

THEOREM 3. *The mean and the variance of the number of inversions I_{2n} in a random 2-ordered permutation of size $2n$ is*

$$\overline{I_{2n}} = \frac{4^{n-1}}{\binom{2n}{n}} \sim \sqrt{\frac{\pi}{16}} n^{3/2}, \qquad \operatorname{var}(I_{2n}) \sim \left(\frac{7}{30} - \frac{\pi}{16}\right) n^3.$$

PROOF. The starting point of the proof is the important 1–1 correspondance between the set of 2-ordered permutations of $2n$ elements and the set of monotone paths from the upper-left corner $(0,0)$ to the bottom-right corner (n,n) of the n-by-n lattice. he kth step of the path is $\downarrow$ if k appears in an odd position in the permutation, and it is $\rightarrow$ if k appears in an even position. The path for a typical permutation σ is given in Figure 1. The sorted permutation has the "staircase path" shown by dotted lines. The important property of this representation is that *the number $I_{2n}[\sigma]$ of inversions in σ is equal to the area between the staircase path and σ's path.*

There is an easy heuristic argument to show why $\overline{I_{2n}}$ should be $\Theta(n^{3/2})$: Intuitively, each half of a random path from $(0,0)$ to (n,n) is like a random walk. (The transition probabilities are slightly different from a random walk since the complete path must have exactly $n \downarrow$ moves and $n \rightarrow$ moves.) It is well known that random walks tend to be $\Theta(\sqrt{n})$ units away from the diagonal after n steps, thus suggesting that the area between the walk and the staircase path is $\Theta(n^{3/2})$.

An extension to the notions of admissibility in Section 1.2 provides an elegant and precise way to count the area, cumulated among all possible 2-ordered permutations. Let us define $\mathcal{P}$ to be the set of all possible paths of length ≥ 0 from $(0,0)$ to another point on the diagonal, and $\widehat{\mathcal{P}}$ to be the subset of all "arches" (paths that meet the diagonal only at the endpoints). Each path in $\mathcal{P}$ can be decomposed uniquely into a concatenation of one or more arches in $\widehat{\mathcal{P}}$. In the language of Theorem 1.2, $\mathcal{P}$ is the sequence class of $\widehat{\mathcal{P}}$:

$$\mathcal{P} = \widehat{\mathcal{P}}^*. \tag{8a}$$

For example, the path $p \in \mathcal{P}$ in Figure 1 consists of four paths in $\widehat{\mathcal{P}}$: from $(0,0)$ to $(3,3)$, from $(3,3)$ to $(4,4)$, from $(4,4)$ to $(6,6)$, and from $(6,6)$ to $(8,8)$. For reasons of symmetry, it is useful to look at paths that stay to one side (say, the right side) of the diagonal. The restrictions of $\mathcal{P}$ and $\widehat{\mathcal{P}}$ to the right side of the diagonal are denoted $\mathcal{S}$ and $\widehat{\mathcal{S}}$, respectively. As above, we have

$$\mathcal{S} = \widehat{\mathcal{S}}^*. \tag{8b}$$

Each path has the same number of $\downarrow$ moves as $\rightarrow$ moves. We define the *size* of path p, denoted $|p|$, to be the number of $\downarrow$ moves in p. The other parameter of interest is the area between p and the staircase path; we call the area the *weight* of p and denote by $w[p]$. The size and weight functions are linear; that is, $|p \parallel q| = |p| + |q|$ and $w[p \parallel q] = w[p] + w[q]$, where $p \parallel q$ denotes the concatenation of paths p and q.

Let $P_{n,k}$ (respectively, $\widehat{P}_{n,k}$, $S_{n,k}$, $\widehat{S}_{n,k}$) be the number of paths $p \in \mathcal{P}$ (respectively, $\widehat{\mathcal{P}}$, $\mathcal{S}$, $\widehat{\mathcal{S}}$) such that $|p| = n$ and $w[p] = k$. We define the OGF

$$P(u,z) = \sum_{k,n} P_{n,k} u^k z^n, \tag{9}$$

and we define the OGFs $\widehat{P}(u,z)$, $S(u,z)$, and $\widehat{S}(u,z)$ similarly. The mean and variance of I_{2n} can be expressed readily in terms of $P(u,z)$:

$$\overline{I_{2n}} = \frac{1}{\binom{2n}{n}} [z^n] \frac{\partial P(1,z)}{\partial u}; \tag{10a}$$

$$\overline{I_{2n}(I_{2n}-1)} = \frac{1}{\binom{2n}{n}} [z^n] \frac{\partial^2 P(1,z)}{\partial u^2}; \tag{10b}$$

$$\operatorname{var}(I_{2n}) = \overline{I_{2n}(I_{2n}-1)} + \overline{I_{2n}} - (\overline{I_{2n}})^2. \tag{10c}$$

We can generalize our admissibility aproach in Theorem 1.2 to handle OGFs with two variables:

LEMMA 1. *The sequence construct is admissible with respect to the size and weight functions:*

$$\mathcal{P} = \widehat{\mathcal{P}}^* \quad \Longrightarrow \quad P(u,z) = \frac{1}{1 - \widehat{P}(u,z)}; \tag{11}$$

$$\mathcal{S} = \widehat{\mathcal{S}}^* \quad \Longrightarrow \quad S(u,z) = \frac{1}{1 - \widehat{S}(u,z)}; \tag{12}$$

PROOF OF LEMMA 1. A equivalent definition of the OGF $P(u,z)$ is

$$P(u,z) = \sum_{p \in \mathcal{P}} u^{w[p]} z^{|p|}.$$

Each nontrivial path $p \in \mathcal{P}$ can be decomposed uniquely into a concatenation $\hat{p}_1 \parallel \hat{p}_2 \parallel \ldots \parallel \hat{p}_\ell$ of nontrivial paths in $\widehat{\mathcal{P}}$. By the linearity of the size and weight functions, we have

$$\begin{aligned} P(u,z) &= \sum_{\substack{\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_\ell \in \widehat{\mathcal{P}} \\ \ell \ge 0}} u^{w[\hat{p}_1] + w[\hat{p}_2] + \cdots + w[\hat{p}_\ell]} z^{|\hat{p}_1| + |\hat{p}_2| + \cdots + |\hat{p}_\ell|} \\ &= \sum_{\ell \ge 0} \Bigg(\sum_{\hat{p} \in \widehat{\mathcal{P}}} u^{w[\hat{p}]} z^{|\hat{p}|} \Bigg)^\ell = \sum_{\ell \ge 0} \left(\widehat{P}(u,z) \right)^\ell = \frac{1}{1 - \widehat{P}(u,z)}. \end{aligned}$$

The proof of (12) is identical. ∎

Lemma 1 gives us two formulæ relating the four OGFs $P(u,z)$, $\widehat{P}(u,z)$, $S(u,z)$, and $\widehat{S}(u,z)$. The following decomposition gives us another two relations, which closes the cycle and lets us solve for the OGFs: Every path $\hat{s} \in \widehat{\mathcal{S}}$ can be decomposed uniquely into the path $\rightarrow \|s\| \downarrow$, for some $s \in \mathcal{S}$, and the size and weight funcitons of $\hat{s}$ and s are related by $|\hat{s}| = |s| + 1$ and $w[\hat{s}] = w[s] + |s| + 1$. Hence, we have

$$\widehat{S}(u,z) = \sum_{s \in S} u^{w[s] + |s| + 1} z^{|s|+1} = uz \sum_{s \in S} u^{w[s]} (uz)^{|s|} = uz S(u, uz). \tag{13}$$

Each path in $\widehat{\mathcal{P}}$ is either in $\widehat{\mathcal{S}}$ or in the reflection of $\widehat{\mathcal{S}}$ about the diagonal, which we call $\operatorname{refl}(\widehat{\mathcal{S}})$. For $\hat{s} \in \widehat{\mathcal{S}}$, we have $|\operatorname{refl}(\hat{s})| = |\hat{s}|$ and $w[\operatorname{refl}(\hat{s})] = w[\hat{s}] - |\hat{s}|$, which gives us

$$\widehat{P}(u,z) = \sum_{\hat{s} \in \widehat{\mathcal{S}} \cup \operatorname{refl}(\widehat{\mathcal{S}})} u^{w[\hat{s}]} z^{|\hat{s}|} = \sum_{\hat{s} \in S} \left(u^{w[\hat{s}]} z^{|\hat{s}|} + u^{w[\hat{s}] - |\hat{s}|} z^{|\hat{s}|} \right) = \widehat{S}(u,z) + \widehat{S}\left(u, \frac{z}{u}\right). \tag{14}$$

Equations (13) and (14) can be viewed as types of admissibility reductions, similar to those in Lemma 1, except that in this case the weight functions are not linear (the relations between the weight functions involve the size function), thus introducing the uz and z/u arguments in the right-hand sides of (13) and (14).

The four relations (11)–(14) allow us to solve for $P(u,z)$. Substituting (13) into (12) gives

$$S(u,z) = uzS(u,z)S(u,uz) + 1, \tag{15}$$

and substituting (13) into (14) and the result into (11), we get

$$P(u,z) = \big(uzS(u,uz) + zS(u,z)\big)P(u,z) + 1. \tag{16}$$

Using (16), we can then express $\frac{\partial}{\partial u}P(1,z)$ and $\frac{\partial^2}{\partial u^2}P(1,z)$, which we need for (10), in terms of derivatives of $S(u,z)$ evaluated at $u=1$, which in turn can be calculated from (15). These calculations are straightforward, but are best done with an symbolic algebra system. ∎

ALTERNATE PROOF. We can prove Theorem 3 in a less elegant way by studying how the file is decomposed after the first pass into the two sorted subfiles

$$X_1 = x[1], x[3], x[5], \ldots \qquad \text{and} \qquad X_2 = x[2], x[4], x[6], \ldots.$$

We can express $\overline{I_{2n}}$ as

$$\overline{I_{2n}} = \frac{1}{\binom{2n}{n}} \sum_{\substack{1\le i\le n \\ 0\le j\le n-1}} A_{i,j}, \tag{17a}$$

where $A_{i,j}$ is the total number of inversions involving the ith element of X_1 (namely, $x[2i-1]$), among all 2-ordered permutations in which there are j elements in X_2 less than $x[2i-1]$. The total number of such 2-ordered permutations is

$$\binom{i+j-1}{i-1}\binom{2n-i-j}{n-j},$$

and a simple caluculation shows that each contributes $|i-j|$ inversions to $A_{i,j}$. Substituting this into (17a), we get

$$\overline{I_{2n}} = \frac{1}{\binom{2n}{n}} \sum_{\substack{1\le i\le n \\ 0\le j\le n-1}} |i-j|\binom{i+j-1}{i-1}\binom{2n-i-j}{n-j}, \tag{17b}$$

and the rest of the derivation consists of manipulation of binomial coefficients. ∎

The increment sequence $(2,1)$ is not very interesting in practice, because the first pass still takes quadratic time, on the average. We can generalize the above derivation to show that the average number $\overline{I_n}$ of inversions in a random h-ordered permutation is

$$\overline{I_n} = \frac{2^{2q-1}q!q!}{(2q+1)!}\left(\binom{h}{2}q(q+1) + \binom{r}{2}(q+1) - \binom{h-r}{2}\frac{q}{2}\right), \tag{18}$$

where $q = \lfloor n/h \rfloor$ and $r = n \bmod h$. This allows us to determine (for large h and larger n) the best two-pass increment sequence $(h,1)$. In the first pass, there are h insertion sorts of size $\sim n/h$; by (4c), the total number of inversions in the subfiles is $\sim n^2/(4h)$. By (18), we can approximate the number of inversions encountered in the second pass by $\overline{I_n} \sim \frac{1}{8}\sqrt{\pi h}n^{3/2}$. The total number of inversions in both passes is thus $\sim n^2/(4h) + \frac{1}{8}\sqrt{\pi h}n^{3/2}$, which is minimized when $h \sim (16n/\pi)^{1/3}$. The resulting running time is $O(n^{5/3})$.

When there are more than two passes, an interesting phemonenon occurs: *when an h-sorted file is k-sorted, the file remains h-sorted.* Yao [1980] shows how to combine this fact with an extension of the approach used in (17a) and (17b) to analyze increments of the form $(h,k,1)$, for constant values h and k. Not much else is known in the average case, except when each increment is a multiple of the next. In that case, the running time can be reduced to $O(n^{1.5+\epsilon/2})$, where $\epsilon = 1/(2^t-1)$ and t is the number of increments.

V. Pratt discovered that we get an $O(n \log^2 n)$-time algorithm in the worst case if we use all increments of the form $2^p 3^q$, for $p, q \geq 0$, in decreasing order. This particular approach is not typically used in practice because the list of increments is hard to generate, and the number of increments (and thus the number of passes) is $O(\log^2 n)$. Sequences with only $O(\log n)$ increments that result in a $O(n^{1+\epsilon})$ running time are reported in [Incerpi and Sedgewick, 1985].

One possible application of shellsort is to the construction of sorting networks, which sort through a sequence of pairwise comparison/exchanges, where the choice of which pair of elements to compare next cannot be based on the outcomes of previous comparisons. Recently, Ajtai, Komlós, and Szemerédi [1983] solved a longstanding open problem by constructing a sorting network that works in $O(\log n)$ parallel steps. Their result is a theoretical breakthrough, but in terms of practicality the network is not very interesting since the coefficient implicit in the big-oh term is huge. If an $O(n \log n)$-time shellsort is found, it is possible that it can be modified to yield a sorting network of depth $O(\log n)$, thus providing a practical design for an optimal sorting network.

3.4. Bubble Sort

The bubble sort algorithm works by a series of passes. In each pass, some final portion $x[\ell+1], x[\ell+2], \ldots, x[n]$ of the array is already known to be in sorted order, and the largest element in the initial part of the array $x[1], x[2], \ldots, x[\ell]$ is "bubbled" to the right by a sequence of $\ell - 1$ pairwise comparisons

$$x[1] : x[2], \quad x[2] : x[3], \quad \ldots, \quad x[\ell-1] : x[\ell].$$

In each comparison, the elements exchange place if they are out of order. The value of ℓ is reset to the largest t such that the comparison $x[t] : x[t+1]$ required an exchange, and then the next pass starts.

The most interesting aspect of bubble sort is its analysis. The algorithm itself is not very useful in practice, since it runs slower than insertion sort and selection sort, yet is more complicated to program. Bubble sort's running time depends upon three quantities: the number of inversions I_n, the number of passes A_n, and the number of comparisons C_n. The analysis of I_n has already been given in Section 3.1.

THEOREM 4 [Knuth, 1973b]. *The average number of passes and comparisons done in bubble sort, on a random permutation of size n, is*

$$\overline{A_n} = n - \sqrt{\frac{\pi n}{2}} + O(1), \qquad \overline{C_n} = \frac{1}{2}\left(n^2 - n \log n - (\gamma + \log 2 - 1)n\right) + O(\sqrt{n}).$$

PROOF. Each pass in bubble sort reduces all the nonzero entries in the inversion table by 1. There are at most k passes in the algorithm when each entry in the original inversion table is $\leq k$. The number of such inversion tables can be obtained via Theorem 1 by substituting $x_i = \delta_{i \leq k}$ into $F(x_0, x_1, \ldots, x_n)$, which gives $k!k^{n-k}$. Plugging this into the definition of expected value, we get

$$\overline{A_n} = n + 1 - \frac{1}{n!} \sum_{0 \leq k \leq n} k! k^{n-k}. \tag{19}$$

The summation can be shown to be equal to $\sqrt{\pi n/2} - 2/3 + O(1/\sqrt{n})$ by an application of Euler's summation formula.

The average number of comparisons $\overline{C_n}$ can be determined in a similar way. For the moment, let us restrict our attention to the jth pass. Let c_j be the number of comparisons done in the jth pass. We have $c_j = \ell - 1$, where ℓ is the upper index of the subarray processed in pass j. We can also characterize ℓ as the last position in the array at the beginning of pass j that contains an element which moved to the left one slot during the previous pass. We denote the value of the element in position ℓ by i. It follows that all the elements in positions $\ell+1, \ell+2, \ldots, n$ have value $> i$. We noted earlier that each nonzero entry in the inversion table decreases by 1 in each pass. Therefore, the number of inversions for element i at the beginning of the jth pass is $b_i - j + 1$, and element i is located in array position $\ell = i + b_i - j + 1$. This gives us $c_j = i + b_i - j$.

Without a *priori* knowledge of ℓ or i, we can calculate c_j by using the formula

$$c_j = \max_{1 \leq i \leq n} \{i + b_i - j \mid b_i \geq j - 1\}. \tag{20}$$

The condition $b_i \geq j-1$ restricts attention to those elements that move left one place in pass $j-1$; it is easy to see that the term in (20) is maximized at the correct i. To make use of (20), let us define $f_j(k)$ to be the number of inversion tables (2a) such that either $b_i < j-1$ or $i+b_i-j \leq k$. We can evaluate $f_j(k)$ from Theorem 1 by substituting $x_i = 1$ if $b_i < j-1$ or $i+b_i-j \leq k$, and 0 otherwise. A simple calculation gives

$$f_j(k) = (j+k)!(j-1)^{n-j+k}, \qquad \text{for } 0 \leq k \leq n-j.$$

By the definition of expected value, we have

$$\begin{aligned}\overline{C_j} &= \frac{1}{n!} \sum_{\substack{1\leq j\leq n\\ 0\leq k\leq n-j}} k(f_j(k) - f_j(k-1)) \\ &= \binom{n+1}{2} - \frac{1}{n!} \sum_{\substack{1\leq j\leq n\\ 0\leq k\leq n-j}} f_j(k) \\ &= \binom{n+1}{2} - \frac{1}{n!} \sum_{0\leq r<s\leq n} s!r^{n-s}. \end{aligned} \tag{21a}$$

The intermediate step follows by summation by parts. The summation in (21*a*) can be simplified using Euler's summation formula into series of sums of the form

$$\frac{1}{m!} \sum_{1\leq t<m} (m-t)!(m-t)^t t^q, \qquad \text{for } q \geq -1, \tag{21b}$$

where $m = n+1$. By applying Stirling's approximation, we find that the summand in (21*b*) decreases exponentially when $t > m^{1/2+\epsilon}$, and we can reduce the problem further to that of approximating $F_q(1/m)$, where

$$F_q(x) = \sum_{k\geq 1} e^{-k^2x/2} k^q, \qquad \text{for } q \geq -1, \tag{22a}$$

as $m \to \infty$. The derivation can be carried out via Euler's summation formula, but things get complicated for the case $q = -1$. A more elegant derivation is to use Mellin transforms. The function $F_q(x)$ is a harmonic sum, and its Mellin transform $F_q^*(s)$ is

$$F_q^*(s) = \Gamma(s)\zeta(2s-q)2^s, \tag{22b}$$

defined in the fundamental strip $\langle 0; \infty\rangle$. The asymptotic expansion of $F_q(1/m)$ follows by computing the residues in the left half-plane $\Re(s) \leq 0$. There are simple poles at $s = -1, -2, \ldots$ because of the term $\Gamma(s)$. When $q = -1$, the $\Gamma(s)$ and $\zeta(2s-q)$ terms combine to contribute a double pole at $s = 0$. When $q \geq 0$, $\Gamma(s)$ contributes a simple pole at $s = 0$, and $\zeta(2s-q)$ yields a simple pole at $s = (q+1)/2$. Putting everything together, we find that

$$\frac{1}{n!} \sum_{0\leq r<s\leq n} s!r^{n-s} = \frac{1}{2} n \log n + \frac{1}{2}(\gamma + \log 2)n + O(\sqrt{n}). \tag{23}$$

The formula for $\overline{C_n}$ follows immediately. ∎

After k passes in the bubble sort algorithm, the number of elements known to be in their final place is typically larger than k; the variable ℓ is always set to be as small as possible so as to minimize the size of the array that must be considered in a pass. The fact that (23) is $o(n^2)$ implies that the number of comparisons is hardly less than that of the more naïve algorithm in which the bubbling process in the kth pass is done on the subarray $x[1], x[2], \ldots, x[n-k+1]$.

3.5. Quicksort

We can get much more efficient exchange-based sorting algorithms using a divide-and-conquer approach. In the *quicksort* method, due to C. A. R. Hoare, an element s is chosen (say, the first element

in the file), the file is partitioned into the part $\leq s$ and the part $> s$, and then each half is sorted recursively. The recursive decomposition can be analyzed using the sophisticated tools we develop in Section 4.2 for binary search trees, which have a similar decomposition, and so we defer the analysis until Section 4.2. The expected number of comparisons is $\sim 2n \log n$. Good references for analysis techniques and results for quicksort can be found in [Knuth 1973b] and [Sedgewick 1977b].

Quicksort is an extremely good general-purpose sorting routine. A big drawback of the version described above is its worst-case performance: it requires $\Theta(n^2)$ time to sort a file that is already sorted in either ascending or descending order! In many computer installations, the resident sort routine is often used (in a wasteful way) to *verify* that sorted files are really sorted, and in this case quicksort will perform poorly. A good way to guard against guaranteed bad behavior is to choose the partioning element s to be a random element in the current subfile, in which case our above analysis applies. Another good method is to choose s to be the median of three elements from the subfile (say, the first, middle, and last elements). This also has the effect of reducing the number of comparisons to $\sim \frac{12}{7} n \log n$.

If the smaller of the two subfiles is always processed first after each partition, then the recursion stack contains at most $\log n$ entries. But by clever programming, we can simulate the stack with only a constant amount of space, at a very slight increase in computing time [Ďurian 1986]. The analysis of quicksort in the presence of some equal keys is given in [Sedgewick 1977a].

3.6. Radix-Exchange Sort

The *radix-exchange sort* algorithm is an exchange-based algorithm that uses divide-and-conquer in a different way than quicksort. The recursive decomposition is based on the individual bits of the keys. In pass k, the keys are partitioned into two groups: the group whose kth least significant bit is 0, and the group whose kth least significant bit is 1. The partitioning is done in a "stable" manner so that the relative order of the keys within each group is the same as before the partitioning. Then the 1-group is appended to the 0-group, and the next pass begins. After t passes, where t is the number of bits in the keys, the algorithm terminates.

In this case, the recursive decomposition is identical to radix-exchange tries, which we shall study in a general context Section 4.3, and the statistics of interest for radix-exchange sorting can be expressed directly in terms of corresponding parameters of tries. We defer the details to Section 4.3.

3.7. Selection Sort

Selection sort in some respects is the inverse of insertion sort, because the order in which the elements are processed is based upon the output rather than upon the input. In the kth pass, for $1 \leq k \leq n-1$, the kth smallest element is selected and is put into its final place $x[i]$.

Straight Selection. In the simplest variant, called *straight selection sort*, the kth smallest element is found by a sequential scan of $x[k]$, $x[k+1]$, ..., $x[n]$, and it changes places with the current $x[k]$. Unlike insertion sort, the algorithm is not stable; that is, two elements with the same value might be output in the reverse order that they appear in the input.

The number of comparisons performed is always

$$C_n = (n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2}.$$

The number of times a new minimum is found (the number of data movements) in the kth pass is the number of left-to-right minima L_{n-k+1} encountered in $x[k]$, $x[k+1]$, ..., $x[n]$, minus 1. All permutations of $x[k]$, $x[k+1]$, ..., $x[n]$ are equally likely. By (5c), the average number of updates of the minimum over the course of the algorithm is

$$\sum_{1 \leq k \leq n-1} (\overline{L_{n-k+1}} - 1) = (n+1)H_n - 2n = n \log n + (\gamma - 2)n + \log n + O(1).$$

The variance, which was recently determined by Yao [1987], is much more difficult to compute, since the contributions L_{n-k+1} from the individual passes are not independent.

Heapsort. A more sophisticated way of selecting the minimum, called *heapsort*, due to J. W. J. Williams, is based upon the notion of tournament elimination. The $n-k+1$ elements to consider in the kth pass are stored in a heap priority queue. A heap is a tree in which the value of the root of each subtree is $\leq$ the values of the other elements in the subtree. In particular, the smallest element is always at the root. The heaps we use for heapsort have the nice property that the tree is always perfectly balanced, except possibly for the rightmost part of the last level. This allows us to represent the heap without need of pointers as an array h: the children of element $h[i]$ are stored in $h[2i]$ nd $h[2i+1]$, for all $1 \leq i \leq \lfloor (n-i+1)/2 \rfloor$.

The kth pass of heapsort consists of outputing $h[1]$ and deleting it from the array; the element stored in $h[n-k+1]$ is then moved to $h[1]$, and it is "filtered" down to its correct place in $O(\log n)$ time. The creation of the initial heap can be done in linear time. The worst-case time to sort the n elements is thus $O(n \log n)$.

In the average case, the analysis is complicated by a lack of randomness: the heap at the start of the kth pass, for $k \geq 2$, is not a random heap of $n-k+1$ elements.

3.8. Merge Sort

The first sorting program ever executed on an electronic computer used the following divide-and-conquer approach, known as merge sort: The file is split into two subfiles of equal size (or nearly equal size), each subfile is sorted recursively, and then the two sorted subfiles are merged together in the obvious linear fashion. When done from bottom-up, the algorithm consists of several passes; in each pass, the sorted subfiles are paired off, and each pair is merged together.

The linear nature of the merging process makes it ideal for input files in the form of a linked list and for external sorting applications, in which the file does not fit entirely in internal memory and must instead be stored in external memory (like disk or tape), which is best accessed in a sequential manner. Typically, the merge is of a higher order than 2; for example, four subfiles at a time might be merged together, rather than just two. Considerations other than the number of comparisons, such as the rewind time on tapes and the seek time on disks, affect the running time. An encyclopedic collection of variants of merge sort and their analyses appears in [Knuth, 1973b].

For simplicity, we restrict our attention to the number of comparisons performed during a binary (order-2) merge sort, when $n = 2^j$, for some $j \geq 0$. All the comparisons take place during the merges. For each $0 \leq k \leq j-1$, there are 2^k merges of pairs of sorted subfiles, each subfile of size $n/2^{k+1} = 2^{j-k-1}$. If we assume that all $n!$ permutation are equally likely, it is easy to see that, as far as relative order is concerned, the two subfiles in each merge form a random 2-ordered permutation, independent of the other merges. The number of comparisons to merge two random sorted subfiles of length p and q is $p+q-\mu$, where μ is the number of elements remaining to be output in one subfile when the other subfile becomes exhausted. The probability that $\mu \geq s$, for $s \geq 1$, is the probability that the s largest elements are in the same subfile, namely,

$$\frac{p^{\underline{s}}}{(p+q)^{\underline{s}}} + \frac{q^{\underline{s}}}{(p+q)^{\underline{s}}},$$

where $a^{\underline{b}}$ denotes the "falling power" $a(a-1)\ldots(a-b+1)$. Hence, we have

$$\overline{\mu} = \sum_{s \geq 1} \left(\frac{p^{\underline{s}}}{(p+q)^{\underline{s}}} + \frac{q^{\underline{s}}}{(p+q)^{\underline{s}}} \right) = \frac{p}{q+1} + \frac{q}{p+1}. \tag{24}$$

By (24), the total number of comparisons used by merge sort, on the average, is

$$\overline{C_n} = j2^j - \sum_{1 \leq k \leq j-1} 2^k \frac{2^{j-k}}{2^{j-k-1}+1} = n \log_2 n - \alpha n + O(1),$$

where

$$\alpha = \sum_{n \geq 0} \frac{1}{2^n+1} = 1.2645\ldots.$$

The analysis when n is not a power 2 involves arithmetic functions based upon the binary representation of n and can be found in [Knuth, 1973b]. Batcher's odd-even merge is analyzed in [Sedgewick 1978].

4. Recursive Decompositions and Algorithms on Trees

In this section we develop a uniform framework for obtaining average-case statistics on four classes of trees—binary and plane trees, binary search trees, radix-exchange tries, and digital search trees. Our statistics, which include number of enumerations, number of nodes, height, and path length, have numerous applications to the analysis of tree-based searching and symbolic processing algorithms, as well as to the analysis of sorting methods not covered in Section 2, such as quicksort, binary tree sort, and radix-exchange sort. Our approach has two parts:

1. Each of the four classes of trees has a recursive formulation that lends itself naturally to the symbolic generating function method described in Part I. The statistic of interest for each tree t corresponds naturally to a *valuation function* (VF) $v[t]$. The key idea which unifies our analyses is an extension of the admissible concept of Part I: A recursive definition of the VF translates directly into a functional equation involving the generating function. The type of generating function used (either OGF or EGF) and the type of functional equation that results depend upon the particular nature of the recursion.
2. We determine the coefficients of the GF based upon the functional equation resulting from step 1. Sometimes an explicit closed form is obtained, but typically we apply the asymptotic methods of Part I, our particular approach depending upon the nature of the equation.

4.1. Binary Trees and Plane Trees

Binary trees and plane trees can be used to represent symbolic expressions and recursive structures and have immense combinatorial significance. This section studies statistical models under which all trees of a given size are equally likely. Such models are not applicable to binary search trees or digital search trees, which we cover in Sections 4.2, 4.3, and 4.4, but when slightly enriched, they provide good models for algorithms operating on expression trees, term trees, or Lisp structures [Clark 1979].

We begin by considering the class $\mathcal{B}$ of *binary trees* defined in Section 1.1:

$$\mathcal{B} = \{\blacksquare\} + (\{\circ\} \times \mathcal{B} \times \mathcal{B}), \tag{1}$$

where $\blacksquare$ represents an external node and $\circ$ an internal node. The size of a binary tree is the number of internal nodes in the tree.

The cartesian product decomposition in (1) suggests that we represent our statistic of interest via an OGF. Further motivation for this choice is given in Eqs. (1.2) and (1.3). We use $v[t]$ to represent the valuation function v applied to tree t. We define v_n to be its cumulated value $\sum_{|t|=n} v[t]$ among all trees of size n, and $v(z)$ to be the OGF $\sum_{n\geq 0} v_n z^n$. The recursive decomposition of $\mathcal{B}$ leads directly to the following fundamental relations:

THEOREM 1. *The sum and recursive product valuation functions are admissible for the class $\mathcal{B}$ of binary trees:*

$$\begin{aligned} v[t] = u[t] + w[t] \quad &\Longrightarrow \quad v(z) = u(z) + w(z); \\ v[t] = u[t_{\text{left}}] \cdot w[t_{\text{right}}] \quad &\Longrightarrow \quad v(z) = z \cdot u(z) \cdot w(z), \end{aligned}$$

where t_{left} and t_{right} denote the left and right subtrees of t.

The proof is similar to that of Theorem 1.1. The importance of Theorem 1 is due to the fact that it provides an automatic translation from VF to OGF, for all VFs of interest.

EXAMPLES. 1. *Enumeration.* The standard trick we shall use throughout this section for counting the number of trees of size n in a certain class is to use the unit VF $I[t] \equiv 1$. For example, let B_n be the number of binary trees with n internal nodes, $n \geq 0$. By our definitions above, B_n is simply equal to I_n, and thus the OGF $B(z)$ is equal to $I(z)$. We can get another derivation of $I(z)$ via Theorem 1 if we use the following recursive definition of $I[t]$,

$$I[t] = \delta_{|t|=0} + I[t_{\text{left}}] \cdot I[t_{\text{right}}], \tag{2a}$$

which is a composition of the two types of VF expressions handled by Theorem 1. Since $B_0 = 1$, the OGF for $\delta_{|t|=0}$ is 1. Theorem 1 translates (2*a*) directly into

$$I(z) = 1 + zI(z)^2. \tag{2b}$$

The solution $B(z) = I(z) = \frac{1}{2z}(1 - \sqrt{1-4z})$ follows directly. By expanding coefficients, we get $B_n = \frac{1}{n+1}\binom{2n}{n}$, as in Section 1.1.

2. *Internal Path Length.* The standard recursive tree traversal algorithm uses a stack to keep track of the ancestors of the current node in the traversal. The average stack size, amortized over the course of the traversal, is related to the internal path length of the tree, divided by n. The VF corresponding to the cumulated path lengths among all binary trees with n nodes can be expressed in the following form suitable for Theorem 1:

$$\begin{aligned} p[t] &= |t| - 1 + \delta_{|t|=0} + p[t_{\text{left}}] + p[t_{\text{right}}] \\ &= |t| - 1 + \delta_{|t|=0} + p[t_{\text{left}}] \cdot I[t_{\text{right}}] + I[t_{\text{left}}] \cdot p[t_{\text{right}}]. \end{aligned} \tag{3a}$$

We computed the OGFs for $I[t] \equiv 1$ and $\delta_{|t|=0}$ in the last example, and the OGF for the size VF $S[t] = |t|$ is easily seen to be $S(z) = \sum_{n\geq 0} nB_n z^n = zB'(z)$. Applying Theorem 1, we have

$$p(z) = zB'(z) - B(z) + 1 + 2zp(z)B(z),$$

which gives us

$$p(z) = \frac{zB'(z) - B(z) + 1}{1 - 2zB(z)} = \frac{1}{1-4z} - \frac{1}{z}\left(\frac{1-z}{\sqrt{1-4z}} - 1\right). \tag{3b}$$

We get p_n by expanding (3b). The result is given below; the asymptotics follow from Stirling's formula. ∎

THEOREM 2. *The cumulated path length over all binary trees of n nodes is*

$$p_n = 4^n - \frac{3n+1}{n+1}\binom{2n}{n},$$

and the expected internal path length p_n/B_n is asymptotically $n\sqrt{\pi n} - 3n + O(\sqrt{n})$.

A similar derivation in [Knuth 1973a] considers the bivariate OGF $B(u,z) = \sum_{n,k\geq 0} b_{n,k} u^k z^n$, where $b_{n,k}$ is the number of binary trees with size n and path length k. It satisfies the functional equation $B(u,z) = 1 + zB(u,uz)^2$ (cf. (2b)). The expected path length and the variance can be formed in the usual way in terms of the coefficients of z^n in the derivatives of $B(u,z)/B_n$, evaluated at $u = 1$.

The two examples above illustrate our general philosophy that it is useful to compute the OGFs for a standard catalogue of valuation functions, so as to handle a large variety of statistics. The most important VFs are clearly $I[t]$ and $S[t]$.

Another important class of trees is the class $\mathcal{G}$ of *plane trees* (also known as *ordered trees*). Each tree in $\mathcal{G}$ consists of a root node and an arbitrary number of ordered subtrees. This suggests the recursive definition

$$\mathcal{G} = \{\circ\} \times \mathcal{G}^*,$$

where $\circ$ represents a node, and $\mathcal{G}^* = \{\circ\} \times \sum_{k\geq 0} \mathcal{G}^k$ is the sequence class of $\mathcal{G}$, defined in Section 1.2. The size of a tree is defined to be its number of nodes. An interesting subclass of $\mathcal{G}$ is the class $\mathcal{T} = \mathcal{G}^\Omega$ of plane trees in which the degrees of the nodes are constrained to be in some subset Ω of the nonnegative integers. We require that $0 \in \Omega$ or else the trees will be infinite. The class $\mathcal{G}$ is the special case of $\mathcal{T}$ when Ω is the set of nonnegative integers. It is possible to mimic (4) and get the corresponding representation for $\mathcal{T}$

$$\mathcal{T} = \{\circ\} \times \sum_{k\in\Omega} \mathcal{T}^k, \tag{5}$$

but we shall see that it is just as simple to deal directly with (4) by using the appropriate VF to restrict the degrees. There are two important correspondences between $\mathcal{B}$, $\mathcal{G}$, and $\mathcal{T}$:

1. The set of binary trees of size n are isomorphic to the set of plane trees of size $n+1$. A standard technique in data structures illustrates the correspondence: We represent a general tree of $n+1$ nodes as a binary tree with no right subtree and with a binary tree of n internal nodes as its left subtree; the left link denotes "first child" and the right link denotes "next sibling."
2. If we think of the bottommost nodes of trees in $\mathcal{T}$ with $\Omega = \{0,2\}$ as "external nodes," we get a 1–1 correspondence between binary trees of size n and plane trees with degree constraint $\{0,2\}$ of size $2n+1$.

Theorem 1 generalizes in a straightforward manner for $\mathcal{G}$:

THEOREM 3. *The sum and recursive product valuation functions are admissible for the class $\mathcal{G}$ of plane trees:*

$$v[t] = u[t] + w[t] \qquad \Longrightarrow \qquad v(z) = u(z) + w(z);$$
$$v[t] = \delta_{\deg t \in \Omega} \prod_{1 \le i \le \deg t} u_{i,\deg t}[t_i] \qquad \Longrightarrow \qquad v(z) = z \sum_{k \in \Omega} \prod_{1 \le i \le k} u_{i,k}(z),$$

where $t_1, t_2, \ldots, t_{\deg t}$ represent the subtrees of t.

EXAMPLES. *Enumerations.* 1. The number G_n of plane trees with n nodes is obtained via the unit VF $I[t] \equiv 1 = \prod_{0 \le i \le \deg t} I[t_i]$. (Plane trees are always non-empty, so $|t| \ge 1$.) By Theorem 3, we get $G(z) = I(z) = z \sum_{k \ge 0} I(z)^k = z/(1 - I(z))$. This implies $G(z) = I(z) = \frac{1}{2}(1 - \sqrt{1 - 4z}) = zB(z)$, and thus we have $G_{n+1} = B_n$, which illustrates correspondence 1 mentioned earlier.

2. For the number T_n of trees of size n with degree constraint Ω, we apply to $\mathcal{G}$ the constrained unit VF $I^\Omega[t] = \delta_{t \in \mathcal{T}} = \delta_{\deg t \in \Omega} \prod_{0 \le i \le \deg t} I^\Omega[t_i]$. For the special case $\Omega = \{0, 2\}$, Theorem 3 gives us $T(z) = I^\Omega(z) = z + zI^\Omega(z)^2$. The solution to this quadratic equation is $T(z) = I^\Omega(z) = \frac{1}{2z}(1 - \sqrt{1 - 4z^2}) = zB(z^2)$, and thus we have $T_{2n+1} = B_n$, illustrating correspondence 2.

3. For $d \ge 2$, let us define the class $\mathcal{D} = \mathcal{D}_d$ of d-ary trees to be $\mathcal{D} = \{\blacksquare\} + (\{\circ\} \times \mathcal{D}^d)$. Binary trees are the special case $d = 2$. The number D_n of d-ary trees can be obtained by generalizing our derivation of B_n at the beginning of this section. The derivation we present, though, comes from generalizing correspondence 2 and staying within the framework of plane trees: Each d-ary tree corresponds to a plane tree of $dn + 1$ nodes with degree constraint $\Omega = \{0, d\}$. The same derivation used in the preceding example gives $T(z) = I(z) = z + zI(z)^d$. By Lagrange-Bürmann inversion, with $f(z) = T(z)$, $\Phi(u) = 1 + u^d$, we get

$$D_n = T_{dn+1} = [z^{dn+1}]T(z) = \frac{1}{dn+1}[u^{dn}]\left((1 + u^d)^{dn+1}\right) = \frac{1}{dn+1}\binom{dn+1}{n}. \quad \blacksquare$$

In each of the examples above, the functional equation involving the OGF was simple enough so that either the OGF could be solved in explicit closed form or else the Lagrange-Bürmann inversion theorem could be applied easily (that is, the coefficients of $\Phi(u)^n$ were easy to determine). More advanced asymptotic methods are needed, for example, to determine the number T_n of plane trees with arbitrary degree constraint Ω. Let us assume for simplicity that Ω is *aperiodic*, that is, Ω consists of 0 and any sequence of positive integers with a greatest common divisor of 1.

To count T_n, we start out as in the second example above. By applying Theorem 3, we get

$$T(z) = z\,\omega(T(z)), \tag{6}$$

where $\omega(u) = \sum_{k \in \Omega} u^k$. Lagrange-Bürmann inversion is of little help when $\omega(u)$ has several terms, so we take another approach. The singularities of $T(z)$ are of an algebraic nature. We know from Section 2.2 that the asymptotic behavior of the coefficients T_n are related to the dominant singularities of $T(z)$, that is, the ones with smallest modulus. To find the singularities of $T(z)$, let us regard $T(z)$ as the solution y of the equation

$$F(z, y) = 0, \qquad \text{where } F(z, y) = y - z\omega(y). \tag{7}$$

The function $y = T(z)$ is defined implicitly as a function of z. By the Implicit Function Theorem, the solution y with $y(z_0) = y_0$ is analytically continuable at z_0 if $F_y(z_0, y_0) \neq 0$, where $F_y(z, y)$ denotes the partial derivative with respect to y. Hence, the singularities of y (the values of z where y does not have an analytic continuation) are the values ρ where

$$F(\rho, \tau) = \tau - \rho\omega(\tau) = 0, \qquad F_y(\rho, \tau) = 1 - \rho\omega'(\tau) = 0. \tag{8}$$

This gives $\rho = \tau/\omega(\tau) = 1/\omega'(\tau)$, where τ is a root of the equation

$$\omega(\tau) - \tau\omega'(\tau) = 0. \tag{9}$$

Let us denote the dominant singularity of $T(z)$ on the positive real line by ρ^*, and let τ^* be the (unique) corresponding value of τ by (9). Since $T(\rho^*) = \tau^*$, it follows that τ^* is real. If Ω is aperiodic, then by examining the power series equation corresponding to (9), we see that τ^* is the unique real solution to (9), and any other solution τ must have larger modulus.

Around the point (ρ^*, τ^*), the dependence between y and z is locally of the form

$$0 = F(z,y) = F_z(\rho^*,\tau^*)(z-\rho^*) + 0\cdot(y-\tau^*) + F_{yy}(\rho^*,\tau^*)(y-\tau^*)^2 + \text{smaller order terms.} \quad (10)$$

By iteration (or bootstrapping) and bounding the coefficients, we can show that $y(z)$ has the form

$$y(z) = f(z) + g(z)\sqrt{1-\frac{z}{\rho}}, \quad (11a)$$

where $f(z)$ and $g(z)$ are analytic at $z=\rho$, and $g(\rho) = -\sqrt{\omega(\tau^*)/\omega''(\tau^*)}$. Hence, we have

$$y(z) = f(z) + g(\rho)\sqrt{1-\frac{z}{\rho}} + O\left(\left(1-\frac{z}{\rho}\right)^{3/2}\right). \quad (11b)$$

Theorem 2.2 shows that the contribution of $f(z)$ to T_n is insignificant. By applying the transfer lemma (Theorem 2.3), we get our final result:

THEOREM 4 [Meir and Moon 1978]. *If Ω is aperiodic, we have*

$$T_n \sim c\rho^{-n}n^{-3/2},$$

where the constants c and ρ are given by $c = -\sqrt{\omega(\tau)/(2\pi\omega''(\tau)}$ and $0 < \rho = \tau/\omega(\tau) < 1$, and τ is the smallest positive root of the equation $\omega(\tau) - \tau\omega'(\tau) = 0$.

For brevity, we expanded only the first terms of $y(z)$ in (11*b*), but we could easily have expanded $y(z)$ further to get the full asymptotic expansion of T_n. In the aperiodic case, which is also considered in [Meir and Moon 1978], the generating function $T(z)$ has more than one dominant singularity, and the contributions of these dominant singularities must be added together.

In the rest of this section we show how several other parameters of trees can be analyzed by making partial use of this approach. The asymptotics are often determined via the techniques described in Sections 2.2–2.4.

Height of Plane Trees. For example, let us consider the expected maximum stack size during a recursive tree traversal. (We earlier considered the expected stack size amortized over the course of the traversal.) The maximum stack size is simply the height of the tree, namely, the length of the longest path from the root to a node.

THEOREM 5 [De Bruijn, Knuth, and Rice 1972]. *The expected height $\overline{H}_n$ of a plane tree with n nodes, where each tree in $\mathcal{G}$ is equally likely, is $\overline{H}_n = \sqrt{\pi n} + O(1)$.*

PROOF. The number $G_n^{[h]}$ of plane trees with height $\le h$ corresponds to using the 0–1 VF $G^{[h]}$, which is defined to be 1 if the height of t is $\le h$, and 0 otherwise. It has a recursive formulation

$$G^{[h+1]}[t] = \prod_{1\le i\le \deg t} G^{[h]}[t_i], \quad (12a)$$

which is in the right form to apply Theorem 3. From it we get

$$G^{[h+1]}(z) = \sum_{k\ge 0} z\left(G^{[h]}(z)\right)^k = \frac{z}{1-G^{[h]}(z)}, \quad (12b)$$

where $G^{[0]}(z) = z$. Note the similarity to the generating function for plane trees, which satisfies $G(z) = z/(1-G(z))$. It is easy to transform (12*b*) into

$$G^{[h+1]}(z) = z\frac{F_h(z)}{F_{h+1}(z)}, \qquad \text{where } F_{h+2}(z) = F_{h+1}(z) - zF_h(z). \quad (13)$$

The polynomials $F_h(z)$ are called Fibonacci-Chebyshev polynomials. From the linear recurrence that $F_h(z)$ satisfies, we can express $F_h(z)$ as a rational function of $G^h(z)$. Then applying Lagrange-Bürmann inversion, we get the following expression:

$$G_{n+1} - G_{n+1}^{[h]} = \sum_{j\geq 0}\left(\binom{2n}{n+1-j(h+2)} - 2\binom{2n}{n-j(h+2)} + \binom{2n}{n-1-j(h+2)}\right). \tag{14}$$

The expected tree height $\overline{H}_{n+1}$ is given by

$$\overline{H}_{n+1} = \frac{1}{G_{n+1}}\sum_{h\geq 1} h\left(G_{n+1}^{[h]} - G_{n+1}^{[h-1]}\right) = \frac{1}{G_{n+1}}\sum_{h\geq 0}\left(G_{n+1} - G_{n+1}^{[h]}\right). \tag{15}$$

By substituting (14), we see that the evaluation of (15) is related to sums of the form

$$S_n = \sum_{k\geq 0} d(k)\binom{2n}{n-k}\Big/\binom{2n}{n}, \tag{16}$$

where $d(k)$ is the number of divisors of k. By Stirling's approximation, we can approximate S_n by $T(1/\sqrt{n})$, where

$$T(x) = \sum_{k\geq 1} d(k)e^{-k^2x^2}. \tag{17}$$

The problem is thus to evaluate $T(x)$ asymptotically as $x \to 0$. This is one of the expansions we did in Section 2.4 using Mellin transforms, where we found that

$$T(x) \sim -\pi\frac{\log x}{x} + \left(\frac{3}{4} - \frac{\log 2}{2}\right)\frac{\pi}{x} + \frac{1}{4} + O(x^m), \tag{18}$$

as $x \to 0$. The theorem follows from the appropriate combination of terms of the form (18). ∎

THEOREM 6 [Flajolet and Odlyzko 1982]. *The expected height $\overline{H}_n$ of a plane tree with degree constraint Ω, where Ω is aperiodic and each tree in $\mathcal{T}$ is equally likely, is $\overline{H}_n \sim \lambda\sqrt{n}$, where $\lambda = (2\pi)^{1/2}(\phi(\tau)\phi''(\tau))^{-1/2}\phi'(\tau)$ and τ is the constant appearing in Theorem 4.*

PROOF SKETCH. By analogy with (12), we use the VF $T^{[h]}[t] = \delta_{\text{height } t\leq h}$, which can be recursively expressed as

$$T^{[h+1]}[t] = \delta_{\deg t\in\Omega}\prod_{1\leq i\leq \deg t} T^{[h]}[t_i]. \tag{19a}$$

Theorem 3 gives us

$$T^{[h+1]}(z) = z\,\omega\left(T^{[h]}(z)\right), \tag{19b}$$

where $T^{[0]}(z) = z$ and $\omega(u) = \sum_{k\in\Omega} u^k$. The generating function $H(z)$ of the cumulated height of trees is equal to

$$H(z) = \sum_{h\geq 0}\left(T(z) - T^{[h]}(z)\right). \tag{20}$$

One way to regard (19*b*) is simply as the iterative approximation scheme to the fixed point equation (6) that determines $T(z)$. A delicate singularity analysis leads to the result. To do the analysis, we need to examine the behavior of the iterative scheme near the singularity $z = \rho$, which is an example of a *singular iteration problem*. We find in the neighborhood of $z = \rho$ that

$$H(z) \sim \frac{d}{1 - z/\rho}\log\frac{1}{1 - z/\rho},$$

where d is the appropriate constant. The theorem follows directly. ∎

Methods similar to those used in the proof of this theorem had been used by Odlyzko to prove the following:

THEOREM 7 [Odlyzko 1982]. *The number E_n of balanced 2–3 plane trees with n "external" nodes is $E_n \sim \frac{1}{n}\phi^n W(\log n)$, where ϕ is the golden ratio $(1+\sqrt{5})/2$, and $W(x)$ is a continuous and periodic function.*

This result actually extends to several families of balanced trees, which are used as search structures with guaranteed $O(\log n)$ access time. The occurrence of the golden ratio in Theorem 7 is not surprising, given our discussion in Section 2.2 of the equation $f(z) = z + f(z^2 + z^3)$, which is satisfied by the OGF of E_n.

Pattern Matching. Another important class of algorithms on trees deal with *pattern matching*, the problem of detecting all occurrences of a given pattern tree inside a larger text tree, which occurs often in symbolic manipulation systems. Unlike the simpler case of string matching, where linear-time worst-case algorithms are known, it is conjectured that no linear-time algorithm exists for tree pattern matching.

The following straightforward algorithm, called *sequential matching algorithm*, has quadratic running time in the worst case, but can be shown to run in linear time on the average. For each node of the tree, we compare the subtree rooted at that node with the pattern tree by doing simultaneous preorder traversals of the subtree and the pattern tree. Whenever a mismatch is found, the preorder traversal is aborted, and the next node in the tree is considered. If a preorder traversal successfully finishes, then a match is found.

THEOREM 8 [Steyaert and Flajolet 1983]. *The expected running time of the sequential matching algorithm, when applied to a fixed pattern P and all trees in $\mathcal{T}$ of size n, is $\sim cn$, where c is a function of the degree constraint Ω and the stucture of pattern P and is uniformly bounded by an absolute constant, for all Ω and P.*

PROOF SKETCH. The proof depends upon a lemma that the probability that P occurs at a random node in the tree is asymptotically $\tau^{e-1}\rho^i$, where i and e are the numbers of internal and external nodes in P. The algebraic part of the proof of the lemma is a direct application of the symbolic operator method of Theorems 1 and 3 applied to multisets of trees. Generating functions for the number of pattern matches have simple expressions in terms of $T(z)$; a singularity analysis finishes the proof. ∎

The same type of analysis can be applied to a large variety of tree algorithms in a semi-automatic way. One illustration is the following:

THEOREM 9 [Flajolet and Steyaert 1987]. *For any set Θ of operators and Δ of differentiation rules with at least one "expanding rule," the average-case complexity of the symbolic differentiation algorithm is asymptotically $cn^{3/2} + O(n)$, where the constant c depends upon Θ and Δ.*

Tree Compaction. A different kind of singular behavior occurs in the problem known as *common subexpression elimination* or *tree compaction*, where a tree is compacted into a directed acyclic graph by avoiding duplication of identical substructures. This has applications to the compaction of Lisp programs and to code optimization.

THEOREM 10 [Flajolet, Sipala, and Steyaert 1987]. *The expected size of the maximally compacted dag representation of a random tree in $\mathcal{T}$ of size n is $cn/\sqrt{\log n} + O(n/\log n)$, where the constant c depends upon Ω.*

The dominant singularity in this case is of the form $1/\sqrt{(1-z)\log(1-z)^{-1}}$. The theorem shows that the space savings to be expected when compacting trees approaches 100% as $n \to \infty$, though convergence is slow.

Register Allocation. The *register allocation* problem consists of finding an optimal strategy for evaluating expressions that can be represented by a tree. The optimal pebbling strategy, due to Ershov, requires only $O(\log n)$ registers to evaluate a tree of size n. The following theorem determines the coefficient in the average case for evaluating expressions involving binary operators:

THEOREM 11 [Flajolet, Raoult, and Vuillemin 1979], [Kemp 1979]. *The expected optimum number of registers to evaluate a random binary tree of size n is $\log_4 n + P(\log_4 n) + o(1)$, where $P(x)$ is a periodic function with period 1 and small amplitude.*

PROOF SKETCH. The analysis involves the combinatorial sum

$$V_n = \sum_{k\geq 1} v_2(k)\binom{2n}{n-k},$$

where $v_2(k)$ is the number of 2s in the decomposition of n into prime factors. If we normalize and approximate the binomial coefficient by an exponential term, as in (17), we can compute the approximations's Mellin transform

$$\frac{1}{2}\frac{\zeta(s)}{2^s-1}\Gamma\left(\frac{1}{2}\right).$$

The set of regularly-spaced poles $s = 2ik\pi/\log 2$ corresponds to periodic fluctuations in the form of a Fourier series. ∎

4.2. Binary Search Trees

We denote by $\mathcal{BST}(S)$ the *binary search tree* formed by inserting a sequence S of elements. It has the recursive decomposition

$$\mathcal{BST}(S) = \begin{cases} \langle \mathcal{BST}(S_<), s_1, \mathcal{BST}(S_>) \rangle, & \text{if } |S| \geq 1; \\ \langle \blacksquare \rangle, & \text{if } |S| = 0, \end{cases} \tag{21}$$

where s_1 is the first element in S, $S_<$ is the subsequence of elements $\leq s_1$, and $S_>$ is the subsequence of elements $> s_1$. An empty binary search tree is represented by the external node ■.

The search for an element x proceeds as follows, starting with the root s_1 as the current node y: We compare x with y, and if $x < y$ we set y to be the left child of y, and if $x > y$ we set y to be the right child of y. The process is repeated until either $x = y$ (successful search) or else an external node is reached (unsuccessful search). (Note that this process finds only the *first* element with value x. If the elements' values are all distinct, this is no problem; otherwise, the left path should be searched until a leaf or an element of smaller value is reached.) Insertion is done by inserting the new element into the tree at the point where the unsuccessful search ended. The importance of binary search trees to sorting and range queries is that a linear-time inorder traversal will output the elements in sorted order.

Well-known data structures, such as 2-3 trees, AVL trees, red-black trees, and self-adjusting search trees, do some extra work to ensure that the insert, delete, and query operations can be done in $O(\log n)$ time, where n is the size of the tree. (In the first three cases, the times are logarithmic in the worst case, and in the latter case they are logarithmic in the amortized sense.) In this section we show that the same logarithmic bounds hold in the average case without need for any balancing overhead. Our probability model assumes that the sequence S of n elements $s_1, s_2, \ldots, s_n$ is picked by random sampling from a real interval, or equivalently, as far as relative ordering is concerned, the elements form a random permutation of size n. The dynamic version of the problem, which corresponds to an average-case amortized analysis, appears in Section 6.

We define $\mathcal{BST}$ to be the class of all binary search trees corresponding to permutations, $\mathcal{BST} = \{\mathcal{BST}(\sigma) \mid \sigma \in S_n\}$. We use K to denote the random variable describing the size of the left subtree; that is, $|S_<| = K$ and $|S_>| = n - 1 - K$. By our probability model, the splitting probabilities become

$$\Pr\{K = k\} = \frac{1}{n}, \qquad \text{for } 0 \leq k \leq n-1. \tag{22}$$

One consequence of this is that not all trees in $\mathcal{BST}$ are equally likely to occur. For example, the perfectly balanced tree of three nodes (which occurs for $\sigma = (2,1,3)$ and $\sigma = (2,3,1)$) is twice as likely to occur as the tree for $\sigma = (1,2,3)$.

The powerful valuation function method that we introduced in the last section applies equally well to binary search trees. In this case, however, the nature of recurrence (21) suggests that we use EGFs of cumulative values (or equivalently OGFs of expected values). For VF $v[t]$, we let v_n be its expected value for trees of size n, and we define $v(z)$ to be the OGF $\sum_{n\geq 0} v_n z^n$.

THEOREM 12. *The sum and subtree product valuation functions are admissible for the class $\mathcal{BST}$ of binary search trees:*

$$v[t] = u[t] + w[t] \quad \Longrightarrow \quad v(z) = u(z) + w(z);$$

$$v[t] = u[t_<] \cdot w[t_>] \quad \Longrightarrow \quad v(z) = \int_0^z u(t)w(t)\,dt,$$

where $t_<$ and $t_>$ denote the left and right subtrees of t.

The subtree product VF typically results in an integral equation on the OGFs, which by differentiation can be put into the form of a differential equation. This differs from the equations that resulted from Theorems 1 and 3, which we used in the last section for binary and plane trees.

A good illustration of these techniques is to compute the expected number of probes $\overline{C_n}$ per successful search on a random binary search tree of size n. We assume that each of the n elements is equally likely to be the object of the search. It is easy to see that $\overline{C_n}$ is equal to the expected internal path length p_n, divided by n, plus 1, so it suffices to compute p_n. The recursive definition of the corresponding VF $p[t]$ is

$$\begin{aligned} p[t] &= |t| + p[t_<] + p[t_>] \\ &= |t| + p[t_<] \cdot I[t_>] + I[t_<] \cdot p[t_>], \end{aligned} \tag{23a}$$

where $I[t] \equiv 1$ is the unit VF, whose OGF is $I(z) = \sum_{n\geq 0} z^n = 1/(1-z)$. The size VF $S[t] = |t|$ has OGF $\sum_{n\geq 0} nz^n = z/(1-z)^2$. Theorem 12 translates (23*a*) into

$$p(z) = \frac{z}{(1-z)^2} + 2\int_0^z \frac{p(t)}{1-t}\,dt. \tag{23b}$$

Differentiating (23*b*), we get a linear first-order differential equation

$$p'(z) - \frac{2p(z)}{1-z} - \frac{1+z}{(1-z)^3} = 0, \tag{23c}$$

which can be solved using the variation-of-parameter method (1.20) to get

$$p(z) = -2\frac{\log(1-z)+z}{(1-z)^2} = 2H'(z) - \frac{2+z}{(1-z)^2},$$

where $H(z) = \sum_{n\geq 0} H_n z^n$ is the OGF of the Harmonic numbers. The following theorem results by extracting $[z^n]p(z)$:

THEOREM 13. *The expected internal path length of a random binary search tree with n internal nodes is*

$$p_n = 2(n+1)H_{n+1} - 2n - 2 \sim 2n\log n + (2\gamma - 3)n + O(\log n).$$

This has direct application to several statistics of interest. For example, $p_n - n$ is the expected number of comparisons used to sort a sequence of n elements by building a binary search tree and then performing an inorder traversal. The expected number $\overline{U_n}$ of probes per unsuccessful search (which is also the average number of probes per insertion, since insertions are preceded by an unsuccessful search) is the average external path length $\overline{EP_n}$, divided by $n+1$. The well-known correspondence

$$EP_n = IP_n + 2n \tag{24a}$$

between the external path length EP_n and the internal path length IP_n of a binary tree with n internal nodes leads to

$$\overline{U_n} = \frac{n}{n+1}(\overline{C_n} + 1), \tag{24b}$$

which readily yields an expression for $\overline{U_n}$ via Theorem 13. We can also derive $\overline{U_n}$ directly as we did for $\overline{C_n}$ or via the use of PGFs. Yet another alternative is an ad hoc proof that combines (24*b*) with different linear relation between $\overline{U_n}$ and $\overline{C_n}$, namely,

$$\overline{C_n} = \frac{1}{n}\sum_{0\leq i\leq n-1} \overline{U_i}. \tag{25}$$

Eq. (25) follows from the observation that the n possible successful searches on a tree of size n retrace the steps taken during the n unsuccessful searches that were done when the elements were originally inserted.

Quicksort. We can apply this machinery to the analysis of quicksort, as mentioned in Section 3.5. Let q_n be the average number of comparisons used by quicksort to sort n elements. Quicksort works by choosing a partitioning element s (say, the first element), dividing the file into the part $\le s$ and the part $> s$, and recursively sorting each subfile. The process is remarkably similar to the recursive decomposition of binary search trees. The version of quicksort in [Knuth 1973b] and [Sedgewick 1977b] uses $n+1$ comparisons to split the file into two parts. (Only $n-1$ comparisons are needed, but the extra two comparisons help speed up the rest of the algorithm in actual implementations.) The initial conditions are $q_0 = q_1 = 0$. The corresponding VF $q[t]$ is

$$\begin{aligned} q[t] &= |t| + 1 - \delta_{|t|=0} - 2\delta_{|t|=1} + q[t_{\le}] + q[t_{>}] \\ &= |t| + 1 - \delta_{|t|=0} - 2\delta_{|t|=1} + q[t_{\le}] \cdot I[t_{>}] + I[t_{\le}] \cdot q[t_{>}] \end{aligned} \tag{26a}$$

As before, the OGFs for $I[t] \equiv 1$ and $S[t] = |t|$ are $1/(1-z)$ and $z/(1-z)^2$. By the translation of Theorem 12, we get

$$q(z) = \frac{z}{(1-z)^2} + \frac{1}{1-z} - 1 - 2z + 2\int_0^z \frac{q(t)}{1-t}\,dt; \tag{26b}$$

$$q'(z) = \frac{2}{(1-z)^3} - 2 + \frac{2q(z)}{1-z} \tag{26c}$$

The linear differential equation (26c) can be solved via the variation-of-parameter method to get

$$q(z) = -2\frac{\log(1-z)}{(1-z)^2} - \frac{2}{3(1-z)^2} + \frac{2}{3}(1-z) = 2H'(z) - \frac{8}{3(1-z)^2} + \frac{2}{3}(1-z). \tag{27a}$$

We can then extract $q_n = [z^n]q(z)$ to get

$$q_n = 2(n+1)\left(H_{n+1} - \frac{4}{3}\right) \sim 2n\log n + 2n\left(\gamma - \frac{4}{3}\right) + O(\log n). \tag{27b}$$

In practice, quicksort can be optimized by stopping the recursion when the size of the subfile is $\le m$, for some parameter m. When the algorithm terminates, a final insertion sort is done on the file. (We know from Section 3.2 that insertion sort is very efficient when the number of inversions is small.) The analysis of quicksort can be modified to give the average running time as a function of n and m. The optimum m can then be determined, as a function of n. This is done in [Knuth 1973b] and [Sedgewick 1977], where it is shown that $m = 9$ is optimum in typical implementations once n gets large enough. The average number of comparisons can be derived using the truncated VF

$$q_m[t] = \delta_{|t|>m}(|t|+1) + q_m[t_{\le}] \cdot I[t_{>}] + I[t_{\le}] \cdot q_m[t_{>}] \tag{28}$$

(cf. (26a)). The truncated unit VF $I_m[t] = \delta_{|t|>m}$ and the truncated size VF $S_m[t] = \delta_{|t|>m}|t|$ have the OGFs $\sum_{n>m} z^n = z^{m+1}/(1-z)$ and $\sum_{n>m} nz^n = \big((m+1)z^{m+1} - mz^{m+2}\big)/(1-z)^2$, respectively. The rest of the derivation precedes as before (and should done with an symbolic algebra system); the result (cf. (27b)) is

$$q_{m,n} = 2(n+1)\left(H_{n+1} - H_{m+2} + \frac{1}{2}\right). \tag{29}$$

Height of Binary Search Trees. The height of binary search trees involve interesting equations over generating functions. By analogy with (12), let $G_n^{[h]}$ denote the probability that a random binary search tree of size n has height $\le h$. The corresponding VF $G^{[h]}[t] = \delta_{\text{height}\,t \le h}$ is of the form

$$G^{[h+1]}[t] = \delta_{|t|=0} + G^{[h]}[t_{\le}] \cdot G^{[h]}[t_{>}]. \tag{30a}$$

Theorem 12 translates this directly into

$$G^{[h+1]}(z) = 1 + \int_0^z \left(G^{[h]}(t)\right)^2 dt, \tag{30b}$$

where $G^{[0]}(z) = 1$ and $G(z) = G^{[\infty]}(z) = 1/(1-z)$. The sequence $\{G^{[h]}(z)\}_{h\ge 0}$ forms a sequence of Picard approximants to $G(z)$. The OGF for the expected height is

$$H(z) = \sum_{h\ge 0}\left(G(z) - G^{[h]}(z)\right). \tag{31}$$

It is natural to conjecture that $H(z)$ has the singular expansion

$$G(z) \sim \frac{c}{1-z}\log\frac{1}{1-z}, \tag{32}$$

as $z \to 1$, for some constant c, but no one has succeeded so far in establishing it directly. Devroye [1986a] has determined the asymptotic form of H_n using the theory of branching processes:

THEOREM 14 [Devroye 1986a]. *The expected height H_n of a binary search tree of size n is $H_n \sim c \log n$, where $c = 4.311070\ldots$ is the root ≥ 2 of the equation $(2e/c)^c = e$.*

Theorems 13 and 14 point out clearly that a random binary search tree is fairly balanced, in constrast to the random plane trees we studied Section 4.1. The expected height and path lengths are $O(\log n)$ and $O(n \log n)$, whereas the corresponding quantities for plane trees are $O(\sqrt{n})$ and $O(n\sqrt{n})$.

Interesting problems in average-case analysis also arise in connection with balanced search trees, but interest is usually focused on storage space rather than running time. For example, a fringe analysis is used in [Yao 1978] and [Brown 1979] to derive upper and lower bounds on the expected storage utilization and number of balanced nodes in random 2-3 trees and B-trees. These techniques can be extended to get better bounds, but the computations become prohibitive.

Multidimensional Search Trees. The binary search tree structure can be generalized in various ways to two dimensions. The most obvious generalization, called *quad trees*, uses internal nodes of degree 4. The quad tree for a sequence $S = s_1\ s_2, \ldots, s_n$ of n inserted elements is defined by

$$\mathcal{Q}(S) = \begin{cases} \langle s_1, \mathcal{Q}(S_{>,>}), \mathcal{Q}(S_{\leq,>}), \mathcal{Q}(S_{\leq,\leq}), \mathcal{Q}(S_{>,\leq})\rangle, & \text{if } |S| \geq 1; \\ \langle\blacksquare\rangle, & \text{if } |S| = 0. \end{cases} \tag{33}$$

Here each element s in S is a two-dimensional number, and the four quadrants determined by s are denoted $S_{>,>}$, $S_{\leq,>}$, $S_{\leq,\leq}$, and $S_{>,\leq}$. Quad trees support general range searching, and in particular partially specified queries of the form "Find all elements $s = (s_x, s_y)$ with $s_x = c$." The search proceeds recursively to all subtrees whose range overlaps the query range.

THEOREM 15 [Flajolet et al 1987]. *The expected number of comparisons $\overline{C_n}$ for a partially specified query in a quad tree of size n is $\overline{C_n} = bn^{(\sqrt{17}-3)/2} + O(1)$, where b is a positive real number.*

PROOF. The splitting probabilities for quad trees are not in as simple a form as in (22), but they can be determined readily. By use of the appropriate VF $c[t]$, we get

$$c_n = 1 + \frac{4}{n(n+1)} \sum_{0\leq \ell \leq n-1} \sum_{0\leq k\leq \ell} c_k. \tag{34a}$$

In terms of the OGF $d(z) = zc(z)$, this becomes a second-order differential equation

$$d''(z) - \frac{4}{z(1-z)^2} d(z) = \frac{2}{(1-z)^3}. \tag{34b}$$

It is not clear how to solve explicitly for $d(z)$, but we can get asymptotic estimates for d_n based upon the fact that $d(z) \sim a(1-z)^{-\alpha}$, as $z \to 1$, for some positive real a and α. We cannot determine a in closed form in general for this type of problem, but α can be determined by substituting $a(1-z)^{-\alpha}$ into (34b) to get the "indicial equation"

$$\alpha(\alpha+1) - 4 = 0, \tag{35}$$

whose positive solution is $\alpha = (\sqrt{17}-1)/2$. The transfer lemma (Theorem 2.3) gives us our final result. ∎

Quad trees can be generalized to k dimensions, $k \geq 2$, but the degrees of the nodes become 2^k, which is too large to be practical. A better alternative, called *k-d trees*, is a binary search tree in which the splitting at each node on level i, $i \geq 0$, is based upon ordinate i mod $k+1$ of the element stored there.

THEOREM 16 [Flajolet and Puech 1986]. *The expected number of real number comparisons for a query in a k-d tree of size n, in which s of the k fields are specified, is $\sim an^{1-s/k+\vartheta(s/k)}$, where $\vartheta(u)$ is the root $\vartheta \in [0,1]$ of the equation $(\vartheta+3-u)^u(\vartheta+2-u)^{1-u} - 2 = 0$.*

PROOF SKETCH. The proof proceeds by first developing a system of integral equations for the OGFs of expected costs using the appropriate VFs and applying Theorem 12. This reduces to a differential system of order $2k - s$. It cannot be solved explicitly in terms of standard transcendental functions, but a singularity analysis can be done to get the result, based upon a generalization of the approach for quad trees. ∎

Heap-Ordered Trees. We conclude this section by considering *heap-ordered trees*, in which the value of the root node of each subtree is $\leq$ the values of the other elements in the subtree. We discussed the classical array representation of a perfectly balanced heap in connection with the heapsort algorithm in Section 3.5. Heap-ordered trees provide the basic and efficient implementations of priority queues, which support the operations *insert*, *find_min*, and *delete_min*. Additional operations sometimes include *merge* and *decrease_key*. Pagodas [Françon, Viennot, Vuillemin 1978] are a direct implementation of heap-ordered trees which also supports the *merge* operation.

For the sequence S of n elements $s_1, s_2, \ldots, s_n$, we define $\mathcal{HOT}(S)$ to be the (canonical) heap-ordered tree formed by S. It has the recursive definition

$$\mathcal{HOT}(S) = \begin{cases} \langle \mathcal{HOT}(S_{\text{left}}), \min(S), \mathcal{HOT}(S_{\text{right}}) \rangle, & \text{if } |S| \geq 1; \\ \langle \blacksquare \rangle, & \text{if } |S| = 0. \end{cases} \tag{36}$$

where $\min(S)$ is the rightmost smallest element in S, S_{left} is the initial subsequence $s_1, \ldots, s_{\min(S)-1}$, and S_{right} is the final subsequence $s_{\min(S)+1}, \ldots, s_n$. We assume in our probability model that S is a random permutation of n elements. Analysis of parameters of heap-ordered trees and pagodas is similar to the analysis of binary search trees, because of the following equivalence principle due to W. Burge (see [Vuillemin 1980]):

THEOREM 17. *For each pair of inverse permutations σ and σ^{-1}, we have*

$$\mathcal{BST}(\sigma) \equiv_{\text{shape}} \mathcal{HOT}(\sigma^{-1}),$$

where $t \equiv_{\text{shape}} u$ iff the unlabeled trees associated with trees t and u are identical.

For purposes of analysis, any parameter of permutations defined inductively over the associated heap-ordered tree can thus be directly analyzed using the admissibility rules of Theorem 12. Heap-ordered trees in the form of cartesian trees can also be used to handle a variety of 2-dimensional search problems [Vuillemin 1980].

4.3. Radix-Exchange Tries

Radix-exchange tries are binary search trees in which the elements are stored in the external nodes, and navigation through the trie at level i is based upon the ith bits of the search argument and the element stored there. Bit 0 means "go left," and bit 1 means "go right." We assume for simplicity that each element is a real number in $[0, 1]$ of infinite precision. The trie $\mathcal{TR}(S)$ for a set S of elements is defined recursively by

$$\mathcal{TR}(S) = \begin{cases} \langle \mathcal{TR}(S_0), \circ, \mathcal{TR}(S_1) \rangle, & \text{if } |S| > 1; \\ \langle \blacksquare \rangle, & \text{if } |S| = 1; \\ \langle \emptyset \rangle, & \text{if } |S| = 0, \end{cases} \tag{37}$$

where S_0 and S_1 are defined as follows: If we take the elements in S that have 0 as the first bit and then throw away that bit, we get S_0, the elements in the left subtrie. The set S_1 of elements in the right subtrie is defined similarly for the elements starting with a 1 bit. The elements are stored in the external nodes of the trie; the size of the trie is the number of external nodes. When S has a single element s, the trie consists of the external node $\blacksquare$ with value s; an empty trie is represented by the null external node $\emptyset$.

The trie $\mathcal{TR}(S)$ does not depend on the order in which the elements in S are inserted; this is quite different from the case of binary search trees, where order can make a big difference on the shape of the tree. In tries, the shape of the trie is based upon the distribution of the elements' values.

We use the probability model that the values of the elements are independent and uniform in the real interval $[0, 1]$. We define the class of all tries to be $\mathcal{TR}$. The probability that a trie of n elements has a left subtrie of size k and a right subtrie of size $n - k$ is the Bernoulli probability

$$p_{n,k} = \frac{1}{2^n} \binom{n}{k}. \tag{38}$$

This suggests that we use EGFs on expected values to represent trie statistics. We denote the expected value of VF $v[t]$ among trees of size n by v_n and the EGF $\sum_{n \geq 0} v_n z^n / n!$ by $v(z)$. The admissibility theorem takes yet another form:

THEOREM 18. *The sum and subtree product valuation functions are admissible for the class $\mathcal{TR}$ of tries:*

$$v[t] = u[t] + w[t] \quad \Longrightarrow \quad v(z) = u(z) + w(z);$$
$$v[t] = u[t_0] \cdot w[t_1] \quad \Longrightarrow \quad v(z) = u\left(\frac{z}{2}\right) w\left(\frac{z}{2}\right),$$

where t_0 and t_1 represent the left and right subtries of t.

In typical cases, the EGFs for the VFs that we encounter are in the form of difference equations that we can iterate.

The expected number of bit inspections per successful search in a trie with n external nodes is equal to the expected external path length p_n, divided by n. The following theorem shows that the search times are logarithmic, on the average, when no balancing is done.

THEOREM 19 [Knuth 1973b]. *The average external path length p_n of a random trie of size n is $p_n = n\log_2 n + (\gamma/\log 2 + \frac{1}{2} + R(\log_2 n))n + O(\sqrt{n})$, where $R(u)$ is a periodic function of small amplitude with period 1 and mean 0.*

PROOF. The VF corresponding to external path length is

$$p[t] = |t| - \delta_{|t|=1} + p[t_0] \cdot I[t_1] + I[t_0] \cdot p[t_1]. \tag{39a}$$

The unit VF $I[t] \equiv 1$ has EGF $\sum_{n\geq 0} z^n/n! = e^z$, and the size VF $S[t] = |t|$ has EGF $\sum_{n\geq 0} nz^n/n! = ze^z$. By Theorem 18, (39a) translates to

$$p(z) = ze^z - z + 2e^{z/2}p\left(\frac{z}{2}\right). \tag{39b}$$

By iterating the recurrence and then extracting coefficients, we get

$$p(z) = z\sum_{k\geq 0}\left(e^z - e^{z(1-1/2^k)}\right); \tag{39c}$$

$$p_n = n\sum_{k\geq 0}\left(1 - \left(1 - \frac{1}{2^k}\right)^{n-1}\right). \tag{39d}$$

It is easy to verify the natural approximation $p_n \sim nP(n)$, where $P(x)$ is the harmonic sum

$$P(x) = \sum_{k\geq 0}\left(1 - e^{-x/2^k}\right). \tag{40}$$

We have already derived the asymptotic expansion of $P(x)$, as $x \to \infty$, by use of Mellin transforms in (2.28). The result follows immediately. ∎

Theorem 19 generalizes to the biased case, where the bits of each element are independently 0 with probability p and 1 with probability $q = 1-p$. The average external path length is asymptotically $(n\log_2 n)/H$, where H is the binary entropy function $H = -p\log_2 p - q\log_2 q$. In the case of unsuccessful searches, a similar approach shows that the average number of bit inspections is $\sim (\log_2 n)/H$. The variance is $O(1)$ with fluctuation in the unbiased case [Jacquet and Régnier 1986], [Kirschenhöfer and Prodinger 1986], and it increases to $\sim c\log n + O(1)$, for some constant c, in the biased case [Jacquet and Régnier 1986]. The limiting distributions are studied in [Jacquet and Régnier 1986] and [Pittel 1986]. The height of a trie has mean $\sim 2\log_2 n$ and variance $O(1)$ [Flajolet 1983]. Limiting distributions of the height are studied in [Flajolet 1983] and [Pittel 1986].

Another important statistic on tries, besides search time, is storage space. Unlike binary search trees, the amount of auxiliary space used by tries, measured in terms of the number of internal nodes, is variable.

THEOREM 20 [Knuth 1973b]. *The expected number i_n of internal nodes in a random unbiased trie with n external nodes is $i_n = (n/\log 2)(1 + Q(\log_2 n)) + O(\sqrt{n})$, where $Q(u)$ is a periodic function of small amplitude with period 1 and mean 0.*

PROOF. The VF corresponding to the number of internal nodes is

$$i[t] = 1 - \delta_{|t|=0} - \delta_{|t|=1} + i[t_0] \cdot I[t_1] + I[t_0] \cdot i[t_1]. \tag{41a}$$

Theorem 18 translates this to

$$i(z) = e^z - 1 - z + 2e^{z/2} i\left(\frac{z}{2}\right). \tag{41b}$$

By iterating the recurrence and then extracting coefficients, we get

$$i(z) = \sum_{k\geq 0} 2^k \left(e^z - \left(1 + \frac{z}{2^k}\right) e^{(1-1/2^k)z}\right); \tag{41c}$$

$$i_n = \sum_{k\geq 0} 2^k \left(1 - \left(1 - \frac{1}{2^k}\right)^n - \frac{n}{2^k}\left(1 - \frac{1}{2^k}\right)^{n-1}\right). \tag{41d}$$

We can approximate i_n to within $O(\sqrt{n})$ in a natural way by $S(n)$, where

$$S(x) = \sum_{k\geq 0} 2^k \left(1 - e^{-x/2^k}\left(1 + \frac{x}{2^k}\right)\right). \tag{42a}$$

Equation (42*a*) is a harmonic sum, and its Mellin transform $S^*(s)$ can be computed readily:

$$S^*(s) = \frac{(s+1)\Gamma(s)}{1 - 2^{s+1}}, \tag{42b}$$

where the fundamental strip of $S^*(s)$ is $\langle -2; -1\rangle$. The result follows by computing the residues in the right half-plane $\Re(s) \geq -1$. There is a simple pole at $s = 0$ due to $\Gamma(s)$ and poles at $-1 + 2ik\pi/\log 2$ due to the denominator of (42*b*). ∎

In the biased case, the expected number of internal nodes is $\approx n/H$. The variance for both the unbiased and biased case is $O(n)$, which includes a fluctuating term [Jacquet and Régnier 1987]; the distribution of the number of internal nodes is normal [Jacquet and Régnier 1986].

Theorem 20, as do a number of the results, generalizes to the case in which each external node in the trie represents a page of secondary storage capable of storing $b \geq 1$ elements. Such tries are generally called *b-tries*. The analysis uses truncated VFs, as in the second quicksort example in Section 4.2, to stop the recursion when the subtrie has $\leq b$ elements. The result applies equally well to the extendible hashing scheme of [Fagin et al 1979], where the trie is built on the hashed values of the elements, rather than on the elements themselves. Extendible hashing will be considered further in Section 5.1.

THEOREM 21 [Knuth 1973b]. *The expected number of pages of capacity b needed to store a file of n records using b-tries or extendible hashing is* $(n/(b\log 2))(1 + R(\log_2 n)) + O(\sqrt{n})$, *where* $R(u)$ *is periodic with period* 1 *and mean* 0.

Radix-Exchange Sorting. It is no accident that radix-exchange tries and the radix-exchange sorting algorithm have a common name. Radix-exchange sorting is related to tries in a way very similar to how quicksort is related to binary search trees, except that the relationship is even closer. All our average-case analysis in this section carries over to the analysis of radix-exchange sorting: The distribution of the number of partitioning stages used by radix-exchange sorting to sort n numbers is the same as the distribution of the number of internal nodes in a trie, and the distribution of the number of bit inspections done by radix-exchange sorting is same as the distribution of the external path length of a trie.

4.4. Digital Search Trees

Digital search trees are like tries except that the elements are stored in the internal nodes, or equivalently they are like binary search trees except that the branching at level i is determined by the $(i+1)$st bit rather than by a key comparison. The digital search tree $DST(S)$ for a sequence S of inserted elements is defined recursively by

$$DST(S) = \begin{cases} \langle DST(S_<), s_1, DST(S_>)\rangle, & \text{if } |S| \geq 1; \\ \langle\blacksquare\rangle, & \text{if } |S| = 0, \end{cases} \tag{43}$$

where s_1 is the first element of S, and the sequence S_0 of elements in the left subtree is formed by taking the elements in $S - \{s_1\}$ that have 0 as the first bit and then throwing away the first bit. The sequence S_1 for the right subtree is defined similarly. Like binary search trees, the size of the tree is its number of internal nodes, and its shape is sensitive to the order in which the elements are inserted. The empty digital search tree is denoted by the external node ■. Our probability model is the same as for tries, except that the probability that a tree of n elements has a left subtree of size k and a right subtree of size $n-k$ is $\binom{n-1}{k}/2^{n-1}$. The class of all digital search trees is denoted $\mathcal{DST}$.

The nature of the decomposition in (43) suggests that we use EGFs of expectations in our analysis, as in the last section, but the admissibility theorem takes a different form:

THEOREM 22. *The sum and subtree product valuation functions are admissible for the class $\mathcal{DST}$ of digital search trees:*

$$v[t] = u[t] + w[t] \quad \Longrightarrow \quad v(z) = u(z) + w(z);$$
$$v[t] = u[t_0] \cdot w[t_1] \quad \Longrightarrow \quad v(z) = \int_0^z u\left(\frac{t}{2}\right) w\left(\frac{t}{2}\right) dt,$$

where t_0 and t_1 denote the left and right subtrees of t.

Tries are preferred in practice over digital search trees, since the key comparison done at each node in a digital search tree takes longer than the bit comparison done in a trie, and the elements in a trie are kept in sorted order. We do not have space in this manuscript to include the relevant analysis; instead we refer the reader to [Flajolet and Sedgewick 1986], [Knuth 1973b], and [Konheim and Newman 1973]. The key difference between the analysis of digital search trees and the analysis of tries in the last section is that the equations on the EGFs that result from Theorem 22 are typically difference-differential equations, to which the Mellin techniques that worked so well for tries cannot be applied directly. Instead the asymptotics come by an application due to S. O. Rice of the following classical formula from the calculus of finite differences; the proof of the formula is an easy application of Cauchy's formula.

THEOREM 23. *Let C be a closed curve encircling the points 0, 1, ..., n, and let $f(z)$ be analytic in C. Then we have*

$$\sum_k \binom{n}{k} (-1)^k f(k) = \frac{1}{2i\pi} \int_C B(n+1, -z) f(z)\, dz,$$

where $B(x,y) = \Gamma(x)\Gamma(y)/\Gamma(x+y)$ is the classical Beta function.

THEOREM 24 [Konheim and Newman 1973], [Knuth 1973b]. *The expected internal path length of a random discrete search tree is*

$$(n+1)\log_2 n + \left(\frac{\gamma - 1}{\log 2} + \frac{1}{2} - \alpha + P(\log_2 n)\right) n + O(\sqrt{n}),$$

where $\gamma = 0.57721\ldots$ is Euler's constant, $\alpha = 1 + \frac{1}{3} + \frac{1}{7} + \frac{1}{15} + \cdots = 1.606695\ldots$, and $P(u)$ is a periodic function with period 1 and very small amplitude.

5. Hashing and Address Computation Techniques

In this section we consider several well-known hashing algorithms, including separate chaining, coalesced hashing, uniform probing, double hashing, secondary clustering, and linear probing, and we also discuss the related methods of interpolation search and distribution sorting. Our machine-independent model of search performance is the number of probes made into the hash table during the search. We are primarily interested in the expected number of probes per search, but in some cases we also consider the distribution of the number of probes and the expected maximum number of probes among all the searches in the table.

With hashing, searches can be performed in constant time, on the average, regardless of the number of elements in the hash table. All hashing algorithms use a pre-defined *hash function*

$$hash : \{\text{all possible elements}\} \rightarrow \{1, 2, \ldots, m\} \tag{1}$$

that assigns a *hash address* to each of the n elements. Hashing algorithms differ from one another in how they resolve the *collision* that results when an element's hash address is already occupied. The two main techniques for resolving collisions are chaining (in which links are used to explicitly link together keys with the same hash address) and open addressing (where the search path through the table is defined implicitly). We study these two classes of hashing algorithms in the next two sections.

We use the *Bernoulli probability model* for our average-case analysis: We assume that all m^n possible sequences of hash addresses (or *hash sequences*) are equally likely. Simulation studies confirm that this is a reasonable assumption for well-designed hashing functions. An unsuccessful search can begin at any of the m slots in the hash table with equal probability, and the object of the successful search is equally likely to be any of the n elements in the table. We assume that each unsuccessful search is followed by an insertion, and that each insertion is preceded by an unsuccessful search. We denote the expected number of probes per unsuccessful search (or insertion) in a hash table with n elements by $\overline{U_n}$, and the expected number of probes per successful search by $\overline{C_n}$.

5.1. Bucket Algorithms and Hashing by Chaining

Separate Chaining. One of the most obvious techniques for resolving collisions is to link together all the elements with the same hash address into a list or chain. The generic name for this technique is *separate chaining*. The first variant we shall study stores the chains in auxiliary memory; the hth slot in the hash table contains a link to the start of the chain of elements with hash address h. This particular variant is typically called *indirect chaining*, because the hash table stores only pointers, not the elements themselves.

Search time clearly depends upon the number of elements in the chain searched. We define ${}_n^m X_i$ (or simply X_i) in the Bernoulli model to be the RV describing the number of elements in bucket i. This model is sometimes called the *urn model*, and the distribution of X_i is called the *occupancy distribution*. Distributions of this sort appear in the analyses of each of the chaining algorithms we consider in this section and they serve to unify our analyses. An unsuccessful search on a chain of length k makes one probe per element, plus one probe to find the link to the beginning of the chain. This allows us to express the expected number of probes per unsuccessful search as

$$\overline{U_n} = \mathbf{E}\left\{\frac{1}{m}\sum_{1\le i\le m}(1+X_i)\right\}. \tag{2a}$$

By symmetry, the expected values $\mathbf{E}\{X_i\}$ are the same for each $1 \le i \le m$, so we can restrict our attention to one particular bucket, say, bucket 1. (For simplicity, we shall abbreviate X_1 by X.) Eq. (2a) simplifies to

$$\overline{U_n} = 1 + \mathbf{E}\{X\}. \tag{2b}$$

For successful searches, each chain of length k contributes $2 + 3 + \cdots + k + 1 = 3k/2 + k^2/2$ probes. The expected number of probes per successful search is thus

$$\overline{C_n} = \mathbf{E}\left\{\frac{1}{n}\sum_{1\le i\le m}\left(\frac{3}{2}X_i + \frac{1}{2}{X_i}^2\right)\right\} = \frac{m}{n}\left(\frac{3}{2}\mathbf{E}\{X\} + \frac{1}{2}\mathbf{E}\{X^2\}\right). \tag{3}$$

We can compute (2*b*) and (3) in a unified way via the PGF

$$X(u) = \sum_{k\geq 0} \Pr\{X = k\}u^k. \tag{4}$$

Eqs. (2*b*) and (3) for $\overline{U_n}$ and $\overline{C_n}$ are expressible simply in terms of derivatives of $X(u)$:

$$\overline{U_n} = 1 + X'(1); \qquad \overline{C_n} = \frac{m}{n}\left(2X'(1) + \frac{1}{2}X''(1)\right). \tag{5}$$

A straightforward combinatorial argument shows that

$$\Pr\{X = k\} = \binom{n}{k}\left(\frac{1}{m}\right)^k\left(\frac{m-1}{m}\right)^{n-k}, \tag{6a}$$

and by the binomial theorem we get

$$X(u) = \left(\frac{m-1+u}{m}\right)^n. \tag{6b}$$

Hence, we have

$$X'(1) = \frac{n}{m}; \qquad X''(1) = \frac{n(n-1)}{m^2}. \tag{6c}$$

We get the following theorem by substituting (6*c*) into (5):

THEOREM 1. *The expected number of probes per unsuccessful and successful seach for indirect chaining, when there are n elements in a hash table of m slots, is*

$$\overline{U_n} = 1 + \frac{n}{m} \sim 1 + \alpha; \qquad \overline{C_n} = 2 + \frac{n-1}{2m} \sim 2 + \frac{\alpha}{2}.$$

We can also derive the formula for $\overline{C_n}$ from the one for $\overline{U_n}$ by noting that a correspondence akin to Eq. (4.25) for binary search trees holds:

$$\overline{C_n} = 1 + \frac{1}{n}\sum_{0\leq i\leq n-1} \overline{U_i}. \tag{7}$$

Another way to derive $X(u)$ is to decompose X into the sum of n independent 0–1 RVs

$$X = x_1 + x_2 + \cdots + x_n,$$

where $x_i = 1$ iff the ith element goes into bucket i. Each x_i has the same distribution as $\frac{m}{1}X_1$. The PGF $x_i(u)$ is clearly $(m-1)/m + u/m$. Formula (6*b*) follows from the observation that the PGF of a sum of independent RVs is equal to the product of the PGFs.

EXAMPLES. 1. *Direct chaining.* A more efficient version of separate chaining, called direct chaining, stores the first element of each chain directly in the hash table itself. This shortens each successful search time by one probe, and the expected unsuccessful search time is reduced by $\Pr\{X > 0\} = 1 - X(0) = 1 - (1 - 1/m)^n \sim 1 - e^{-\alpha}$. We get

$$\overline{U_n} = \left(1 - \frac{1}{m}\right)^n + \frac{n}{m} \sim e^{-\alpha} + \alpha; \qquad \overline{C_n} = 1 + \frac{n-1}{2m} \sim 1 + \alpha. \tag{8}$$

2. *Direct chaining with relocation.* The above variant is wasteful of hash table slots, because auxiliary space is used to store colliders even though there might be empty slots available in the hash table. (And for that reason, the load factor $\alpha = n/m$ is not a true indication of space usage.) The direct chaining with relocation method stores all elements directly in the hash table. A special situation arises when an element x with hash address $hash(x)$ collides during insertion with another element x'. If x' is the first element in its chain, then x is inserted into an empty slot in the table and linked to the end of the chain. Otherwise, x' is not at the start of the chain, so x' is relocated

to an empty slot in order to make room for x; the link to x' from its predecessor in the chain must be updated. The successful search time is the same as before. Unsuccessful searches can start in the middle of a chain; each chain of length $k > 0$ contributes $k(k+1)/2$ probes. A search starting at one of the $m - n$ empty slots takes one probe. This gives us

$$\overline{U_n} = X'(1) + \frac{1}{2}X''(1) + \Pr\{\text{slot } hash(x) \text{ is empty}\} = 1 + \frac{n(n-1)}{2m^2} \sim 1 + \frac{\alpha^2}{2}. \tag{9}$$

The main difficulty with this algorithm is the overhead of moving elements, which can be expensive for large record elements and might not be allowed if there are pointers to the elements from outside the hash table. Updating the previous link requires either the use of bidirectional or circular chains or recomputing the hash address of x' and following links until x' is reached. None of these alternatives are attractive, and we shall soon consider a better alternative called *coalesced hashing*, which has nearly the same number of probes per search, but without the overhead. ∎

An Admissible Construction. Another route to the analysis of separate chaining, which will be useful for our later analysis, is to consider the hash table as the m-ary partitional product of the individual buckets

$$\mathcal{H} = \mathcal{B} * \mathcal{B} * \cdots * \mathcal{B} \tag{10a}$$

à la Section 1.3. The new twist here is that some of the elements in the table are "marked." (We shall explain shortly how this relates to separate chaining.) We let $H_{k,n,m}$ be the number of m^n hash sequences for which k of the elements are marked, and we denote its EGF by

$$H(u,z) = \sum_{k,n\geq 0} H_{k,n,m} u^k \frac{z^n}{n!}. \tag{10b}$$

By analogy with Theorem 1.4 for EGFs, the following theorem shows that the partitional product in (10a) translates into a product of EGFs; the proof is similar to that of Theorem 1.4 and is omitted for brevity.

THEOREM 2. *If the number of marked elements in bucket i is a function of the number of elements in bucket i, then the EGF $H(u,z) = \sum_{k,n\geq 0} H_{k,n,m} u^k z^n/n!$ can be decomposed into*

$$H(u,z) = B_1(u,z) \cdot B_2(u,z) \cdot \ldots \cdot B_m(u,z), \tag{10c}$$

where

$$B_i(u,z) = \sum_{t\geq 0} u^{f_i(t)} \frac{z^t}{t!}, \tag{10d}$$

and $f_i(t)$ is the number of marked elements in bucket i when there are t elements in bucket i.

We are interested in the number of elements in bucket 1, so we adopt the marking rule that all elements in bucket 1 are marked, and the other elements are left unmarked. In terms of Theorem 2, we have $f_1(t) = t$ and $B_1(u,z) = \sum_{t\geq 0} u^t z^t/t! = e^{uz}$ for bucket 1, and $f_i(t) = 0$ and $B_i(u,z) = \sum_{t\geq 0} z^t/t! = e^z$ for the other buckets $2 \leq i \leq m$. Substituting this into Theorem 2, we have

$$H(u,z) = e^{(m-1+u)z}. \tag{11a}$$

We can obtain the EGF of $X_1(u)$ by dividing each $H_{k,n,m}$ term in (10b) by m^n, or equivalently by replacing z by z/m. Combining this with (11a) gives

$$H\left(u, \frac{z}{m}\right) = \sum_{n\geq 0} X(u) \frac{z^n}{n!} = e^{(m-1+u)\frac{z}{m}} \tag{11b}$$

(cf. (6b)). Hence, we have $X'(1) = [\frac{z^n}{n!}]\frac{\partial}{\partial u} H(u, z/m)\big|_{u=1}$ and $X''(1) = [\frac{z^n}{n!}]\frac{\partial^2}{\partial u^2} H(u, z/m)\big|_{u=1}$, and the formulæ for $\overline{U_n}$ and $\overline{C_n}$ follow from (5).

Distribution Sorting. Bucketing can also be used to sort efficiently when the values of the elements are smoothly distributed. Suppose for simplicity that the n values are real numbers in the unit interval $[0, 1)$. The *distribution sort* algorithm works by breaking up the range into m buckets $[0, \frac{1}{m})$, $[\frac{1}{m}, \frac{2}{m})$, $\ldots$, $[\frac{m-1}{m}, 1)$, and partitioning the elements into the buckets based upon their values. Each bucket is sorted using selection sort (cf. Section 3.5) or some other simple quadratic sorting method. The sorted buckets are then appended together to get the final sorted output.

Selection sort uses $\binom{k}{2}$ comparisons to sort k elements. The average number of comparisons $\overline{C_n}$ used by distribution sort is thus

$$\overline{C_n} = \mathbf{E}\left\{\sum_{1\le i\le m}\binom{X_i}{2}\right\} = \sum_{1\le i\le m}\mathbf{E}\left\{\frac{X_i(X_i-1)}{2}\right\} = \frac{1}{2}\sum_{1\le i\le m} X_i''(1). \tag{12a}$$

By (6c), we have $\overline{C_n} = n(n-1)/(2m)$ when the values of the elements are independently and uniformly distributed. The other work done by the algorithm takes $O(n+m)$ time, so this gives a linear-time sorting algorithm when we choose $m = \Theta(n)$. (Note that this does not contradict the well-known $\Omega(n \log n)$ lower bound on the average sorting time in the comparison model, since this is not a comparison-based algorithm.)

The assumption that the values are uniformly distributed is not always easy to justify, unlike for the case of hashing, because there is no hash function to scramble the values; the partitioning is based upon the elements' raw values. Suppose the elements are distributed according to density function $f(x)$. We assume that $\int_0^1 f(x)^2\,dx < \infty$ which assures that $f(x)$ is well-behaved. We define $p_i = \int_{A_i} f(x)\,dx$ to be the probability that an element falls into the ith bucket $A_i = [\frac{i-1}{m}, \frac{i}{m})$. It is easy to see that for general $f(x)$ formula (6b) becomes

$$X_i(u) = \big((1-p_i) + p_i u\big)^n. \tag{12b}$$

By (12a) and (12b) we have

$$\overline{C_n} = \binom{n}{2}\sum_{1\le i\le m} {p_i}^2 = \binom{n}{2}\sum_{1\le i\le m}\left(\int_{A_i} f(x)\,dx\right)^2. \tag{12c}$$

The last summation in (12c) can be bounded by an applications of Jensen's inequality:

$$\begin{aligned}\sum_{1\le i\le m}\left(\int_{A_i} f(x)\,dx\right)^2 &= \frac{1}{m^2}\sum_{1\le i\le m}\left(\int_{A_i} f(x)\,d(mx)\right)^2\\ &\le \frac{1}{m^2}\sum_{1\le i\le m}\int_{A_i} f(x)^2\,d(mx)\\ &= \frac{1}{m}\int_0^1 f(x)^2\,dx.\end{aligned} \tag{12d}$$

We can show that the bound in (12d) is asymptotically tight by computing a matching lower bound:

$$\begin{aligned}\frac{1}{m^2}\liminf_{n\to\infty}\sum_{1\le i\le m}\left(\int_{A_i} f(x)\,d(mx)\right)^2 &= \frac{1}{m^2}\liminf_{n\to\infty}\sum_{1\le i\le m}\int_{A_i} f_n(x)^2\,d(mx)\\ &= \frac{1}{m}\liminf_{n\to\infty}\int_0^1 f_n(x)^2\,dx,\end{aligned}$$

where $f_n(x) = mp_i$, for $x \in A_i$, is the histogram approximation of $f(x)$, which converges to $f(x)$ almost everywhere. By Fatou's lemma, we can get a lower bound by moving the liminf operator inside the integral:

$$\frac{1}{m}\liminf_{n\to\infty}\int_0^1 f_n(x)^2\,dx \ge \frac{1}{m}\int_0^1 \liminf_{n\to\infty} f_n(x)^2\,dx = \frac{1}{m}\int_0^1 f(x)^2\,dx.$$

Hence, substituting our approximation into (12c), we find that the average number of comparisons is

$$\overline{C_n} \sim \frac{n^2}{2m} \int_0^1 f(x)^2 \, dx.$$

The coefficient of the linear term when $m = \Theta(n)$ is proportional to $\int_0^1 f(x)^2 \, d(x)$, which can be very large. The erratic behavior due to nonuniform $f(x)$ can be alleviated by one level of recursion, in which the above algorithm is used to sort the individual buckets: If the number X_i of elements in bucket i is more than c, for some predetermined constant c, we sort the bucket by breaking up the range $[\frac{i-1}{m}, \frac{i}{m})$ into X_i subbuckets, and proceed as before. The surprising fact, which can be shown by techniques similar to above, is that $\overline{C_n}$ is bounded by $n/2$ in the limit, regardless of $f(x)$ (assuming of course that our assumption $\int_0^1 f(x)^2 \, dx < \infty$ is satisfied).

THEOREM 3 [Devroye 1986b]. *The expected number of comparisons $\overline{C_n}$ done by two-level bucketing to sort n elements that are independently distributed with density function $f(x)$, which satisfies $\int_0^1 f(x)^2 \, dx < \infty$, is*

$$\overline{C_n} \sim \frac{n}{2} \int_0^1 e^{-f(x)} \, dx \leq \frac{n}{2}.$$

The variance and higher moments of the number of probes are also small. If the unit interval assumption is not valid and the values of the elements are not bounded, we can redefine the interval to be $[x_{\min}, x_{\max}]$ and apply the same basic idea given above. The analysis becomes a litle more complicated. Details appear in [Devroye 1986b].

For the actual implementation, a hashing scheme other than separate chaining can be used to store the elements in the table. Sorting with linear probing is discussed at the end of the section. An application of bucketing to fast sorting on associative secondary storage devices appears in [Lindstrom and Vitter 1985].

Interpolation Search. Newton's method and the secant method are well-known schemes for determining the leading k bits of a zero of a continuous function $f(x)$ in $O(\log_2 k)$ iterations. (By "zero," we mean a solution x to the equation $f(x) = 0$.) Starting with an initial approximation x_0, the methods produce a sequence of refined approximations $x_1, x_2, \ldots$ that converge to a zero x^*, assuming that $f(x)$ is well-behaved. The discrete analogue is called *interpolation search*, in which the n elements are in a sorted array, and the goal is to find the element x^* with a particular value c.

THEOREM 4 [Yao and Yao 1976], [Gonnet, Rogers, and George 1980]. *The average number of comparisons per successful search using interpolation search on a sorted array is $\sim \log_2 \log_2 n$, assuming that the elements' values are independently and identically distributed.*

This bound is similar to that of the continous case above; we can think of the accuracy k as being $\log_2 n$, the number of bits needed to specify the array position of x^*.

PROOF SKETCH. Gonnet, Rogers, and George [1980] show by some probabilistic arguments that the probability that at least k probes are needed to find x^* is bounded by

$$p_k(t) = \prod_{1 \leq i \leq k} \left(1 - \frac{1}{2} e^{-t2^{-i}}\right), \tag{13a}$$

where $t = \log(\pi n/8)$. Hence, the expected number of probes is bounded by

$$F(t) = \sum_{k \geq 0} p_k(t), \tag{13b}$$

which can be expressed in terms of the harmonic sum

$$F(t) = \frac{1}{Q(t)} \sum_{k \geq 0} Q(t2^k), \tag{13c}$$

where

$$Q(t) = \prod_{i \geq 1} \left(1 - \frac{1}{2} e^{-t2^{-i}}\right)^{-1}. \tag{13d}$$

The sum in (13c) is a harmonic sum to which Mellin transforms can be applied to get

$$F(t) \sim \log_2 t + \alpha + P(\log_2 t) + o(1), \qquad \text{as } t \to \infty, \tag{13e}$$

where α is a constant and $P(u)$ is a periodic function associated with the poles at $\chi_k = 2ik\pi/\log 2$. The $\log_2 \log_2 n$ bound follow by substituting $t = \log(\pi n/8)$ into (13e). ∎

Maximum Bucket Occupancy. An interesting statistic that lies between average-case and worst-case search times is the expected number of elements in the *largest* bucket (also known as the *maximum bucket occupancy*). It has special significance in parallel processing applications in which elements are partitioned randomly into buckets and then the buckets are processed in parallel, each in linear time; in this case, the expected maximum bucket occupancy determines the average running time.

We can make use of the product decomposition (10*a*) and Theorem 2 to count the number of hash sequences that give a hash table with $\leq b$ elements in each bucket. We mark all the elements in a bucket if the bucket has $> b$ elements; otherwise, the elements are left unmarked. In this terminology, our quantity of interest is simply the number $H_{0,n,m}$ of hash sequences with no marked elements:

$$H_{0,n,m} = \left[u^0 \frac{z^n}{n!}\right] H(u,z) = \left[\frac{z^n}{n!}\right] H(0,z) = \left[\frac{z^n}{n!}\right] \left(B_1(0,z) \cdot B_2(0,z) \cdot \ldots \cdot B_m(0,z)\right), \tag{14a}$$

where

$$B_i(0,z) = \sum_{0 \leq n \leq b} \frac{z^n}{n!}, \tag{14b}$$

for $1 \leq i \leq m$. The sum in (14*b*) is the truncated exponential, which we denote by $e_b(z)$. Hence, the number of hash sequences with $\leq b$ elements per bucket is

$$H_{0,n,m} = \left[\frac{z^n}{n!}\right] \left(e_b(z)\right)^m. \tag{14c}$$

We use $q_n^{[b]}$ to denote the probability that a random hash sequence gives $\leq b$ elements per bucket. As in (11*b*), we can convert from the enumeration $H_{0,n,m}$ to the probability $q_n^{[b]}$ by replacing z in (14*c*) by z/m:

$$q_n^{[b]} = \left[\frac{z^n}{n!}\right] \left(e_b\left(\frac{z}{m}\right)\right)^m. \tag{14d}$$

There is a close relation between the EGF of Bernoulli statistics and the corresponding Poisson statistic:

THEOREM 5. *If $g(z) = \sum_{n \geq 0} g_n z^n/n!$ is the EGF for a measure g_n (for example, probability, expectation, moment) in the Bernoulli model, then $e^{-\alpha} g(\alpha)$ is the corresponding measure if the total number of elements $X_1 + \cdots + X_m$ is Poisson with mean α.*

PROOF. The measure in the Poisson model is the conditional expectation of the measure in the Bernoulli model, namely,

$$\sum_{n \geq 0} g_n \Pr\{X_1 + \cdots + X_m = n\} = \sum_{n \geq 0} g_n \frac{e^{-\alpha}\alpha^n}{n!} = e^{-\alpha} g(\alpha). \quad \blacksquare$$

We shall use Theorem 5 and direct our attention to the Poisson model, where the number of elements in each bucket is Poisson distributed with mean α, and hence the total number of elements is a Poisson RV with mean $m\alpha$. The analysis of the Bernoulli case can be handled in much the same way that we shall handle the analysis of extendible hashing later in this section, so covering the Poisson case will present us with a different perspective.

We let $p_\alpha^{[b]}$ denote the probability that a random hash sequence yields $\leq b$ elements per bucket in the Poisson model. By Theorem 5, we have

$$p_\alpha^{[b]} = \left(e^{-\alpha} e_b(\alpha)\right)^m.$$

(This can also be derived directly by noting that the m buckets are independent and that the Poisson probability that a given bucket has $\leq b$ elements is $e^{-\alpha} e_b(\alpha)$.) What we want is to compute the expected maximum bucket size

$$\overline{M_\alpha} = \sum_{b \geq 1} b\left(p_\alpha^{[b]} - p_\alpha^{[b-1]}\right) = \sum_{b \geq 0} \left(1 - p_\alpha^{[b]}\right). \tag{15}$$

We shall consider the case $\alpha = o(\log m)$ (although the basic principles of our analysis will apply for any α). A very common occurrence in occupancy RVs is a sharp rise in the distribution $p_\alpha^{[b]}$ from ≈ 0 to ≈ 1 in the "central region" near the mean value. Intuitively, a good approximation for the mean is the value $b = \tilde{b}$ such that $p_\alpha^{[\tilde{b}]}$ is sufficiently away from 0 and 1. We choose the value $b = \tilde{b}$ such that

$$\frac{e^{-\alpha}\alpha^{\tilde{b}+1}}{\tilde{b}!} \approx \frac{\lambda}{m}, \tag{16}$$

for some positive constant λ. The following bound illustrates the sharp increase in $p_\alpha^{[b]}$ in the vicinity $b \approx \tilde{b}$:

$$\begin{aligned} p_\alpha^{[\tilde{b}+k]} &= \left(e^{-\alpha} e_{\tilde{b}+k}(\alpha)\right)^m = \left(1 - e^{-\alpha} \sum_{b>\tilde{b}+k} \frac{\alpha^b}{b!}\right)^m \\ &\sim \left(1 - \frac{e^{-\alpha}\alpha^{\tilde{b}+k+1}}{(\tilde{b}+k+1)!}\right)^m \sim \left(1 - \frac{\lambda\alpha^k}{m\tilde{b}^k}\right)^m \sim e^{-\lambda\alpha^k/\tilde{b}^k}. \end{aligned} \tag{17}$$

The approximation is valid uniformly for $k = o(\sqrt{\tilde{b}})$. For larger k, (17) no longer holds, but $p_\alpha^{[\tilde{b}+k]}$ is still exponentially decreasing with respect to k. The net effect is that we can get an asymptotic approximation for $\overline{M_\alpha}$ by approximating $1 - p_\alpha^{[b]}$ in (15) by a 0–1 step function with the step at $b = \tilde{b}$:

$$\overline{M_\alpha} \sim \sum_{0\le b<\tilde{b}} (1) + \sum_{b\ge\tilde{b}} (0) = \tilde{b}. \tag{18}$$

When $\alpha = \Theta(1)$, it is easy to see from (16) that $\tilde{b} \sim \Gamma^{-1}(m) \sim (\log m)/\log\log m$. (Here $\Gamma^{-1}(y)$ denotes the inverse of the Gamma function $\Gamma(x)$.) The same techniques can be applied for general α. The asymptotic behavior of $\overline{M_\alpha}$ for the Bernoulli and Poisson models is the same. A heuristic argument for why this should be so is given in [Gonnet 1981].

THEOREM 6 [Kolchin et al 1978]. *In the Bernoulli model with n elements inserted in m buckets ($\alpha = n/m$) or in the Poisson model in which the occupancy of each bucket is an independent Poisson RV with mean α, the expected maximum bucket occupancy is*

$$\overline{M_\alpha} \sim \begin{cases} \dfrac{\log m}{\log\log m}, & \text{if } \alpha = \Theta(1); \\ \tilde{b}, & \text{if } \alpha = o(\log m); \\ \alpha, & \text{if } \alpha = \omega(\log m), \end{cases}$$

where $\tilde{b}$ is given by (16).

When $\alpha = o(\log m)$, the maximum bucket size is equal to $\tilde{b}$ with probability $\sim e^{-\lambda}$ and to $\tilde{b}+1$ with probability $\sim 1 - e^{-\lambda}$. When α gets large, $\alpha = \omega(\log m)$, the bucket occupancies become fairly uniform; the difference $\overline{M_\alpha} - \alpha$ converges in probability to $\sim \sqrt{2\alpha\log m}$, provided that $\alpha = m^{O(1)}$.

Extendible Hashing. A related problem is the expected directory size used in *extendible hashing*, in which the hash table is allowed to grow and shrink dynamically. Each slot in the hash table models a page of secondary storage capable of storing up to b elements. If a bucket overflows, the number of buckets in the table is successively doubled until each bucket has $\le b$ elements. The directory acts a b-trie, based upon the infinite precision hash addresses of the elements (cf. Section 4.3). For this reason, the analyses of directory size and trie height are very closely related.

At any given time, the directory size is equal to the number of buckets in the table, which is always a power of 2. The probability $\pi_n^{[h]}$ that the directory size is $\le 2^h$ is

$$\pi_n^{[h]} = \left[\frac{z^n}{n!}\right] \left(e_b\left(\frac{z}{2^h}\right)\right)^{2^h}. \tag{19a}$$

This is identical to (14d) with $m = 2^h$, except that in this case m is the parameter that varies, and the bucket capacity b stays fixed. We can also derive (19a) via the admissibility theorem for tries (Theorem 4.18): The probability $\pi_n^{[h]}$ is the probability that the height of a random b-trie is $\le h$. The EGF $\pi^{[h]}(z) = \sum_{n\ge0} \pi_n^{[h]} z^n/n!$ satisfies

$$\pi^{[h]}(z) = \left(\pi^{[h-1]}\left(\frac{z}{2}\right)\right)^2; \qquad \pi^{[0]}(z) = e_b(z). \tag{19b}$$

Hence, (19a) follows.

THEOREM 7 [Flajolet 1983]. *In the Bernoulli model, the expected directory size in extendible hashing for bucket size $b > 1$ when there are n elements present is*

$$\overline{S_n} \sim Q\left(\left(1+\frac{1}{b}\right)\log_2 n\right) n^{1+1/b},$$

where $Q(u)$ is a periodic function with period 1 and mean value $\Gamma(1-1/b)/\big((\log 2)(b+1)!^{1/b}\big)$.

PROOF. We can express the average directory size $\overline{S_n}$ in terms of $\pi_n^{[h]}$ in a way similar to (15):

$$\overline{S_n} = \sum_{h\geq 1} 2^h(\pi_n^{[h]} - \pi_n^{[h-1]}) = \sum_{h\geq 0} 2^h(1-\pi_n^{[h]}). \tag{20}$$

The first step in the derivation is to apply the saddle point method of Section 2. We omit the details for brevity. As for the maximum bucket occupancy, the probabilities $\pi_n^{[h]}$ change quickly from ≈ 0 to ≈ 1 in a "central region," which in this case is where $h = \tilde{h} = (1+1/b)\log_2 n$. By saddle point, we get the uniform approximation

$$\pi_n^{[h]} \sim \exp\left(\frac{-n^{b+1}}{(b+1)!2^{bh}}\right), \tag{21}$$

for $|h-\tilde{h}| < \log_2 \log n$. When $h \geq \tilde{h}+\log_2 \log n$, approximation (21) is no longer valid, but the terms $1-\pi_n^{[h]}$ continue to decrease exponentially with respect to h. Hence, we can subsitute approximation (21) into (20) without affecting the leading terms of $\overline{S_n}$. We get $\overline{S_n} \sim T\big(n^{b+1}/(b+1)!\big)$, where $T(x)$ is the harmonic sum

$$T(x) = \sum_{h\geq 0} 2^h\left(1-e^{-x/2^{bh}}\right), \tag{22a}$$

whose Mellin transform is

$$T^*(s) = \frac{-\Gamma(s)}{1-2^{bs+1}} \tag{22b}$$

in the fundamental strip $\langle -1; -1/b\rangle$. The asymptotic expansion of $T(x)$, as $x \to \infty$, is given by the poles of $T^*(s)$ to the right of the strip. There is a simple pole at $s = 0$ due to $\Gamma(s)$ and simple poles at $s = -1/b + 2ik\pi/\log_2$ due to the denominator. The result follows immediately. ∎

Theorem 7 shows that the leading term of $\overline{S_n}$ oscillates with period $(1+1/b)\log_2 n$. An intuition as to why there is oscillation can be found in the last summation in (20). The sum samples the terms $1-\pi_n^{[h]}$ at each nonnegative integer h using the exponential weight function 2^h. The value of h where the approximation (21) changes quickly from ≈ 0 to ≈ 1 is close to an integer value only when $(1+1/b)\log_2 n$ is close to an integer, thus providing the periodic effect.

It is also interesting to note from Theorem 7 that the directory size is asymptotically superlinear, that is, the directory becomes larger than the file itself when n is large! Fortunately, convergence is slow, and the nonlinear growth of $\overline{S_n}$ is not noticable in practice when b is large enough, say $b \geq 40$. Similar results for the Poisson model appear in [Régnier 1981].

The same techniques apply to the analysis of the expected height $\overline{H_n}$ of tries:

$$\overline{H_n} = \sum_{h\geq 1} h(\pi_n^{[h]} - \pi_n^{[h-1]}) = \sum_{h\geq 0}(1-\pi_n^{[h]}). \tag{23}$$

This is the same as (20), but without the weight factor 2^h. (When trie height grows by 1, directory size doubles.)

THEOREM 8 [Flajolet 1983]. *The expected height in the Bernoulli model of a random b-trie having n keys is*

$$\overline{H_n} = \left(1+\frac{1}{b}\right)\log_2 n + P\left(\left(1+\frac{1}{b}\right)\log_2 n\right) + o(1),$$

where $P(u)$ is periodic with period 1 and small amplitude about the mean $1/2 + \big(\gamma - \log\big((b+1)!\big)\big)/(b\log 2)$.

In the biased case, where 0 occurs with probability p and 1 occurs with probability $q = 1-p$, we have

$$\pi^{[h]}(z) = \pi^{[h-1]}(pz)\cdot\pi^{[h-1]}(qz),$$

which gives us

$$\pi^{[h]}(z) = \prod_{1\leq k\leq h}\left(e_b(p^k q^{n-k} z)\right)^{\binom{h}{k}}$$

(cf. (19a) and (19b)). Multidimensional versions have been studied in [Régnier 1985].

Coalesced Hashing. We can bypass the problems of direct chaining with relocation by using *hashing with coalescing chains* (or simply *coalesced hashing*). Part of the hash table is dedicated to storing elements that collide when inserted. We redefine m' to be the number of slots in the hash table. The range of the hash function is restricted to $\{1, 2, \ldots, m\}$. We call the first m slots the *address region*; the bottom $m' - m$ slots, which is for storing colliders, is called the *cellar*.

When a collision occurs during insertion because the element's hash address is already occupied, the element is inserted instead into an empty slot in the cellar and linked to the end of the chain. If there is no room in the cellar, the element must be stored directly in the address region. Elements inserted later might collide with this element, and their chains would "coalesce."

If the cellar size is chosen so that it can accomodate all the colliders, then coalesced hashing reduces to separate chaining. It is somewhat surprising that performance can be improved by choosing a smaller cellar size so that coalescing occurs. The intuition is that by making the address region larger, the hash addresses of the elements are spread out over a larger area, which helps reduce collisions. This offsets the disadvantages of coalescing, which typically occurs much later. We might be tempted to go the the extreme and eliminate the cellar completely (this variant is called *standard coalesced hashing*), but performance deteriorates. The theorem below gives the expected search times as a function of the load factor $\alpha = n/m$ and the address factor $\beta = m/m'$. We can use the theorem to determine the optimum β. It turns out that β_{opt} is a function of α and of the type of search, but the compromise value $\beta = 0.86$ gives near optimum search performance for a large range of α and is recommended for general use.

THEOREM 9 [Vitter and Chen 1987]. *The expected number of probes per search for coalesced hashing in an m'-slot table with address size $m = \beta m'$ and with $n = \alpha m'$ elements is*

$$\overline{U_n} \sim \begin{cases} e^{-\alpha/\beta} + \dfrac{\alpha}{\beta}, & \text{if } \alpha \le \lambda\beta; \\ \dfrac{1}{\beta} + \dfrac{1}{4}\left(e^{2(\alpha/\beta - \lambda)} - 1\right)\left(3 - \dfrac{2}{\beta} + 2\lambda\right) - \dfrac{1}{2}\left(\dfrac{\alpha}{\beta} - \lambda\right), & \text{if } \alpha \ge \lambda\beta; \end{cases}$$

$$\overline{C_n} \sim \begin{cases} 1 + \dfrac{\alpha}{2\beta}, & \text{if } \alpha \le \lambda\beta; \\ 1 + \dfrac{\beta}{8\alpha}\left(e^{2(\alpha/\beta - \lambda)} - 1 - 2\left(\dfrac{\alpha}{\beta} - \lambda\right)\right)\left(3 - \dfrac{2}{\beta} + 2\lambda\right) & \\ \quad + \dfrac{1}{4}\left(\dfrac{\alpha}{\beta} + \lambda\right) + \dfrac{\lambda}{4}\left(1 - \dfrac{\lambda\beta}{\alpha}\right), & \text{if } \alpha \ge \lambda\beta, \end{cases}$$

where λ is the unique nonnegative solution to the equation $e^{-\lambda} + \lambda = 1/\beta$.

The method described above is formally known as *late-insertion coalesced hashing* (LICH). Vitter and Chen also analyze two other methods called *early-insertion coalesced hashing* (EICH) and *varied-insertion coalesced hashing* (VICH). In EICH, a colliding element is inserted immediately after its hash address in the chain, by rerouting pointers. VICH uses the same rule, except when there is a cellar slot in the chain following the element's hash address; in that case, the element is linked into the chain immediately after the last cellar slot in the chain. VICH requires slightly fewer probes per search than the other variants, and appears optimum among all possible linking methods. Deletion algorithms and implementation issues are also covered in [Vitter and Chen 1987].

PROOF. We first consider the unsuccessful search case. We count the number $m^{n+1}\overline{U_n}$ of probes needed to perform all possible unsuccessful searches in all possible hash tables of m elements. Each chain of length ℓ (which we call an (ℓ, t)-chain) contributes

$$\delta_{\ell=0} + \ell + (\ell - t - 1) + (\ell - t - 2) + \cdots + 1 = \delta_{\ell=0} + \ell + \binom{\ell - t}{2} \tag{24a}$$

probes, and we have

$$m^{n+1}\overline{U_n} = m^{n+1}p_n + \sum_{\ell,t} \ell\, c_n(\ell, t) + \sum_{\ell,t} \binom{\ell - t}{2} c_n(\ell, t), \tag{24b}$$

where p_n is the probability that the hash address of the element is unoccupied, and $c_n(\ell,t)$ is the number of (ℓ,t)-chains. The second term is easily seen to be nm^n, since there are n elements in each hash table. The evaluations of the first term $m^{n+1}p_n$ and the third term

$$S_n = \sum_{\ell,t} \binom{\ell-t}{2} c_n(\ell,t) \tag{24c}$$

are similar; for brevity we restrict our attention to the latter. There does not seem to be a closed form expression for $c_n(\ell,t)$, but we can develop a recurrence for $c_n(\ell,t)$ which we can substitute into (24c) to get

$$S_n = (m+2)^{n-1} \sum_{0\le j\le n-1} \left(\frac{m}{m+2}\right)^j (j-m'+m)\frac{F_j}{m^j}, \tag{25}$$

where F_j is the number of hash sequences of j elements that yield full cellars. In terms of probabilities, F_j/m^j is the probability that the cellar is full after j insertions.

This provides a link to the occupancy distributions studied earlier. We define the RV N_j to be the number of elements that collide when inserted, in a hash table of j elements. The cellar is full after k insertions if and only if $N_j \ge m'-m$; by taking probabilities, we get

$$\frac{F_j}{m^j} = \Pr\{N_j \ge m'-m\}. \tag{26}$$

This expresses F_j/m^j as a tail of the probability distribution for N_j. We can determine the distribution by applying Theorem 2. We let $H_{k,j,m}$ be the number of hash sequences of n elements for which k elements are marked. Our marking rule is that all elements that collide when inserted are marked; that is, for each bucket (hash address) i with t elements, there are $f_i(t) = t-1+\delta_{t=0}$ marked elements. By Theorem 2, we have

$$H(u,z) = \sum_{k,j\ge 0} H_{k,j,m} u^k \frac{z^j}{j!} = \prod_{1\le i\le m} B_i(u,z), \tag{27a}$$

where

$$B_i(u,z) = \sum_{t\ge 0} u^{t-1+\delta_{t=0}} \frac{z^t}{t!} = \frac{e^{uz}-1+u}{u}, \tag{27b}$$

for each $1 \le i \le m$. Substituting (27b) into (27a), we get

$$H(u,z) = \left(\frac{e^{uz}-1+u}{u}\right)^m. \tag{27c}$$

This allows us to solve for $F_j/m^j = \Pr\{N_j \ge m'-m\}$:

$$\Pr\{N_j = k\} = \frac{H_{k,j,m}}{m^j} = \left[u^k \frac{z^j}{j!}\right] H\left(u, \frac{z}{m}\right); \tag{28a}$$

$$F_j/m^j = 1 - \left[u^{c-1}\frac{z^j}{j!}\right] \frac{1}{1-u} H\left(u, \frac{z}{m}\right). \tag{28b}$$

We can get asymptotic estimates for (28b) by use of saddle point as in our analysis of extendible hashing or by Chernoff bounds as in [Vitter and Chen 1987]; the details are omitted for brevity. The distribution F_j/m^j increases sharply from ≈ 0 to ≈ 1 in the "central region" where

$$\overline{N_j} \approx m'-m. \tag{29a}$$

The expected number of collisions $\overline{N_j}$ is

$$\overline{N_j} = \left[\frac{z^j}{j!}\right] \frac{\partial}{\partial u} H\left(u, \frac{z}{m}\right)\Big|_{u=1} = j - m + m\left(1-\frac{1}{m}\right)^j. \tag{29b}$$

We can solve for the value of $j = \tilde{j}$ that satisfies (29a) by combining (29a) and (29b) and using the ratios $\tilde{\alpha} = \tilde{j}/m'$ and $\beta = m/m'$. We have

$$e^{-\tilde{\alpha}} + \tilde{\alpha} \approx \frac{1}{\beta}. \tag{29c}$$

In a way similar to (18), we approximate F_j/m^j by a 0–1 step function with the step at $j = \tilde{j}$, and we get

$$S_n \sim (m+2)^{n-1} \sum_{\tilde{j} \le j \le n-1} \left(\frac{m}{m+2}\right)^j (j - m' + m), \tag{30}$$

which can be summed easily. The error of approximation can be shown to be negligible as in the analysis of maximum bucket size and extendible hashing. The analysis of the term $m^{n+1}p_n$ in (24b) is based upon similar ideas and is omitted for brevity.

For the case of successful searches, the formula for $\overline{C_n}$ follows from a formula similar to (7):

$$\overline{C_n} = 1 + \frac{1}{n} \sum_{0 \le i \le n-1} (\overline{U_i} - p_i), \tag{31}$$

where p_i is the term in (24b). ∎

A unified analysis of all three variants of coalesced hashing—LICH, EICH, and VICH—is given in [Vitter and Chen, 1987]. The maximum number of probes per search among all the searches in the same table, for the special case of standard LICH (when there is no cellar and $m = m'$), is shown in [Pittel 1987b] to be $\log_c n - 2\log_c \log n + O(1)$ with probability ~ 1 for successful searches and $\log_c n - \log_c \log n + O(1)$ with probability ~ 1 for unsuccessful searches, where $c = 1/(1 - e^{-\alpha})$.

5.2. Hashing by Open Addressing

The alternative to chaining is to probe an implicitly defined sequence of locations while looking for an element. If the element is found, the search is successful; if an "open" (empty) slot is encountered, the search is unsuccessful, and the new element is inserted into the empty slot that terminated the search.

Uniform Probing. A simple scheme to analyze, which serves as a good approximation to more practical methods, is *uniform probing*. The probing sequence for each element x is a random permutation $(h_1(x), h_2(x), \ldots, h_m(x))$ of $\{1, 2, \ldots, m\}$. For an unsuccessful search, the probability that k probes are needed when there are n elements in the table is

$$p_k = \frac{n^{\underline{k-1}}(m-n)}{m^{\underline{k}}} = \binom{m-k}{m-n-1} \Big/ \binom{m}{n}. \tag{32a}$$

Hence, we have

$$\overline{U_n} = \sum_{k \ge 0} k p_k = \frac{1}{\binom{m}{n}} \sum_{k \ge 0} k \binom{m-k}{m-n-1}. \tag{32b}$$

We split k into two parts $k = (m+1) - (m-k+1)$, because we can handle each separately; the $m-k+1$ term gets "absorbed" into the binomial coefficient.

$$\begin{aligned}
\overline{U_n} &= \frac{1}{\binom{m}{n}} \sum_{k \ge 0} (m+1) \binom{m-k}{m-n-1} - \frac{1}{\binom{m}{n}} \sum_{k \ge 0} (m-k+1) \binom{m-k}{m-n-1} \\
&= m + 1 - \frac{m-n}{\binom{m}{n}} \sum_{k \ge 0} \binom{m-k+1}{m-n} \\
&= m + 1 - \frac{m-n}{\binom{m}{n}} \binom{m+1}{m-n+1} \\
&= \frac{m+1}{m-n+1}.
\end{aligned} \tag{32c}$$

Successful search time for open addressing algorithms satisfy the same relation as (4.25):

$$\overline{C_n} = \frac{1}{n} \sum_{0 \le i \le n-1} \overline{U_i}. \tag{33}$$

Putting this all together, we get

THEOREM 10. *The expected number of probes per unsuccessful and successful seach for uniform probing, when there are $n = m\alpha$ elements in a hash table of m slots, is*

$$\overline{U_n} = \frac{m+1}{m-n+1} \sim \frac{1}{1-\alpha}; \qquad \overline{C_n} = \frac{m+1}{n}(H_{m+1} - H_{m-n+1}) \sim \frac{1}{\alpha}\log\frac{1}{1-\alpha}.$$

The asymptotic formula $\overline{U_n} \sim 1/(1-\alpha) = 1 + \alpha + \alpha^2 + \ldots$ has the followng intuitive interpretation: With probability α we need more than one probe, with probability α^2 we need more than two probes, and so on. The expected maximum search time per hash table is studied in [Gonnet 1981].

Double Hashing and Secondary Clustering. The practical limitation of uniform probing is that computing several hash functions is very expensive. However, the performance of uniform probing can be approximated well by use of just two hash functions. In the *double hashing* method, the ith probe is made at slot

$$h_1(x) + ih_2(x) \bmod m, \qquad \text{for } 0 \le i \le m-1; \tag{34}$$

that is, the probe sequence starts at slot $h_1(x)$ and steps cyclically through the table with step size $h_2(x)$. (For simplicity, we have renumbered the m slots to be $0, 1, \ldots, m-1$.) The two values of the two hash function must be relatively prime to one another so that the probe sequence (34) gives a full permutation. Guibas and Szemerédi [1978] show using interesting probabilistic techniques that when $\alpha < 0.319$ the number of probes per search is asymptotically equal to that of uniform probing.

A variation that requires only one hash function, called *hashing with secondary clustering*, defines $h_2(x)$ implicitly in terms of $h_1(x)$. For example, if m is prime, we can set

$$h_2(x) = \begin{cases} 1, & \text{if } h_1(x) = 0; \\ m - h_1(x), & \text{if } h_1(x) > 0. \end{cases} \tag{35}$$

THEOREM 11 [Knuth 1973b]. *The expected number of probes per unsuccessful and successful seach for hashing with secondary clustering, when there are $n = m\alpha$ elements in a hash table of m slots, is*

$$\overline{U_n} \sim \frac{1}{1-\alpha} - \alpha + \log\frac{1}{1-\alpha}; \qquad \overline{C_n} \sim 1 + \log\frac{1}{1-\alpha} + \frac{\alpha}{2}.$$

The proof is a generalization of the method we will use for linear probing below. The number of probes per search for secondary clustering is slightly more than for double hashing and uniform probing, but secondary clustering is often faster in practice, since the overhead of computing two hash functions is eliminated.

Linear Probing. Perhaps the simplest implementation of open addressing is a further extension of (34) and (35), called *linear probing*, in which the unit step size $h_2(x) \equiv 1$ is used. This causes *primary clustering* in the table, because all the elements with the same hash address follow the same probing sequence.

THEOREM 12 [Knuth 1973b]. *The expected number of probes per unsuccessful and successful seach for linear probing, when there are $n = m\alpha$ elements in a hash table of m slots, is*

$$\overline{U_n} = \frac{1}{2}\big(1 + Q_0(m, n-1)\big); \qquad \overline{C_n} = \frac{1}{2}\big(1 + Q_1(m, n)\big), \tag{36}$$

where

$$Q_r(m, n) = \sum_{k \ge 0} \binom{r+k}{k} \frac{n^{\underline{k}}}{m^k}.$$

If $\alpha = n/m$ is a constant bounded away from 1, then we have

$$\overline{C_n} \sim \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right); \qquad \overline{U_n} \sim \frac{1}{2}\left(1 + \frac{1}{(1-\alpha)^2}\right).$$

For full tables, we have

$$\overline{C_m} \sim \frac{1}{2}\sqrt{\frac{\pi m}{2}} + \frac{1}{3} + \frac{1}{24}\sqrt{\frac{\pi}{2m}} + O(1/m); \qquad \overline{U_{m-1}} = \frac{m+1}{2}.$$

PROOF. The derivation of the exact formulæ for $\overline{C_n}$ and $\overline{U_n}$ is an exercise in combinatorial manipulation. The number of the m^n hash sequences such that slot 0 is empty is

$$f(m,n) = \left(1 - \frac{n}{m}\right) m^n. \tag{37}$$

By decomposing the the hash table into two separate parts, we find that the number of hash sequences such that position 0 is empty, positions 1 through k are occupied, and position $k+1$ is empty is

$$g(m,n,k) = \binom{n}{k} f(k+1,k) f(m-k-1, n-k). \tag{38}$$

The probability that $k+1$ probes are needed for an unsuccessful search is thus

$$p_k = \frac{1}{m^n} \sum_{k \le j \le n} g(m,n,j). \tag{39}$$

The formulæ (36) for $\overline{U_n} = \sum_{0 \le k \le n} (k+1) p_k$ and $\overline{C_n} = \frac{1}{n} \sum_{0 \le i \le n-1} \overline{U_i}$ in terms of $Q_0(m, n-1)$ and $Q_1(m,n)$ follow by applications of Abel's identity (cf. Section 1.4).

When α is bounded below 1, then we can evaluate $Q_r(m,n)$ asymptotically by approximating $n^{\underline{k}}/m^k$ in the summations by $n^k/m^k = \alpha^k$. The interesting case as far as analysis is concerned is for the full table when $\alpha \approx 1$. It is convenient to define the new terminology

$$Q_{\{b_k\}} = \sum_{k \ge 0} b_k \frac{(m-1)^{\underline{k}}}{m^k}. \tag{40a}$$

The link between our new terminology and the old terminology is

$$Q_r(m, m-1) = Q_{\{\binom{r+k}{k}\}}(m). \tag{40b}$$

Note that the Q functions each have only a finite number of nonzero terms. The following powerful theorem provides asymptotic expansions for several choices of $\{b_k\}$:

THEOREM 13. *We have*

$$Q_{\{b_k\}}(m) = \frac{m!}{m^{m-1}} [z^m] A(y(z)), \tag{41a}$$

where $y(z)$ is defined implicitly by the equation

$$y(z) = z e^{y(z)}, \tag{41b}$$

and $A(u)$ is the antiderivative of $B(u) = \sum_{k \ge 0} b_k u^k$.

PROOF OF THEOREM 13. The proof uses an application of Lagrange-Bürmann inversion (Theorem 1.6) applied to $y(z)$. The motivation is based upon the fact that sums of the form (40a) are associated with the implicitly defined function $y(z)$ in a natural way; the number of labeled oriented trees on m vertices is m^{m-1}, and its EGF is $y(z)$. Also, $y(z)$ was used in Section 1.4 to derive Abel's identity, and Abel's identity was used above to derive (36). By applying Lagrange-Bürmann inversion to the right-hand side of (41a), with $\Phi(u) = e^u$ and $\varphi(u) = A(u)$, we get

$$\begin{aligned}
\frac{m!}{m^{m-1}} \frac{1}{m} [u^{m-1}] e^{um} A'(u) &= \frac{(m-1)!}{m^{m-1}} [u^{m-1}] e^{um} B(u) \\
&= \frac{(m-1)!}{m^{m-1}} \sum_{0 \le k \le m-1} b_k \frac{m^{m-k-1}}{(m-k-1)!} \\
&= \sum_{0 \le k \le m-1} b_k \frac{(m-1)^{\underline{k}}}{m^k} \\
&= Q_{\{b_k\}}(m). \quad \blacksquare
\end{aligned}$$

By (40*b*) the sequences $\{b_k\}$ corresponding to $Q_0(m, m-1)$ and $Q_1(m, m-1)$ have generating functions $B_0(u) = 1/(1-u)$ and $B_1(u) = 1/(1-u)^2$, and the corresponding antiderivatives are $A_0(u) = \log\big(1/(1-u)\big)$ and $A_1(u) = 1/(1-u)$. By applying Theorem 13, we get

$$Q_0(m, m-1) = \frac{m!}{m^{m-1}}[z^m]\log\frac{1}{1-y(z)}; \qquad Q_1(m, m-1) = \frac{m!}{m^{m-1}}[z^m]\frac{1}{1-y(z)}. \tag{42}$$

It follows from the implicit formula (41*b*) for $y(z)$ that the dominant singularity of $\log\big(1/(1-y(z))\big)$ (the z of smallest modulus for which $y(z) = 1$) is at $z = 1/e$. Using the method we used to expand $T(z)$ in the analysis of tree height in Section 4.1, we get

$$\log\frac{1}{1-y(z)} = \frac{1}{2}\log\frac{1}{1-ez} - \frac{1}{2}\log 2 + O(\sqrt{1-ez}). \tag{43}$$

The approximation for $\overline{C_n}$ follows by extracting coefficients from (43). The formula $\overline{U_{m-1}} = (m+1)/2$ can be obtained by an analysis of the right-hand side of (42), or more directly by counting the number of probes needed for the m unsuccessful searches in a hash table with only one empty slot. ∎

The length of the maximum search per table for fixed $\alpha < 1$ is shown in [Pittel 1987a] to be $\Theta(\log n)$, on the average. Amble and Knuth [1974] consider a variation of linear probing, in which each cluster of elements in the table is kept in the order that would occur if the elements were inserted in increasing order by element value. Elements are relocated within a cluster, if necessary, during an insertion. The average number of probes per unsuccessful search decreases to $\overline{C_{n-1}}$. Linear probing can also be used for sorting, in a way similar to distribution sort described earlier, except that the elements are stored in a linear probing hash table rather than in buckets. The n elements in the range $[a, b)$ are inserted into the table using the hash function $h_1(x) = \lfloor (x-a)/(x-b) \rfloor$. The elements are then compacted and sorted via insertion sort. The hash table size m should be chosen to be somewhat larger than n (say, $n/m \approx 0.8$), so that insertion times for linear probing are fast.

6. Dynamic Algorithms

In this section we study performance measures that reflect the dynamic performance of a data structure over an interval of time, during which several operations may occur. In Section 6.1, we analyze the performance of some well-known implementations of priority queues, in which all possible sequences of *insert* and *delete_min* operations are equally likely, under the constraint that the initial and final priority queues are empty. In Section 6.2 we turn our attention to continuous models, in particular the M/M/∞ queue and a non-Markovian generalization called hashing with lazy deletion. The statistic of interest is the size of the queue, and especially the *maximum* size attained over time, which is related to questions of preallocating storage for sweepline structures. We conclude in Section 6.3 with the probabilistic analysis of union-find algorithms.

6.1. Dynamic Analyses under Discrete Models

Dynamic data structures such as lists, search trees, and heap-ordered trees can be analysed in a dynamic context under the effect of *sequences of operations*. Françon [1978] first proposed a model called the "history model" which amounts to analyzing dynamic structures under all possible evolutions up to order-isomorphism. Using combinatorial interpretations of continued fractions and orthogonal polynomials [Flajolet 1981], a number of data structures can be analysed under this model [Flajolet, Françon, and Vuillemin 1980]. In this section, we shall cover some of the theory, with special emphasis on priority queue algorithms.

A priority queue implementation (see Section 4.2) supports the operations *insert* and *delete_min*. Let $I(x)$ denote the *insert* operation putting element x into the queue, and let D denote *delete_min*; an example of a particular sequence of operations is

$$s = I(3.1)\, I(1.7)\, I(2.9)\, I(3.7)\, D\, D\, I(3.4). \tag{1}$$

Such a sequence s consists of a *schema* $IIIIDDI$, from which we see that s causes the queue size to increase by 3. If we restrict attention to structures that operate by comparison between keys,

the effect of s on an initially empty queue is fully characterized by the following information: The second operation $I(1.7)$ inserts an element smaller than the first, the third operation $I(2.9)$ inserts an element that falls in between the two previous ones, and so on. We define the *history* associated to a sequence s to consist of the schema of s in which each operation is labeled by the *rank* of the element on which it operates (relative to the current state of the structure). If we make the convention that ranks are numbered starting at 0, then all *delete_min* operations must be labeled by 0, and each *insert* is labeled between 0 and k, where k is the size of the priority queue at the time of the *insert*. The history associated with (1) is

$$I_0\, I_0\, I_1\, I_3\, D_0\, D_0\, I_1. \tag{2}$$

We let $\mathcal{H}$ denote the set of all histories containing as many *inserts* as *delete_mins*, and we let $\mathcal{H}_n$ be the subset of those that have length n. We define $H_n = \mathrm{card}(\mathcal{H}_n)$ and $H(z) = \sum_{n\geq 0} H_n z^n$.

THEOREM 1. *The OGF of priority queue histories has the continued fraction expansion*

$$H(z) = \cfrac{1}{1 - \cfrac{1\cdot z^2}{1 - \cfrac{2\cdot z^2}{1 - \cfrac{3\cdot z^2}{1 - \cfrac{4\cdot z^2}{\ddots}}}}} \tag{3}$$

Theorem 1 is a special case of a general theorem of [Flajolet 1981] that expresses generating functions of labeled schemas in terms of continued fractions. Another special case is the enumeration of plane trees, given in equations (12) and (13) of Section 4.1. Returning to priority queue histories, from a classical theorem of Gauß applied to continued fraction (3), we find that

$$H_{2n} = 1\cdot 3\cdot 5\cdot \ldots \cdot (2n-1),$$

with $H_{2n+1} = 0$. Thus the set of histories has an explicit and simple counting expression. From the same theory, it follows that the OGF $H^{[h]}(z)$ for histories with height bounded by an integer h is the hth convergent of (3)

$$H^{[h]}(z) = \frac{P_h(z)}{Q_h(z)}, \tag{4}$$

where P_h and Q_h are closely related to Hermite polynomials. From (3) and (4), it is possible to determine generating functions for extended sets of histories (such as the set of histories $\mathcal{H}^{(k)}$ that starts at size 0 and ends at size k), and then to find the number of times a given operation is performed on a priority queue structure of size k in the course of all histories of $\mathcal{H}_n$. Expressions again involve the continued fraction $H(z)$ in (3) and its convergents given in (4). From there, for a given priority queue structure, we can compute the *integrated cost* $\overline{K_n}$ defined as the expected cost of a random history in $\mathcal{H}_n$: If $\overline{CI_k}$ and $\overline{CD_k}$ are the individual expected costs for *insert* and *delete_min* performed on the priority queue structure when it has size k, we have

$$\overline{K_n} = \frac{1}{H_n}\sum_k \left(\overline{CI_k}\cdot NI_{n,k} + \overline{CD_k}\cdot ND_{n,k}\right), \tag{9}$$

where $NI_{n,k}$ (respectively, $ND_{n,k}$) is the number of times operation *insert* (respectively, *delete_min*) occurs at level k inside all histories in $\mathcal{H}_n$. Manipulations with EGFs make it possible to express the final result in simple form. For instance, we have the following two EGFs of histories $\mathcal{H}$ with a simple expression:

$$\sum_{n\geq 0} H_n \frac{z^n}{n!} = e^{z^2/2} \qquad \text{and} \qquad \sum_{n\geq 0} H_{2n}\frac{z^n}{n!} = \frac{1}{\sqrt{1-2z}}.$$

The main theorem is that the following two GFs

$$C(x) = \sum_{k\geq 0}(\overline{CI_k} + \overline{CD_{k+1}})x^k \qquad \text{and} \qquad K(z) = \sum_{n\geq 0}\overline{K_{2n}}H_{2n}\frac{z^n}{n!}, \tag{6}$$

where $C(x)$ is an OGF of individual costs and $K(z)$ is a modified EGF of integrated costs (after normalization by H_n), are closely related:

THEOREM 2. *The GFs $C(x)$ and $K(z)$ defined above satisfy*

$$K(z) = \frac{1}{\sqrt{1-2z}} C\left(\frac{z}{1-z}\right). \tag{7}$$

If we plug into (7) the OGF $C(x)$ corresponding to a particular implementation of priority queues, and then extract coefficients, we get the integrated cost for that implementation. For instance, for histories of length $2n$ for sorted lists (SL) and binary search trees (BST), we have

$$\overline{K_{2n}^{\mathrm{SL}}} = \frac{n(n+5)}{6} \qquad \text{and} \qquad \overline{K_{2n}^{\mathrm{BST}}} = n\log n + O(n). \tag{8}$$

A variety of dynamic data structures (dictionaries, symbol tables, etc.) can be analysed with these techniques under the history model. To each data type is associated a continued fraction of the form (3), a class of orthogonal polynomials (Laguerre, Poisson-Charlier, etc.) related to (4), and finally a transformation analogous to (6) that describes the transition from a GF of individual costs to the corresponding GF of integrated costs and which is usually expressed as an integral transform.

6.2. Continuous-Time Queueing Processes

Sweepline algorithms are designed to process a sequence of items over time At time t, the data structure stores the items that are "living" at time t. Let us think of the ith item as being an interval $[s_i, t_i]$ in the unit interval, containing a unique key k_i of supplementary information; the ith item is "born" at time s_i, "dies" at time t_i, and is "living" when $t \in [s_i, t_i]$. The data structure must be able to support searching the living items based on key value.

It is natural to think of the data structure as a queue, as far as size is concerned. Let us denote the queue size at time t by $Need(t)$, the number of items that need to be included in the data structure. If we think of the items as horizontal intervals, then $Need(t)$ is just the number of intervals "cut" by the vertical line at position t. In a typical application, only square root of the total number of items tend to be present in the queue at any given time. It is thus very inefficient to devote a separate storage location to every item; the data structure should be dynamic.

In the *hashing with lazy deletion* (HwLD) data structure, items are stored in a hash table of H buckets, based upon the hash value of the key. The distinguishing feature of HwLD is that an item is not deleted as soon as it dies; the "lazy deletion" strategy deletes a dead item only when a later insertion accesses the same bucket. The number H of buckets is chosen so that the expected number of items per bucket is small. HwLD is thus more time-efficient than doing "vigilant-deletion," at a cost of storing some dead items.

Expected Queue Sizes. We define $Use(t)$ to be the number of items in the HwLD data structure at time t; that is, $Use(t) = Need(t) + Waste(t)$, where $Waste(t)$ is the number of dead items present at time t. Let us consider the M/M/∞ queueing model, in which the births form a Poisson process, and the lifespans of the intervals are independently and exponentially distributed.

THEOREM 3 [Feller 1968], [Van Wyk and Vitter 1986]. *In the stationary $M/M/\infty$ model, both Need and Use $-$ H are identically Poisson distributed with mean λ/μ, where λ is the birth rate of the intervals and $1/\mu$ is the average lifetime per item.*

PROOF. We define $p_{m,n}(t) = \Pr\{Need(t) = m,\ Waste(t) = n\}$ for $m, n \geq 0$, and 0 otherwise.

$$\begin{aligned} p_{m,n}(t+\Delta t) = {} & \big((1-\lambda\,\Delta t)(e^{-\mu\,\Delta t})^m + o(\Delta t)\big)\, p_{m,n}(t) \\ & + \big((1-\lambda\,\Delta t)(m+1)(1-e^{-\mu\,\Delta t})(e^{-\mu\,\Delta t})^m + o(\Delta t)\big)\, p_{m+1,n-1}(t) \\ & + \delta_{n=0}\,(\lambda\,\Delta t + o(\Delta t)) \sum_{j\geq 0} p_{m-1,j}(t) + o(\Delta t). \end{aligned} \tag{9a}$$

By expanding the exponential terms in (9a) and rearranging, and letting $\Delta t \to 0$, we get

$$p'_{m,n}(t) = (-\lambda - m\mu)\, p_{m,n}(t) + (m+1)\mu p_{m+1,n-1}(t) + \delta_{n=0}\lambda \sum_{j\geq 0} p_{m-1,j}(t). \tag{9b}$$

In the stationary model, the probabilities $p_{m,n}(t)$ are independent of t, and thus the left-hand side of (9b) is 0. For notational simplicity we shall drop the dependence upon t. The rest of the derivation proceeds by considering the multidimensional OGF $P(z,w) = \sum_{m,n} p_{m,n} z^m w^n$. Equation (9*b*) becomes

$$\mu(z-w)\frac{\partial P(z,w)}{\partial z} = -\lambda P(z,w) + \lambda z P(z,1). \tag{9c}$$

This provides us with the distribution of *Need*:

$$\Pr\{Need = m\} = [z^m]P(z,1) = [z^m]e^{(z-1)\lambda/\mu} = \frac{(\lambda/\mu)^m}{m!}e^{\lambda/\mu}. \tag{10}$$

To find the distribution of *Use*, we replace w by z in (9*c*), which causes the left-hand-side of (9*c*) to become 0. We get

$$\Pr\{Use = k\} = [z^k]P(z,z) = [z^k]zP(z,1). \tag{11}$$

The rest follows from (10). ∎

Maximum Queue Size. A more interesting statistic, which has direct application to matters of storage preallocation, is the maximum values of $Need(t)$ and $Use(t)$ attained over time t.

Orthogonal polynomials arise in an interesting way when considering the more general model of a birth-and-death process, which is a Markov process in which transitions from level k are allowed only to levels $k+1$ and $k-1$. The infinitesimal birth and death rates at level k are denoted by λ_k and μ_k:

$$\Pr\{Need(t+\Delta t) = j \mid Need(t) = k\} = \begin{cases} \lambda_k \Delta t + o(\Delta t), & \text{if } j = k+1; \\ \mu_k \Delta t + o(\Delta t), & \text{if } j = k-1; \\ o(\Delta t), & \text{otherwise.} \end{cases}$$

For the special case of the M/M/∞ model, we have $\lambda_0 = \lambda_1 = \cdots = \lambda$ and $\mu_k = k\mu$; for the M/M/1 model, we have $\lambda_0 = \lambda_1 = \cdots = \lambda$ and $\mu_0 = \mu_1 = \cdots = \mu$.

THEOREM 4 [Mathieu and Vitter 1987]. *The distribution of* $\max_{0\le t\le 1}\{Need(t)\}$ *can be expressed simply in terms of Fibonacci polynomials (for the M/M/1 process) and Poisson-Charlier polynomials (for the M/M/∞ process). For several types of linear birth-and-death processes, of the form* $\lambda_k = \alpha k + \beta$, $\mu_k = \gamma k + \delta$, $Q_j(x)$ *can be expressed in terms of either Laguerre polynomials or Meixner polynomials of the second kind.*

The approach of Theorem 4 is interesting, given how orthogonal polynomials arose in a similar way in Section 6.1, and it can be used to calculate the distributions numerically, but it does not yield asymptotics directly.

Relation to Maximum Bucket Occupancy. We can instead get bounds on the maximum queue size by considering the discrete version of the problem, namely, the maximum bucket occupancy in hashing, which we covered in Section 5.2. The bounds we get are tight, up to a constant factor, when we make very weak restrictions on μ and H, which are always met in practice.

THEOREM 5 [Mathieu and Vitter 1987]. *In the stationary M/M/∞ model, we have*

$$\mathbf{E}\{\max_{0\le t\le 1}\{Need(t)\}\} = O(\mathbf{E}\{Need\}) = O(\lambda/\mu), \tag{12}$$

under the condition that $\mu = O(\lambda/\log\lambda)$, *and*

$$\mathbf{E}\{\max_{0\le t\le 1}\{Use(t)\} - \max_{0\le t\le 1}\{Need(t)\}\} = O(\mathbf{E}\{Use - Need\}) = O(H), \tag{13}$$

where the number of buckets in HwLD is assumed to be $H = \Omega(\log\lambda)$.

PROOF. Our approach for each result is to approximate the queueing process by a sequence of stages (successive refinements) of a discrete analog, which we call *time hashing*. The statistic of interest in the time hashing is the maximum bucket occupancy. We shall always use the terminology "bucket" when discussing the hashing inherent in HwLD and the term "slot" when discussing time hashing.

To prove (12), we can assume that $H = 1$, since the bucketing does not affect $Need(t)$ in any way. Stages $k = 0, 1, 2, \ldots$ of time hashing are defined as follows: All items (intervals) with lifespan in

the range $(\frac{1}{\mu}2^{k-1}, \frac{1}{\mu}2^k]$ are put into stage k. In addition, we put all items with lifespan $< \frac{1}{\mu}2^{k-1}$ into stage 0. Each stage consists of a hash table of $\lceil \mu 2^{-k} + 1 \rceil$ slots, as pictured in Figure 3.1. The jth slot, for $0 \le j \le 2^k$, represents the interval of time $(\frac{1}{\mu}(j-1)2^k, \frac{1}{\mu}j2^k]$. An item in stage k is placed into the slot corresponding to its birthtime. We limit the above process to $T^* \approx \lg\ln\mu$ stages in the M/M/∞ case, after which we group together all the leftover items into a single slot.

We define $N_k(j)$ to be the number of stage k items in slot j, for $j \le T^*$. We can bound

$$\max_{0\le t\le 1}\{Need(t)\} \le 2\sum_k \max_j\{N_k(j)\}.$$

An application of our analysis for the maximum bucket occupancy in hashing gives us

$$\mathbf{E}\{\max_j\{N_k(j)\}\} = O\left(\frac{\lambda}{\mu 2^k}\right),$$

The expected number of items remaining after T^* stages is bounded by λ/μ. The bound (12) follows immediately.

To prove (13), we use the fact that

$$\mathbf{E}\{\max_{0\le t\le 1}\{Use(t)\} - \max_{0\le t\le 1}\{Need(t)\}\} \le \mathbf{E}\{\max_{0\le t\le 1}\{Waste(t)\}\},$$

and bound the right-hand side. We define the *waste* of an item to be the amount of time after its death that it remains in the HwLD data structure. In this instance of time hashing, stage k, for $k = 0, 1, \ldots$, consists of those items whose waste is in the range $(\frac{H}{\lambda}2^{k+1}, \frac{H}{\lambda}2^{k+2}]$; we put items of waste $\le 2\frac{H}{\lambda}$ into stage 0. Each of the H buckets in the HwLD implementation has its own stages. Every stage is further decomposed into five substages, each consisting of a hash table of $\lceil \lambda/(5H2^k) \rceil$ slots. The five substages for stage k are identical; each one is offset from the previous substage by time $\frac{H}{\lambda}2^k$. WLOG, we shall treat only the first substage and shall refer to it simply as stage k.

The jth slot represents the time interval $(5j2^k\frac{H}{\lambda}, 5(j+1)2^k\frac{H}{\lambda}]$. Each slot is subdivided into fifths. The first fifth of each slot is called the *death zone*, the next fifth is called the *twilight zone*, and the last three fifths comprise the *birth zone*. An item in bucket h, $1 \le h \le H$, is placed into slot j for that bucket if its death time occurs within the death zone of that slot. For each hash bucket h, $1 \le h \le H$, each stage k, and each slot j, we define

$$w_{h,k}(j) = \begin{cases} \#\text{ deaths in death zone}, & \text{if } \exists \text{ birth in birth zone and} \\ & \nexists \text{ birth in twilight zone}; \\ 0, & \text{otherwise}; \end{cases}$$

$$W_k(j) = \sum_{1\le h\le H} w_{h,k}(j).$$

We have

$$\mathbf{E}\{\max_{0\le t\le 1}\{Waste(t)\}\} \le 5\sum_k \mathbf{E}\{\max_j\{W_k(j)\}\}.$$

We want to bound the sum by $O(E(Waste)) = O(H)$. We cut off the number of stages at $T^* \approx \lg\ln(\lambda/H)$, and put the remaining items into one large slot. A key observation for the derivation is that the death rate in the M/M/∞ model is a Poisson process with the same intensity as the birth rate, and thus also reversible [Kelly, 1979]. We can prove lemmas similar to those in the last section to get our desired bound. A big difference between this application of time hashing and the one in the last section is that the random variables $w_{h,k}(j)$ (and hence also $W_k(j)$) are almost always 0 as k grows: $\Pr\{w_{h,k}(j) = 0\} \approx 1 - e^{-2^k}$. This causes the maximum slot occupancy to behave wildly. In fact, to get (13), it is not enough to bound $\mathbf{E}\{\max_j\{w_{h,k}(j)\}\}$ and then multiply by H, because the result will be too large: the load factor in the analysis of $\max_j\{w_{h,k}(j)\}$ is too small, and the ratio between the average maximum slot occupancy and the average slot occupancy is no longer $O(1)$. The solution is to consider the H buckets *in toto* and to bound $\mathbf{E}\{\max_j\{W_k(j)\}\}$ directly using saddle point techniques. ∎

6.3. Set Union-Find Algorithms

The *set union-find* data type is useful in several computer applications, such as computing minimum spanning trees, testing equivalence of finite state machines, and handling COMMON blocks in FORTRAN compilers. The operation *union*(x, y) merges the equivalence classes (or simply *components*) containing x and y and chooses a unique "representative" element for the combined component. Operation *find*(x) returns the representative of x's component, and *make_set*(x) creates a singleton component $\{x\}$ with representative x.

Union-find algorithms have been studied extensively in terms of worst-case and amortized performance. Tarjan and van Leeuwen [1984] give matching upper and lower amortized bounds of $\Theta(n + m\alpha(m + n, n))$ for the problem, where n is the number of *make_set* operations, m is the number of *finds*, and $\alpha(a, b)$ denotes a functional inverse of Ackermann's function. The lower bounds holds in a separable pointer machine model, and the upper bound is achieved by the well-known tree data structure that does weighted merges and path compression.

In this section we study the average-case running time of more simple-minded algorithms, called "quick find" (QF) and "quick find weighted" (QFW). The data structure consists of an array called *rep*, with one slot per element; *rep*$[x]$ is the representative for element x. Each *find* can thus be done in constant time. In addition, the elements in each component are linked together in a list. In the QF algorithm, *union*(x, y) is implemented by setting *rep*$[z]$:= *rep*$[x]$, for all $z \in y$'s component. The QFW algorithm is the same, except that when x's component is smaller than y's, we take the quicker route and set *rep*$[z]$:= *rep*$[y]$, for all $z \in x$'s component. An auxiliary array is used to keep track of the size of each component.

Since all *finds* take constant time, we shall confine our attention to the *union* operations. We consider $n-1$ *unions* performed on a set of n elements, so that at the end a single component of size n remains. Our performance measure, which we denote by T_n^{QF} and T_n^{rmQFW}, is the total number of updates to slots of *rep* made during the *unions*. We consider three models of "random" input.

Random Graph Model. Probably the most realistic model was proposed in [Yao 1976], based on the random graph model of [Erdös and Rényi 1959]. Each of the $\binom{n}{2}$ undirected edges between n vertices "fires" independently, governed by a Poisson process. Each order of firings is thus equally likely. When an edge $\{x, y\}$ fires, we execute *union*(x, y) if x and y are in different components.

THEOREM 6 [Knuth and Schönhage 1978], [Bollobas and Simon 1985]. *The average number of updates done by QF and QFW in the random graph model is*

$$\overline{T_n^{\mathrm{QF}}} = \frac{n^2}{8} + o(n(\log n)^2);$$

$$\overline{T_n^{rmQFW}} = cn + o(n/\log n), \qquad \text{where } c = 2.0847\ldots.$$

We shall restrict ourselves to showing that $\overline{T_n^{\mathrm{QF}}} \sim n^2/8$ and $\overline{T_n^{rmQFW}} = O(n)$ using the derivation from [Knuth and Schönhage 1978]. The techniques in [Bollobás and Simon 1985] are needed to determine the coefficient c and to get better bounds on the second-order terms. In addition, [Bollobás and Simon 1985] consider sequences of less than n *unions*. They show that, on the average, QF performs $(\frac{1}{2} - \epsilon)n$ *unions* in $O(n \log n)$ time, and QFW does k *unions* in $O(k)$ time, for any $k \le n$.

SKETCH OF PROOF. The proof is based on the intuition from [Erdös and Renyi, 1959] that with probability $1 - O(1/\log n)$ a random graph on n vertices with $\frac{1}{2}n \log n + \frac{1}{2}cn$ edges, where c is a constant, consists of one giant connected component of size $\ge n - \log \log n$ and a set of isolated vertices. The graph is connected with probability $e^{-e^{-c}}$. In terms of *union* operations, it is very likely that the last few *unions* joined the giant component to singleton components; the cost for each such *union* would be $O(n)$ for QF and $O(1)$ for QFW. The proof of Theorem 6 consists in showing this behavior extends over the entire sequence of *unions*.

For QFW, we find by recurrences and asymptotic approximations that

$$E_{n,k,m} = O\left(\frac{n}{k^{3/2}m^{3/2}(k+m)^{3/2}}\right), \qquad \text{for } k, m < n^{2/3} \text{ and } k, m > n^{2/3}, \tag{14}$$

where $E_{n,k,m}$ is the expected number of times a component of size k is merged with one of size m. Hence,

$$\overline{T_n^{rmQFW}} = \sum_{1\le k,m<n} \min\{k,m\}E_{n,k,m} \le \sum_{1\le k\le m<n} k(E_{n,k,m} + E_{n,m,k}). \tag{15}$$

For the portion of the sum to which (14) applies, we can bound (15) by $O(n)$. For the rest of the range, in which $1 \le k \le n^{2/3} \le m < n$, the sum is bounded by n, since each element can be merged at most once from a component of size $< n^{2/3}$ into one of size $\ge n^{2/3}$. The analysis of QF is similar. ∎

Several combinatorial algorithms, have been designed and analyzed using the random graph model. For example, [Babai, Erdös, and Selkow 1980] give an algorithm that runs in $O(n^2)$ average time for testing graph isomorphism, which is an *NP*-complete problem. We refer the reader to the chapter on probabilistic analysis.

Random Spanning Tree Model. Each sequence of *union* operations corresponds to a "union tree," in which the directed edge $\langle x, y\rangle$ means that the component with representative y is merged into the component with representative x. In the random spanning tree model, all possible union trees are equally likely; there are n^{n-2} possible oriented trees and $(n-1)!$ firing orders of the edges in each tree.

THEOREM 7 [Yao 1976], [Knuth and Schönhage 1978]. *The average number of updates done by QF and QFW in the random spanning tree model is*

$$\overline{T_n^{QF}} = \sqrt{\frac{\pi}{8}}n^{3/2} + O(n\log n);$$
$$\overline{T_n^{rmQFW}} = \frac{1}{\pi}n\log n + O(n).$$

SKETCH OF PROOF. An admissibility argument similar to those in Section 4 allows us to compute the probability $p_{n,k}$ that the last *union* merges components of size k and $n-k$:

$$p_{n,k} = \frac{1}{2(n-1)}\binom{n}{k}\left(\frac{k}{n}\right)^{k-1}\left(\frac{n-k}{n}\right)^{n-k-1}. \tag{16}$$

And it is easy to show that

$$\overline{T_n} = c_n + 2\sum_{0<k<n} p_{n,k}\overline{T_k}, \tag{17}$$

where $c_n = \sum_{1<k<n} kp_{n,k}$ for QF and $c_n = \sum_{1<k<n}\min\{k, n-k\}p_{n,k}$ for QFW. By symmetry we have $\sum_{1<k<n} kp_{n,k} = n/2$, and arguments similar to those used for Theorem 6 show that $\sum_{1<k<n}\min\{k,n-k\}p_{n,k} = (2n/\pi)^{1/2} + O(1)$. Recurrence (17) is in a special linear form that allows us to solve it "by repertoire": the solution for $c_n = a_n + b_n$ is the sum of the solutions for $c_n = a_n$ and for $c_n = b_n$. Hence, if we can find a "basis" of different c_n for which (17) can be solved easily, then we can solve (17) for QF and QFW by linear combinations of the basis functions. It turns out that the basis in our case is the set of Q-functions $Q_{\{1/k^r\}}(n)$, for $r = -1, 0, 1, \ldots$, which we studied in connection with linear probing in Section 5.2. ∎

Random Components Model. In the simplest model, and also the least realistic, we assume that at any given time each pair of existing components is equally likely to be merged next. The union tree is this framework is nothing more than a random binary search tree, which we studied extensively in Section 4.2. Admissibility arguments lead directly to the following result:

THEOREM 8 [Doyle and Rivest 1976]. *The average number of updates done by QF and QFW in the random components model is*

$$\overline{T_n^{QF}} = n(H_n - 1) = n\log n + O(n);$$
$$\overline{T_n^{rmQFW}} = nH_n - \frac{1}{2}nH_{\lfloor n/2\rfloor} - \lceil n/2\rceil = \frac{1}{2}n\log n + O(n).$$

References

O. AMBLE AND D. E. KNUTH [1974]. "Ordered Hash Tables," *The Computer Journal* **17**(2), May 1974, 135–142.

L. BABAI, P. ERDÖS, AND S. M. SELKOW [1980]. "Random Graph Isomorphisms," *SIAM Journal on Computing* **9**, 628–635.

M. AJTAI, J. KOMLÒS, AND E. SZEMERÉDI [1983]. "An $O(n \log n)$ Sorting Network," *Proceedings of the 15th Annual Symposium on Theory of Computer Science*, Boston, May 1983, 1–9.

E. BENDER [1973]. "Central and Local Limit Theorems Appplied to Asymptotic Enumeration," *Journal of Combinatorial Theory, Series A* **15**, 1973, 91-111.

E. BENDER [1974]. "Asymptotic Methods in Enumerations," *SIAM Review* **16**, 1974, 485–515.

C. BENDER AND S. ORSZAG [1978]. *Advanced Mathematical Methods for Scientists and Engineers.* McGraw-Hill, 1978.

P. BILLINGSLEY [1986]. *Probability and Measure*, Second Edition, J. Wiley, New York, 1986.

B. BOLOBÁS [1985]. *Random Graphs*, Academic Press, New York 1985.

B. BOLLOBÁS AND I. SIMON [1985]. "On the Expected Behavior of Disjoint Set Union Algorithms," *Proceedings of the 18th Annual Symposium on Theory of Computing*, Providence, RI, May 1985, 224–231.

M. R. BROWN [1979]. "A Partial Analysis of Random Height-Balanced Trees," *SIAM Journal on Computing* **8**(1), February 1979, 33–41.

E. R. CANFIELD [1977]. "Central and Local Limit Theorems for Coefficients of Binomial Type," *Journal of Combinatorial Theory, Series A* **23**, 1977, 275-290.

D. W. CLARK [1979]. "Measurements of Dynamic List Structure Use in Lisp," *IEEE Transactions on Software Engineering* **SE–5**(1), January 1979, 51–59.

L. COMTET [1974]. *Advanced Combinatorics.* Reidel, Dordrecht, 1974.

L. COMTET [1969]. "Calcul pratique des coefficients de Taylor d'une fonction algébrique," *L'Enseignement Mathématique* **10**, 1969, 267-270.

B. DAVIES [1978]. *Integral Transforms and Their Applications*, Springer, New York 1978.

N. G. DE BRUIJN [1981]. *Asymptotic Methods in Analysis.* Dover, New York, 1981.

N. G. DE BRUIJN, D. E. KNUTH, AND S. O. RICE [1972]. "The Average Height of Planted Plane Trees," in *Graph Theory and Computing*, edited by R.-C. Read, Academic Press, New York, 1972, 15–22.

L. DEVROYE [1986a]. "A Note on the Height of Binary Search Trees," *Journal of the ACM* **33**, 1986, 489–498.

L. DEVROYE [1986b]. *Lecture Notes on Bucket Algorithms*, Birkhäuser, Boston, 1986.

G. DOETSCH [1955]. *Handbuch der Laplace Transformation.* Volumes 1–3. Birkhäuser Verlag, Basel, 1955.

J. DOYLE AND R. L. RIVEST [1976]. "Linear Expected Time of a Simple Union-Find Algorithm," *Information Processing Letters* **5**, 1976, 146–148.

B. ĎURIAN [1986]. "Quicksort without a Stack," *Proceedings of the 12th Annual Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science* **233**, 1986, 283–289.

R. FAGIN, J. NIEVERGELT, N. PIPPENGER, AND H. R. STRONG [1979]. "Extendible Hashing—A Fast Access Method for Dynamic Files," *ACM Transactions on Database Systems* **4**(3), September 1979, 315–344.

W. FELLER [1968]. *An Introduction to Probability Theory and Its Applications*, Volume 1, Wiley, New York, third edition 1968.

PH. FLAJOLET [1981]. *Analyse d'algorithmes de manipulation d'arbres et de fichiers*, in *Cahiers du BURO* **34–35**, 1981, 1–209.

PH. FLAJOLET [1983]. "On the Performance Evaluation of Extendible Hashing and Trie Searching," *Acta Informatica* **20**, 1983, 345–369.

PH. FLAJOLET [1985]. "Mathematical Methods in the Analysis of Algorithms and Data Structures," INRIA, Research Report 400, 1985. To appear in *A Graduate Course in Computer Science*, Computer Science Press, 1987.

PH. FLAJOLET [1987]. "Analytic Models and Ambiguity of Context-Free Languages," *Theoretical Computer Science* **49**, 1987.

PH. FLAJOLET, G. GONNET, C. PUECH, AND M. ROBSON [1987]. "Analytic Variations on Quad Trees," in preparation.

PH. FLAJOLET AND A. M. ODLYZKO [1982]. "The Average Height of Binary Trees and Other Simple Trees," *J. Computer and System Sciences* **25**, 1982, 171-213.

PH. FLAJOLET AND A. M. ODLYZKO [1987]. "Singularity Analysis of Generating Functions," preprint.

PH. FLAJOLET AND C. PUECH [1986] "Partial Match Retrieval of Multidimensional Data," *Journal of the ACM* **33**(2), April 1986, 371–407. 1986.

PH. FLAJOLET, J.-C. RAOULT, AND J. VUILLEMIN [1977]. "The Number of Registers Required to Evaluate Arithmetic Expressions," *Theoretical Computer Science* **9**, 1979, 99–125.

PH. FLAJOLET, M. RÉGNIER AND R. SEDGEWICK [1985]. "Some Uses of the Mellin Integral Transform in the Analysis of Algorithms," in *Combinatorics on Words*, Springer NATO ASI Series F, Volume 12, Berlin, 1985.

PH. FLAJOLET AND R. SEDGEWICK [1986]. "Digital Search Trees Revisited," *SIAM Journal on Computing* **15**(3), August 1986, 748–767.

PH. FLAJOLET, P. SIPALA, AND J.-M. STEYAERT [1987]. "The Analysis of Tree Compaction in Symbolic Manipulations," preprint.

PH. FLAJOLET AND J.-M. STEYAERT [1987]. "A Complexity Calculus for Recursive Tree Algorithms," *Mathematical Systems Theory* **19**, 1987, 301–331.

J. FRANÇON [1978]. "Histoire de fichiers," *RAIRO Inform. Theor.* **12**, 1978, 49–67.

J. FRANÇON, G. VIENNOT, AND J. VUILLEMIN [1978]. "Description and Analysis of an Efficient Priority Queue Representation," *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, Ann Arbor, October 1978, 1–7.

G. H. GONNET [1984] *Handbook of Algorithms and Data Structures.* Addison-Wesley, Reading, 1984.

G. H. GONNET [1981]. "Expected Length of the Longest Probe Sequence in Hash Code Searching," *Journal of the ACM* **28**(2), April 1981, 289–304.

G. H. GONNET, L. D. ROGERS, AND A. GEORGE [1980]. "An Algorithmic and Complexity Analysis of Interpolation Search," *Acta Informatica* **13**(1), January 1980, 39–46.

I. GOULDEN AND D. JACKSON [1983]. *Combinatorial Enumerations.* Wiley, New York, 1983.

D. H. GREENE [1983]. "Labelled Formal Languages and Their Uses," Stanford University, Technical Report STAN-CS-83-982, 1983.

D. H. GREENE AND D. E. KNUTH [1982]. *Mathematics for the Analysis of Algorithms.* Birkhäuser, Boston, second edition 1982.

L. J. GUIBAS AND E. SZEMERÉDI [1978]. "The Analysis of Double Hashing," *Journal of Computer and System Sciences* **16**(2), April 1978, 226–274.

B. HARRIS AND L. SCHOENFELD [1968]. "Asymptotic Expansions for the Coefficients of Analytic Functions," *Illinois J. Math.* **12**, 1968, 264-277.

W. K. HAYMAN [1956]. "A Generalization of Stirling's Formula," *J. Reine und Angewandte Mathematik* **196**, 1956, 67-95.

P. HENRICI [1977]. *Applied and Computational Complex Analysis.* Volumes 1–3. Wiley, New York, 1977.

J. M. INCERPI AND R. SEDGEWICK [1985]. "Improved Upper Bounds on Shellsort," *Journal of Computer and System Sciences* **31**, 1985, 210–224.

PH. JACQUET AND M. RÉGNIER [1986]. "Trie Partitioning Process: Limiting Distributions," in *Proceedings of CAAP '86, Lecture Notes in Computer Science* **214**, 1986, 196-210.

PH. JACQUET AND M. RÉGNIER [1987]. "Normal Limiting Distribution of the Size of Tries," in *Performance '87, Proceedings of 13th International Symposium on Computer Performance*, Brussels, December 1987.

R. KEMP [1979]. "The Average Number of Registers Needed to Evaluate a Binary Tree Optimally," *Acta Informatica* **11**(4), 1979, 363–372.

R. KEMP [1984]. *Fundamentals of the Average Case Analysis of Particular Algorithms.* Teubner-Wiley, Stuttgart, 1984.

D. E. KNUTH [1973a]. *The Art of Computer Programming.* Volume 1: *Fundamental Algorithms.* Addison-Wesley, Reading, MA, second edition 1973.

D. E. KNUTH [1973b]. *The Art of Computer Programming.* Volume 3: *Sorting and Searching.* Addison-Wesley, Reading, MA, 1973.

D. E. KNUTH [1981]. *The Art of Computer Programming.* Volume 2: *Semi-Numerical Algorithms.* Addison-Wesley, Reading, MA, second edition 1981.

D. E. KNUTH AND A. SCHÖNHAGE [1978]. "The Expected Linearity of a Simple Equivalence Algorithm," *Theoretical Computer Science* **6**, 1978, 281–315.

V. F. KOLCHIN, B. A. SEVAST'YANOV, AND V. P. CHISTYAKOV [1978]. *Random Allocations.* V. H. Winston & Sons, Washington, 1978.

KONHEIM AND NEWMAN [1973]. "A Note on Growing Binary Trees," *Discrete Mathematics* **4**, 1973, 57–63.

E. E. LINDSTROM AND J. S. VITTER [1985]. "The Design and Analysis of BucketSort for Bubble Memory Secondary Storage," *IEEE Transactions on Computers* **C–34**(3), March 1985.

C. M. MATHIEU AND J. S. VITTER [1987]. "Maximum Queue Size and Hashing with Lazy Deletion," Brown University, Technical Report, 1987.

A. MEIR AND J. W. MOON [1978]. "On the Altitude of Nodes in Random Trees," *Canadian Journal of Mathematics* **30**, 1978, 997–1015.

A. M. ODLYZKO [1982]. "Periodic Oscillations of Coefficients of Power Series that Satisfy Functional Equations," *Advances in Math.* **44**, 1982, 180-205.

A. M. ODLYZKO AND L. B. RICHMOND [1985]. "Asymptotic Expansions for the Coefficients of Analytic Generating Functions," *Aequationes Mathematicae* **28**, 1985, 50-63.

B. PITTEL [1986]. "Paths in a Random Digital Tree: Limiting Distributions," *Advances in Applied Probability* **18**, 1986, 139–155.

B. PITTEL [1987a]. "Linear Probing: The Probable Largest Search Time Grows Logarithmically with the Number of Records," *Journal of Algorithms* **8**(2), June 1987, 236–249.

B. PITTEL [1987b]. "On Probabilistic Analysis of a Coalesced Hashing Algorithm," *Annals of Probability* **15**(2), July 1987.

G. POLYA [1937]. "Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und chemische Verbindungen", *Acta Mathematica* **68**, 1937, 145-254. Translated in: G. Polya and R. C. Read, *Combinatorial Enumeration of Groups, Graphs and Chemical Compounds*, Springer, New York, 1987.

P. W. PURDOM AND C. A. BROWN [1985]. *The Analysis of Algorithms.* Holt, Rinehart and Winston, New-York, 1985.

M. RÉGNIER [1981]. "On the Average Height of Trees in Digital Search and Dynamic Hashing," *Information Processing Letters* **13**, 1981, 64–66.

M. RÉGNIER [1985]. "Analysis of Grid File Algorithms," *BIT* **25**, 1985, 335–357.

V. N. SACHKOV [1978]. *Verojatnosnie Metody v Kombinatornom Analize*, Nauka, Moscow, 1978.

R. SEDGEWICK [1977a]. "Quicksort with Equal Keys," *SIAM Journal on Computing* **6**(2), June 1977, 240–267.

R. SEDGEWICK [1977b]. "The Analysis of Quicksort Programs," *Acta Informatica* **7**, 1977, 327–355.

R. SEDGEWICK [1978]. "Data Movement in Odd-Even Merging," *SIAM Journal on Computing* **7**(3), August 1978, 239–272.

R. SEDGEWICK [1983a]. "Mathematical Analysis of Combinatorial Algorithms," in *Probability Theory and Computer Science*, edited by G. Louchard and G. Latouche, Academic Press, London, 1983.

R. SEDGEWICK [1983b]. *Algorithms.* Addison-Wesley, Reading, 1983.

D. L. SHELL [1959]. "A High-Speed Sorting Procedure," *Communications of the ACM* **2**(7), July 1959, 30–32.

R. P. STANLEY [1978]. "Generating Functions," in *Studies in Combinatorics*, edited by G.-C. Rota, MAA Monographs, 1978.

R. P. STANLEY [1986]. *Enumerative Combinatorics*, Wadsworth and Brooks/Cole, Monterey, 1986.

J.-M. STEYAERT AND PH. FLAJOLET [1983]. "Patterns and Pattern-Matching in Trees: an Analysis," *Information and Control* **58**, 1983, 19–58.

C. J. VAN WYK AND J. S. VITTER [1986]. "The Complexity of Hashing with Lazy Deletion," *Algorithmica* **1**(1), March 1986, 17–29.

J. S. VITTER AND W-C. CHEN [1987]. *Design and Analysis of Coalesced Hashing.* Oxford University Press, New York, 1987.

J. VUILLEMIN [1980]. "A Unifying Look at Data Structures," *Communications of the ACM* **23**(4), April 1980, 229–239.

A. C.-C. YAO [1976]. "On the Average Behavior of Set Merging Algorithms," *Proceedings of the 8th Annual ACM Symposium on Theory of Computing*, 1976, 192–195.

A. C.-C. YAO [1978]. "On Random 2-3 Trees," *Acta Informatica* **9**(2), 1978, 159–170.

A. C.-C. YAO [1980]. "An Analysis of $(h, k, 1)$ Shellsort," *Journal of Algorithms* **1**(1), 1980, 14–50.

A. C.-C. YAO [1987]. Personal communication.

A. C.-C. YAO AND F. F. YAO [1976]. "The Complexity of Searching an Ordered Random Table," *Proceedings of the 17th Annual Symposium on Foundations of Computer Science*, Houston, October 1976, 173–177.

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique