Probabilistic Analysis of Some Searching and Sorting Algorithms

By

Janice Lent

B.A. May 1983, University of Virginia
M.S. June 1985, Virginia Tech

A Dissertation submitted to

The Faculty of

The Columbian School of Arts and Sciences
of the George Washington University in partial satisfaction
of the requirements for the degree of Doctor of Philosophy

January 30, 1996

Dissertation directed by

Hosam M. Mahmoud
Professor of Statistics

Dedication

To my dear parents

Peter and Mary Lent

Acknowledgments

I would like to thank my advisor, Dr. Hosam Mahmoud, for all his help and encouragement during the time I worked on this dissertation. I would also like to thank Dr. Robert Smythe for his advice and teaching. Dr. Smythe and several others, including Dr. James Fill, Dr. Reza Modarres, and Dr. Sudip Bose, all read this dissertation with care and provided comments which improved it.

Special thanks are also due to Ms. Shail Butani of the Bureau of Labor Statistics, who consistently supported me in my course work and in my work on this dissertation.

Abstract

We use binary trees to analyze two algorithms, insertion sort and multiple quickselect. In each case, we consider the number of comparisons consumed as a measure of performance. We assume that the ranks of the n data values being searched or sorted form a random permutation of the integers $\{1, ..., n\}$. For insertion sort, we consider the limiting distribution of the number of comparisons consumed in the process of sorting the n keys. We present an average-case analysis of the number of comparisons multiple quickselect (MQS) requires for simultaneously finding several order statistics in the data set.

Using the concept of a "tree-growing" search strategy, we prove that for most practical insertion sorting algorithms, the number of comparisons needed for sorting n keys has asymptotically a normal distribution. We prove and apply a sufficient condition for asymptotic normality. The condition specifies a relationship between the variance of the number of comparisons and the rate of growth in height of the sequence of binary trees that the search strategy "grows."

MQS is a variant of the popular Quicksort algorithm, modified to search for several order statistics simultaneously. We show that, when p is an integer fixed with respect to

n, the size of the data set, MQS requires an average of $(2H_p + 1)\,n - 8p \ln n + O(1)$ comparisons to find p order statistics, where H_p is the pth harmonic number. We assume that the set of ranks of the p order statistics is selected at random from the collection of all $\binom{n}{p}$ possible sets of p ranks. Our result therefore represents a "grand" average over all possible sets of order statistics. In the process of calculating the grand average, we find the distribution and moments of the number of descendants of a node ranked j in a binary search tree of size n.

The main results presented in this dissertation will also appear in articles by Lent, Mahmoud, and Bose (1996); and Lent and Mahmoud (1996).

Contents

List of Figures

List of Tables

Chapter 1
Introduction

Sorting data files is a central problem in computing, with countless applications. Since in many instances sorting consumes considerable computing time and data storage space, practitioners must choose a sorting method suitable for the application at hand and the available computing resources. Alternative sorting algorithms may often be compared according to known performance measures such as speed and space requirements. Classical computer science literature provides average-case performance measures for some algorithms (see Knuth 1973b, for example). Our intent in this dissertation is two-fold. In Chapter 3, we seek to take sorting problems beyond average-case analysis into the domain of distribution theory, which offers a much fuller understanding of a sorting algorithm's performance. We show that for most popular implementations of insertion sort, the number of comparisons required for sorting a data set has asymptotically normal behavior. In Chapter 4 we consider an algorithm designed to search a data set for a specified collection

of order statistics—a problem closely related to that of sorting a data set. We present an average-case analysis of multiple quickselect, a variant of the popular Quicksort algorithm (Hoare 1962) modified to select several order statistics simultaneously. Further motivation for each problem is provided below.

Throughout this dissertation, we take the number of comparisons an algorithm performs as a measure of its time complexity. We recognize that, in the case of insertion sort, which we investigate in Chapter 3, practitioners may also consider the number of movements of data items which must be performed in the course of the sorting. Comparisons, however, consume more time than other operations whose analysis is generally much less interesting.

Binary trees serve as important tools in both Chapters 3 and 4. Chapter 2 therefore provides an overview of their basic properties and the derivation of one property heretofore unknown: the distribution and moments of the number of descendants of a node in a random binary search tree. We use this new result in our analysis of multiple quickselect in Chapter 4. In Chapter 5 we expand our results and discuss avenues of future research.

1.1 Motivation for Analysis of Tree-growing Search Strategies

Insertion sort is a popular on-line sorting algorithm. At each stage of the sorting, the data obtained so far make up a sorted array. The algorithm reads in a new datum, searches for its proper position in the array, and inserts it. When a data file of size $n \geq 2$ is to be sorted, the ith search is for the position of the $(i+1)$st key, $i = 1, 2, \ldots$. We shall

denote the algorithm that performs the ith search by S_i and call the collection of search algorithms $S = \{S_i\}_{i=1}^{\infty}$ a *search strategy*. Doberkat (1982) and Panny (1986) have studied the moments of the number of comparisons required for insertion sort with one particular (linear) search strategy. In Chapter 3 we show that if the values in a data array have been randomly selected from a continuous distribution, the number of comparisons needed to sort the values by most practical search strategies has asymptotically normal behavior. Thus we investigate properties of search strategies in general, with insertion sort serving as a conspicuous application.

In order to insert a new key in a sorted data array, an implementation of insertion sort selects a sequence of *probes:* keys in the sorted array to which the algorithm compares the new key. If the new key is larger than a given probe, the algorithm selects a larger probe; if the probe's value exceeds that of the new key, the search continues in the segment of the data array containing keys smaller than the probe. In general, the search algorithms applied for different keys in an insertion sort may be independent of each other. One may, for example, imagine a search strategy $S = \{S_i\}_{i=1}^{\infty}$ in which S_k is binary search and S_{k+1} is linear search. Practitioners, however, usually prefer strategies in which all the algorithms applied are coded as a single procedure which is then invoked with different parameters at different stages of the sorting. Most efficient, easy-to-use strategies thus possess certain consistency properties, which we will specify and use to prove our main result. Rudimentarily, when all the search algorithms S_i in a strategy S specify probe positions through a reasonable function of the *length* of the fragment of the data array being searched, we will call S a *consistent* strategy.

In Chapter 3, we show that for all consistent strategies, the number of comparisons used has asymptotically normal behavior. We derive the asymptotic means and variances for two particular search strategies: binary search and linear search. In so doing we show that, when a binary search strategy is used, the behavior of insertion sort compares favorably to that of Quicksort, as regards the number of comparisons consumed. The number of comparisons performed by binary insertion sort has a smaller mean and a smaller variance than the number of comparisons performed by Quicksort, and it enjoys asymptotically normal behavior. Though insertion sort requires more movements[1] of data records—an inherent disadvantage of on-line sorting algorithms—it is often favored in practice for its simplicity and versatility.

1.2 Motivation for Analysis of Multiple Quickselect

In statistical inference, statistics are frequently constructed based on a few order statistics from a data set. Supposing the observed data set is $X_1, ..., X_n$, statistics based on order statistics usually take the form

$$A_n = f\left(X_{(i_1)}, X_{(i_2)}, ..., X_{(i_p)}\right),$$

where f is a p–dimensional measurable function and $X_{(k)}$ is the kth order statistic in the data set, with p typically ranging between two and five. Applications of such statistics abound in the literature of non-parametric methods. Distribution-free tolerance limits (Wilks 1962, p. 334), distribution-free confidence intervals of quantiles—like Geert-

[1] The number of movements required during an insertion sort may be reduced through the use of pointers, as in a linked list. The pointers, however, substantially increase the algorithm's space requirements and complicate the search process, so they are not often used with large data sets.

sema's (1970) U-statistic confidence interval for median estimation—and the Hodges–Lehmann class of statistics (Serfling 1980) are only a few examples in which a pair of order statistics is needed. The five-statistic summary, comprising the smallest, the first quartile, the median, the third quartile, and the largest in a data set, is commonly used in business and other applications as a profile of the data.

Several algorithms exist for the complete sorting of a data file. Fewer are known, however, for the direct computation of a single order statistic, and even fewer for selecting several order statistics simultaneously. Some randomized algorithms have been developed for finding an order statistic (see Floyd and Rivest 1975), and some optimal algorithms exist for certain cases, e.g., for finding the smallest value. These optimal algorithms, however, are designed to find specific order statistics and may not be easily modified to work in other applications. The optimal algorithm for finding the smallest value, for example, cannot be adapted in any direct way to search for a specified quantile like the median in a file of arbitrary size. And specialized algorithms are generally not suitable for use with weighted data, since the weights affect the rank of a desired quantile in the data set (see Lent and Modarres 1995).

A general algorithm for selecting any order statistic, or group of order statistics, is thus desirable in practice. Variants of Quicksort, the fastest known in-situ sorting algorithm, suit this task. A modified version of Quicksort with occasional truncation of one side, to be called *multiple Quickselect* (MQS), may be used to find several order statistics from a given data set. This algorithm is especially useful for computing L-estimates, as discussed in Chapter 4, where we study the average speed of MQS and describe a

specific application of the algorithm involving weighted data.

Chapter 2
Binary Trees

A binary tree is a hierarchical structure comprising nodes organized in levels. A node can have either (a) no children—in which case the node is called a *leaf*, (b) a left child, (c) a right child, or (d) two distinct children, one left and one right. In all the binary trees considered in this dissertation, the nodes carry labels having the following property: All left descendants of a node—that is, the node's left child and all descendants of the left child—have label values less than that of the node, and all right descendants have label values greater than that of the node. It is customary to draw a binary tree with the *root*—the only node with no ancestors—at the top and the leaves at the bottom. The *level* or *depth* of a specific node is the length of the path (the number of edges) from the node to the root node, which is on level zero. The *size* of a binary tree is the number of its nodes. If level k of a tree has 2^k nodes—the maximum number possible—it is said to be *saturated*. A tree in which all existing levels, except possibly the lowest, are saturated

is *complete*.[2] The *height* of the tree is the length of the longest root-to-leaf path. We extend a binary tree by adding enough *external* nodes to ensure that each original *internal*[3] node has two children. Each leaf thus has two external nodes as children. These external nodes, which appear as squares in our diagrams (see, for example, Figure 3.2.1), represent possible insertion positions in the tree. Since in many cases the display of external nodes only obscures the shape of a tree, most figures in this dissertation depict unextended binary trees.

When a binary tree T_π is constructed from a permutation (or set whose ranks form a permutation) $\pi = (\pi(1), ..., \pi(n))$ of $\{1, ..., n\}$, $\pi(1)$ goes to the root of T_π; each subsequent element $\pi(j)$ goes to the unique position found by comparing $\pi(j)$ with the root and then making a right or left turn, according as $\pi(j) > \pi(1)$ or $\pi(j) < \pi(1)$. The procedure is repeated in the subtrees until an empty subtree—an insertion position—is found. When the permutation comprising the ranks of the n data values is selected at random from the set of all permutations of the integers $\{1, ..., n\}$ in such a way that each permutation has an equal probability of selection, we call the tree a *random binary search tree*.[4] This model of randomness is suitable when the data values are selected at random from any continuous distribution. Known as the random permutation model, it is discussed in detail by Mahmoud (1992, Chapter 2). In this dissertation, we base all applications of random binary search trees on the random permutation model. Considering only the relative ranks of the data values, we may treat a data array of size

[2]Some authors require that the nodes on the lowest level of a complete tree be positioned as far to the left as possible, but the positioning of the nodes within a level is irrelevant to our analysis.

[3]Using a parallel term for the nodes of the unextended tree, we call the original nodes *internal*.

[4]Random binary search trees should not be confused with the deterministic *decision* trees we introduce in the next chapter.

n as a random permutation of the integers $\{1, ..., n\}$. Figure 2.1 shows a binary search tree being "grown" from a random permutation of the integers 1 through 8.

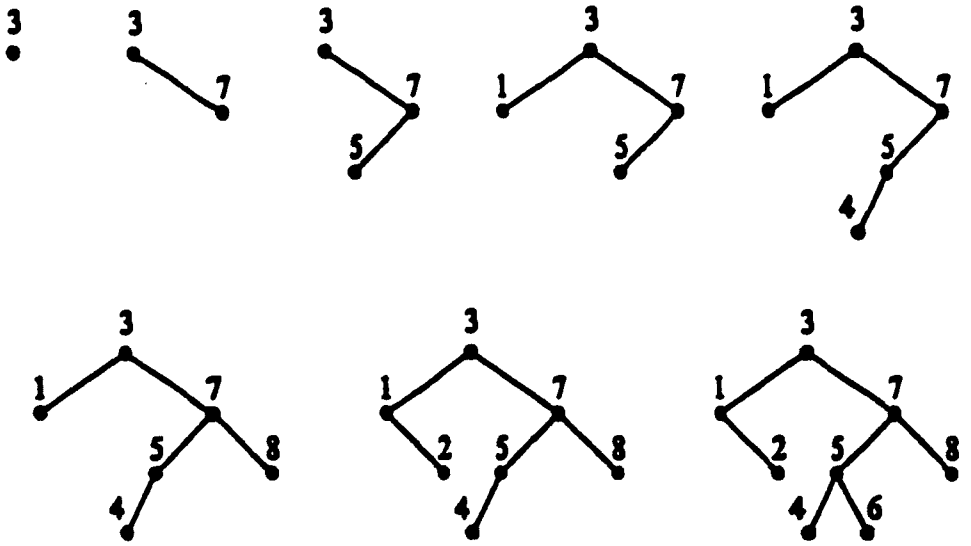

Figure 2.1. A binary search tree grown from the permutation 3 7 5 1 4 8 2 6.

In Section 2.2, we derive the distribution of the number of descendants of a node in a random binary search tree. We will use this result in analyzing the number of comparisons performed by MQS; it may have further applications in the analysis of other algorithms. The distribution is related to some well-known properties of binary search trees, which we now review.

2.1 Some Basic Properties of Binary Trees

Binary trees are widely used for storing data records in computers. Their properties have been extensively studied by Lynch (1965), Arora and Dent (1969), Devroye (1991),

Mahmoud (1992), Dobrow and Fill (1995), and many others. In this section, we review those properties of binary trees that relate to the material of this dissertation.

2.1.1 The Number of External Nodes in a Binary Tree

Recall that a binary tree of size n is constructed from n nodes and that we create the extended tree by adding enough external nodes to ensure that each internal node has two children. An extended tree of size n thus has a total of $2n$ nodes—internal or external—that are children. Since all nodes in a binary tree are children except the root node, $n-1$ of the $2n$ children are internal nodes, and the rest are external nodes. Thus the number of external nodes in an extended binary tree of size n is always $2n-(n-1)=n+1$.

2.1.2 The Height of a Binary Tree

The *height* of a binary tree—the length of the longest root-to-leaf path—is one less than the maximum number of comparisons performed in an unsuccessful search for a data value presumed to be stored in the tree. We will use this fact in Chapter 3 to establish a sufficient condition for the asymptotic normality of the number of comparisons performed in an insertion sort. Here we consider the range of h_n, the height of a tree of size n. It is easily seen (Mahmoud 1992, p. 58) that

$$\lfloor \log_2 n \rfloor \leq h_n \leq n-1.$$

The upper bound is attained by trees having only one node on each level. Complete trees contain the maximum possible number of nodes on each level, except possibly the lowest, and attain the lower bound. For $i = 0, \ldots, \lfloor \log_2 n \rfloor - 1$, a complete tree of size

n has 2^i nodes on the ith level. (Recall that we consider the root node the zeroth level.) The lowest level of a complete tree may not be fully saturated, since there may not be an integer j such that $n = 2^j - 1$. (Fill 1995 studies the probabilities of complete trees.)

2.1.3 External Paths in a Random Binary Search Tree

The external nodes of a binary search tree represent potential insertion positions. An unsuccessful search for a particular value in a tree follows one of the tree's *external paths*—paths from the root node to an external node. Let U_n denote the length of a randomly selected external path in a binary search tree created by successive insertion from a random permutation of the integers $\{1, ..., n\}$. That is, from a random binary search tree of size n, we select one external node (out of the $n + 1$ external nodes), giving each external node an equal probability of selection. We consider the length of the path from the root node to the selected external node. Thus U_n is subject to "double randomness" and denotes the number of comparisons performed during an unsuccessful search in a random binary tree of size n. Lynch (1965) has shown that, for $k = 1, ..., n$,

$$P\{U_n = k\} = \frac{2^k}{(n+1)!}\begin{bmatrix} n \\ k \end{bmatrix},$$

where $\begin{bmatrix} n \\ k \end{bmatrix}$ is the kth signless Stirling number of the first kind of order n. Mahmoud (1992, pp. 71–78) provides a detailed analysis of U_n and includes derivations of its mean and variance:

$$E[U_n] \sim 2 \ln n,$$

and

$$Var[U_n] \sim 2 \ln n.$$

On average, U_n is of the same order of magnitude as the minimum value of the height h_n. We may say, therefore, that randomly grown binary trees tend to be short and "bushy." Note that U_n may be thought of as the number of ancestors of a randomly selected external node in a random binary search tree. In the next subsection, we consider the number of ancestors of an internal node.

2.1.4 The Number of Ancestors of a Node in a Random Binary Search Tree

Let $A_j^{(n)}$ represent the number of ancestors of a node ranked j in a random binary search tree of size n. For notational simplicity (especially in the next section), we consider each node an ancestor of itself but *not* a descendant of itself. Arora and Dent (1969) showed that

$$E\left[A_j^{(n)}\right] = H_{n-j+1} + H_j - 1,$$

where $H_n = \sum_{i=1}^{n} 1/i$, the nth harmonic number. The number of ancestors of j is the number of comparisons that would be performed in a successful search for j in a binary tree. Let $A^{(n)}$ be the number of ancestors of a randomly selected internal node in a random binary search tree. Like U_n above, $A^{(n)}$ is subject to double randomness. We can find $E\left[A^{(n)}\right]$ by averaging $E\left[A_j^{(n)}\right]$ over all values of j:

$$\begin{aligned} E\left[A^{(n)}\right] &= \frac{1}{n}\left[\sum_{j=1}^{n} H_j + \sum_{j'=1}^{n} H_{j'}\right] - 1 \\ &= \frac{2}{n}\left[(n+1)H_n - n\right] - 1 \\ &= 2\left(1+\frac{1}{n}\right)H_n - 3 \\ &= 2\ln n + O(1). \end{aligned}$$

To obtain the second equality above, we have used the identity

$$\sum_{j=1}^{n} H_j = (n+1)\, H_n - n, \tag{2.1}$$

found in Mahmoud (1992), p. 10.

We may also relate $E\left[A_j^{(n)}\right]$ to the average number of descendants of j, as derived in the next section: Suppose we are given a random permutation π of the integers $\{1, ..., n\}$ that results in a binary tree in which $j' \neq j$ is an ancestor of j. We may form a second permutation, π' say, by swapping the positions of j and j' in π. Then (as is clear from the Claim on p. 15 below) π' results in a binary tree in which j is a *descendant* of j'. We thus establish a one-to-one correspondence between the set of permutations resulting in trees in which j is an ancestor of j' and the set of permutations resulting in trees in which j is a descendant of j'. Let $S_j^{(n)}$ be the number of descendants of j in a random binary search tree of size n. Then

$$\begin{aligned} E\left[S_j^{(n)}\right] &= \sum_{\substack{j'=1 \\ j' \neq j}}^{n} P\{j' \text{ is a descendant of } j\} \\ &= \sum_{\substack{j'=1 \\ j' \neq j}}^{n} P\{j' \text{ is an ancestor of } j\} \\ &= E\left[A_j^{(n)}\right] - P\{j \text{ is an ancestor of itself}\} \\ &= H_{n-j+1} + H_j - 2. \end{aligned} \tag{2.2}$$

In the next section, we derive this average directly from the distribution of $S_j^{(n)}$. The random variable $A_j^{(n)} - 1$ denotes the level of the node ranked j in the tree, and it is clear that $A_j^{(n)} - 1$ has the mean of $S_j^{(n)}$. Note, however, that the distributions of these two random variables are quite different. Consider, for example, their probabilities of taking

the value zero: for $j = 1, ..., n$,

$$P\left\{A_j^{(n)} - 1 = 0\right\} = P\{j \text{ is the root of the tree}\} = \frac{1}{n}.$$

while

$$P\left\{S_j^{(n)} = 0\right\} = \begin{cases} \frac{1}{2}, & \text{if } j \in \{1, n\}; \\ \frac{1}{3}, & \text{otherwise.} \end{cases}$$

These probabilities are easily proved. First,

$$P\left\{S_1^{(n)} = 0\right\} = P\{2 \text{ precedes } 1 \text{ in the permutation}\} = \frac{1}{2},$$

and

$$P\left\{S_n^{(n)} = 0\right\} = P\{n-1 \text{ precedes } n \text{ in the permutation}\} = \frac{1}{2};$$

similarly, for $j \in \{2, 3, ..., n-1\}$,

$$P\left\{S_j^{(n)} = 0\right\} = P\{j-1 \text{ and } j+1 \text{ precede } j \text{ in the permutation}\} = \frac{1}{3}.$$

The full distribution of $S_j^{(n)}$ is derived in the next section.

2.2 The Number of Descendants of a Node in a Random Binary Search Tree

As before, let $S_j^{(n)}$ be the number of descendants of j in a binary search tree constructed from a random permutation of the integers 1 through n. For a cleaner notation, we suppress the superscript n. To find $P\{S_j = k\}$, $k = 0, ..., n-1$, we find the number of permutations of the n integers which correspond to binary trees in which j has exactly k descendants.

Claim:[5] Let π be a permutation of the integers $\{1, \ldots, n\}$, and let T_π be the binary tree grown from π. Let $j \in \{1, \ldots, n\}$, and let $T_\pi^{(j)}$ be the subtree of T_π rooted at j. The following are equivalent:

1. $T_\pi^{(j)}$ comprises nodes corresponding to the integers $d_1 < \cdots < d_{k+1}$.
2. All of the following hold:
   (a) The integers $d_1, \ldots, d_{k+1}$ are consecutive integers in $\{1, \ldots, n\}$ and include j.
   (b) All integers in the set $\{d_1, \ldots, d_{k+1}\} - \{j\}$ follow j in π.
   (c) The integers $d_1 - 1$ and $d_{k+1} + 1$, if included in $\{1, \ldots, n\}$, precede j in π.

Proof of Claim: First we show that (1) implies (2). Since T_π is constructed from a permutation, all integers in a subtree of T_π clearly must form a consecutive set. Also, descendants of j must follow j in π, so all members of $\{d_1, \ldots, d_{k+1}\} - \{j\}$ follow j in π. We show by contradiction that $d_1 - 1$ and $d_{k+1} + 1$, if they are in $\{1, \ldots, n\}$, lie on the path from the root to j in T_π. Suppose $d_1 - 1$ is in $\{1, \ldots, n\}$ and does not lie on the path from the root to j. If d_1 precedes $d_1 - 1$ in π, then when $d_1 - 1$ is inserted into T_π, it must be compared with d_1 and so must be a descendant of d_1 and thus also of j—a contradiction. If $d_1 - 1$ precedes d_1 in π, then when d_1 is inserted into T_π, it must be compared with $d_1 - 1$. Then since $d_1 - 1$ is not a descendant of j and does not lie on the path from the root to j, d_1 cannot be a descendant of j—again a contradiction. A similar argument shows that if $d_{k+1} + 1$ is in $\{1, \ldots, n\}$, it must lie on the path from the

[5] Devroye (1991) states this fact without proof.

root to j. So clearly $d_1 - 1$ and $d_{k+1} + 1$, if they are in $\{1, ..., n\}$, must precede j in π. and condition (2) is satisfied.

Next assuming condition (2), let π_j be the (truncated) permutation obtained by truncating π just before j, so that π_j comprises integers preceding j in π. and let T_{π_j} be the binary tree grown from π_j. Then T_{π_j} contains $d_1 - 1$ and $d_{k+1} + 1$, if they are in $\{1, ..., n\}$. Also, if $d_1 - 1$ is in $\{1, ..., n\}$, it is the largest number in T_{π_j} that is less than j. Thus when j is inserted into T_π, j must be compared with $d_1 - 1$, so $d_1 - 1$ lies on the path from the root to j in T_π. Similarly, if $d_{k+1} + 1$ is in $\{1, ..., n\}$, then $d_{k+1} + 1$ is the smallest number greater than j in T_{π_j} and so must lie on the path from the root to j in T_π. Let δ be the first number in the set $\{d_1, ..., d_{k+1}\} - \{j\}$ to appear in π. Let π_δ be the permutation obtained by truncating π just before δ, and let T_{π_δ} be the binary tree grown from π_δ. If $\delta < j$, then j is the smallest number greater than δ in T_{π_δ}, so when δ is inserted, it must be compared with j and thus become a descendant of j in T_π. If $\delta > j$, then j is the greatest number less than δ in T_{π_δ}, so again, δ must join $T_\pi^{(j)}$. Similarly, when subsequent members of the set $\{d_1, ..., d_{k+1}\} - \{j\}$ are inserted, they will be "sandwiched" (in rank) between either (*a*) $d_1 - 1$ and some member of $T_\pi^{(j)}$; or (*b*) $d_{k+1} + 1$ and some member of $T_\pi^{(j)}$; or (*c*) two members of $T_\pi^{(j)}$. In any case, they must join $T_\pi^{(j)}$. So $T_\pi^{(j)}$ includes $d_1, ..., d_{k+1}$. Since $T_\pi^{(j)}$ does not include $d_1 - 1$ or $d_{k+1} + 1$ (which, if they are in T_π, lie on the path from the root to j), $T_\pi^{(j)}$ comprises only $d_1, ..., d_{k+1}$. ■

So to find the number of permutations corresponding to trees in which j has k

descendants, imagine the numbers 1 through n lined up in order from left to right. Suppose a sliding window, just wide enough to cover $k+1$ numbers, is placed over the number j. The numbers appearing in the window make up the ordered set d_1 through d_{k+1}—the integers in the subtree rooted at j. (Later we will slide the window from side to side as we consider all possible sets of k descendants of j.) Imagine a frame around the window, with left and right sides just wide enough to cover one extra number each. The sides of the window frame cover the numbers $d_1 - 1$ and $d_{k+1} + 1$, if they lie between 1 and n inclusive. If $d_1 = 1$, the left side of the window frame will not cover any numbers; similarly for the right side of the frame if $d_{k+1} = n$. To find the number of permutations resulting in trees in which j has k descendants, we consider the number of ways to position the window around j; that is, we consider the number of choices for the set of k descendants. For each set, we take into account the position of the window frame, since any integers covered by the frame must precede j in the permutation.

For $k = 0, \ldots, n-2$, let $\theta \in \{1, 2\}$ be the number of integers covered by the window frame when j and its (fixed) descendants $\delta_1, \ldots, \delta_k$ are visible in the window. Then $P\{j \text{ has descendants } \delta_1, \ldots, \delta_k\}$ may be expressed as

$$\frac{\left[\begin{array}{c}\text{number of permutations of}\\ \text{integers covered by frame}\end{array}\right]\left[\begin{array}{c}\text{number of permutations of}\\ \delta_1, \ldots, \delta_k\end{array}\right]}{\left[\begin{array}{c}\text{number of permutations of all integers}\\ \text{covered by window and frame}\end{array}\right]} = \frac{\theta!\, k!}{(k+\theta+1)!}.$$

Simplifying this expression, we obtain

$$P\{j \text{ has descendants } \delta_1, \ldots, \delta_k\} = \begin{cases} \frac{1}{(k+2)(k+1)}, & \text{if } \theta = 1, \\ \frac{2}{(k+3)(k+2)(k+1)}, & \text{if } \theta = 2. \end{cases}$$

So, for example, when $k \leq \min\{(j-2), (n-j-1)\}$, the window is narrow enough

so that we can slide it all the way to either side of j and each of the two sides of the frame will still cover one integer. We have $k+1$ possible sets of k descendants, each with two frame integers, i.e., $\theta = 2$. Thus, for the case $k \leq \min\{(j-2),(n-j-1)\}$.

$$P\{S_j = k\} = \frac{2(k+1)}{(k+3)(k+2)(k+1)} = \frac{2}{(k+3)(k+2)}.$$

For other values of k, we must consider cases in which one or both sides of the frame do not cover any integers. When $j-1 \leq k \leq n-j-1$, the left frame will not cover any integers if we slide the window all the way to the left side, but we can still slide it all the way to the right side without sending it over the edge. So we have one set of k descendants for which there is only one frame integer, and $(j-1)$ sets of k descendants for which there are two frame integers. Thus for the case $j-1 \leq k \leq n-j-1$,

$$P\{S_j = k\} = \frac{1}{(k+2)(k+1)} + \frac{2(j-1)}{(k+3)(k+2)(k+1)}.$$

In general, for $i \in \{1,2\}$, let a_i denote the number of possible sets of k descendants of j for which $\theta = i$. Considering all possible positions for the sliding window, we have

$$a_1 = 0,\ a_2 = k+1, \qquad \text{if } k \leq \min\{j-2, n-j-1\};$$

$$a_1 = 1,\ a_2 = j-1, \qquad \text{if } j-1 \leq k \leq n-j-1;$$

$$a_1 = 1,\ a_2 = n-j, \qquad \text{if } n-j \leq k \leq j-2;$$

$$a_1 = 2,\ a_2 = n-k-2, \qquad \text{if } k \geq \max\{j-1, n-j\}.$$

Then for $k = 0, \ldots, n-2$,

$$P\{S_j = k\} = \frac{a_1}{(k+2)(k+1)} + \frac{2a_2}{(k+3)(k+2)(k+1)},$$

with a_1 and a_2 as given above. And trivially,

$$P\{S_j = n-1\} = P\{j \text{ appears first in the permutation}\} = \frac{1}{n}.$$

So for $j \leq \frac{n+1}{2}$. and for a positive integer m.

$$\begin{aligned} E\left[S_j^m\right] &= 2\sum_{k=0}^{j-2} \frac{k^m}{(k+3)(k+2)} \\ &\quad + \sum_{k=j-1}^{n-j-1} \frac{(k+2j+1)k^m}{(k+3)(k+2)(k+1)} \\ &\quad + 2(n+1) \sum_{k=n-j}^{n-2} \frac{k^m}{(k+3)(k+2)(k+1)} \\ &\quad + \frac{(n-1)^m}{n}. \end{aligned} \tag{2.3}$$

After simplifying this expression (using the MAPLE software) for the cases $m = 1$ and $m = 2$, we obtain, for $j \leq \frac{n+1}{2}$,

$$E[S_j] = H_{n-j+1} + H_j - 2. \tag{2.4}$$

and

$$\begin{aligned} E\left[S_j^2\right] &= 10 + \frac{3}{j+1} + \frac{3}{n-j+2} \\ &\quad - (2j+5) H_{j+1} + (2j-2n-7) H_{n-j+2} \\ &\quad + 2(n+1) H_{n+1} + 2n. \end{aligned} \tag{2.5}$$

The expression derived in Chapter 4 for the average number of comparisons used by MQS involves sums, over all values of j, of the moments of S_j. By symmetry we have, for n even,

$$\begin{aligned} \sum_{j=1}^{n} E[S_j] &= 2\sum_{j=1}^{n/2} E[S_j] \\ &= 2\{(n+1) H_n - 2n\} \\ &= 2(n+1) H_n - 4n, \end{aligned}$$

where again we have used equation 2.1 to sum the Harmonic numbers. Using similar identities, we can sum $E\left[S_j^2\right]$ over j:

$$\begin{aligned}\sum_{j=1}^{n} E\left[S_j^2\right] &= 2\sum_{j=1}^{n/2} E\left[S_j^2\right] \\ &= 2\left\{\frac{3}{2}n^2 + \frac{17}{2}n - 5(n+1)H_n\right\} \\ &= 3n^2 + 17n - 10(n+1)H_n.\end{aligned}$$

For n odd,

$$\sum_{j=1}^{(n-1)/2} E[S_j] = \sum_{j=1}^{(n-1)/2} H_{n-j+1} + \sum_{j=1}^{(n-1)/2} H_j - (n-1),$$

and

$$E\left[S_{\frac{n+1}{2}}\right] = 2H_{\frac{n+1}{2}} - 2.$$

By symmetry,

$$\sum_{j=1}^{n} E[S_j] = 2\sum_{j=1}^{(n-1)/2} E[S_j] + E\left[S_{\frac{n+1}{2}}\right].$$

Plugging in the expressions above for $\sum_{j=1}^{\frac{n-1}{2}} E[S_j]$ and $E\left[S_{\frac{n+1}{2}}\right]$ and simplifying gives, for n odd,

$$\sum_{j=1}^{n} E[S_j] = 2(n+1)H_n - 4n,$$

the same formula obtained for even values of n. Similarly, for n odd,

$$\begin{aligned}\sum_{j=1}^{(n-1)/2} E\left[S_j^2\right] &= 5(n-1) + 3\sum_{j=1}^{(n-1)/2} \frac{1}{j+1} + 3\sum_{j=1}^{(n-1)/2} \frac{1}{n-j+2} \\ &\quad -2\sum_{j=1}^{(n-1)/2} (j+1)H_{j+1} - 2\sum_{j=1}^{(n-1)/2} (n-j+2)H_{n-j+2} \qquad (2.6) \\ &\quad -3\sum_{j=1}^{(n-1)/2} H_{j+1} - 3\sum_{j=1}^{(n-1)/2} H_{n-j+2} + (n+1)(n-1)H_{n+1} \\ &\quad +n(n-1),\end{aligned}$$

and

$$E\left[S^2_{\frac{n+1}{2}}\right] = 10 + \frac{12}{n+1} - 2(n+3)H_{\frac{n+1}{2}} - 6H_{\frac{n+1}{2}} \qquad (2.7)$$
$$+2(n+1)H_{n+1} + 2n.$$

Again we have

$$\sum_{j=1}^{n} E\left[S_j^2\right] = 2\sum_{j=1}^{(n-1)/2} E\left[S_j^2\right] + E\left[S^2_{\frac{n+1}{2}}\right],$$

so, using expressions 2.6 and 2.7 above, we obtain

$$\sum_{j=1}^{n} E\left[S_j^2\right] = 3n^2 + 17n - 10(n+1)H_n,$$

again the same formula we derived for n even. For each fixed $m \geq 3$ and uniformly in $j \leq \frac{n+1}{2}$, we have, from equation 2.3,

$$\begin{aligned}
E\left[S_j^m\right] &= 2\sum_{k=1}^{j-2}\left[k^{m-2} + O\left(k^{m-3}\right)\right] + \sum_{\substack{k=j-1\\k\neq 0}}^{n-j-1}\left[k^{m-2} + O\left(k^{m-3}\right)\right] \\
&\quad +2j\sum_{\substack{k=j-1\\k\neq 0}}^{n-j-1}\left[k^{m-3} + O\left(k^{m-4}\right)\right] + 2(n+1)\sum_{k=n-j}^{n-2}\left[k^{m-3} + O\left(k^{m-4}\right)\right] \\
&\quad +n^{m-1} + O\left(n^{m-2}\right) \\
&= \frac{2(j-2)^{m-1}}{m-1} + \frac{(n-j-1)^{m-1}}{m-1} - \frac{(j-2)^{m-1}}{m-1} \\
&\quad +2j\left[\frac{(n-j-1)^{m-2}}{m-2} - \frac{(j-2)^{m-2}}{m-2}\right] \\
&\quad +2n\left[\frac{(n-2)^{m-2}}{m-2} - \frac{(n-j-1)^{m-2}}{m-2}\right] + n^{m-1} + O\left(n^{m-2}\right) \\
&= \frac{(j-2)^{m-1}}{m-1} + \frac{(n-j-1)^{m-1}}{m-1} + \frac{2j(n-j)^{m-2}}{m-2} - \frac{2j(j-2)^{m-2}}{m-2} \\
&\quad +\frac{2n(n-2)^{m-2}}{m-2} - \frac{2n(n-j-1)^{m-2}}{m-2} + n^{m-1} + O\left(n^{m-2}\right) \\
&= \frac{(n-j)^{m-1}}{m-1} - \frac{2(n-j)^{m-1}}{m-2} + j^{m-1}\left(\frac{1}{m-1} - \frac{2}{m-2}\right)
\end{aligned}$$

$$
\begin{aligned}
&+\frac{2n^{m-1}}{m-2}+n^{m-1}+O\left(n^{m-2}\right) \\
&= \left(\frac{1}{m-1}-\frac{2}{m-2}\right)\left[(n-j)^{m-1}+j^{m-1}\right]+n^{m-1}\left(1+\frac{2}{m-2}\right) \\
&\quad +O\left(n^{m-2}\right).
\end{aligned}
$$

So for n even,

$$
\begin{aligned}
\sum_{j=1}^{n/2} E\left[S_j^m\right] &= \left(\frac{1}{m-1}-\frac{2}{m-2}\right)\left[\sum_{j=1}^{n/2}(n-j)^{m-1}+\sum_{j=1}^{n/2} j^{m-1}\right] \\
&\quad +n^m\left(\frac{1}{2}+\frac{1}{m-2}\right)+O\left(n^{m-1}\right) \\
&= \left(\frac{1}{m-1}-\frac{2}{m-2}\right)\left[\sum_{k=n/2}^{n-1} k^{m-1}+\sum_{j=1}^{n/2} j^{m-1}\right]+n^m\left(\frac{1}{2}+\frac{1}{m-2}\right) \\
&\quad +O\left(n^{m-1}\right) \\
&= \left(\frac{1}{m-1}-\frac{2}{m-2}\right)\left[\frac{(n-1)^m}{m}-\frac{(n/2-1)^m}{m}+\frac{(n/2)^m}{m}\right] \\
&\quad +n^m\left(\frac{1}{2}+\frac{1}{m-2}\right)+O\left(n^{m-1}\right) \\
&= \left(\frac{1}{m-1}-\frac{2}{m-2}\right)\left(\frac{n^m}{m}\right)+n^m\left(\frac{1}{2}+\frac{1}{m-2}\right)+O\left(n^{m-1}\right) \\
&= n^m\left[\frac{1}{2}+\frac{1}{m-2}+\frac{1}{m(m-1)}-\frac{2}{m(m-2)}\right]+O\left(n^{m-1}\right) \\
&= n^m\left(\frac{1}{2}+\frac{1}{m-1}\right)+O\left(n^{m-1}\right).
\end{aligned}
$$

Then by symmetry we have, for n even and each fixed $m \geq 3$,

$$
\sum_{j=1}^{n} E\left[S_j^m\right] = n^m\left(1+\frac{2}{m-1}\right)+O\left(n^{m-1}\right).
$$

It is easy to verify that this result also holds for odd values of n: only three terms of the sum over j may differ in this case, and these do not affect the term of order n^m.

Chapter 3
Tree-growing Search Strategies

In this chapter we study classes of search strategies which may be used for sorting a data set by insertion sort. Insertion sort is an on-line sorting algorithm. At each point in time, all data values obtained so far are stored in a sorted array. When a new datum appears, the algorithm finds its correct position in the array and inserts it—perhaps moving some previously obtained data to new positions in order to make room. Let S_i denote the algorithm used to perform the ith search (the search for the correct position of the $(i+1)$st key), and let $\mathcal{S} = \{S_i\}_{i=1}^{\infty}$. We will call $\mathcal{S}$ a *search strategy*. Most commonly used search strategies possess certain consistency properties, which we define in this chapter. Using these properties, we will show that when, for each integer i, the search algorithm S_i specifies a sequence of probes defined as reasonable functions of the lengths of the data subarrays being searched, the number of comparisons consumed in the sorting has asymptotically normal behavior.

3.1 Classes of Search Strategies

To identify a class of reasonable probe-selection functions, we introduce *decision trees:* deterministic binary trees whose nodes correspond to the positions of the probes selected by a search algorithm. The position of the first probe chosen becomes the root of the decision tree; the positions of the second probes—at most one on each side of the first probe—become its children, and so forth. A probe falling at one of the ends of the data array will not have two internal nodes as children; one or both of its children will then be external nodes in the extended tree. The algorithm S_i is represented by T_i, a decision tree of size i whose root-to-leaf paths represent the possible probe sequences of S_i. We shall call the search strategy S a *tree-growing strategy* if, for every positive integer i, the shape of T_{i+1} may be obtained by adding an internal node in one of the insertion positions—one of the external nodes—of T_i. Since the tree-growing property provides an element of consistency across algorithms in the search strategy, we will restrict our class of consistent strategies to those possessing this property.

Assume the ranks of the data elements being sorted form a random permutation of the integers $\{1, ..., n\}$, as is the case when the data values are selected at random from any continuous distribution. Let the random variable C_n be the total number of times S compares a new key to a probe during the sorting of the first n keys. The class of *normal* search strategies comprises strategies for which C_n is asymptotically normal; that is

$$\frac{C_n - E[C_n]}{\sqrt{Var[C_n]}} \xrightarrow{D} N(0,1).$$

Although the asymptotic mean and variance of C_n obviously depend on the particular

strategy S, we shall see that the normality of a subclass of tree-growing strategies—the consistent strategies—may be established by relating the variance of C_n to the rate of growth in height of the decision trees in the family $\mathcal{T} = \{T_i\}_{i=1}^{\infty}$. Figure 3.1 shows the relationships between the three classes of search strategies: consistent strategies are tree-growing by definition, and we will show that they are normal.

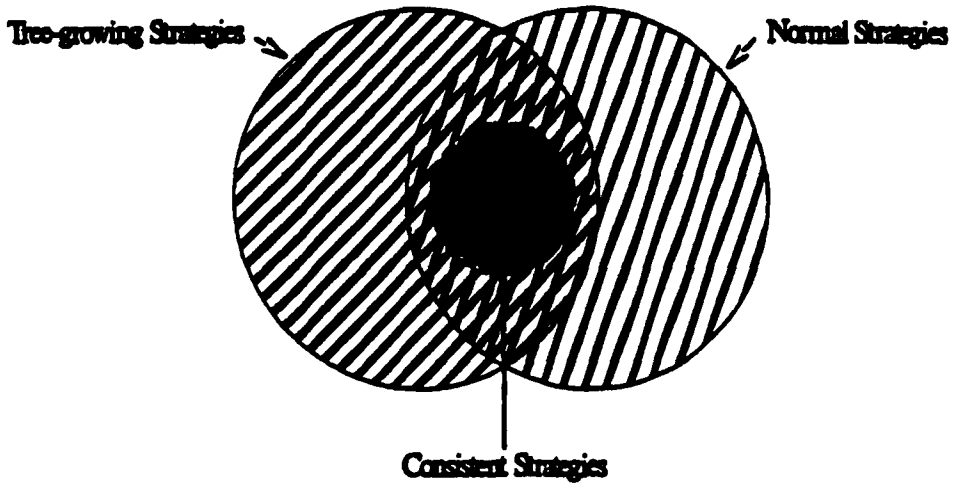

Figure 3.1. Classes of search strategies

The next section provides examples of tree-growing search strategies. In Section 3.3 we identify a subclass of tree-growing strategies as *consistent*. The remaining sections focus on the normality of consistent strategies. Section 3.4 gives a sufficient condition for the normality of tree-growing strategies. We use this condition to establish the normality of two commonly used strategies—binary search and linear search—in Section 3.5. We also establish their full asymptotic distributions, including the asymptotic means and variances which serve as centering and scaling factors. In Section 3.6, we use the sufficient condition to prove our main result: all consistent strategies are normal. We

extend this result further in Section 3.7, to include some tree-growing implementations of Fibonaccian search.

3.2 Examples of Tree-growing Strategies

We illustrate the tree-growing property through simple examples. One commonly used strategy known as Linear Search from the Bottom always selects as a first probe the largest value in the data array being searched. (We assume here that the largest value is at the "bottom" of the array and the smallest value at the "top.") When searching for the position of the $(i+1)$st key, Linear Search from the Bottom probes positions $i, i-1, i-2$, and so forth, until it discovers a key less than the $(i+1)$st key. If it finds such a key in position j, it inserts the new key in position $j+1$. Though Linear Search from the Bottom is inefficient, it is useful for searching sequential access data files (e.g., data stored on a tape, as discussed by Hu and Wachs 1987). Figure 3.2.1 shows the decision trees of Linear Search from the Bottom. The shape of T_{i+1} is obtained from T_i by adjoining a leaf to replace the leftmost external node of T_i. The i nodes of T_i are relabeled in T_{i+1}, and it is the *shape* of T_{i+1} that evolves from that of T_i by "tree-growing."

Variants of Linear Search from the Bottom include the obvious Linear Search from the Top as well as a slightly more complex method called c–jump search. Still based on sequential searching, c–jump search starts at the "top" of the data array and advances c positions at a time. When it finds a probe larger than the new key, it reverses direction and performs a linear search "from the bottom" of the relevant fragment (a fragment of size $c-1$) of the array. Three–jump search is illustrated by the three decision trees of

Figure 3.2.2.

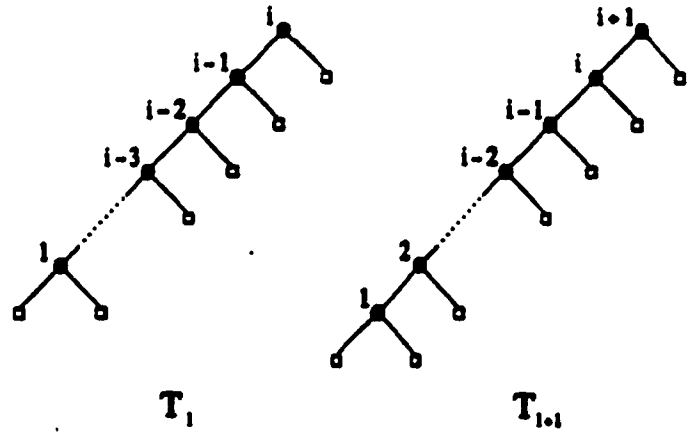

Figure 3.2.1. The extended decision trees of Linear Search From the Bottom

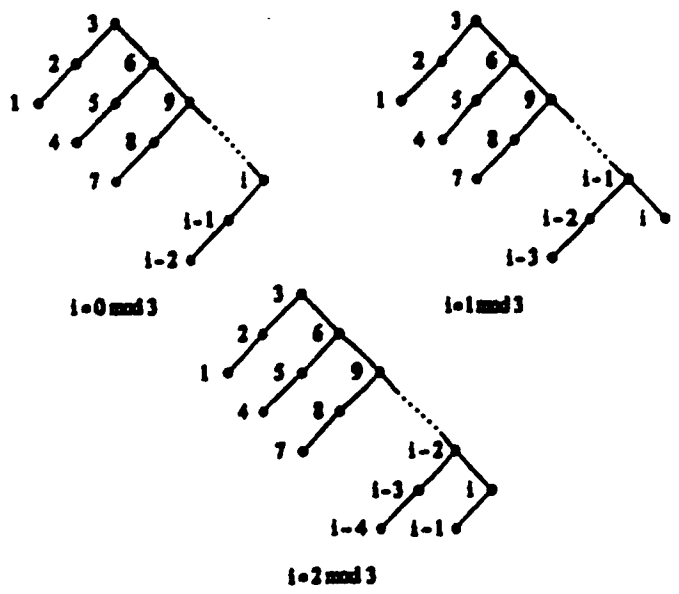

Figure 3.2.2. The (unextended) decision trees of three-jump search

A more efficient search method, also commonly used, is binary search, which always probes the middle position of the remaining portion of the list. Thus when the search

has been confined to a fragment of the data array between an upper bound u and a lower bound l, binary search chooses the probe in position $\left\lceil \frac{l+u}{2} \right\rceil$. For $i = 6$, the decision tree representing binary search is that of Figure 3.2.3.

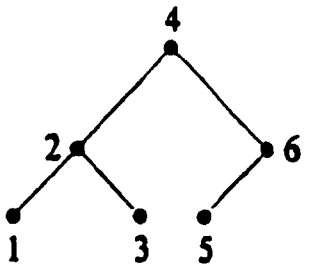

Figure 3.2.3. The (unextended) decision tree of binary search

Though all of the search strategies discussed above lie in the class of consistent strategies defined in the next section, strategies reasonable for some applications lie outside this class. Examples include Alternating Linear Search—a hybrid of Linear Search from the Bottom and Linear Search from the Top—which is represented by the decision trees of Figure 3.2.4. Figure 3.2.5 shows a decision tree for Fibonaccian search, a method so called because its decision tree is endowed with a recursive partition that follows the Fibonacci number sequence. When the tree size is $F_{k+1} - 1$, where F_j is the jth Fibonacci number,[6] the two subtrees of the root node have sizes $F_k - 1$ and $F_{k-1} - 1$, and the property propagates recursively in the subtrees. Fibonaccian search is sometimes preferred to binary search because Fibonaccian search requires only the operations of addition and subtraction to compute the position of the next probe (see Knuth 1973b, Section 6.2.1), whereas binary search requires division—a more complex computing operation.

[6]The usual definition of the Fibonacci number F_j is $F_j = F_{j-1} + F_{j-2}$, with $F_0 = 0$, $F_1 = 1$.

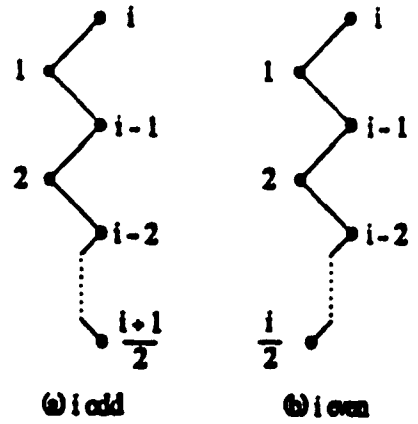

Figure 3.2.4. The (unextended) decision trees of Alternating Linear Search

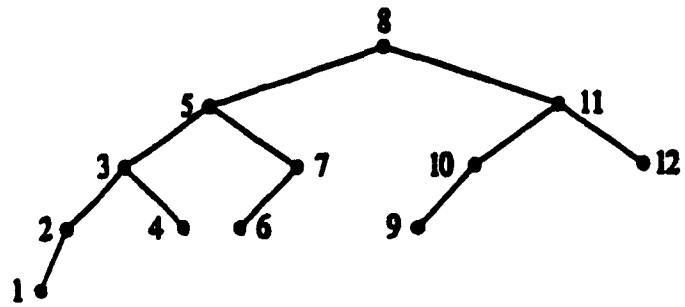

Figure 3.2.5. The (unextended) decision tree of Fibonaccian search in an array of 12 keys

Like Alternating Linear Search, Fibonaccian search fails to meet our consistency criteria. We will see, however, that all of the search strategies discussed here are normal, except possibly some implementations of Fibonaccian search.[7]

3.3 Consistent Strategies

In this section we present a condition on the search strategy S that is equivalent to the

[7]The normality of Fibonaccian search may depend on the search strategy used when the number of keys in the array being searched is not of the form $F_k - 1$. In Section 3.7 we discuss two implementations of Fibonaccian search for which we can prove normality.

tree-growing property and give a rigorous definition of consistency.

Let T_i be the tree representing the search algorithm S_i—the algorithm used to insert the $(i+1)$st key. Let A_i be the sorted array formed by the first i keys. Think of A_i as a string of beads laid on a table and numbered from left to right. We search for the correct position of the $(i+1)$st key by successively selecting probes in A_i, at each step comparing the $(i+1)$st key to the selected probe. Think of the probes as beads which we remove from the string, thus breaking the string into many beaded string fragments. At the first step in the insertion, we select one probe (remove one bead); at the second step, we again select one probe, but the probe we select depends on the outcome of our comparison of the $(i+1)$st key to the first probe. Thus there are two possible second probes: one in case the $(i+1)$st key is larger than the first probe and another in case it is smaller (unless, of course, the first probe selected falls at one of the ends of A_i—in this case, the algorithm would specify only one probe at the second step). In general then, at the jth step in the insertion ($j \geq 1$), the algorithm specifies up to 2^{j-1} possible probes.

Let P_{ij} be the set of all possible positions for the first j probes in the insertion of the $(i+1)$st key. Then P_{ij} has at most $2^j - 1$ elements, which correspond to the nodes on levels zero through $j-1$ of T_i and also to the collection of beads we have taken out of the string in the first j steps. Let $A(i,j,1)$, $A(i,j,2)$, ..., $A(i,j,2^j)$ be the subarrays of A_i (some may be empty) created by the partitioning which results from deleting all elements of P_{ij}. Think of these as the beaded string fragments left after taking out all the beads corresponding to the elements of P_{ij}. Note that there are at most 2^j such string fragments; there will be less than 2^j fragments if at any stage we have taken out some

beads that were adjacent to each other or some that were on the ends of the original string. So if we have some adjacent probes or probes at the ends of the array, some of the subarrays $A(i,j,k)$ will be empty. Let l_{ijk} be the length of $A(i,j,k)$, $k = 1.2, ..., 2^j$ (zero if $A(i,j,k)$ is empty). At the $(j+1)$st step in the insertion, we select one probe from each of the nonempty subarrays (or remove one bead from each of the string fragments that are left). Let p_{ijk} be the position of the next probe within the subarray $A(i,j,k)$, if $A(i,j,k)$ is not empty. That is, we number the beads in each of the string fragments separately from left to right and let p_{ijk} be the position of the bead to be removed from the fragment corresponding to $A(i,j,k)$, relative to the other beads in the fragment.

Claim: The search strategy S has the tree-growing property if and only if, for all i, j, and k such that $A(i,j,k)$ is not empty,

$$p_{ijk} = f(l_{ijk}, j, k),$$

for some function f such that

$$f(l_{ijk}+1, j, k) = f(l_{ijk}, j, k) + \alpha(l_{ijk}, j, k),$$

and $\alpha(l_{ijk}, j, k)$ takes values in the set $\{0,1\}$.

Remark: The condition above, which we will call the tree-growing condition, implies that (1) the probe selected in $A(i,j,k)$ at the $(j+1)$st step cannot depend directly on $i+1$, the label of the insertion; and (2) if the array $A(i,j,k)$ were larger by one element, the position of the probe selected would be either p_{ijk} or $p_{ijk}+1$. In other words, if the

string were one bead longer, we would select either the same bead or the bead to the right of this bead (recalling that the beads are laid on a table and numbered from left to right).

Proof of Claim: We first show that the tree-growing condition implies the tree-growing property. To visualize the argument we present, think of the beads in the original string of i beads as corresponding to the internal nodes of T_i and the beaded string fragments as corresponding to subtrees within T_i. The next bead to be removed from one string fragment corresponds to the root of the subtree formed by the beads within the fragment; the beads to the left of this bead (if any) correspond to the left sub-subtree, while the beads to the right of it (if any) correspond to the right sub-subtree. In order to compare T_i and T_{i+1} (the trees representing the search strategies for inserting the $(i+1)$st and $(i+2)$nd keys), we wish to number the internal nodes on each level of these trees from left to right and compare the subtrees rooted at each level. To make the numbering easier, visualize two infinite complete binary trees τ_i and τ_{i+1}. We superimpose T_i onto τ_i by marking the nodes in τ_i which also appear in T_i; we superimpose T_{i+1} similarly onto τ_{i+1}. At each level j in τ_i and τ_{i+1}, number the 2^j subtrees rooted at level j from left to right. Note that the sizes of the left and right subtrees rooted at level one of T_i are $l_{i,1,1}$ and $l_{i,1,2}$, respectively (the sizes of the subarrays $A(i,1,1)$ and $A(i,1,2)$). That is, after we have removed one bead from the original string, we are left with two string fragments, one with $l_{i,1,1}$ beads and one with $l_{i,1,2}$ beads. The tree-growing condition says that if the original string of beads we had started with had been longer by one bead, we would

have selected either the same bead or the bead to the right of that bead. Therefore, the lengths of the two string fragments we would get if the original string had been one bead longer are the same as those we get by partitioning the shorter string, except that, if we had started with the longer string, one of the two string fragments would be longer by one bead. In symbols, by the tree-growing condition we have

$$p_{i+1,0,1} = \begin{cases} p_{i,0,1}, & \text{if } \alpha(i,0,1) = 0; \\ p_{i,0,1} + 1, & \text{if } \alpha(i,0,1) = 1, \end{cases}$$

where $p_{i+1,0,1}$ is the first probe chosen in the $(i+2)$nd insertion. In terms of the lengths of the subarrays $A(i+1,1,1)$ and $A(i.1,1)$, we must have either

$$l_{i+1,1,1} = l_{i,1,1} \text{ and } l_{i+1,1,2} = l_{i,1,2} + 1$$

or

$$l_{i+1,1,1} = l_{i,1,1} + 1 \text{ and } l_{i+1,1,2} = l_{i,1,2}.$$

Next consider the lengths of the beaded string fragments left after removing the one or two beads selected at the second step in the $(i+2)$nd insertion. The tree-growing condition implies that the lengths of the four (or fewer) fragments left after the second removal of beads would be the same for a string with i or $i+1$ beads, except that one of the fragments obtained from the longer string would have one additional bead. The lengths of the beaded string fragments from a string of length $i+1$ correspond to the numbers of marked nodes in the subtrees rooted at level two in τ_{i+1}. So in terms of the subtrees of τ_{i+1} and τ_i, the tree-growing condition, as it applies to the choices for $p_{i+1,1,1}$ and $p_{i+1,1,2}$, implies that the number of marked nodes in the kth subtree $(k = 1, ..., 4)$

rooted at level two in τ_{i+1} must be the same as the number of marked nodes in the kth subtree rooted at level two in τ_i, except that one of the four subtrees in τ_{i+1} will have one additional marked node. In general, l_{ijk} is the number of marked nodes in the kth subtree rooted at level j in τ_i. And by the tree-growing condition, applied successively to the probes selected at steps one through j, the number of marked nodes in the kth subtree rooted at level j in τ_{i+1} will be the same as the number of marked nodes in the kth subtree rooted at level j in τ_i, except that one of the subtrees in τ_{i+1} will have one additional marked node. Since this implies that the extra node contained in one of the subtrees of T_{i+1} must eventually fall to one of the insertion positions (external nodes) of T_i, the tree-growing property must hold.

Arguing by contradiction, it is easy to see that the tree-growing property implies the tree-growing condition. For if the tree-growing condition did not hold, the size of one of the subtrees of T_{i+1} would differ from that of the corresponding subtree of T_i by more than one. Thus we could not obtain T_{i+1} from T_i simply by inserting a node at one of the insertion positions. ■

The tree-growing property provides some similarity of probe selection functions within and across the algorithms S_i, $i = 1, ..., n$. We restrict our class of *consistent* strategies to those which possess additional consistency properties:

Definition: Let $S = \{S_i\}_{i=1}^{\infty}$ be a tree-growing strategy for insertion sort. We call S a *consistent* strategy if, for all i, j, and k such that $A(i, j, k)$ is not empty, S_i specifies the

relative rank (within $A(i,j,k)$) of the probe selected in $A(i,j,k)$ by

$$p_{ijk} = \varphi_S(l_{ijk}), \qquad 1 \le p_{ijk} \le l_{ijk},$$

where the function φ_S has the following property: if $g_S(n) = \min(\varphi_S(n) - 1, n - \varphi_S(n))$, then $\lim_{n\to\infty} g_S(n)/n$ exists.

In words, $g_S(n)$ is the size of the smaller subtree rooted at level one of T_n, and consistency requires that the proportion of nodes belonging to the smaller subtree approach a limit as n approaches infinity. Note also that for the consistent strategies, the position of the probe p_{ijk} within $A(i,j,k)$ depends only on l_{ijk}, the length of the subarray. Since the consistent strategies are tree-growing, we have

$$\varphi_S(l_{ijk} + 1) = \varphi_S(l_{ijk}) + \alpha_S(l_{ijk}),$$

and $\alpha_S(l_{ijk})$ takes values in the set $\{0, 1\}$. Examples of consistent strategies include c–jump search, which is specified by the function

$$\varphi_S(l_{ijk}) = \begin{cases} c, & \text{if } l_{ijk} > c; \\ l_{ijk}, & \text{otherwise.} \end{cases}$$

In this case, $\lim_{n\to\infty} g_S(n)/n = 0$.

For Alternating Linear Search, however, the position of the next probe is a function of both l_{ijk} and j:

$$p_{ijk} = \begin{cases} 1, & j \text{ even;} \\ l_{ijk}, & j \text{ odd.} \end{cases}$$

Thus p_{ijk} is not a function of the length only—it depends explicitly on j (the step number). So, though Alternating Linear Search is tree-growing (see Figure 3.2.4), it is not consistent.

3.4 A Sufficient Condition for Normality of Tree-Growing Strategies

Let T_i be the decision tree for S_i, the search algorithm used to insert the $(i+1)$st key by the consistent search strategy $S = \{S_i\}_{i=1}^{\infty}$. Let h_i denote the height of T_i, let X_i denote the number of comparisons made by S_i, and let $C_n = \sum_{i=1}^{n-1} X_i$, the total number of comparisons required by S for sorting n keys. For each positive integer n, the random variables $X_1, ..., X_{n-1}$ and C_n may be defined on the space of random permutations of the integers $\{1, ..., n\}$. Each insertion is performed independently of all others, so we may take the X_i's to be independent random variables. Let $s_n^2 = Var[C_n]$. If we show that, for every $\varepsilon > 0$,

$$\lim_{n\to\infty} \frac{1}{s_n^2} \sum_{i=1}^{n-1} \int_{\{|X_i - E[X_i]| > \varepsilon s_n\}} (X_i - E[X_i])^2 \, dP = 0,$$

we will know by the Lindeberg Theorem (see Billingsley 1986, p. 368) that C_n is asymptotically normal.

Lemma 3.1: *If $h_n = o(s_n)$, then S is a normal strategy.*

Proof: For $i = 1, ..., n-1$, let $Y_i = X_i - E[X_i]$, and let $\mathcal{F}_i$ denote the distribution function of Y_i. Then

$$P\{|Y_i| > h_n + 1\} \le P\{\max(X_i, E[X_i]) > h_i + 1\} = 0,$$

since X_i cannot exceed $h_i + 1$. Now if $h_n = o(s_n)$ and $\varepsilon > 0$, there is a constant N_ε such that, for all $n > N_\varepsilon$, $h_n + 1 < \varepsilon s_n$. Then for $i = 1, \ldots, n-1$,

$$\int_{\{|y| \geq \varepsilon s_n\}} y^2 d\mathcal{F}_i(y) \leq \int_{\{|y| > h_n + 1\}} y^2 d\mathcal{F}_i(y) = 0,$$

because the integral is taken over a set of zero probability. Thus for $n > N_\varepsilon$,

$$\sum_{i=1}^{n-1} \int_{\{|y| \geq \varepsilon s_n\}} y^2 d\mathcal{F}_i(y) = 0.$$

So

$$\lim_{n \to \infty} \frac{1}{s_n^2} \sum_{i=1}^{n-1} \int_{\{|y| \geq \varepsilon s_n\}} y^2 d\mathcal{F}_i(y) = 0,$$

and the normality of S follows from the Lindeberg theorem. ■

As we will see in Chapter 5 (subsection 5.1.2), there are tree-growing strategies that do not satisfy the condition of Lemma 3.1.

3.5 Normality of Two Commonly Used Search Strategies

We illustrate the use of the sufficient condition for normality by showing the normality of two search strategies commonly used for insertion sort: binary search and linear search. For these we obtain the full asymptotic distributions, using the asymptotic means for centering and variances for scaling. We begin by introducing some notation to be used here and in later sections.

Notation: Let $S = \{S_i\}_{i=1}^{\infty}$ be a consistent strategy, and let $\mathcal{T} = \{T_i\}_{i=1}^{\infty}$ be the sequence of decision trees representing S. For $i = 1, \ldots, n$, let h_i be the height of T_i, and for $k = 1, 2, \ldots$, let $L_k = \min\{i : h_i = k\}$, and $U_k = \max\{i : h_i = k\}$. That is,

$T_{L_k}, T_{L_k+1}, ..., T_{U_k}$ are the only trees in $\mathcal{T}$ that have height k, and thus $U_k + 1 = L_{k+1}$. Define $U_0 = 1$, and for $k = 1, 2, ...$, let $m_k = U_k - U_{k-1}$. As before, let C_n denote the number of comparisons needed to sort the first n keys by insertion sort using S, and X_i the number of comparisons required for inserting the $(i+1)$st key. (Again C_n and the X_i's may be defined on the space of random permutations of the integers $\{1, ..., n\}$.)

We define the symbol Ω as used here and in later sections: If χ and ψ are functions of n, we say that $\chi(n) = \Omega(\psi(n))$ if there are positive constants N and k such that for all $n > N$,

$$|\chi(n)| > k|\psi(n)|.$$

Since C_n and the X_i's are defined on the same probability space, we may write

$$C_n = \sum_{i=1}^{n-1} X_i.$$

So

$$E[C_n] = \sum_{i=1}^{n-1} E[X_i],$$

and, since we assume the X_i's are independent,

$$s_n^2 = Var[C_n] = \sum_{i=1}^{n-1} Var[X_i].$$

Binary Search is a consistent strategy which, when searching a subarray of length l, selects as a probe the element whose rank (relative to the other elements in the subarray) is $\lceil \frac{l}{2} \rceil$. This strategy results in a sequence of complete decision trees. The external nodes of the complete decision tree representing the binary search algorithm S_i lie only on one or two levels: the lowest level $\lfloor \log_2 i \rfloor$ (if $i+1$ is a power of 2) or the two lowest

levels, $\lfloor \log_2 i \rfloor$ and $\lfloor \log_2 i \rfloor + 1$. Thus $X_i = \lfloor \log_2 i \rfloor + B_i$, where B_i is a Bernoulli random variable that assumes the value one with probability

$$\frac{2\left(i+1-2^{\lfloor \log_2 i \rfloor}\right)}{i+1},$$

the proportion of external nodes at level $\lfloor \log_2 i \rfloor + 1$ (see Knuth 1973a, Exercise 2.3.4.5–3). So

$$E[X_i] = \frac{2\left(i+1-2^{\lfloor \log_2 i \rfloor}\right)}{i+1} + \lfloor \log_2 i \rfloor,$$

and

$$\begin{aligned} E[C_n] &= \sum_{i=1}^{n-1} \frac{2\left(i+1-2^{\lfloor \log_2 i \rfloor}\right)}{i+1} + \sum_{i=1}^{n-1} \lfloor \log_2 i \rfloor \\ &= n \log_2 n + O(n). \end{aligned}$$

For binary search, $U_k = 2^{k+1} - 1$, the number of nodes in a complete tree of height k, so $m_k = 2^k$. To compute $s^2_{U_j}$, we may first sum the variances of the X_i's corresponding to trees of height k, for $k = 1, \ldots, j$, and then sum over k:

$$s^2_{U_j} = \sum_{k=1}^{j} \sum_{i=0}^{2^k-1} Var[X_{L_k+i}].$$

Again using the result about the number of nodes on levels $\lfloor \log_2 i \rfloor$ and $\lfloor \log_2 i \rfloor + 1$,

$$\sum_{i=0}^{2^k-1} Var[X_{L_k+i}] = \sum_{i=0}^{2^k-1} \left(\frac{2(i+1)}{2^k+i+1}\right)\left(1 - \frac{2(i+1)}{2^k+i+1}\right).$$

Some straightforward algebraic manipulation shows that

$$\begin{aligned} \sum_{i=0}^{2^k-1} Var[X_{L_k+i}] &= 2\sum_{i=0}^{2^k-1} \frac{i+1}{2^k+i+1} - 4\sum_{i=0}^{2^k-1} \frac{(i+1)^2}{(2^k+i+1)^2} \\ &= 2\left[2^k - 2^k\left(H_{2^{k+1}} - H_{2^k}\right)\right] \\ &\quad -4\left[2^k - 2^{k+1}\left(H_{2^{k+1}} - H_{2^k}\right) + 4^k\left(H^{(2)}_{2^{k+1}} - H^{(2)}_{2^k}\right)\right] \\ &= 6\left(2^k\right)\left[H_{2^{k+1}} - H_{2^k}\right] - 4^{k+1}\left[H^{(2)}_{2^{k+1}} - H^{(2)}_{2^k}\right] - 2^{k+1}, \end{aligned}$$

where $H_n^{(m)} = \sum_{i=1}^{n} 1/i^m$, the nth harmonic number of order m. (We omit the superscript when m is 1.) To simplify this expression, we use the asymptotic approximations

$$H_n = \ln n + \gamma + O\left(\frac{1}{n}\right),$$

where γ is Euler's constant, and

$$H_n^{(2)} = \frac{\pi^2}{6} - \frac{1}{n} + O\left(\frac{1}{n^2}\right).$$

Using these we obtain

$$\sum_{i=0}^{2^k-1} Var\left[X_{L_k+i}\right] = (6\ln 2 - 4)2^k + O(1).$$

Then

$$\begin{aligned} s_{U_j}^2 &= \sum_{k=1}^{j} \sum_{i=0}^{2^k-1} Var\left[X_{L_k+i}\right] \\ &= (6\ln 2 - 4)\sum_{k=1}^{j} 2^k + O(j) \\ &= \Omega(U_j). \end{aligned}$$

Also, if $0 \le l \le 2^j - 1$,

$$\sum_{i=0}^{l} Var\left[X_{L_j+i}\right] = 4^{j+1}\left(\frac{1}{2^j+l+1} - \frac{1}{2^j}\right) + 6\left(2^j\right)\left(H_{2^j+l+1} - H_{2^j}\right) - 2l + O(1).$$

Thus for general $n = L_j + i$, where $0 \le i \le 2^j - 1$,

$$s_{L_j+i}^2 = s_{U_{j-1}}^2 + 4^{j+1}\left(\frac{1}{2^j+i+1} - \frac{1}{2^j}\right) + 6\left(2^j\right)\left(H_{2^j+i+1} - H_{2^j}\right) - 2i + O(j).$$

Since, in the case of binary search, $h_{U_j} = O(\ln U_j)$, there is a positive constant c such that $s_{U_j} = \Omega\left(\sqrt{U_j}\right) = \Omega\left(\exp\left(ch_{U_j}\right)\right)$. Then for $i = 0, ..., m_j - 1$,

$$s_{L_j+i} > s_{U_{j-1}}$$

$$= \Omega\left(\exp\left(c\left\lceil h_{L_j+i} - 1\right\rceil\right)\right)$$

$$= \Omega\left(\exp\left(ch_{L_j+i}\right)\right).$$

That is, $s_n = \Omega(\exp(ch_n))$, so $h_n = o(s_n)$, and the condition of Lemma 3.1 is satisfied. Writing $n = L_j + i$ and simplifying the above expression for $s^2_{L_j+i}$ gives

$$s_n^2 = n\left[\frac{6(1+f_n)\ln 2 - 6}{2^{f_n}} - 2 + \frac{4}{4^{f_n}}\right] + O(\ln n),$$

where

$$f_n \equiv \log_2 n - \lfloor \log_2 n \rfloor .$$

(The sequence $\{f_n\}_{n=1}^{\infty}$ is dense, but not uniformly dense, on the interval $[0,1)$, as discussed by Kuipers and Niederreiter 1974.) Then

$$\frac{C_n - n\log_2 n}{\sqrt{nA_n}} \xrightarrow{D} N(0,1),$$

where

$$A_n = \frac{6(1+f_n)\ln 2 - 6}{2^{f_n}} - 2 + \frac{4}{4^{f_n}}.$$

Figure 3.5.1 shows the graph of the analytic continuation of the function $A_n = A(f_n)$ to the real line, as f_n ranges over the interval $[0,1)$. The function attains its minimum value (approximately 0.1519119) at the point

$$\frac{W\left(-\frac{8}{3e^2}\right) + 2}{\ln 2} - 1 = 0.14146968...,$$

where $W(x)$ is the principal branch of the omega function. (See Fritsch et al. 1973.) It attains its maximum value (approximately 0.1744492) at

$$\frac{W\left(0, -\frac{8}{3e^2}\right) + 2}{\ln 2} - 1 = 0.70705937...,$$

where $W(0,x)$ is the zeroth branch of the omega function. Intuitively, we can think of A_n as an asymptotic average of the variances of $\{X_1, ..., X_n\}$. Thus A_n increases in n as long as $Var[X_n] > A_{n-1}$. This occurs while the numbers of external nodes on the two lowest levels of T_n are close and thus f_n is close to $\frac{1}{2}$. When one level of T_n has many more external nodes than the other, $Var[X_n]$ is small, and A_n tends to decrease. As each level of the tree fills up, and f_n takes on increasing values in the interval $[0, 1)$, A_n completes one cycle: it moves from $6\ln 2 - 4$ (approximately 0.1588831) down to its minimum (for that cycle), then up to its maximum (for that cycle) and finally back down to $6\ln 2 - 4$. With each new cycle, A_n more closely approximates its analytic continuation.

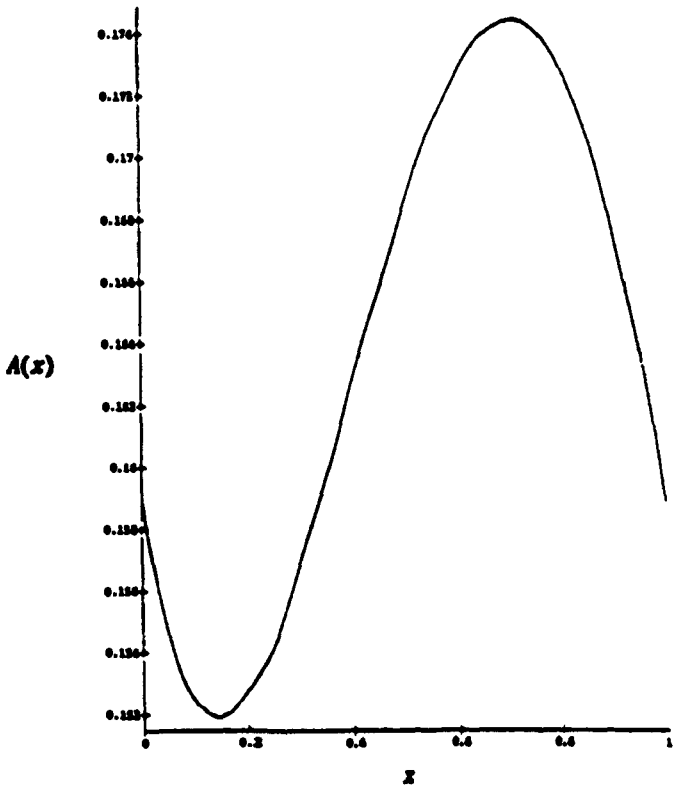

Figure 3.5.1. The analytic continuation of the function $A(f_n)$ onto the interval (0,1].

A second commonly used consistent strategy is linear search which, when searching an array of size l, always selects as a probe the data element of rank l (Linear Search from the Bottom) or the data element of rank 1 (Linear Search from the Top). For linear search it is known (see Gonnet and Baeza-Yates 1991) that

$$E[C_n] = \frac{n^2}{4} + O(n),$$

and

$$Var[C_n] \sim \frac{n^3}{36}.$$

Since $s_n^2 = \Omega(n^3)$, and $h_n = n - 1$, the sufficient condition holds for the linear search strategy; thus

$$\frac{C_n - n^2/4}{n^{3/2}} \xrightarrow{D} N\left(0, \frac{1}{36}\right).$$

3.6 Normality of Consistent Strategies

In this section we use the sufficient condition for normality to prove that all consistent strategies are normal. Let $\mathcal{S} = \{S_i\}_{i=1}^{\infty}$ be a consistent strategy represented by the decision trees $\mathcal{T} = \{T_i\}_{i=1}^{\infty}$, and let X_i, C_n, s_n^2, L_k, U_k, and m_k be as defined. We begin by establishing some properties of consistent strategies.

Property 1: *For each positive integer i, the decision tree T_i representing an algorithm from a consistent strategy has at least one external node on each unsaturated level.*

The right subtree of the tree in Figure 3.6.1, for example, has the following shape characteristic: its left sub-subtree is complete down to level two while its right sub-

subtree has neither internal nor external nodes on level two. The third level of the tree is therefore both unsaturated *and* bereft of external nodes. We will show that this tree cannot be part of a sequence of decision trees representing a consistent strategy.

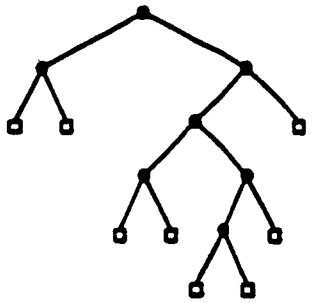

Figure 3.6.1. An extended decision tree that cannot represent an algorithm from a consistent strategy

Proof of Property 1: The first probe specified by S_i, $i = 1, 2, ...$, divides an array of size i into two subarrays. Let $g_S(i)$ denote the size of the smaller of the two subarrays and $g'_S(i)$ the size of the larger. (Naturally we define $g_S(0) = g'_S(0) = 0$.) For a cleaner notation, we suppress the subscript S. If T_i had an incomplete level with no external nodes, T_i would contain a subtree whose two sub-subtrees (left and right) had the following property: one sub-subtree would be complete down to a level λ, while the other would have neither internal nor external nodes at level λ. Since S_i is a consistent algorithm, this would imply that for some number $n' \leq i$, the size of the subtree,

$$g^{(\lambda)}(n') = 0, \text{ and } g^{(\lambda)}(g'(n')) > 0$$

(where $g^{(p)}(i) = g(g(g \cdots (i)))$, the function g composed p times, for a positive integer p). This is a contradiction, since g must be nondecreasing but $g'(n') < n'$. ■

Property 2: *For consistent strategies, the sequence* $\{m_k\}_{k=1}^{\infty}$ *is nondecreasing in* k, *so* $U_k/m_k \leq k+1$.

Proof of Property 2: Let g and g' be as defined. (Again we suppress the subscript S.) Then for each positive integer i, the larger subtree (rooted at level one) of T_i has the same shape as $T_{g'(i)}$, the decision tree for $S_{g'(i)}$. Since height is monotonic in size, the larger subtree of $T_{L_{k+1}}$—$T_{L_{k+1}}$ being the smallest decision tree in $\mathcal{T}$ having height $k+1$—has the shape of T_{L_k}, the smallest decision tree in $\mathcal{T}$ whose height is k. So $g'(L_{k+1}) = L_k$; similarly, $g'(U_{k+1}) = U_k$. Also, by the tree-growing property, if $i_2 > i_1$, then $g'(i_2) \geq g'(i_1)$, and $i_2 - g'(i_2) \geq i_1 - g'(i_1)$. Since for all i, $g'(i) < i$, this implies (taking $i_1 = g'(i)$ and $i_2 = i$) that

$$i - g'(i) \geq g'(i) - g'(g'(i)).$$

Then

$$\begin{aligned} m_{k+1} &= U_{k+1} - U_k \\ &= U_{k+1} - g'(U_{k+1}) \\ &\geq g'(U_{k+1}) - g'(g'(U_{k+1})) \\ &= U_k - U_{k-1} \\ &= m_k. \end{aligned}$$

Since the m_k's are nondecreasing in k,

$$\frac{U_k}{m_k} = \frac{1 + m_1 + m_2 + \cdots + m_k}{m_k} \leq k+1. \quad \blacksquare$$

We now present the main result of this chapter.

Theorem 3.1: *All consistent strategies are normal.*

Proof: If $\lim_{n\to\infty} g(n)/n = 1/2$, the strategy is similar to binary search: the rate of growth in h_n is $O(\ln n)$, while the rate of growth in s_n^2 is $\Omega(n)$, so the sufficient condition clearly holds. Now assume $\lim_{n\to\infty} g(n)/n = c$ for some constant $c \in [0, 1/2)$. Let h_n be the height and β_n the number of complete levels of a decision tree of size n. We first show that in this case, there are positive constants N_1 and c_1 such that $c_1 < 1$ and for $n > N_1$ we have

$$\beta_n \leq c_1 h_n.$$

If β_n is bounded above by some constant, there is nothing to prove, so assume that $\lim_{n\to\infty} \beta_n = \infty$. (The limit must exist, because $\{\beta_n\}_{n=1}^{\infty}$ is nondecreasing in n.) Since $\lim_{n\to\infty} g(n)/n = c < 1/2$, there is a positive constant $c_2 < 1/2$ and an integer N_2 such that for $n > N_2$, $g(n) < c_2 n$. That is, the smaller subtree of a decision tree of size $n > N_2$ has fewer than $c_2 n$ nodes. Further, each subtree of size $n > N_2$ also has this property; that is, its smaller sub-subtree has at most $c_2 n$ nodes. Consider the number of times we can successively apply the function g to the size of the tree (or subtree) and still obtain as a result an integer greater than N_2. As n grows, the number of such applications also grows, approaching a limit of infinity. Thus there is a constant integer c_3 such that for n large enough,

$$g^{(\beta_n - 1)}(n) < c_2^{\beta_n - c_3} \left(\frac{1}{2}\right)^{c_3} n,$$

where, by definition, $g^{(k)}(n) = g\left(g^{(k-1)}(n)\right)$. Since $g^{(\beta_n - 1)}(n) = 1$, we have

$$c_2^{\beta_n - c_3} \left(\frac{1}{2}\right)^{c_3} n > 1.$$

Taking logarithms on both sides of the inequality gives

$$(\beta_n - c_3)\ln c_2 - c_3 \ln 2 + \ln n > 0,$$

and, rearranging, we obtain

$$\beta_n < \frac{\ln n}{\ln\left(\frac{1}{c_2}\right)} + c_3\left(1 - \frac{\ln 2}{\ln\left(\frac{1}{c_2}\right)}\right).$$

Since $h_n \geq \lfloor \log_2 n \rfloor$, and $c_2 < 1/2$, this implies that there are positive constants N_1 and c_1 such that $c_1 < 1$ and for $n > N_1$,

$$\beta_n \leq c_1 h_n.$$

In the notation introduced at the beginning of the previous section, we have shown that there is a constant j_0 such that for $j > j_0$,

$$\beta_{ij} \leq c_1 j,$$

where β_{ij} is the number of complete levels of T_{L_j+i}, $i = 0, 1, ..., m_j - 1$. By Property 1, T_{L_j+i} has at least one external node on every level below β_{ij}, so for $i = 0, ..., m_j - 1$,

$$\begin{aligned} Var\left(X_{L_j+i}\right) &\geq \frac{1}{L_j + i + 1} \sum_{k=\beta_{ij}}^{j} \left[k - E\left(X_{L_j+i}\right)\right]^2 \\ &\geq \frac{1}{U_j} \sum_{k'=1}^{j'} \left[k' - \mu'\right]^2, \end{aligned}$$

where $j' = j - \beta_{ij} + 1$, and $\mu' = E\left(X_{L_j+i}\right) - \beta_{ij} + 1$. Since the sum of squares above may be minimized by setting $\mu' = (j' + 1)/2$, we have

$$Var\left(X_{L_j+i}\right) \geq \frac{1}{U_j} \sum_{k'=1}^{j'} \left[k' - \frac{j' + 1}{2}\right]^2$$

$$
\begin{aligned}
&= \frac{1}{U_j}\left\{\frac{j'(j'+1)(2j'+1)}{6} - \frac{j'(j'+1)^2}{2} + \frac{j'(j'+1)^2}{4}\right\} \\
&= \frac{1}{U_j}\left\{\frac{j'^3}{12} + O\left(j'^2\right)\right\} \\
&= \frac{\Omega(j^3)}{U_j}.
\end{aligned}
$$

Recalling that $j' \geq (1-c_1)j$ for sufficiently large j, we have

$$
Var\left(X_{L_j+i}\right) = \frac{\Omega(j^3)}{U_j}.
$$

Then

$$
\begin{aligned}
\sum_{i=0}^{m_j-1} Var\left(X_{L_j+i}\right) &= \sum_{i=0}^{m_j-1} \frac{\Omega(j^3)}{U_j} \\
&= \left(\frac{m_j}{U_j}\right)\Omega\left(j^3\right) \\
&\geq \left(\frac{1}{j+1}\right)\Omega\left(j^3\right) \\
&= \Omega\left(j^2\right),
\end{aligned}
$$

where we have used Property 2 to obtain the inequality. So

$$
\begin{aligned}
s_{U_j}^2 &= \sum_{k=1}^{j}\sum_{i=0}^{m_k-1} Var\left(X_{L_k+i}\right) \\
&= \sum_{k=1}^{j}\Omega\left(k^2\right) \\
&= \Omega\left(j^3\right).
\end{aligned}
$$

Thus $s_{U_j} = \Omega\left(j^{\frac{3}{2}}\right)$, which implies that $j = o\left(s_{L_j}\right)$, or $h_n = o(s_n)$. ■

3.7 Normality of Other Tree-growing Search Strategies

Properties 1 and 2 of consistent strategies, as is clear from their proofs, apply to all tree-growing strategies for which the position of each probe depends only on the length of the

data subarray being searched. Here we consider tree-growing strategies which, though not consistent, retain this characteristic and thus possess properties 1 and 2. The proof given above clearly works for strategies in which the limit $\lim_{n\to\infty} g(n)/n$ does not exist, as long as $\overline{\lim}_{n\to\infty} g(n)/n < 1/2$. One example of such a strategy is the following tree-growing implementation of Fibonaccian search:[8] Let the position of the probe to be selected in an array of size n be specified by

$$\xi_1(n) = \begin{cases} n, & \text{if } n \leq 3; \\ n - F_{k(n)-1} + 1, & \text{if } F_{k(n)+1} \leq n \leq F_{k(n)+1} + F_{k(n)-1} - 1; \\ F_{k(n)+1}, & \text{if } F_{k(n)+1} + F_{k(n)-1} \leq n \leq F_{k(n)+2} - 1; \end{cases}$$

where, for $n \geq 3$, $k(n)$ is such that

$$F_{k(n)+1} \leq n < F_{k(n)+2},$$

and F_i is the ith Fibonacci number.

Using this strategy, we "grow" the Fibonacci tree of size $F_{k+1} - 1$ into a Fibonacci tree of size $F_{k+2} - 1$ in the following way: We first add F_{k-1} nodes to the larger subtree of $T_{F_{k+1}-1}$, bringing the size of this subtree to $F_{k+1} - 1$. Then we add F_{k-2} nodes to the smaller subtree, bringing its size to $F_k - 1$. The new Fibonacci tree then has size $F_{k+1} - 1 + F_{k-1} + F_{k-2} = F_{k+2} - 1$. We may think of this tree-growing strategy as a "fast growing" implementation of Fibonaccian search, since the corresponding decision trees grow rapidly in height as n grows. Figure 3.7.1 shows the trees representing this strategy for $n = 13$ and $n = 14$.

[8]Not all implementations of Fibonaccian search have the tree-growing property (see Knuth 1973b, Section 6.2.1). We restrict our discussion to tree-growing implementations.

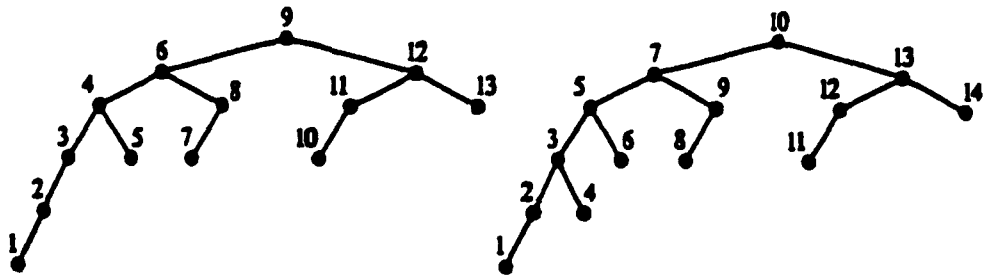

Figure 3.7.1. Decision trees of a fast-growing implementation of Fibonaccian search

With this strategy we have, using our previous notation,

$$\begin{aligned}\overline{\lim}_{n\to\infty}\frac{g(n)}{n} &= \lim_{k\to\infty}\frac{F_{k-1}-1}{F_{k+1}-1}\\ &= \lim_{k\to\infty}\frac{\left(1/\sqrt{5}\right)\phi^{k-1}}{\left(1/\sqrt{5}\right)\phi^{k+1}}\\ &= \frac{1}{\phi^2}\\ &= 0.381966...,\end{aligned}$$

where

$$\phi = \frac{1+\sqrt{5}}{2},$$

the golden ratio (as in Knuth 1973b). This strategy is not consistent, because

$$\begin{aligned}\underline{\lim}_{n\to\infty}\frac{g(n)}{n} &= \lim_{k\to\infty}\frac{F_{k-1}-1}{F_{k+1}+F_{k-1}-1}\\ &= \lim_{k\to\infty}\frac{\phi^{k-1}}{\phi^{k+1}+\phi^{k-1}}\\ &= \frac{1}{\phi^2+1}\\ &= 0.276393....\end{aligned}$$

Since we do, however, have $\overline{\lim}_{n\to\infty} g(n)/n < 1/2$, this strategy is normal.

The following proposition further expands the class of strategies for which we can prove normality.

Proposition 3.1: *Any search strategy S for which $\underline{\lim}_{n\to\infty} g(n)/n = k_1 \in (0, 1/2)$ is normal.*

Proof: Let $g'(n)$ be as defined. Since $\underline{\lim}_{n\to\infty} g(n)/n = k_1 > 0$, there are constants j_0 and k_2 such that $k_2 < 1$ and for $j > j_0$ we have $g'(U_j) \leq k_2 U_j$. Then for $j > j_0$,

$$g'^{(j-j_0)}(U_j) \leq k_2^{j-j_0} U_j.$$

For example, for $j = j_0 + 2$ we have

$$g'^{(2)}(U_{j_0+2}) = g'(g'(U_{j_0+2})) = g'(U_{j_0+1}) \leq k_2 U_{j_0+1} = k_2 g'(U_{j_0+2}) \leq k_2^2 U_{j_0+2}.$$

Since $g'^{(j-j_0)}(U_j) \geq 1$, this implies that

$$1 \leq k_2^{j-j_0} U_j,$$

and, taking logarithms gives

$$(j - j_0)\ln k_2 + \ln U_j \geq 0.$$

Rearranging, we obtain

$$j \leq \frac{\ln U_j}{\ln\left(\frac{1}{k_2}\right)} + j_0.$$

Thus there is some constant $k_3 > 0$ such that for $j > j_0$, $U_j > \exp(k_3 j)$. Using the facts that $s_n^2 = \Omega(n)$ for all tree-growing strategies,[9] and, writing $j = h_n$, that $U_{j-1} < n$, we

[9]It is easily proved by induction that the variance of C_n is minimized in the case of binary search, for which we have shown that $s_n^2 = \Omega(n)$.

have

$$s_n^2 = \Omega(n) = \Omega(\exp(k_3 h_n)),$$

so $h_n = o(s_n)$. ■

Another implementation of Fibonaccian search provides a practical example of a strategy for which Proposition 3.1 applies. Let the position of the probe to be selected in an array of size n be determined by the function

$$\xi_2(n) = \begin{cases} n, & \text{if } n \le 2; \\ F_{k(n)}, & \text{if } F_{k(n)+1} \le n \le F_{k(n)+1} + F_{k(n)-2} - 1; \\ n - F_{k(n)} + 1, & \text{if } F_{k(n)+1} + F_{k(n)-2} \le n \le F_{k(n)+2} - 1; \end{cases}$$

where $F_{k(n)}$ is as previously defined. This strategy is similar to the one determined by ξ_1 above, except we add nodes to the smaller subtree first, then to the larger—this is a slow-growing implementation of Fibonaccian search. Figure 3.7.2 shows its decision trees for $n = 13$ and $n = 14$.

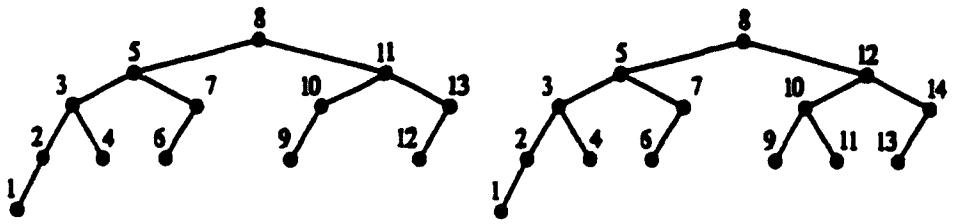

Figure 3.7.2. Decision trees of a slow-growing implementation of Fibonaccian search

With this strategy we have, using our previous notation,

$$\overline{\lim}_{n\to\infty} \frac{g(n)}{n} = \lim_{k\to\infty} \frac{F_k - 1}{F_{k+1} + F_{k-2} - 1}$$

$$= \lim_{k\to\infty} \frac{F_k}{2F_k}$$
$$= \frac{1}{2}.$$

Here, again, we do not have consistency, since

$$\begin{aligned}\lim_{n\to\infty} \frac{g(n)}{n} &= \lim_{k\to\infty} \frac{F_{k-1}-1}{F_{k+1}-1} \\ &= \lim_{k\to\infty} \frac{\left(1/\sqrt{5}\right)\phi^{k-1}}{\left(1/\sqrt{5}\right)\phi^{k+1}} \\ &= \frac{1}{\phi^2},\end{aligned}$$

but the normality of this Fibonaccian strategy follows from Proposition 3.1.

We have seen that all strategies for which either

$$\overline{\lim}_{n\to\infty} g(n)/n < 1/2$$

or

$$\underline{\lim}_{n\to\infty} g(n)/n > 0$$

are normal. One remaining case, the class of strategies for which $\overline{\lim}_{n\to\infty} g(n)/n = 1/2$ and $\underline{\lim}_{n\to\infty} g(n)/n = 0$, is more problematic. For a subset of these strategies, a modification of the proof for consistent strategies suffices: If there are positive constants N_1 and c_1 such that $c_1 < 1$ and for $n > N_1$,

$$\beta_n < c_1 h_n,$$

then the sufficient condition holds, as for the case of consistent strategies where the limit of $g(n)/n$ is less than one half. If for every number c in the interval $[0, 1)$ we have $\beta_n \geq ch_n$ infinitely often, the tree becomes nearly complete infinitely often. In

this case, we have alternating "stages" of growth in height in the sequence $\mathcal{T}$. The following argument shows that the number of nodes added between "shifts" of $g(n)/n$ from its upper limit to its lower limit, approaches infinity very quickly: For every ε in the interval $(0, 1/4)$, there are sequences of integers $\{n_{i\varepsilon}\}_{i=1}^{\infty}$ and $\{n'_{i\varepsilon}\}_{i=1}^{\infty}$ such that for each positive integer i we have $n_{i\varepsilon} < n'_{i\varepsilon} < n_{(i+1)\varepsilon}$, $g(n_{i\varepsilon})/n_{i\varepsilon} > \frac{1}{2} - \varepsilon$, and $g(n'_{i\varepsilon})/n'_{i\varepsilon} < \varepsilon$. Since by the tree-growing property $g(n'_{i\varepsilon}) > g(n_{i\varepsilon})$, we may write

$$\begin{aligned} n'_{i\varepsilon} &> \frac{g(n_{i\varepsilon})}{\varepsilon} \\ &> \frac{(1/2-\varepsilon)\, n_{i\varepsilon}}{\varepsilon}, \end{aligned}$$

for each i. Then since

$$\lim_{\varepsilon \to 0} \frac{1/2-\varepsilon}{\varepsilon} = \infty,$$

we have

$$\lim_{i \to \infty} \lim_{\varepsilon \to 0} \frac{n'_{i\varepsilon}}{n_{i\varepsilon}} = \infty.$$

It is also clear from the tree-growing property that

$$n_{(i+1)\varepsilon} \geq n'_{i\varepsilon}(2-2\varepsilon),$$

so the number of nodes added between shifts of $g(n)/n$ from its lower to its upper limit also approaches infinity. During stages in which $g(n)/n$ is less than $1/2$, $\mathcal{T}$ will resemble a sequence of trees representing a consistent strategy with $\lim_{n \to \infty} g(n)/n < 1/2$. If $g(n)/n$ remains close to $1/2$ for a large number of consecutive integers (as it must for the tree to fill out), the family $\mathcal{T}$ may contain subsequences of nearly complete trees. If the length of these subsequences approaches infinity as n approaches infinity,

the subsequences will resemble sequences of trees representing consistent strategies with $\lim_{n\to\infty} g(n)/n = 1/2$: that is, we will have $h_{U_j} = O(\ln U_j)$, so $U_j = \Omega(\exp(kj))$ for some $k > 0$. In each type of stage then, as the number of nodes added within a stage approaches infinity, we have $h_n = o(s_n)$. Intuition thus suggests that normality holds in the case of these "oscillating" strategies. Examples of such strategies, however, are difficult to concoct and unlikely to be used.

Chapter 4
Average-case Analysis of Multiple Quickselect

Quicksort, a sorting algorithm invented by Hoare (1962), is among the most efficient known methods of sorting an array of data values selected at random from a continuous distribution. Quicksort begins by selecting one data value (often the first value) from the array to use as a *pivot*. By comparing the pivot to each of the other values in the array, Quicksort places the pivot in its correct sorted position. In the process, Quicksort assigns all values less than the pivot to positions left of the pivot and all values greater than the pivot to positions right of the pivot. Thus the original array is *partitioned* into two smaller unsorted arrays, one on each side of the pivot. In recursive fashion, Quicksort then begins the partitioning process afresh with each of the smaller arrays.

Many have studied the distribution of the number of comparisons required by Quicksort. Régnier (1989), for example, used martingale theory to show that the number of

comparisons, when suitably standardized, has a distribution that converges to a limiting distribution. Rösler (1991) characterized this distribution as a fixed point of a contraction, and Hennequin (1987) computed its first five cumulants. In addition, Hennequin (1991) gave a general formula for the pth cumulant of this distribution (where p is an integer greater than one). Tan and Hadjicostas (1995) provided more information about the tail of the limiting distribution. All of these researchers assumed that the data values being sorted had been selected at random from a continuous probability distribution. Under this model of randomness, which we also adopt here, the ranks of the data elements form a random permutation of the integers 1 through n, the size of the data set. Eddy and Schervish (1995) examined the performance of Quicksort under alternative models of randomness, e.g., assuming the ranks of the data values follow a triangular distribution.

Mahmoud, Modarres, and Smythe (1995) analyzed the distribution of the number of comparisons required by Quickselect, a variant of Quicksort, to find one order statistic in a set of random values from a continuous distribution. Quickselect performs the same partitioning routine prescribed in Quicksort, but, seeking only the ith order statistic, partitions only subarrays containing the ith position. After each partition step, if the pivot is not the order statistic sought, Quickselect continues its search in the subarray to the left or right of the pivot—whichever contains the ith position—and abandons the other subarray. Multiple quickselect (MQS) is a variant of Quicksort modified to search for two or more order statistics at a time. Like Quickselect, MQS relies on the partitioning procedure found in Quicksort. Since more than one order statistic is sought, however, MQS specifies one *or both* of the subarrays created in a partition step as containing

desired statistics. Thus after completing a partition step, MQS may or may not abandon one fragment of the data array.

4.1 Operation of Multiple Quickselect

Operationally, MQS may be implemented as a procedure that receives four integral parameters F, L, B, and T, while globally accessing the data array $A[1..n]$ and the array $OS[1..p]$ containing the (sorted) ranks of the desired order statistics. The parameters F and L indicate the *first* and *last* positions in the portion of A to be searched; similarly B and T denote the *bottom* and *top* positions of the subarray of the array OS for whose elements MQS is to search within A. The initial call is

$$MQS(1, n, 1, p);$$

and the algorithm first goes through the usual partition[10] of $A[F..L]$ as in Quicksort (see, for example, Sedgewick 1988). The partitioning procedure selects a pivot and moves it to its correct position in the list, position k say, creating two random subarrays $A[F..k-1]$ and $A[k+1..L]$. Next MQS searches $OS[B..T]$ for the rank k of the pivot. If k is not found in $OS[B..T]$ but there is an integer r such that all elements of $OS[B..r]$ are less than k and all elements of $OS[r+1..T]$ exceed k, then MQS searches for $OS[B..r]$ within $A[F..k-1]$ and for $OS[r+1..T]$ within $A[k+1..L]$. These searches are accomplished by the two recursive calls

[10]We assume here that the partition procedure requires $n-1$ comparisons to partition an array of size n; a modification of the standard partitioning algorithm accomplishes this.

$$MQS(F, k-1, B, r);$$
$$MQS(k+1, L, r+1, T);$$

If, on the other hand, k is found within OS at position r, the algorithm announces its finding of one of the order statistics, stores its value $A[k]$, and proceeds with the calls

$$MQS(F, k-1, B, r-1);$$
$$MQS(k+1, L, r+1, T);$$

The recursive process continues until all elements of OS have been found. Figure 4.1.1 gives formal code for a Pascal-like implementation of MQS. (In this implementation, the arrays A, OS, and R—an array in which MQS stores the values of the order statistics after finding them—are accessed globally.)

Figure 4.1.1. A Pascal-like Implementation of MQS

```
procedure MQS(F, L, B, T: integer);

var k, Q: integer;

{OS contains the ranks of the p order statistics; R contains their values, once found.}

begin {MQS}

if B<T then

  begin {if}
  Partition(F, L, k);

{"Partition" returns k, the correct position of the pivot.}

  Search (k, Q);

{"Search" searches OS for k, returning a value of p if k exceeds the largest value in OS and a value of 0 if k is less than the smallest
value in OS; otherwise, if k is not in OS, Search returns the position of the highest value less than k in OS.}

  if Q=0 then MQS(F, k-1, B, T);
  else if OS[Q] = k then

{One order statistic has been found.}

    begin {if}
    R[Q]:=A[k];
    MQS(F, k-1, B, Q-1);
    MQS(k+1, L, Q+1, T);
    end {if}
  else
    begin {else}
    MQS(F, k-1, B, Q);
    MQS(k+1, L, Q+1, T);
    end {else}
  end {if}
end; {MQS}
```

The algorithm is particularly useful for computing L-estimates, a class of linear statistics discussed in detail by Serfling (1980, Chapter 8). An L-estimate is a function of the form

$$L_n = \sum_{i=1}^{n} c_i X_{(i)}.$$

Often many of the coefficients c_i are identical, and one may compute L_n without completely sorting the data array. If we wish to compute, for example, the α-trimmed mean, defined as

$$T_n = \frac{1}{n - 2\lfloor n\alpha \rfloor} \sum_{i=\lfloor n\alpha \rfloor + 1}^{n-\lfloor n\alpha \rfloor} X_{(i)},$$

we may use MQS to identify the two bounding order statistics $X_{(\lfloor n\alpha \rfloor + 1)}$ and $X_{(n-\lfloor n\alpha \rfloor)}$. In the process of selecting these, MQS will place all intermediate order statistics in the array positions $(\lfloor n\alpha \rfloor + 2), ..., (n - \lfloor n\alpha \rfloor - 1)$. We need not sort these intermediate values, since we seek only their average.

Chapter 1 provides additional examples of applications in which "all purpose" selection algorithms like MQS prove useful. In the next section, we calculate the average number of comparisons MQS uses in finding p order statistics in a data array of size n, assuming that p is fixed with respect to n. Our formula for the average number of comparisons involves the moments of $S_j^{(n)}$, the number of descendants of a node of rank j in a random binary search tree of size n. The distribution of this random variable and an expression for its moments are derived in Chapter 2. In Section 4.3, we discuss an example in which we apply a "weighted" variant of MQS to a data set.

4.2 The Average Number of Comparisons Performed by MQS

In this section we examine the number of comparisons MQS requires for finding p order statistics. Having noted the algorithm's versatility, we assume here that the ranks of the p order statistics have been selected at random without replacement from the integers $\{1, ..., n\}$. With this assumption, our average number of comparisons will represent a "grand" average *over all possible sets of p order statistics* as well as over all possible permutations of the ranks in the data array. Our method of analysis relies on the distribution of random "Quicksort trees"—binary trees whose subtrees correspond to the partition steps performed by Quicksort, the root of each subtree being the data value of the pivot in the corresponding partition. Figure 4.2.1 shows an example of a Quicksort tree in which $n = 15$, and the partition procedure selects the first element in $A[F..L]$ as the pivot.

The "MQS tree" is formed by "pruning" from the Quicksort tree all nodes corresponding to data values *not* used as pivots by MQS. In Figure 4.2.1, the MQS tree for the example is shown with solid edges, while the pruned nodes are joined to the tree with dashed edges. We use the Quicksort trees and MQS trees to analyze the number of comparisons consumed by MQS. (As in Chapter 2, we consider each node in the tree an ancestor of itself but not a descendant of itself.) By analogy with Quicksort, we see that if we use MQS to find the p order statistics ranked $j_1, j_2, ..., j_p$ in a data array, the total number of comparisons needed will be the same as the total number of descendants of all ancestors of $j_1, j_2, ..., j_p$ in the Quicksort tree corresponding to the data array, or the

total number of descendants—in the Quicksort tree—of *all* nodes in the MQS tree. By counting the descendants of all nodes in the MQS tree of Figure 4.2.1, we can see that, in this example, MQS uses a total of 38 comparisons to find the three order statistics.

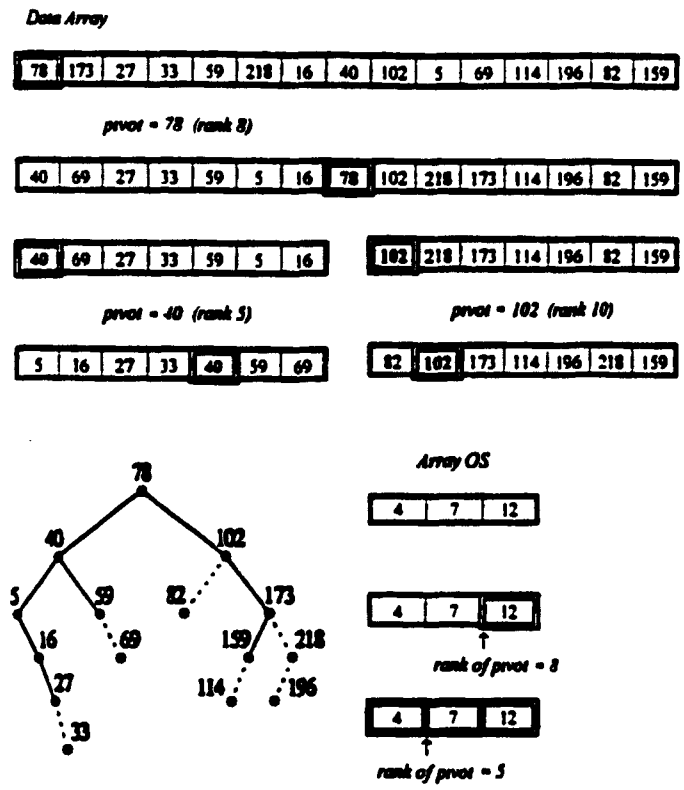

Figure 4.2.1. A data array in which MQS searches for three order statistics. The corresponding MQS tree (solid) includes all data elements that serve as pivots.

We assume our implementation of the Quicksort partition step ensures that the relative ranks of the data elements in a subarray of size n', created by the partitioning, form a random permutation of the integers $\{1, ..., n'\}$. It is a classical result (Régnier 1989) that the distribution of the number of comparisons required by Quicksort for a partition step in which the data value ranked j is the pivot is the same as the distribution of the number

of descendants of a node j in a binary search tree formed by successive insertion from a random permutation of the integers $\{1, ..., n\}$. We use this equivalence principle to prove Theorem 4.1. We also need the following lemma:

Lemma 4.1:

$$\sum_{i=1}^{p} \frac{(-1)^{i+1}}{i} \binom{p}{i} = H_p.$$

Proof: We prove this by induction on p. The basis ($p = 1$)is easily verified. Assuming the assertion true for p, we have, by Pascal's identity,

$$\begin{aligned} \sum_{i=1}^{p+1} \frac{(-1)^{i+1}}{i} \binom{p+1}{i} &= \sum_{i=1}^{p+1} \frac{(-1)^{i+1}}{i} \left[\binom{p}{i} + \binom{p}{i-1} \right] \\ &= \sum_{i=1}^{p+1} \frac{(-1)^{i+1}}{i} \binom{p}{i} + \sum_{i=1}^{p+1} \frac{(-1)^{i+1}}{p+1} \binom{p+1}{i}, \end{aligned}$$

where, in the second sum, we have used the classical identity

$$\frac{1}{n+1} \binom{n+1}{m+1} = \frac{1}{m+1} \binom{n}{m}.$$

So

$$\sum_{i=1}^{p+1} \frac{(-1)^{i+1}}{i} \binom{p+1}{i} = \sum_{i=1}^{p} \frac{(-1)^{i+1}}{i} \binom{p}{i} + \frac{1}{p+1} \left[\sum_{i=0}^{p+1} (-1)^{i+1} \binom{p+1}{i} + \binom{p+1}{0} \right].$$

By the induction hypothesis, the first sum is H_p, and by the binomial theorem, the second sum is $-(1-1)^{p+1} = 0$. Therefore,

$$\sum_{i=1}^{p+1} \frac{(-1)^{i+1}}{i} \binom{p+1}{i} = H_p + \frac{1}{p+1} = H_{p+1}. \quad \blacksquare$$

Suppose n data values are selected at random from a continuous distribution. Let p be a positive integer fixed with respect to n, and assume the ranks of p order statistics are selected at random without replacement from $\{1, ..., n\}$. Under these assumptions, we have the following theorem:

Theorem 4.1: *MQS requires an average of*

$$(2H_p + 1)\, n - 8p \ln n + O(1)$$

comparisons, where $H_p = \sum_{i=1}^{p} 1/i$, *the pth harmonic number.*

Proof: Let $C_p^{(n)}$ denote the number of comparisons *between data values* that MQS performs in searching for the p order statistics. Then

$$C_p^{(n)} = \sum_{j=1}^{n} S_j^{(n)} I_{jp}^{(n)}, \tag{4.1}$$

where $S_j^{(n)}$ denotes the number of descendants of node j in the Quicksort tree corresponding to the data array; and

$$I_{jp}^{(n)} = \begin{cases} 1, & \text{if } j \text{ is an ancestor of at least one of the } p \text{ order statistics;} \\ 0, & \text{otherwise.} \end{cases}$$

The number of descendants of the node of rank j is the number of comparisons Quicksort performs in the partition step in which j is the pivot—Quicksort compares the value of the root node to that of each of its descendants. But unless one of the desired order statistics lies in the subarray corresponding to the subtree rooted at j, MQS will not perform the partition step, and we will have $I_{jp}^{(n)} = 0$.

Since all the random variables in this chapter depend on n, the size of the data array, we suppress the superscript. We find $E[C_p]$ by conditioning on the value of S_j. First note that for $k = 0, \ldots, n-1$.

$$\begin{aligned} E[I_{jp} | S_j = k] &= P\{I_{jp} = 1 | S_j = k\} \\ &= P\{j \text{ is an ancestor of at least one of the } p \text{ order} \\ &\quad \text{statistics } | S_j = k\} \\ &= 1 - P\{j \text{ is not an ancestor of any of the } p \text{ order} \\ &\quad \text{statistics } | S_j = k\} \\ &= 1 - \frac{\binom{n-k-1}{p}}{\binom{n}{p}}, \end{aligned}$$

where, to obtain the last equality, we have used our assumption that the ranks of the p order statistics were selected at random without replacement from the integers $\{1, \ldots, n\}$. Then

$$\begin{aligned} E\left[\sum_{j=1}^{n} S_j I_{jp}\right] &= \sum_{j=1}^{n}\sum_{k=0}^{n-1} k\left[1 - \frac{\binom{n-k-1}{p}}{\binom{n}{p}}\right] P\{S_j = k\} \\ &= \sum_{j=1}^{n}\sum_{k=0}^{n-1} kP\{S_j = k\} \\ &\quad -\frac{1}{\binom{n}{p}}\sum_{j=1}^{n}\sum_{k=0}^{n-1} k\binom{n-k-1}{p} P\{S_j = k\}. \end{aligned} \tag{4.2}$$

Now

$$\sum_{j=1}^{n}\sum_{k=0}^{n-1} kP\{S_j = k\} = \sum_{j=1}^{n} E[S_j] = E[C_n], \tag{4.3}$$

where C_n is the number of comparisons Quicksort would require to sort the entire data array. (Note that we may consider Quicksort a special case of MQS, the case in which

all order statistics are to be computed.) Then from equations 4.1, 4.2, and 4.3,

$$E[C_p] = E[C_n] - \frac{1}{\prod_{i=0}^{p-1}(n-i)} \sum_{j=1}^{n} \sum_{k=0}^{n-1} k \left[\prod_{i=1}^{p} (n-k-i) \right] P\{S_j = k\}. \tag{4.4}$$

To simplify equation 4.4, we will use Pochhammer's symbol for the rising factorial:

$$\langle x \rangle_p = x(x+1)\cdots(x+p-1) = \sum_{r=1}^{p} \begin{bmatrix} p \\ r \end{bmatrix} x^r, \tag{4.5}$$

where $\begin{bmatrix} p \\ r \end{bmatrix}$ is the rth signless Stirling number of the first kind, of order p. Differentiating equation 4.5 gives

$$\langle x \rangle_p \sum_{j=0}^{p-1} \frac{1}{x+j} = \sum_{r=1}^{r} r \begin{bmatrix} p \\ r \end{bmatrix} x^{r-1}. \tag{4.6}$$

Rewriting equation 4.4 with this notation, we have

$$\begin{aligned} E[C_p] &= E[C_n] - \frac{1}{\langle n-p+1 \rangle_p} \sum_{j=1}^{n} \sum_{k=0}^{n-1} k \langle n-p-k \rangle_p P\{S_j = k\} \\ &= E[C_n] - \frac{1}{\langle n-p+1 \rangle_p} \sum_{j=1}^{n} \sum_{k=0}^{n-1} k \left[\sum_{r=1}^{p} \begin{bmatrix} p \\ r \end{bmatrix} (n-p-k)^r \right] P\{S_j = k\} \\ &= E[C_n] \\ &\quad - \frac{1}{\langle n-p+1 \rangle_p} \sum_{j=1}^{n} \sum_{k=0}^{n-1} k \left[\sum_{r=1}^{p} \begin{bmatrix} p \\ r \end{bmatrix} \sum_{t=0}^{r} (-1)^t \binom{r}{t} k^t (n-p)^{r-t} \right] P\{S_j = k\}, \end{aligned}$$

where we have used identity 4.5 and the binomial theorem. Isolating the terms containing the first and second moments of S_j, we obtain

$$\begin{aligned} E[C_p] &= E[C_n] \\ &\quad - \frac{1}{\langle n-p+1 \rangle_p} \sum_{j=1}^{n} \sum_{r=1}^{p} \begin{bmatrix} p \\ r \end{bmatrix} (n-p)^r E[S_j] && (4.7) \\ &\quad + \frac{1}{\langle n-p+1 \rangle_p} \sum_{j=1}^{n} \sum_{r=1}^{p} \begin{bmatrix} p \\ r \end{bmatrix} r (n-p)^{r-1} E\left[S_j^2\right] && (4.8) \\ &\quad + \frac{1}{\langle n-p+1 \rangle_p} \sum_{j=1}^{n} \sum_{r=2}^{p} \begin{bmatrix} p \\ r \end{bmatrix} \sum_{t=2}^{r} (-1)^{t+1} \binom{r}{t} (n-p)^{r-t} E\left[S_j^{t+1}\right]. && (4.9) \end{aligned}$$

We examine each of expressions 4.7, 4.8, and 4.9 separately. First, using equations 4.3 and 4.5 and the expression for $\sum_{j=1}^{n} E[S_j] = E[C_n]$ given in Section 2.2,

$$\begin{aligned}
-\frac{1}{\langle n-p+1\rangle_p}\sum_{j=1}^{n}\sum_{r=1}^{p}\left[\begin{matrix}p\\r\end{matrix}\right](n-p)^r E[S_j] &= -\frac{\sum_{r=1}^{p}\left[\begin{matrix}p\\r\end{matrix}\right](n-p)^r}{\langle n-p+1\rangle_p}\sum_{j=1}^{n}E[S_j] \\
&= -\frac{\langle n-p\rangle_p}{\langle n-p+1\rangle_p}E[C_n] \\
&= -\left(1-\frac{p}{n}\right)E[C_n] \\
&= -E[C_n]+2pH_n+O(1). \qquad (4.10)
\end{aligned}$$

Similarly, applying identity 4.6 in expression 4.8 and using the expression for $\sum_{j=1}^{n} E\left[S_j^2\right]$ derived in Section 2.2, we have

$$\begin{aligned}
\frac{1}{\langle n-p+1\rangle_p}\sum_{j=1}^{n}\sum_{r=1}^{p}\left[\begin{matrix}p\\r\end{matrix}\right]r(n-p)^{r-1}E\left[S_j^2\right] &= \sum_{j=1}^{n}E\left[S_j^2\right]\frac{\langle n-p\rangle_p}{\langle n-p+1\rangle_p} \\
&\quad\times\sum_{i=0}^{p-1}\frac{1}{n-p+i} \\
&= \frac{1}{n}\sum_{j=1}^{n}E\left[S_j^2\right]\left(1-\frac{p}{n}\right) \\
&\quad\times\sum_{i=0}^{p-1}\left[1+O\left(\frac{1}{n}\right)\right] \\
&= \frac{1}{n}\left[3n^2-10(n+1)H_n+17n\right] \\
&\quad\times\left(1-\frac{p}{n}\right)\left[p+O\left(\frac{1}{n}\right)\right] \\
&= 3pn-10pH_n+O(1). \qquad (4.11)
\end{aligned}$$

Rearranging expression 4.9 gives

$$\sum_{r=2}^{p}\left[\begin{matrix}p\\r\end{matrix}\right]\frac{(n-p)^r}{\langle n-p+1\rangle_p}\sum_{t=2}^{r}(-1)^{t+1}\binom{r}{t}\left(1-\frac{p}{n}\right)^{-t}\sum_{j=1}^{n}\frac{E\left[S_j^{t+1}\right]}{n^t}. \qquad (4.12)$$

To simplify this, we use the asymptotic approximation

$$\left(1-\frac{p}{n}\right)^{-t} = 1+O\left(\frac{1}{n}\right)$$

along with a result from Section 2.2: For fixed $t > 2$,

$$\sum_{j=1}^{n} \frac{E\left[S_j^{t+1}\right]}{n^t} = n\left(1 + \frac{2}{t}\right) + O(1).$$

When we substitute these into expression 4.12, the sum on t becomes (for fixed $r \geq 1$)

$$\begin{aligned}\sum_{t=2}^{r}(-1)^{t+1}\binom{r}{t}\left[n\left(1+\frac{2}{t}\right)+O(1)\right] &= n\left[\sum_{t=0}^{r}(-1)^{t+1}\binom{r}{t}+1-r\right] \\ &\quad +n\left[2\sum_{t=1}^{r}\frac{(-1)^{t+1}\binom{r}{t}}{t}-2r\right]+O(1) \\ &= n\left[0+1+2H_r-3r\right]+O(1),\end{aligned}$$

where we have used Lemma 4.1 to obtain the second equality. Next we observe that

$$\frac{(n-p)^p}{\langle n-p+1\rangle_p} = 1 + O\left(\frac{1}{n}\right)$$

and that for each $r < p$, we have

$$\begin{bmatrix}p\\r\end{bmatrix}\frac{(n-p)^r}{\langle n-p+1\rangle_p}(2H_r+1-3r)n = O(1).$$

Then we may rewrite expression 4.12 as

$$\frac{(n-p)^p}{\langle n-p+1\rangle_p}(2H_p+1-3p)n + \sum_{r=1}^{p-1}\begin{bmatrix}p\\r\end{bmatrix}\frac{(n-p)^r}{\langle n-p+1\rangle_p}(2H_r+1-3r)n + O(1),$$

and simplifying this gives

$$(2H_p+1-3p)n + O(1). \tag{4.13}$$

Finally, substituting expressions 4.10, 4.11, and 4.13 for 4.7, 4.8, and 4.9, respectively, gives

$$E[C_p] = (2H_p+1)n - 8pH_n + O(1).$$

This result is consistent with that of Mahmoud et al. (1995), who showed that

$$E[C_1] = 3n - 8H_n + 13 - \frac{8H_n}{n}.$$

When p exceeds one, however, an exact expression for the total number of comparisons involved will depend on the particular implementation of MQS: some comparisons will be needed *between* partition steps to determine which segments of the partitioned data array may be abandoned. To determine this, we find the relative rank of the final position of the pivot within the array OS; in the code of Figure 4.1, this is accomplished by the call

$$Search\,(OS, k, Q)\,.$$

The comparisons performed during the search of OS may not be the same type of comparisons performed within the partition steps; the data elements may be real numbers, for example, and the ranks of the order statistics integers. We show in Appendix A, however, that the average number of such comparisons is $O(1)$, assuming any reasonable search method is used. Thus the average number of comparisons MQS performs in selecting the p order statistics is

$$(2H_p + 1)n - 8p\ln n + O(1)\,. \quad \blacksquare$$

4.3 An Example

To illustrate the flexibility and performance of MQS, we used it to compute several sets of quantiles from family income data collected through the Current Population Survey (CPS).

The CPS is a monthly household survey sponsored by the Bureau of Labor Statistics and designed primarily for measuring unemployment levels and rates. Each year in March, supplementary questions related to personal and family income are added to the regular CPS questionnaire. Median incomes for various demographic groups are estimated from the March supplement data and published in the Census Bureau's P-60 series.

Each CPS sample family represents a large number of families in the total population and is therefore assigned a sample weight—an estimate of the number of families that the sample family represents. These weights reflect the family's probability of being selected for the CPS as well as the results of a series of ratio adjustments. The CPS sample design is state-based: families living in different states have different probabilities of selection, which vary from about 1/100 to 1/3000. Sample weights are also adjusted to be consistent with a set of independently derived population estimates for states and various demographic groups. A series of adjustment factors is applied to each weight, so the weight for each sample family may be unique. (See Appendix B for a more detailed discussion of CPS estimation procedures.)

The sum of the weights of all families in the sample is an estimate of the number of families in the population. A proper algorithm for selecting income quantiles must therefore take the sample weights into account. The weighted median income, for example, is not necessarily the median income of the sample but rather the sample estimate of the *population* median, defined as follows: Assume the data array A comprises n records, each consisting of a family income value $A[i]$ and a sample weight w_i, and suppose the array is sorted by income. The weighted median may be defined as $A[i_0]$, where i_0 is

an integer such that

$$\sum_{i=1}^{i_0} w_i > \frac{1}{2}\sum_{i=1}^{n} w_i$$

and

$$\sum_{i=1}^{i_0-1} w_i \leq \frac{1}{2}\sum_{i=1}^{n} w_i.$$

That is, half the total weight of the sample is assigned to sample families with incomes equal to or below the weighted median. A variant of MQS, incorporating a weight variable, may be used. The weighted version of MQS keeps track of the total weight—a sum of family weights—on each side of the pivot and selects "weighted" quantiles.

Table 4.1 shows the number of comparisons the weighted variant of MQS performed between data values. Note that weighted MQS consumed only about $6.3n$ comparisons, where n denotes the size of the data array, to select a set of five weighted quantiles: the quantiles 0.10, 0.25, 0.50, 0.75, and 0.90. Five runs of Quickselect, by contrast, performed $12.3n$ comparisons, nearly twice the number needed by MQS, to find the same values.

Theorem 4.1 shows that, if the five quantiles desired were randomly selected, MQS would perform approximately $5.6n$ comparisons, on average, in its search for the quantile values. Intuitively, it is clear that MQS needs more comparisons to find quantiles spaced far apart from one another in the data array than it needs to find quantiles spaced closer together: quantiles spaced closer together will, on average, have a larger number of common ancestors in the MQS tree. Thus it makes sense that the number of comparisons consumed in the search for the five quantiles 0.10, 0.25, 0.50, 0.75, and 0.90 should slightly exceed the average required for the case of randomly selected quantiles. When

the four additional income quantiles 0.01, 0.05, 0.95, and 0.99, were added to the set of desired statistics, MQS required approximately $6.7n$ comparisons—about the same number predicted theoretically for randomly selected quantiles—to find the new set of nine quantiles.

MQS may be accelerated by a simple modification: the insertion of imaginary pivots or *dividers*. In the case of the CPS income data, the dividers may also be used to compute estimates of the numbers of families in specific income ranges. A set D comprising j dividers in sorted order is inserted as follows: the median of D is selected as the first "pivot" and used to partition the data array A into right and left subarrays, A_{left} and A_{right}, say. (Elements of D are not inserted into the data set.) By removing the median from D, we break D into two sublists D_{left} and D_{right}. The dividing algorithm is then applied recursively to (A_{left}, D_{left}) and (A_{right}, D_{right}).

Table 4.1. Comparisons Needed by MQS
1993 CPS Income Data (n = 65,582)

	Number of Comparisons	Number of Comparisons Divided by n	Approximate Average Number of Comparisons (from formula) Divided by n
Finding the 5 quantiles 0.10, 0.25, 0.50, 0.75, and 0.90			
5 runs of Quickselect, no dividers	808,568	12.3	5.6
MQS, no dividers	412,026	6.3	5.6
MQS, 5 tight dividers	390,110	5.9	5.6
MQS, 5 loose dividers	374,003	5.7	5.6
MQS, 10 loose dividers	347,989	5.3	5.6
Finding the 9 quantiles 0.01, 0.05, 0.10, 0.25, 0.50, 0.75, 0.90, 0.95, and 0.99			
MQS, no dividers	437,927	6.7	6.7
MQS, 10 loose dividers	389,715	5.9	6.7

The five dividers were placed at 10,000, 20,000, 30,000, 40,000, and 50,000; the ten dividers were placed at 10,000 through 55,000, at increments of 5,000.

If we are using the dividers merely as a means of accelerating MQS, the data values that tie with the divider may be left on either side of the data array. In this case we call the dividers "loose" dividers. Inserting a loose divider requires only one standard Quicksort partition step with the divider value serving as the pivot. The partition step places all data values larger than the divider on one side of the array and all values less than the divider on the other side. Data values that tie with the divider are simply left in their original positions. If estimates of the numbers of families in various income ranges are desired, we must insert the dividers in such a way that the data records of all families whose incomes tie with a divider are placed on the same side of the divider. The partition step used for inserting these "tight" dividers must be modified to check for ties and place them on a particular side of the data array, thus consuming some additional comparisons.

Appropriate divider values for a particular application depend on both the user's prior knowledge about the data set and the information the user seeks to obtain from the set. If little or no prior information is available, a sample of data values may be selected and its quantiles used as loose dividers. In most practical applications, however, the user's prior knowledge of the distribution of the data values is sufficient to suggest a suitable set of dividers. If the number of values falling within particular ranges must also be computed, as is often the case when the data represent an income distribution, predefined tight dividers must be used.

Table 4.1 shows results of several runs of MQS, with dividers, on the CPS family income data set. Inserting dividers at constant increments improved the performance of MQS on this data set, even when tight dividers were used, providing estimates of

the numbers of families in various income ranges (the ranges between the dividers) at virtually no additional cost.

It should be noted that survey income data are subject to "heaping"—many respondents round their incomes to the nearest hundred or thousand, creating spikes in the sample income distribution. Heaping improves the performance of the Quicksort partitioning process: when partitioning an array of identical values, the partition step breaks the array in the middle.

Chapter 5
Avenues of Future Research

5.1 Problems on Tree-growing Search Strategies

5.1.1 More Normal Tree-growing Strategies

In addition to the consistent strategies discussed in Chapter 3, many other tree-growing strategies are normal. Intuitively, if a strategy corresponds to a set of decision trees $\mathcal{T} = \{T_i\}_{i=1}^{\infty}$ for which the height of T_i grows at a steady rate as i grows, the strategy is likely to be normal. In fact, we can prove the normality of all search strategies S whose corresponding decision trees $\mathcal{T}$ satisfy one of the two sets of conditions specified by the following two propositions.

Proposition 5.1: *Let S be a tree-growing strategy whose corresponding decision trees $\mathcal{T}$ satisfy the following conditions:*

1. $m_k = O(k^{1+\varepsilon})$ *for some* ε *in the interval* $(0, 1)$; *and*

2. $U_k/m_k = O\left(k^{1+\delta}\right)$ *for some* δ *in the interval* $(0, 1)$; *and*

3. *the tree* T_i *is complete down to a certain level; below this level,* T_i *has at least one external node on every* cth *level for some constant* c.

Then S *is normal.*

Proof: Let k_0 and c_1 be such that, for $k \geq k_0$, we have $m_k/k^{1+\varepsilon} \leq c_1$. For $k \geq k_0$, let M_k be the number of nodes inserted between nodes labeled U_{k_0} and U_k. Then

$$\begin{aligned} M_k &\leq \sum_{i=k_0}^{k} c_1 i^2 \\ &\leq c_2 k^3, \end{aligned}$$

for some positive constant c_2. Let h_{M_k} denote the height of a complete tree with M_k nodes. Then

$$h_{M_k} \leq \left\lfloor \log_2\left(c_2 k^3\right) \right\rfloor = \left(\frac{3}{\ln 2}\right) \ln k + O(1),$$

and there are constants c_1' and k_0' such that, for $k > k_0'$,

$$h_{M_k} \leq c_1' \ln k.$$

So for $k > k_0'$, the tree T_{U_k} can only be complete down to level $k_0' + c_1' \ln k$. Then T_{U_k} has at least one external node on every cth level below level $k_0' + c_1' \ln k$. For each positive integer j, let $j' = \lfloor j/c \rfloor$. Then for $j' > k_0'$ and $i = 0, 1, \ldots, m_j - 1$, we have

$$Var\left[X_{L_j+i}\right] \geq \frac{1}{L_j + i + 1} \left\{ \sum_{k=k_0'}^{j'} \left[ck - E\left(X_{L_j+i}\right)\right]^2 - c_1' j^2 \ln j \right\}$$

$$\geq \frac{c^2}{U_j}\left\{\sum_{k''=1}^{j''}[k''-\mu']^2\right\} - \frac{c_1' j^2 \ln j}{U_j}.$$

where $j'' = j' - k_0' + 1$ and

$$\mu' = \frac{E\left[X_{L_j+i}\right] - k_0' + 1}{c}.$$

Following the argument used for consistent strategies in Chapter 3, we note that the sum of squares

$$\sum_{k''=1}^{j''}[k''-\mu']^2$$

may be minimized by setting $\mu' = (j''+1)/2$. Then

$$\begin{aligned} Var\left[X_{L_j+i}\right] &\geq \frac{c^2}{U_j}\left\{\sum_{k''=1}^{j''}\left[k''-\frac{j''+1}{2}\right]^2\right\} - \frac{c_1' j^2 \ln j}{U_j} \\ &= \frac{c^2}{U_j}\left\{\frac{j''^3}{12} + O\left(j^2 \ln j\right)\right\}. \end{aligned}$$

Recalling that $j'' = \lfloor j/c \rfloor - k_0 + 1$, there is a positive constant c_1'' such that, for sufficiently large j, we have $j'' > c_1'' j$. Thus

$$Var\left[X_{L_j+i}\right] = \frac{\Omega(j^3)}{U_j}.$$

Then

$$\begin{aligned} \sum_{i=0}^{m_j-1} Var\left[X_{L_j+i}\right] &= \sum_{i=0}^{m_j-1}\frac{\Omega(j^3)}{U_j} \\ &= \frac{\Omega(j^3)}{U_j/m_j} \\ &= \frac{\Omega(j^3)}{O(j^{1+\delta})} \\ &= \Omega\left(j^{1+\epsilon}\right), \end{aligned}$$

where $\epsilon = 1 - \delta$, and we have used condition (2) to obtain the third equality. So

$$s_{U_j}^2 = \sum_{k=1}^{j}\sum_{i=0}^{m_j-1} Var\left[X_{L_j+i}\right] = \Omega\left(j^{2+\epsilon}\right).$$

and the condition of Lemma 3.1 holds. ■

Proposition 5.1 applies to strategies whose corresponding decision trees grow quickly and steadily in height. It is clear, for example, that Alternating Linear Search satisfies the conditions of the proposition and is therefore normal.

Proposition 5.2: *Let S be a tree-growing strategy whose corresponding decision trees T satisfy the following conditions:*

1. $m_k = \Omega(k^{1+\epsilon})$ *for some* $\epsilon > 0$; *and*
2. $Var(X_i) = \Omega(1)$.

Then S is normal.

Proof: In this case, we have

$$\begin{aligned} s_{U_j}^2 &= \sum_{k=1}^{j} \sum_{i=0}^{m_k-1} Var[X_{L_k+i}] \\ &= \sum_{k=1}^{j} \sum_{i=0}^{m_k-1} \Omega(1) \\ &= \sum_{k=1}^{j} \Omega\left(k^{1+\epsilon}\right) \\ &= \Omega\left(j^{2+\epsilon}\right). \end{aligned}$$

For $i = 0, \ldots, m_j - 1$,

$$s_{L_j+i} > s_{U_{j-1}} = \Omega\left(j^{1+\epsilon}\right),$$

so the condition of Lemma 3.1 is satisfied. ■

Proposition 5.2 applies to strategies whose decision trees grow slowly in height but never approach completeness. Consider, for example, a strategy that selects the probe p_{ijk} according to the function

$$p_{ijk} = \begin{cases} \lfloor l_{ijk}/3 \rfloor , & j \text{ even;} \\ \lfloor 2l_{ijk}/3 \rfloor , & j \text{ odd.} \end{cases}$$

For this strategy, it is easy to show that the height of the decision tree of size n is $O(\ln n)$ as n approaches infinity, so clearly $m_k = \Omega(k^2)$. In each set of siblings in these decision trees, one sibling has roughly twice as many descendants as the other (provided neither are leaves). Since every subtree of the decision trees is lop-sided, the variances of the X_i's cannot approach zero, so Proposition 5.2 applies. We may also see that this strategy is normal by simply noting that the operation of taking mirror images of trees—or of subtrees within trees—does not affect normality, since the number of external nodes on each level of a tree remains the same when subtrees are "flipped" from side to side. Every strategy that is consistent up to the operation of taking mirror images of subtrees is therefore normal. The "rightmost tape-optimal" strategies developed by Hu and Wachs (1987) provide interesting examples of this type of strategy.

The conditions of Propositions 5.1 and 5.2 are stated as properties of the sequence of decision trees $\mathcal{T}$. Future research may be directed toward restating them as properties of the corresponding search strategy S.

5.1.2 Randomized Search Strategies

We can show, however, that not all tree-growing strategies are normal. Consider, for example, a strategy whose sequence of decision trees is identical to the sequence of random binary search trees grown by successive insertion of the ranks of the data values in the array. Each key inserted into the data array would carry a label identifying its order of appearance—the first key would carry a label of 1, the second a label of 2, and so forth. When searching a given subarray, the algorithm would simply probe the element with the lowest "order of appearance" value of all elements in that subarray. In the random permutation of Figure 2.1, for example, the number 3 would carry an order of appearance value of 1, so that each new integer to be inserted into the array would first be compared to the number 3. Similarly, the number 7 would carry an order of appearance value of 2; each new integer greater than 3 would next be compared to the number 7, and so forth. The algorithm would thus generate a sequence of decision trees identical to the one shown in Figure 2.1. The process of finding the next probe, of course, would require more searching—and thus additional comparisons—but the number of comparisons performed *between data values* would have the same distribution as the sum of the depths of all internal nodes in the random binary search tree. As noted in Section 4.2, this is also the distribution of the number of comparisons Quicksort performs when sorting a set of data values whose ranks form a random permutation. Régnier (1989) has shown that this number of comparisons has a limiting distribution that is not normal. The condition of Lemma 3.1 does not apply, since the X_i's in this case are not independent and the height of each decision tree is a random variable.

A modification of this strategy renders it normal but robs it of its tree-growing prop-

erty: Suppose that, when searching a given subarray, we simply select a probe at random, giving each element of the subarray an equal probability of being selected as the probe. In this strategy, the probe sequence used in the search for the correct position of each key is completely independent of the probe sequences used for the other keys. That is, we have introduced double randomness: the final position of the new key is random, and so is the tree branch "grown" by the selected probe sequence. Suppose, for example, that the algorithm inserts a new key into an array of eight keys, and the value of the new key lies between the values of the fourth and fifth keys in the existing array. The sequence of randomly selected probes in this case might be $\{2, 7, 4, 5\}$. Then if the next key to be inserted into the new array (of nine keys) has a value between the first and second values in the existing array, the algorithm might randomly generate the probe sequence $\{6, 4, 3, 1, 2\}$ or perhaps the sequence $\{9, 2, 1\}$. The new probe sequence need not be similar to the one used for inserting the previous key.

Since this algorithm selects the probe sequences randomly and independently, the distribution of the number of comparisons consumed by the search for the correct position of the $(i+1)$st key is the same as that of U_i, the number of comparisons performed during an unsuccessful search in a random binary search tree of size i. Then the number of comparisons required for sorting n keys by this insertion sort strategy is given by

$$R_n = \sum_{i=1}^{n-1} U_i,$$

and the U_i's are independent. To show the asymptotic normality of R_n, we use the Lyapunov central limit theorem, as stated by Mahmoud (1992, p. 41):

Lyapunov's Theorem: *Let $X_1, X_2, X_3, \ldots$ be a sequence of independent random variables. Let $E[X_n] = \mu_n$, $Var[X_n] = \sigma_n^2$, and $E\left[|X_n - \mu_n|^3\right] = \beta_n$ exist for each n ($\sigma_n \neq 0$ for at least one value of n). Furthermore, let*

$$A_n = \left(\sum_{i=1}^{n} \beta_i\right)^{1/3},$$

$$B_n = \left(\sum_{i=1}^{n} \sigma_i^2\right)^{1/2}.$$

If $\lim_{n\to\infty} A_n/B_n = 0$, the random variable $Y_n = \left(\sum_{i=1}^{n} (X_i - \mu_i)\right)/B_n$ converges in distribution to $N(0,1)$.

We first show that

$$\beta_n = E\left[|U_n - E[U_n]|^3\right] \leq 8H_{n+1}^3 + O\left(\ln^3(\ln n)\right).$$

The distribution of U_n, as stated in Section 2.1.3, is given by

$$P\{U_n = k\} = \frac{2^k}{(n+1)!}\begin{bmatrix} n \\ k \end{bmatrix},$$

for $k = 1, \ldots, n$. Its mean and variance are

$$\begin{aligned} E[U_n] &= 2(H_{n+1} - 1) \\ &\sim 2\ln n, \end{aligned}$$

and

$$\begin{aligned} Var[U_n] &= 2H_{n+1} - 4H_{n+1}^{(2)} + 2 \\ &\sim 2\ln n. \end{aligned}$$

Thus

$$\begin{aligned} E[R_n] &= \sum_{i=1}^{n-1} E[U_i] \\ &= 2\sum_{i=1}^{n-1} H_{n+1} - n \\ &\sim 2n\ln n, \end{aligned}$$

and, by the independence of the U_i's,

$$\begin{aligned} Var[R_n] &= \sum_{i=1}^{n-1} Var[U_i] \\ &= 2\sum_{i=1}^{n-1} H_{i+1} - 4\sum_{i=1}^{n-1} H_{i+1}^{(2)} + 2n \\ &\sim 2n\ln n. \end{aligned}$$

We show in Appendix A that

$$\sum_{k=1}^{n} k^3 \frac{2^k}{(n+1)!}\begin{bmatrix} n \\ k \end{bmatrix} = 8H_{n+1}^3 + O\left(\ln^2 n\right).$$

Thus

$$\begin{aligned} \beta_n &= \sum_{k=1}^{n} |k - E[U_n]|^3 \frac{2^k}{(n+1)!}\begin{bmatrix} n \\ k \end{bmatrix} \\ &< \sum_{k=1}^{n} [\max(k, E[U_n])]^3 \frac{2^k}{(n+1)!}\begin{bmatrix} n \\ k \end{bmatrix} \\ &< \sum_{k=1}^{n} \left[k^3 + (E[U_n])^3\right] \frac{2^k}{(n+1)!}\begin{bmatrix} n \\ k \end{bmatrix} \end{aligned}$$

$$
\begin{aligned}
&= \sum_{k=1}^{n} k^3 \frac{2^k}{(n+1)!} \begin{bmatrix} n \\ k \end{bmatrix} + (E[U_n])^3 \\
&= 8H_{n+1}^3 + [2(H_{n+1} - 1)]^3 + O\left(\ln^2 n\right) \\
&= 16H_{n+1}^3 + O\left(\ln^2 n\right).
\end{aligned}
$$

Now it is clear that

$$
\sum_{i=1}^{n} H_i^3 \leq nH_n^3 = O\left(n \ln^3 n\right).
$$

So with A_n and B_n as defined in Lyapunov's theorem, we have

$$
\begin{aligned}
\lim_{n\to\infty} \frac{A_n}{B_n} &\leq \lim_{n\to\infty} \frac{\left(16 \sum_{i=1}^{n} H_{i+1}^3 + O\left(\ln^2 n\right)\right)^{1/3}}{\left(2 \sum_{i=1}^{n} \ln i + O(n)\right)^{1/2}} \\
&= \lim_{n\to\infty} \frac{\left(O\left(n \ln^3 n\right)\right)^{1/3}}{(2n \ln n)^{1/2}} \\
&= 0.
\end{aligned}
$$

So

$$
\frac{R_n - 2n \ln n}{\sqrt{2n \ln n}} \xrightarrow{\mathcal{D}} N(0,1).
$$

Section 3.7 provided examples of normal tree growing strategies that are not consistent. In this section, we have discussed an example of a non-normal tree-growing strategy and a non-tree-growing normal strategy. Thus we have now seen at least one example of each possible type of strategy shown in Figure 3.1.

5.2 Problems on Analysis of MQS

The average-case analysis of MQS presented in Chapter 4 has been extended by Prodinger (1996), who calculated the average number of comparisons MQS uses in searching for a specific set of order statistics. This average, of course, depends on the ranks of the order

statistics sought. Remaining problems in the analysis of MQS include the calculation of the higher moments and the probability distribution of C_p.

Using the notation and method from the average-case analysis, we begin the calculation of the variance with

$$Var[C_p] = E\left[C_p^2\right] - E^2[C_p].$$

We know that $E^2[C_p] \sim n^2(1+2H_p)^2$, so we must find $E\left[C_p^2\right]$. Using our previous notation,

$$\begin{aligned} E\left[C_p^2\right] &= E\left[\left(\sum_{j=1}^{n} S_j I_{jp}\right)^2\right] \\ &= \sum_{j=1}^{n} E\left[S_j^2 I_{jp}^2\right] + 2\sum_{j=1}^{n-1}\sum_{j'=j+1}^{n} E\left[S_j S_{j'} I_{jp} I_{j'p}\right]. \end{aligned}$$

We can find the leading term of $\sum_{j=1}^{n} E\left[S_j^2 I_{jp}^2\right]$ using the following asymptotic approximation, derived in Section 2.2, for a fixed integer $t > 0$:

$$\frac{\sum_{j=1}^{n} E\left[S_j^{t+2}\right]}{n^t} = n^2\left(1+\frac{2}{t+1}\right) + O(n).$$

Then from the development shown in Section 4.2,

$$\begin{aligned} \sum_{j=1}^{n} E\left[S_j^2 I_{jp}^2\right] &= \sum_{j=1}^{n}\sum_{k=0}^{n-1} E\left[S_j^2 I_{jp} | S_j = k\right] P\{S_j = k\} \\ &= \sum_{j=1}^{n}\sum_{k=0}^{n-1} k^2\left[1 - \frac{\binom{n-k-1}{p}}{\binom{n}{p}}\right] P\{S_j = k\}. \end{aligned}$$

We proceed as in our calculation of $E[C_p]$:

$$\begin{aligned} \sum_{j=1}^{n} E\left[S_j^2 I_{jp}^2\right] &= \sum_{j=1}^{n}\sum_{k=0}^{n-1} k^2 P\{S_j = k\} \\ &\quad - \frac{1}{\langle n-p+1\rangle_p}\sum_{j=1}^{n}\sum_{k=0}^{n-1} k^2\left[\sum_{r=1}^{p}\left[{p \atop r}\right]\sum_{t=0}^{r}(-1)^t\binom{r}{t}k^t(n-p)^{r-t}\right] \end{aligned}$$

$$
\begin{aligned}
&\times P\{S_j = k\} \\
&= \sum_{j=1}^{n} E\left[S_j^2\right] - \left(1 - \frac{p}{n}\right) \sum_{j=1}^{n} E\left[S_j^2\right] \\
&\quad + \sum_{r=1}^{p} \begin{bmatrix} p \\ r \end{bmatrix} \frac{(n-p)^r}{\langle n-p+1 \rangle_p} \sum_{t=1}^{r} (-1)^{t+1} \binom{r}{t} \left(1 - \frac{p}{n}\right)^{-t} \sum_{j=1}^{n} \frac{E\left[S_j^{t+2}\right]}{n^t} \\
&= \frac{p}{n} \sum_{j=1}^{n} E\left[S_j^2\right] \\
&\quad + \sum_{r=1}^{p} \begin{bmatrix} p \\ r \end{bmatrix} \frac{(n-p)^r}{\langle n-p+1 \rangle_p} \sum_{t=1}^{r} (-1)^{t+1} \binom{r}{t} \left[n^2 \left(1 + \frac{2}{t+1}\right) + O(n)\right] \\
&= \frac{(n-p)^p n^2}{\langle n-p+1 \rangle_p} \left[\sum_{t=1}^{p} (-1)^{t+1} \binom{p}{t} + 2 \sum_{t=1}^{p} (-1)^{t+1} \frac{\binom{p}{t}}{t+1}\right] \\
&\quad + \sum_{r=1}^{p-1} \begin{bmatrix} p \\ r \end{bmatrix} \frac{(n-p)^r}{\langle n-p+1 \rangle_p} \sum_{t=1}^{r} (-1)^{t+1} \binom{r}{t} n^2 \left(1 + \frac{2}{t+1}\right) + O(n).
\end{aligned}
$$

Observing that, for $r < p$,

$$
\frac{(n-p)^r n^2}{\langle n-p+1 \rangle_p} = O(n),
$$

we have

$$
\begin{aligned}
\sum_{j=1}^{n} E\left[S_j^2 I_{jp}^2\right] &= n^2 \left[1 - 2\left\{\sum_{t=0}^{p} (-1)^t \frac{\binom{p}{t}}{t+1} - 1\right\}\right] + O(n) \\
&= n^2 \left[1 - 2\left(\frac{1}{p+1} - 1\right)\right] + O(n) \\
&= \left(3 - \frac{2}{p+1}\right) n^2 + O(n),
\end{aligned}
$$

where we have used the identity

$$
\sum_{t=0}^{p} (-1)^t \frac{\binom{p}{t}}{t+1} = \frac{1}{p+1}
$$

to obtain the second equality. To prove this identity, we simply note that, by the binomial theorem,

$$
\sum_{t=0}^{p} \int_0^1 (-1)^t \binom{p}{t} x^t dx = \int_0^1 (1-x)^p dx = \frac{1}{p+1}.
$$

Next we must find an expression for $E[S_j S_{j'} I_{jp} I_{j'p}]$ when $j < j'$. Let T_j be the subtree rooted at j. We will use the symbols $\perp$ and $\nearrow$ to mean, respectively, "is disjoint from" and "is an ancestor of." Thus for example, if the two subtrees T_j and $T_{j'}$ are disjoint, we write $T_j \perp T_{j'}$, and if j is an ancestor of j', we will write $j \nearrow j'$. With this notation we have

$$\begin{aligned}
E[S_j S_{j'} I_{jp} I_{j'p}] &= \sum_{k=0}^{n-1}\sum_{k'=0}^{n-1} kk' E[I_{jp} I_{j'p} | S_j = k, S_{j'} = k']\, P\{S_j = k, S_{j'} = k'\} \\
&= \sum_{k=1}^{n-1}\sum_{k'=0}^{k-1} kk' E[I_{jp} I_{j'p} | S_j = k, S_{j'} = k', j \nearrow j'] \\
&\quad \times P\{S_j = k, S_{j'} = k', j \nearrow j'\} \\
&\quad + \sum_{k'=1}^{n-1}\sum_{k=0}^{k'-1} kk' E[I_{jp} I_{j'p} | S_j = k, S_{j'} = k', j' \nearrow j] \\
&\quad \times P\{S_j = k, S_{j'} = k', j' \nearrow j\} \\
&\quad + \sum_{k=0}^{n-3}\sum_{k'=0}^{n-k-3} kk' E[I_{jp} I_{j'p} | S_j = k, S_{j'} = k', T_j \perp T_{j'}] \\
&\quad \times P\{S_j = k, S_{j'} = k', T_j \perp T_{j'}\}.
\end{aligned}$$

The first two terms in the above expression are clearly equal. So when the ranks of the p order statistics are selected at random, without replacement, from the integers $\{1, \ldots, n\}$, we have

$$\begin{aligned}
E[S_j S_{j'} I_{jp} I_{j'p}] &= 2\sum_{k=1}^{n-1}\sum_{k'=0}^{k-1} kk' \left[1 - \frac{\binom{n-k-1}{p}}{\binom{n}{p}}\right] P\{S_j = k, S_{j'} = k', j \nearrow j'\} \\
&\quad + \sum_{k=0}^{n-3}\sum_{k'=0}^{n-k-3} kk' \left[1 - \frac{\binom{n-k-k'-2}{p}}{\binom{n}{p}}\right] P\{S_j = k, S_{j'} = k', T_j \perp T_{j'}\}.
\end{aligned}$$

To compute these sums, we need the joint distribution of $S_j^{(n)}$ and $S_{j'}^{(n)}$, for each integer

j and $j' \neq j$. Known results, however, provide grounds for the following conjecture:

$$Var[C_p] \sim n^2.$$

Mahmoud et al. (1995) have shown that this result holds for the case $p = 1$, while Régnier (1989) has shown it for the case $p = n$—that is, for Quicksort. It seems likely, therefore, that it holds for all intermediate values of p. Simulation results discussed by Lent and Modarres (1995) support this conjecture.

Appendix A
Additional Calculations

A.1 Calculation for Analysis of Tree-growing Search Strategies

Proposition A.1: *For* $n \geq 3$,

$$\sum_{k=1}^{n} k^3 \frac{2^k}{(n+1)!} \begin{bmatrix} n \\ k \end{bmatrix} = 8H_{n+1}^3 + O\left(\ln^2 n\right).$$

Proof: By the definition of the kth Stirling number of the first kind of order n,

$$\sum_{k=1}^{n} k(k-1)(k-2)\, x^{k-3} \begin{bmatrix} n \\ k \end{bmatrix} = \frac{d^3}{dx^3}\left[x(x+1)\cdots(x+n-1)\right].$$

So to find $\sum_{k=1}^{n} k(k-1)(k-2)\, 2^k \begin{bmatrix} n \\ k \end{bmatrix}$, we compute the third derivative of $x(x+1)\cdots(x+n-1)$ and evaluate the result at $x = 2$. This gives

$$\begin{aligned} \sum_{k=1}^{n} k(k-1)(k-2)\, 2^k \begin{bmatrix} n \\ k \end{bmatrix} &= 8(n+1)!\left[(H_{n+1}-1)^3 - 3(H_{n+1}-1)\left(H_{n+1}^{(2)}-1\right)\right. \\ &\qquad \left. + 2\left(H_{n+1}^{(3)}-1\right)\right], \end{aligned}$$

where $H_n^{(m)}$ is the nth harmonic number of order m. Thus

$$\sum_{k=1}^{n} k(k-1)(k-2)\frac{2^k}{(n+1)!} \begin{bmatrix} n \\ k \end{bmatrix} = 8H_{n+1}^3 + O\left(H_{n+1}^2\right).$$

Since Mahmoud (1992, p. 75) shows that

$$\sum_{k=1}^{n} k^2 \frac{2^k}{(n+1)!} \begin{bmatrix} n \\ k \end{bmatrix} \sim 4H_{n+1}^2,$$

and

$$\sum_{k=1}^{n} k \frac{2^k}{(n+1)!} \begin{bmatrix} n \\ k \end{bmatrix} \sim 2H_{n+1},$$

the proposition follows. ■

A.2 Calculations for Analysis of MQS

Proposition A.2: *Under the assumptions of Theorem 4.1, the average number of partition steps performed by MQS is* $2p \ln n + O(1)$.

Proof: Let $D_p^{(n)}$ represent the number of partition steps MQS performs. Then

$$D_p^{(n)} = \sum_{j=1}^{n} Q_j^{(n)} I_{jp}^{(n)},$$

where

$$Q_j^{(n)} = \begin{cases} 1, & \text{if } j \text{ has any descendants;} \\ 0, & \text{otherwise,} \end{cases}$$

and $I_{jp}^{(n)}$ is as defined in Section 4.2. Let $S_j^{(n)}$ be defined as in Chapters 2 and 4, and again we will suppress the superscript. Since

$$E[I_{jp} | S_j = k] = 1 - \frac{\binom{n-k-1}{p}}{\binom{n}{p}},$$

we have, by conditioning on S_j,

$$E\left[\sum_{j=1}^{n} Q_j I_{jp}\right] = \sum_{j=1}^{n} \sum_{k=1}^{n-1} \left[1 - \frac{\binom{n-k-1}{p}}{\binom{n}{p}}\right] P\{S_j = k\}$$

$$= \sum_{j=1}^{n}\sum_{k=1}^{n-1} P\{S_j = k\} - \frac{1}{\prod_{i=0}^{p-1}(n-i)} \sum_{j=1}^{n}\sum_{k=0}^{n-1}\left[\prod_{i=1}^{p}(n-k-i)\right] P\{S_j = k\}.$$

Proceeding as in Section 4.2, we obtain

$$E[D_p] = \sum_{j=1}^{n}\sum_{k=1}^{n-1} P\{S_j = k\} - \frac{1}{\langle n-p+1\rangle_p}\sum_{j=1}^{n}\sum_{r=1}^{p}\begin{bmatrix}p\\r\end{bmatrix}(n-p)^r \sum_{k=1}^{n-1} P\{S_j = k\} \quad \text{(A.1)}$$

$$+\frac{1}{\langle n-p+1\rangle_p}\sum_{j=1}^{n}\sum_{r=1}^{p}\begin{bmatrix}p\\r\end{bmatrix} r(n-p)^{r-1} E[S_j] \quad \text{(A.2)}$$

$$+\frac{1}{\langle n-p+1\rangle_p}\sum_{j=1}^{n}\sum_{r=2}^{p}\begin{bmatrix}p\\r\end{bmatrix}\sum_{t=2}^{r}(-1)^{t+1}\binom{r}{t}(n-p)^{r-t} E\left[S_j^t\right]. \quad \text{(A.3)}$$

We examine each of the above expressions separately. Simplifying expression A.1 gives

$$-\frac{\langle n-p\rangle_p}{\langle n-p+1\rangle_p}\sum_{j=1}^{n}\sum_{k=1}^{n-1} P\{S_j = k\} = -\left(1-\frac{p}{n}\right)\sum_{j=1}^{n}\sum_{k=1}^{n-1} P\{S_j = k\}$$

$$= -\sum_{j=1}^{n}\sum_{k=1}^{n-1} P\{S_j = k\} + \frac{p}{n}\sum_{j=1}^{n}\sum_{k=1}^{n-1} P\{S_j = k\}$$

$$= -\sum_{j=1}^{n}\sum_{k=1}^{n-1} P\{S_j = k\} + O(1). \quad \text{(A.4)}$$

We simplify expression A.2 using the identity

$$\langle x\rangle_p \sum_{j=0}^{p-1}\frac{1}{x+j} = \sum_{r=1}^{p} r\begin{bmatrix}p\\r\end{bmatrix} x^{r-1},$$

introduced in Section 4.2. Thus

$$\sum_{j=1}^{n} E[S_j]\frac{\langle n-p\rangle_p}{\langle n-p+1\rangle_p}\sum_{i=0}^{p-1}\frac{1}{n-p+i} = \frac{1}{n}E[C_n]\left(1-\frac{p}{n}\right) \times \sum_{i=0}^{p-1}\left[1+O\left(\frac{1}{n}\right)\right]$$

$$= \frac{1}{n}\left[2(n+1)H_n - 4n\right]\left(1-\frac{p}{n}\right) \times \left[p + O\left(\frac{1}{n}\right)\right]$$

$$= 2pH_n + O(1). \qquad \text{(A.5)}$$

Expression A.3 equals

$$\sum_{r=1}^{p} \begin{bmatrix} p \\ r \end{bmatrix} \frac{(n-p)^r}{(n-p+1)_p} \sum_{t=2}^{r} (-1)^{t+1} \binom{r}{t} \left(1-\frac{p}{n}\right)^{-t} \sum_{j=1}^{n} \frac{E\left[S_j^t\right]}{n^t}. \qquad \text{(A.6)}$$

Recalling from Section 2.2 that for fixed $t > 1$,

$$\sum_{j=1}^{n} \frac{E\left[S_j^t\right]}{n^t} = O(1),$$

we can reduce expression A.6 to

$$\sum_{r=1}^{p} \begin{bmatrix} p \\ r \end{bmatrix} \left[1 + O\left(\frac{1}{n}\right)\right] \left[\sum_{t=2}^{r} (-1)^{t+1} \binom{r}{t} \left(1-\frac{p}{n}\right)^{-t}\right] O(1) = O(1). \qquad \text{(A.7)}$$

Substituting expressions A.4, A.5, and A.7 for A.1, A.2, and A.3, respectively, we have

$$E[D_p] = 2pH_n + O(1). \quad \blacksquare$$

Recall from Chapter 2 that the average level (or average number of ancestors) of a node in a random binary search tree is $2\ln n + O(1)$. Thus the average number of partition steps needed for finding one order statistic by Quickselect is $2\ln n + O(1)$. The average number of paritition steps needed for p independent runs of Quickselect is therefore $2p\ln n + O(1)$—the same as the average number required for finding p order statistics by MQS. So it is clear that the number of partition steps MQS needs to completely separate the searches for the p order statistics is $O(1)$. This result reveals the rudimentary shape of a typical MQS tree. The number of nodes on each of the

highest levels varies, but below the first $O(1)$ levels the tree has p "branches," each corresponding to a search for one of the order statistics and each having an average height of $2 \ln n + O(1)$.

Appendix B
Overview of CPS Estimation Procedures

The CPS sample consists of selected *clusters* of housing units, some identified as collections of addresses (from a list frame) and others as small geographic areas (from an area frame). Each cluster is expected to include approximately four housing units. Interviewers collect data on all persons living in the selected housing units. The U.S. Census Bureau, using all the sample data, computes a weight for each sample person. These are estimates of the number of actual persons each sample person represents. Weights for sample families are simply the person weights of family heads. The estimation process involves four main steps.

1. data "cleaning,"
2. basic weighting,
3. noninterview adjustment, and
4. ratio estimation.

Data cleaning refers to the process of refining the raw data—correcting for inconsistent

or missing items—to render them suitable for use in estimation. Weights for all sample persons are computed through the remaining three steps.

B.1 Basic Weighting

In the basic weighting procedure, data from each sample person are weighted by the inverse of the person's probability of selection—a rough estimate of the number of persons the sample person represents. Adding the "base weights" of all sample persons having a given characteristic (e.g., all persons in a given income range) yields a simple unbiased estimate of the number of persons in the population possessing the characteristic. Under the current CPS sample design, almost all persons in the same state have the same probability of selection.

When a selected cluster of housing units is found to contain many more units than expected, field subsampling is carried out. (This happens only when the cluster is identified as a geographic area.) Appropriate adjustment factors are then applied to the base weights to account for the subsampling.

B.2 Noninterview Adjustment

In the noninterview adjustment procedure, the weights of persons in all interviewed households are adjusted to account for occupied sample households whose residents were not interviewed because of impassable roads, refusals, or unavailability of respondents. Households not interviewed make up about six percent of the occupied sample households each month. Noninterview clusters are formed by grouping together households in

geographic areas deemed similar in labor force characteristics, and a noninterview adjustment factor—the inverse of the proportion of occupied sample households interviewed—is computed for each cluster. This factor is then applied to the weight of each interviewed person in the cluster.

B.3 Ratio Estimation

Two stages of ratio adjustments are applied to CPS weights. To understand the purpose of the first-stage ratio adjustment, we must briefly review the CPS sample design. In the CPS, large geographic areas, such as counties and cities, are designated as primary sampling units (PSUs). Similar PSUs are grouped together, within state, to form *strata*. One PSU is selected from each stratum, and sample households from the selected PSU represent the entire stratum. Some densely populated PSUs are in strata by themselves and thus are selected with certainty and called self-representing (SR). All other strata are called non-self-representing (NSR), since the sample PSUs in these strata represent not only themselves but also the other PSUs in their respective strata.

The first-stage ratio adjustment reduces the contribution to variance that results from selecting a sample of PSUs rather than drawing sample households from every PSU in the nation. For each state, the following ratio is computed, using data from the most recent decennial census, for two race cells (black and non-black):

$$\frac{\text{total population of all NSR strata}}{\text{estimated population of all NSR strata, based on sample PSUs}}.$$

Since the adjustment factors are based entirely on decennial census data, they do not change from month to month.

The sample distribution may differ somewhat from the population distribution in characteristics such as age, race, sex, and area of residence. Since these characteristics are correlated with labor force status and other characteristics of interest, the sample weights are adjusted to agree with distributions of various population characteristics, as estimated from other data sources (primarily the decennial census, adjusted for census undercount.)

Second-stage ratio adjustment is performed both to reduce variability of the survey estimates and to correct for CPS undercoverage. The adjustment procedure weights the sample person records to provide sample estimates consistent with three sets of population estimates, called controls:

1. state (for persons aged 16 or over)—51 cells
2. age/sex/ethnic origin (Hispanic or non-Hispanic)—19 cells, and
3. age/sex/race—118 cells

Second-stage ratio adjustment is performed separately for each of the eight panels or rotation groups[11] that make up the monthly CPS sample proceeds as follows.

1. CPS sample records are cross-classified by state.
2. Weighted sample population estimates are calculated for states by adding the weights of sample persons in each state.

[11]Households in the same rotation group enter and leave the CPS sample together; households remain in sample for four months, leave the sample for eight months, and then re-enter for another four months.

3. The weight of each person is multiplied by the following ratio, calculated for the person's state of residence:

$$\frac{\text{independently derived state population estimate}}{\text{state population estimate from CPS sample}}.$$

4. Steps 1 through 3 (which accomplish adjustment to state population controls) are repeated for age/sex/ethnic origin and age/sex/race groups.

5. Steps 1 through 4 are repeated five times, for a total of six iterations.

This three-way raking procedure results in CPS sample population estimates for each control characteristic (age, sex, etc.) that virtually equal the corresponding independent population estimates.

References

[1] Arora, S. and Dent, W. (1969). Randomized binary search technique. *Communications of the ACM*, Vol. 12, pp. 77–80.

[2] Billingsley, P. (1986). *Probability and Measure*. Wiley, New York.

[3] Devroye, L. (1991). Limit laws for local counters in random binary search trees. *Random Structures and Algorithms*, Vol. 2, pp. 303–315.

[4] Doberkat, E. (1982). Asymptotic estimates for the higher moments of the expected behavior of straight insertion sort. *Information Processing Letters*, Vol. 14, pp. 179–182.

[5] Dobrow, R. and Fill, J. (1995). On the Markov Chain for the move-to-root rule for binary search trees. *Annals of Applied Probability*, Vol. 5, pp. 1–19.

[6] Eddy, W. and Schervish, M. (1995). How Many Comparisons Does Quicksort Use? *Journal of Algorithms* (to appear).

[7] Fill, J. (1995). On the distribution for binary search trees under the random permutation model. *Random Structures and Algorithms* (to appear).

[8] Floyd, R. and Rivest, R. (1975). Algorithm 489. *Communications of the ACM*, Vol. 18, p. 173.

[9] Fritsch, F., Schafer, R., and Crowley, W. (1973). Solution of the transcendental equation $we^w = x$. *Communications of the ACM*, Vol. 16, No.2.

[10] Geertsema, J. (1970). Sequential confidence intervals based on rank tests. *Annals of Mathematical Statistics*, Vol. 41, pp. 1016–1026.

[11] Gonnet, G. and Baeza-Yates, R. (1991). *Handbook of Algorithms and Data Structures, Second Edition*. Addison–Wesley, Reading, Massachusetts.

[12] Hennequin, P. (1987). Combinatorial analysis of quicksort algorithm. *RAIRO, Theoretical Informatics and Applications*, Vol. 23, pp. 317–333.

[13] Hennequin, P. (1991). Analyse en moyenne d'algorithmes, tri rapide et arbres de recherche, Ph.D. Dissertation, L'École Polytechnique Palaiseau.

[14] Hoare, C. (1962). Quicksort. *The Computer Journal*, Vol. 5, pp. 10–15.

[15] Hu, T. and Wachs, M. (1987). Binary search on a tape. *SIAM Journal on Computing*, Vol. 16, No. 3, pp. 573–590.

[16] Knuth, D. (1973a). *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. Addison–Wesley, Reading, MA.

[17] Knuth, D. (1973b). *The Art of Computer Programming, Vol. 3: Sorting and Searching*. Addison–Wesley, Reading, MA.

[18] Kuipers, L. and Niederreiter, H. (1974). *Uniform Distribution of Sequences*. Wiley, New York.

[19] Lent, J. and Mahmoud, H. (1996). Average-case analysis of multiple quickselect: an algorithm for finding order statistics. *Statistics and Probability Letters* (to appear).

[20] Lent, J., Mahmoud, H., and Bose, S. (1996). On tree-growing search strategies. *Annals of Applied Probability* (provisionally accepted).

[21] Lent, J. and Modarres, R. (1995). An algorithm for simultaneous selection of order statistics (presented at the 1995 Joint Statistical Meetings).

[22] Lynch, W. (1965). More combinatorial problems on certain trees. The Computer Journal, Vol. 7, pp. 299–302.

[23] Mahmoud, H. (1992). *Evolution of Random Search Trees*. Wiley, New York.

[24] Mahmoud, H., Modarres, R., and Smythe, R. (1995). Analysis of quickselect: an algorithm for order statistics. *RAIRO, Theoretical Informatics and Applications* (to appear).

[25] Panny, W. (1986). A note on the higher moments of the expected behavior of straight insertion sort. *Information Processing Letters*, Vol. 22, pp. 175–177.

[26] Prodinger, H. (1996). Analysis of multiple quickselect—Hoare's FIND algorithm for several elements. *Information Processing Letters* (to appear).

[27] Régnier, M. (1989). A limiting distribution for quicksort. *RAIRO, Theoretical Informatics and Applications*, Vol. 23, pp. 335–343.

[28] Rösler, U. (1991). A limit theorem for QUICKSORT. *RAIRO, Theoretical Informatics and Applications*, Vol. 25, pp. 85–100.

[29] Sedgewick, R. (1988). *Algorithms*, 2nd ed. Addison–Wesley, Reading, MA.

[30] Serfling, R. (1980). *Approximation Theorems of Mathematical Statistics*. Wiley, New York.

[31] Tan, K. and Hadjicostas, P. (1995). Some properties of a limiting distribution in Quicksort. *Statistics and Probability Letters* (to appear).

[32] Wilks, S. (1962). *Mathematical Statistics*, Wiley, New York.