Scan A7077
A6696

Connolly - Knight
Preprint

add to 2 seqs

Hold for reprint
Contains possible sequence

A 7077
A 6696

SEARCH IN AN ARRAY IN WHICH PROBE COSTS GROW EXPONENTIALLY OR FACTORIALLY

MARY V. CONNOLLY* and WILLIAM J. KNIGHT‡

Abstract. We find efficient strategies for searching an ordered array in which the cost of a probe into the array increases exponentially or factorially as the probe location moves from left to right. We show that binary search is often significantly inferior to certain other simple search strategies. This is true both in the case where *expected* search costs form the basis of comparison and also in the case where it is desired to minimize the *maximum possible* search cost.

Key words. array search, search trees, binary search

AMS(MOS) subject classification. 68P10

1. Introduction. Steiglitz and Parks [4] have described a filter-design problem that turns out to be equivalent to the problem of searching an ordered array for which a probe into the k-th location has an associated cost that increases as k increases. The equivalence of the two problems is explained in [2]. That paper confirmed a conjecture of Steiglitz and Parks to the effect that although binary search in such an array might be expected to perform poorly in comparison with other seach strategies, in fact binary search is surprisingly near optimal *when the probe costs are given by a polynomial in* k. In the present paper we consider what happens when the probe costs grow exponentially or factorially. We shall show that in these two cases, binary search is often inferior to certain other simple search strategies.

2. Preliminaries. Throughout this paper we discuss the situation encountered by Steiglitz and Parks [4], namely an array of n problems of some kind, one for each array subscript k from 1 to n. The amount of time required to solve the k-th problem is given by a "penalty function" $P(k)$. Each problem has a "yes" or "no" answer, and when the answer is "yes" for some k_0 it is "yes" for all larger subscripts k. We seek the smallest value of k for which the answer is "yes", provided there is such a k. To find this smallest value we want to use an optimal search strategy on the array. There are two separate criteria for deciding whether one strategy is better than

* Department of Mathematics, St. Mary's College, Notre Dame, Indiana 46556.

‡ Department of Mathematics and Computer Science, Indiana University, P.O. Box 7111, South Bend, Indiana 46634.

another. The first compares the *expected* search costs of the competing strategies; the second compares their *worst possible* costs. In this paper we consider both criteria. We look at exponential and factorial penalty functions because the solutions of many combinatorial problems require exponential or factorial time.

Every search strategy for an array of length n corresponds to a unique binary tree with node labels $1, ..., n$. The tree is recursively defined as follows. The label, call it r, of the root of the tree is the subscript of the array cell that the strategy tells us to probe first. The left subtree contains the labels $1, ..., r-1$. It corresponds to the search strategy to be used for locations $1, ..., r-1$ if the first probe yields the answer "yes". The right subtree contains the labels $r+1, ..., n$ and corresponds to the strategy to be used if the first probe yelds the answer "no". Thus the tree and its labels form a *binary search tree* (see [3, p. 321]). Conversely, every n-node binary search tree with node labels $1, ..., n$ corresponds to exactly one search strategy. It is essential to note that the nature of the problem we are considering forces a search to continue until it reaches one of the $n+1$ *external nodes* (see [3, p. 239]), at which point we can identify the location of the first "yes".

In the right subtree of a search tree of the type described above, the node labels do not begin at 1. In order to deal recursively with the left and right subtrees, it is convenient to introduce the following terminology and notation. A **search tree** will be a binary tree whose (internal) nodes have been labeled with a sequence of successive positive integers in such a way that the tree has become a binary search tree. When T denotes an unlabeled binary tree with n nodes, then $T_{1+t,n+t}$ will denote the same tree made into a search tree using the node labels $1+t, 2+t, ..., n+t$.

3. Optimal strategies for expected costs. Consider the case where search strategies are to be compared on the basis of their expected costs. We have no prior information about the array location of the first "yes", and so we assume that it is equally likely to be found in any of the n locations or not to be found at all. That is, each possibility has probability $1/(n+1)$. In [2] it is shown that in this case an optimal search strategy corresponds to a search tree $T_{1,n}$ that minimizes

$$\sum_{k=1}^{n} P(k)\ W_k(T_{1,n}) \quad , \tag{1}$$

where $W_k(T_{1,n})$ denotes the weight (number of nodes) of the subtree of $T_{1,n}$ whose root has the label k. We shall denote the sum in (1) by the expression $S_{P(k)}(T_{1,n})$. More generally, for any n-node search tree $T_{1+t,n+t}$ and any penalty function $P(k)$ defined for $1+t \le k \le n+t$ we shall write

$$S_{P(k)}(T_{1+t,n+t}) = \sum_{k=1+t}^{n+t} P(k)\ W_k(T_{1+t,n+t}) \ . \tag{2}$$

This notation allows us to write compactly a general recurrence relation:

$$(3) \qquad S_{P(k)}(T_{1+t,n+t}) = P(r)\,n + S_{P(k)}(L_{1+t,r-1}) + S_{P(k)}(R_{r+1,n+t}),$$

where r is the label on the root of T, and L and R are its left and right subtrees, with $r - t - 1$ and $n + t - r$ nodes respectively. If we let $S^*_{P(k)}(n, t)$ denote the optimal (i.e., minimal) possible value of $S_{P(k)}(T_{1+t,n+t})$ as T ranges over all n-node binary trees, then (3) implies that

$$(4) \quad S^*_{P(k)}(n, t) = \min_{1+t \le r \le n+t} \{ P(r)\,n + S^*_{P(k)}(r - t - 1, t) + S^*_{P(k)}(n + t - r, t + r) \},$$

and the values of r that produce the minimum in (4) are the roots of optimal search trees for the prescribed values of n and t. Note that an n-node binary tree T may have the property that $T_{1+t,n+t}$ is optimal for one value of t, but not optimal for a different value of t. For example, when $P(k) = k!$ and $n = 4$ the tree on the left in Figure 1 is optimal when made into a search tree with labels 1,2,3,4, but not with 3,4,5,6, while the tree on the right is optimal with labels 3,4,5,6, but not 1,2,3,4 . These facts can be verified by examining all 14 possible 4-node binary trees.

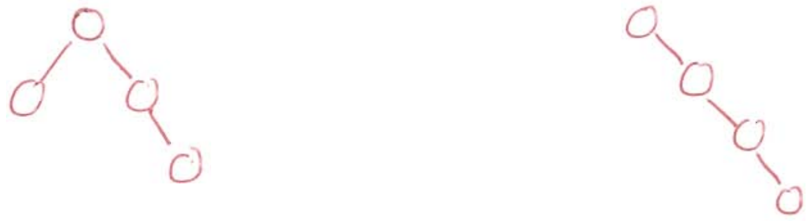


Figure 1

If we are compelled to obtain an *exact* optimal search strategy for a prescribed P(k) and array length N, then we must calculate $S^*_{P(k)}(N, 0)$ as well as complete information concerning at least one optimal search tree whose search cost is this optimal number. Recurrence relation (4), together with the fact that $S^*_{P(k)}(0, t) = 0$ for all t, makes it possible to use dynamic programming to compute all this information: we compute a triangular array of optimal costs and roots for all pairs (n, t) satisfying $n \ge 0$, $t \ge 0$, $n + t \le N$ (cf. [2, pp. 119-123], where a similar dynamic programming problem is solved). This appears to require $O(N^3)$ time, since each of the (roughly) $N^2/2$ pairs (n, t) seems to require us to examine n values of r in order to find the minimum in formula (4). However, when P(k) is a fast growing function, the following theorem drastically limits the number of values of r that must be examined for a given pair (n, t) .

THEOREM 1. Let P(k) be any positive penalty function. Let $n \ge 1$ and $t \ge 0$ be prescribed integers. Then any root r that minimizes the expression on the right in equation (4) must satisfy

$$(5) \qquad \frac{P(r)}{r - t} \le \frac{n + 1}{2} P(1 + t).$$

Proof. We use induction on n. If $n = 1$, then $r = 1 + t$ and the inequality to be proved is trivial. Now suppose the theorem is true for $n = 1, 2, \ldots, m - 1$, where $m \geq 2$. We seek to prove that it is also true when $n = m$. Take any r that minimizes the expression in (4) when $n = m$. Then r is the root of some optimal n-node search tree $T_{1+t,m+t}$ for the prescribed values of n and t. Let c denote the left child of r in $T_{1+t,m+t}$. Perform a simple tree rotation (see [3, p. 306]) to produce a new search tree having root c with right child r. Only two weights $W_k(T_{1+t,m+t})$ have changed: the weight on c in the new tree is m, whereas in $T_{1+t,m+t}$ it is $r + t - 1$; and the new weight on r is $m + t - c$, whereas in $T_{1+t,m+t}$ it is m. It follows that the net change in search cost in going from $T_{1+t,m+t}$ to the new tree is

$$P(c)\,(\,(m - (r - t - 1)) \;+\; P(r)\,((m + t - c) - m)\,.$$

Since the new tree must have search cost at least as great as that of the optimal tree $T_{1+t,m+t}$, the expression above must be non-negative. From this we deduce the inequality

$$(6) \qquad P(r) \;\leq\; (m - r + t + 1)\,P(c)\,/\,(c - t)\;.$$

Now examine the subtree with root c in $T_{1+t,m+t}$. This subtree has $r - t - 1$ nodes, and it must itself be an optimal search tree, for if not then we could replace it in $T_{1+t,m+t}$ by a search tree with lower search cost, and by (3) this would produce a better tree than the optimal $T_{1+t,m+t}$. By the induction hypothesis then, we must have $P(c) / (c - t) \leq [((r - t - 1) + 1) / 2]\,P(1 + t)$. Combining this with (6) gives $P(r) \leq (m - r + t + 1)(r - t)\,P(1 + t) / 2$. Since $m - r + t + 1 < m + 1$, inequality (5) holds when $n = m$, as required to complete the induction. $\square$

The power of Theorem 1 can be seen by taking an example. Suppose $P(k) = k!$ and $t = 0$. Then the inequality in Theorem 1 takes the form $2\,(r - 1)! \leq n + 1$. Since $2^{r-1} \leq 2\,(r - 1)!$ for all positive r, Theorem 1 tells us that when seeking the root r that minimizes the right side of (4), we can (in this special case) restrict our consideration to r satisfying $r \leq 1 + \lg(n + 1)$ instead of $r \leq n$. The inequality is even stronger when $t > 0$ and $P(k) = k!$. Thus, in this case the amount of time required to compute an exact optimal search strategy for a given array length N is only $O(N^2 \log N)$. The entries in Table 1 were computed using this dynamic programming method.

Although we have indicated how to calculate the optimal search strategy, the point of this paper is to demonstrate that for all practical purposes such calculations are not needed when $P(k)$ grows exponentially or factorially. We shall show that certain very simple search strategies are so nearly optimal that it is unlikely to be worthwhile to try to replace them by exact optimal strategies.

We begin with a factorial penalty function. The following theorem implies that when $P(k) = k!$, linear search is so nearly optimal that no other more complicated strategy need even be considered. In particular, binary search is inferior to linear search.

	$P(k) = 1.5^k$		$P(k) = 2^k$		$P(k) = 3^k$		$P(k) = k!$	
n	Optimal Search Cost	Roots of Optimal Trees	Optimal Search Cost	Roots of Optimal Trees	Optimal Search Cost	Roots of Optimal Trees	Optimal Search Cost	Roots of Optimal Trees
1	1.500	1	2	1	3	1	1	1
2	5.250	1	8	1	15	1	4	1
3	11.625	2	22	1,2	54	1	13	1,2
4	22.313	2	50	2	174	1,2	45	2
5	38.906	2	110	1,2	534	2	197	2
6	64.734	3	226	2	1620	1	1069	2
7	104.180	3	464	3	4872	2	6981	1,2
8	163.559	3	938	2	14640	1	53207	2
9	254.104	3	1888	3	43932	2	462313	2
10	389.968	4	3794	2	131826	1	4500208	3
11	594.385	3	7598	2	395490	2	48454894	3
12	900.390	4	15208	3	1186503	3	5.714E08	2
13	1.363E3	5	30438	4	3559530	2	7.321E09	2
14	2.057E3	4	60890	2	1.068E7	3	1.012E11	3
15	3.095E3	3	121792	3	3.204E7	2	1.503E12	3
16	4.651E3	4	243606	4	9.611E7	2	2.383E13	2
17	6.989E3	5	487238	2	2.883E8	3	4.018E14	2
18	1.050E4	6	974488	3	8.650E8	1	7.182E15	3
19	1.577E4	3	1948998	4	2.595E9	2	1.356E17	3
20	2.366E4	4	3898034	2,5	7.785E9	3	2.697E18	2

Table 1

A6696

THEOREM 2. Suppose $P(k) = k!$.

(a) The search cost of a linear search in an array of size $n \geq 7$ is less than

$$n! + 2(n-1)! + 3(n-2)! + 4(n-3)! + 9(n-4)! .$$

(b) The search cost of every search strategy in an array of size $n \geq 7$ exceeds

$$n! + 2(n-1)! + 3(n-2)! + 4(n-3)! + (n-4)! .$$

(c) The search cost of binary search in an array of size $n \geq 7$ is less than

$$n! + 3(n-1)! + 8(n-2)! ,$$

but for infinitely many n, the search cost for a binary search in an array of size n exceeds

$$n! + 3(n-1)! + (n-2)! .$$

Proof. The proofs of these assertions will be shown to depend on the following inequalities, each of which can easily be verified by a simple induction argument.

(7) $4(n-4)! > 6(n-5)! + 7(n-6)! + \ldots + n(1)!$ when $n \geq 7$;

(8) $n! > 2(n-1)! + 3(n-2)! + 4(n-3)! + \ldots + n(1)!$ when $n \geq 4$;

(9) $(n-2)! > 3(n-3)! + 4(n-4)! + 5(n-5)! + \ldots + (n-1)(1)!$ when $n \geq 7$;

(10) $2(n-2)! > n[(n-3)! + (n-4)! + (n-5)! + \ldots + 1!]$ when $n \geq 7$.

By equation (1), the cost of a linear search in an array of length n is given by

(11) $1(n)! + 2(n-1)! + 3(n-2)! + \ldots + n(1)!$.

This together with inequality (7) gives part (a) of the theorem. We prove part (b) by proving it for all *optimal* trees. Take any optimal search tree $T_{1,n}$ with $n \geq 7$ nodes. Then node n must be a leaf of $T_{1,n}$, for otherwise we would have this contradiction: the search cost of $T_{1,n}$ would be strictly greater than $2n!$, which by (8) would exceed linear search cost (11). Since node n is a leaf, by the structure of a search tree the only possible parent of n in $T_{1,n}$ is node $n-1$. The left subtree of node $n-1$ must be empty, for if not then the weight of the subtree rooted at $n-1$ would be at least 3, and thus the cost of $T_{1,n}$ would exceed $n! + 3(n-1)! + (n-2)! + (n-3)! + \ldots + 1!$, which by (8), with n replaced by $n-1$, would exceed linear search cost (11). Since the left subtree of node $n-1$ is empty, by the structure of a search tree the only possible parent of $n-1$ in $T_{1,n}$ is node $n-2$. The left subtree on $n-2$ must also be empty, for if not then the weight of the subtree rooted at $n-2$ would be at least 4, and thus the cost of $T_{1,n}$ would exceed $n! + 2(n-1)! + 4(n-2)! + (n-3)! + \ldots + 1!$, which by (9) would exceed linear search cost (11). By the structure of a search tree the only possible parent of $n-2$ is node $n-3$, which therefore has weight at least 4, so the search cost of $T_{1,n}$ exceeds $n! + 2(n-1)! + 3(n-2)! + 4(n-3)! + (n-4)!$. Finally, for part (c) let $B_{1,n}$ denote the search tree corresponding to the classical binary search strategy in an array of length n. The first inequality of part (c) is proved by noting that in $B_{1,n}$ the weight of the subtree rooted at n is always 1, the weight of the subtree rooted at $n-1$ is at most 3, the weight of the subtree rooted at $n-2$ is at most 6, and the remainder of the search cost is bounded above by the expression on the right side of (10). The second inequality of (c) is proved by noting that whenever n has the form $2^p - 1$ for an integer p, then $B_{1,n}$ contains the 3-node subtree with root $n-1$, left child $n-2$, and right child n . □

Parts (a) and (b) of Theorem 2, together with some algebra and some of the entries in Table 1, can be used to show that only when n is 4 or 5 does the cost of linear search differ from the cost of an optimal search by more than one percent. When $n > 10$, it differs by less than one one-hundredth of a percent. Parts (b) and (c) show that the second-highest order term in the cost of binary search is infinitely often larger than the second order term in the cost of linear search. Thus, although the highest order term (n!) for binary search is "correct" and the percent excess over the optimal cost approaches zero as n goes to infinity, the approach is far slower than with linear search. For example, when $n = 15$ binary search is almost six percent worse than linear search.

Now we turn to the case in which the penalty for probing in location k is proportional to b^k for some constant $b > 1$. The constant of proportionality does not materially affect the computations, so we assume simply that $P(k) = b^k$. Then it is easy to see from (2) that

$$(12) \qquad S_{b^k}(T_{1+t,n+t}) = b^t S_{b^k}(T_{1,n}) .$$

This equation has the happy consequence that when $P(k) = b^k$ for some constant b , then an n-node binary tree T is optimal for labels 1+t, ..., n+t if and only if it is optimal for labels 1, ..., t (cf. the comments following equation (4)). Thus we can dispense with the "translation parameter" t and deal only with search trees of the form $T_{1,n}$. We write $S^*_{b^k}(n)$ for the optimal search cost in an array of length n , and observe that (4) now takes the simpler form

$$(13) \qquad S^*_{b^k}(n) = \min_{1 \le r \le n} \{ b^r n + S^*_{b^k}(r-1) + b^r S^*_{b^k}(n-r) \} .$$

Again, values of r that produce the minimum in (13) are roots of optimal trees labeled 1, ..., n. A dynamic programming solution for $S^*_{b^k}(N)$ now requires only that we compute and store $S^*_{b^k}(n)$ for each $n \le N$. Although (13) makes it appear that the computation will require time proportional to N^2, Theorem 1 can be used in an obvious way to reduce this to $O(N \log N)$.

Table 2 shows the formulas for $S^*_{b^k}(n)$ for $n = 0,1,...,7$ and all $b > 1$. Using (13) and a mathematical program, the authors have generated many more of these formulas, and each is uglier than its predecessors. This suggests that an elegant theory for exact optimal trees is impossible (note, for example, the mystifying alternation of the root of the optimal tree when $n = 7$). For this reason, we have investigated simple search trees that can be shown to have costs that are very nearly optimal. Here is a summary of our results for penalty functions of the form $P(k) = b^k$.

(a) If $1 < b < 1.18$ (approximately), then binary search is so near optimal (at worst about 4 1/2 percent above optimal) that for most purposes binary search would be the search strategy of choice.

n	$S^*_{b^k}(n)$		Root of optimal search tree
0	0		—
1	b		1
2	$2b + b^2$		1
3	$b + 3b^2 + b^3$	if $1 < b \le 2$	2
	$3b + 2b^2 + b^3$	if $2 \le b$	1
4	$b + 4b^2 + 2b^3 + b^4$	if $1 < b \le 3$	2
	$4b + 3b^2 + 2b^3 + b^4$	if $3 \le b$	1
5	$b + 5b^2 + b^3 + 3b^4 + b^5$	if $1 < b \le 2$	2
	$b + 5b^2 + 3b^3 + 2b^4 + b^5$	if $2 \le b \le 4$	2
	$5b + 4b^2 + 3b^3 + 2b^4 + b^5$	if $4 \le b$	1
6	$2b + b^2 + 6b^3 + b^4 + 3b^5 + b^6$	if $1 < b \le 1.58457^*$	3
	$b + 6b^2 + b^3 + 4b^4 + 2b^5 + b^6$	if $1.58458^* \le b \le 2.8637$	2
	$6b + b^2 + 5b^3 + 3b^4 + 2b^5 + b^6$	if $2.8638 \le b \le 3.6180$	1
	$b + 6b^2 + 4b^3 + 3b^4 + 2b^5 + b^6$	if $3.6181 \le b \le 5$	2
	$6b + 5b^2 + 4b^3 + 3b^4 + 2b^5 + b^6$	if $5 \le b$	1
7	$b + 3b^2 + b^3 + 7b^4 + b^5 + 3b^6 + b^7$	if $1 < b \le 1.3416$	4
	$2b + b^2 + 7b^3 + b^4 + 4b^5 + 2b^6 + b^7$	if $1.3417 \le b \le 2.2360$	3
	$7b + b^2 + 6b^3 + b^4 + 4b^5 + 2b^6 + b^7$	if $2.2361 \le b \le 2.6415$	1
	$b + 7b^2 + b^3 + 5b^4 + 3b^5 + 2b^6 + b^7$	if $2.6416 \le b \le 3.8454$	2
	$7b + b^2 + 6b^3 + 4b^4 + 3b^5 + 2b^6 + b^7$	if $3.8455 \le b \le 4.7320$	1
	$b + 7b^2 + 5b^3 + 4b^4 + 3b^5 + 2b^6 + b^7$	if $4.7321 \le b \le 6$	2
	$7b + 6b^2 + 5b^3 + 4b^4 + 3b^5 + 2b^6 + b^7$	if $6 \le b$	1

* This number is an approximation to one of the irrational roots of the polynomial equation $2b + b^2 + 6b^3 + b^4 + 3b^5 + b^6 = b + 6b^2 + b^3 + 4b^4 + 2b^5 + b^6$. Similarly for the other non-integer values.

Table 2

(b) If $1.19 < b < 1.68$ (approximately), then a "quarter-linear" search strategy (to be described) yields results that are at worst about 4 1/2 percent above optimal. Over most of this interval, binary search is always inferior to the quarter-linear strategy, sometimes by as much as 4 percent. The quarter-linear strategy can be roughly described as one in which we begin at the left and probe in every fourth location moving to the right until we find we have gone too far.

(c) If $1.68 < b < 3.2$ (approximately), then a "semi-linear" search strategy (starting at the left and probing in every second location) is better than both the quarter-linear strategy and binary search, and is at worst about one percent above optimal. Binary search cost can exceed semi-linear search cost by as much as 7 1/2 percent.

(d) If $b > 3.2$ (approximately), then linear search is better than semi-linear, quarter-linear, and binary search, and is at worst about 1/2 of one percent above optimal. For every $b > 3.2$, binary search cost can be at least 7 1/2 percent above linear search cost.

The remainder of this section is taken up with proving the (unpleasantly technical) theorems from which we can derive the claims we have just made. The first gives us a useful lower bound for the search cost $\overset{*}{S}_{bk}(n)$ in optimal trees with n nodes.

THEOREM 3. For each real number $b > 1$ there exist (non-unique) positive constants σ_b and τ_b such that for all $n \geq 0$,

$$\overset{*}{S}_{bk}(n) \geq \sigma_b b^n - n - \sigma_b/b - \tau_b = \sigma_b b^n + O(n). \tag{14}$$

Proof. Let $b > 1$ be given. Fix any positive integer M. Use recurrence relation (13) to calculate $\overset{*}{S}_{bk}(n)$ for $n = 0, \ldots, M-1$. The calculations can be numerical, for a prescribed value of b (cf. Table 1), or algebraic (cf. Table 2). Now define two constants τ_b and σ_b by

(15) $\tau_b = \min\{ r + (M - r + 1)/b^r : r = 1, \ldots, M \}$,

(16) $\sigma_b = \min\{ (\overset{*}{S}_{bk}(n) + n + \tau_b) / (b^n - 1/b) : n = 0, \ldots, M - 1 \}$.

Note that (16) implies that for $n = 0, \ldots, M - 1$,

$$\overset{*}{S}_{bk}(n) \geq \sigma_b (b^n - 1/b) - n - \tau_b . \tag{17}$$

We shall now show by induction that (17), and hence (14), holds for all $n \geq 0$. Take any integer $m \geq M$ for which (17) holds for all $n \leq m - 1$. To prove that (17) holds when $n = m$, note that by formula (13) and the inductive hypothesis,

$$\overset{*}{S}_{bk}(n) \geq \min_{1 \leq r \leq m} \{ b^r m + \sigma_b (b^{r-1} - 1/b) - (r - 1) - \tau_b + b^r \sigma_b (b^{m-r} - 1/b) - b^r (m - r) - b^r \tau_b \}$$

$$= \sigma_b (b^m - 1/b) - m - \tau_b + \min_{1 \le r \le m} \{ m - r + 1 + b^r (r - \tau_b) \} .$$

This shows that (17) will hold when $n = m$ provided that for all $m \ge M$,

$$\min_{1 \le r \le m} \{ m - r + 1 + b^r (r - \tau_b) \} \ge 0 .$$

For this it would suffice to prove that $M - r + 1 + b^r (r - \tau_b) \ge 0$ for $r = 1, ..., M$. But this is true just by definition (15) of τ_b. This completes the proof that (17) holds for all n. □

Generally speaking, the larger the value one chooses for M in the proof of Theorem 3, the larger the value one is able to obtain for σ_b, although that is not invariably the case. Computer calculations of σ_b for various values of b and M suggest that for values of b less than 3, the value of σ_b "settles down" to a constant before M reaches 100.

Next we estimate the cost of binary search. As before, let $B_{1,n}$ denote the search tree corresponding to classical binary search. Its root, of course, is $\lfloor (1 + n)/2 \rfloor$. The exact search costs $S_{b^k} (B_{1,n})$ can be computed numerically (for a single prescribed b) or algebraically (as expressions in the variable b) by using the following special cases of equations (3) and (12):

$$\text{(18)} \qquad S_{b^k} (B_{1,n}) = \begin{cases} b^q (2q) + S_{b^k} (T_{1,q-1}) + b^q S_{b^k} (B_{1,q}) & \text{if } n = 2q , \\ b^{q+1} (2q + 1) + S_{b^k} (B_{1,q}) + b^{q+1} S_{b^k} (B_{1,q}) & \text{if } n = 2q + 1 . \end{cases}$$

THEOREM 4. Assume that $P(k) = b^k$, where $b > 1$. Then there exist (non-unique) constants α_b, β_b, γ_b, and δ_b such that for all $n \ge 0$,

$$\text{(19)} \qquad \alpha_b b^n - 2n - \beta_b \le S_{b^k} (B_{1,n}) \le \gamma_b b^n - 2n - \delta_b .$$

Proof. Fix any positive integer M. Use (18) to compute $S_{b^k} (B_{1,n})$ for all $n \le 2M$. Let the following equations define the constants on the left sides:

$$\text{(20)} \qquad \alpha_b = \min \{ (S_{b^k} (B_{1,n}) + 2n) / (b^n - 1/b) : n = M, M+1, ..., 2M \} ,$$

$$\text{(21)} \qquad \varepsilon_b = \max \{ \alpha_b (b^n - 1/b) - 2n - S_{b^k} (B_{1,n}) : n = 0,1, ..., M-1 \} ,$$

$$\text{(22)} \qquad \beta_b = \alpha_b / b + \max \{ 0, \varepsilon_b \} ,$$

$$\text{(23)} \qquad \kappa_b = \max \{ 1 + 2 / (e \ln b) , 2b / (e \ln b) \} ,$$

$$\text{(24)} \qquad \gamma_b = \max \{ (S_{b^k} (B_{1,n}) + 2n + \kappa_b) / (b^n - 1/b) : n = M, M+1, ..., 2M \} ,$$

$$\text{(25)} \qquad \zeta_b = \min \{ \gamma_b b^n - 2n - \gamma_b/b - \kappa_b - S_{b^k} (B_{1,n}) : n = 0,1, ..., M-1 \} ,$$

$$\text{(26)} \qquad \delta_b = \gamma_b/b + \kappa_b + \min \{ 0, \zeta_b \} .$$

Then (24) implies that for all $n = M, M+1, ..., 2M$,

$$\text{(27)} \qquad S_{b^k} (B_{1,n}) \le \gamma_b (b^n - 1/b) - 2n - \kappa_b$$

We shall now show by induction that (27) holds for all $n \geq M$. The base cases are $n = M, M+1, \ldots, 2M$. Now take any integer $m > 2M$ such that (27) holds for $n = M, \ldots, m-1$. To prove that (27) holds for $n = m$, note that if m is odd, then by (18) and the inductive hypothesis we have

$$\begin{aligned} S_{b^k}(B_{1,m}) \;\leq\; & b^{(m+1)/2} m + \gamma_b (b^{(m-1)/2} - 1/b) - 2(m-1)/2 - \kappa_b \\ & + b^{(m+1)/2} \gamma_b (b^{(m-1)/2} - 1/b) - b^{(m+1)/2} (m-1) - b^{(m+1)/2} \kappa_b \\ = \;& \gamma_b (b^m - 1/b) - 2m - \kappa_b + [m + 1 - (\kappa_b - 1) b^{(m+1)/2}] . \end{aligned}$$

It now suffices to prove that $m + 1 - (\kappa_b - 1)b^{(m+1)/2} \leq 0$. Since $\kappa_b \geq 1 + 2/(e \ln b)$, it suffices to prove that $m + 1 - 2b^{(m+1)/2} / (e \ln b) \leq 0$. This can be done by using elementary calculus to prove that the function $2x - 2b^x / (e \ln b)$ has maximum value 0. Similarly, if m is even, then by (18) and the inductive hypothesis,

$$\begin{aligned} S_{b^k}(B_{1,m}) \;\leq\; & b^{m/2} m + \gamma_b (b^{m/2-1} - 1/b) - 2(m/2 - 1) - \kappa_b \\ & + b^{m/2} \gamma_b (b^{m/2} - 1/b) - b^{m/2} m - b^{m/2} \kappa_b \\ = \;& \gamma_b (b^m - 1/b) - 2m - \kappa_b + [m + 2 - \kappa_b b^{m/2}] . \end{aligned}$$

Since $\kappa_b \geq 2b / (e \ln b)$, it suffices to prove that $m + 2 - \kappa_b b^{m/2} \leq 0$, which amounts (again) to showing that $2x - 2b^x / (e \ln b) \leq 0$. We have completed the proof that (27) holds for all $n \geq M$.

Now we are ready to prove the right half of (19) for all $n \geq 0$. By (26), $\delta_b \leq \gamma_b/b + \kappa_b$, so $-\gamma_b/b - \kappa_b \leq -\delta_b$, which together with (27) implies the right half of (19) when $n \geq M$. When $n = 0, \ldots, M-1$, we have

$$\delta_b \leq \gamma_b/b + \kappa_b + \zeta_b \leq \gamma_b/b + \kappa_b + \gamma_b b^n - 2n - \gamma_b/b - \kappa_b - S_{b^k}(B_{1,n})$$

by (26) and (25), so again we obtain the right half of (19).

The left half of (19) is proved similarly, but more simply. Start with (20) and prove inductively that $\alpha_b (b^n - 1/b) - 2n \leq S_{b^k}(B_{1,n})$ for all $n \geq M$. By (21), $\alpha_b (b^n - 1/b) - 2n \leq S_{b^k}(B_{1,n})$ for all n. Combine these facts with (22). We omit details. $\square$

Theorem 4 suggests that if we form the ratio $S_{b^k}(B_{1,n}) / b^n$ and allow n to grow without bound, we *might* see the ratio approach a limit. The next proposition says that this definitely *does not* happen. Instead, we can exhibit distinct constants $\lambda_b < \mu_b$ such that the ratio oscillates infinitely often below λ_b and above μ_b.

THEOREM 5. Assume that $P(k) = b^k$, where $b > 1$. Then there exist (non-unique) constants $\lambda_b < \mu_b$ such that

(a) if $n = 2^{p+1} + 2^p$ for some integer $p \geq 2$, then $S_{b^k}(B_{1,n}) \leq \lambda_b b^n$;

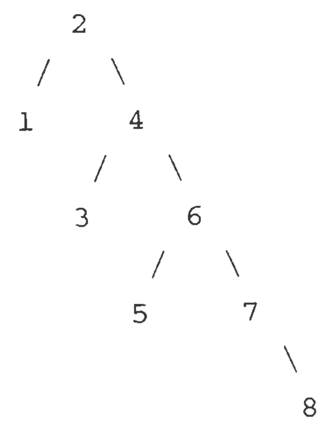
(b) if $n = 2^{p+1} + 2^p - 1$ for some integer $p \geq 2$, then $S_{b^k}(B_{1,n}) \geq \mu_b b^n$.

Proof. The constants λ_b and μ_b are given by the following formulas:

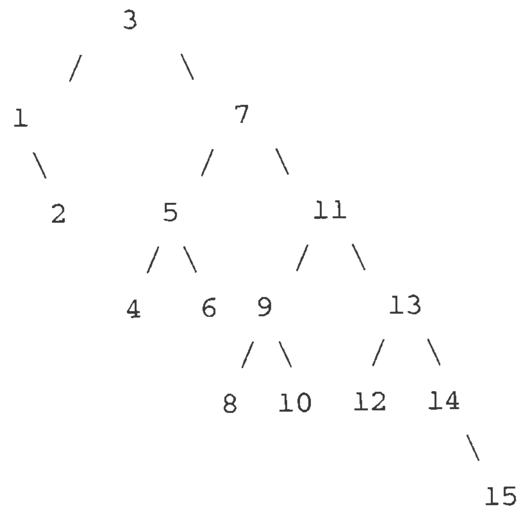
$$\lambda_b = 1 + 3/b + 1/b^2 + 6/b^3 + 12/b^6 + 24/b^{12} + (48b - 46)/((b-1)^2 b^{23})$$
$$+ (b+2)/((b^3-1)b^2) + 5/((b^6-1)b^3) + 11/((b^{12}-1)b^6) + 1/((b-1)(b^{12}-1)) ,$$

$$\mu_b = (b^3 + 2b^2)/(b^3-1) + 5b^3/(b^6-1) + 11b^6/(b^{12}-1) .$$

Use formula (18) to generate algebraic expressions in b for $S_{b^k}(B_{1,n})$ when $n = 2,3,5,6,11,12, \ldots, 2^{p+1} + 2^p - 1, 2^{p+1} + 2^p$. The formula for μ_b is obtained by simple truncation. The formula for λ_b requires that certain finite sums be estimated above by infinite series of the form $\Sigma_{k=1}^{\infty} k x^{k-1} = 1/(1-x)^2$. It also uses Bernoulli's inequality: $t^n \geq 1 + n(t-1)$ for all $t \geq 1$. We omit the details, which are available on request. $\square$



Semi-linear tree when $n = 8$.
Figure 2



Quarter-linear tree when $n = 15$.
Figure 3

Our final theorem gives the search costs for semi-linear and quarter linear search trees. **Semi-linear** search trees are defined for all $n \geq 3$ as follows: the root of the tree is $r = 2 - (n \bmod 2)$, the right child of r is $r + 2$, the right child of that is $r + 4$, and so on to $n - 2$; then $n - 1$ is the right child of $n - 2$, and n is the right child of $n - 1$; all other nodes are left children of the nodes along the path already constructed. See Figure 2 for the case where $n = 8$. A **quarter-linear** tree is defined for all $n \geq 5$ as follows: the root of the tree is $r = 4$ if n is divisible by 4 , and $r = n$

mod 4 otherwise; the right child of r is $r + 4$, the right child of that is $r + 8$, and so on to $n - 4$; the right child of $n - 4$ is $n - 2$, which has right child $n - 1$, which has right child n; the remaining nodes are placed in trees of the form $B_{1,1}$, $B_{1,2}$, or $B_{1,3}$. See Figure 3 for $n = 15$.

THEOREM 6. Assume that $P(k) = b^k$, where $b > 1$.
(a) The search cost of a linear search in an array of length n has the form $L_b\, b^n + O(n)$, where L_b is a constant given by $b^2/(b - 1)^2$.
(b) The search cost of a semi-linear tree with n nodes has the form $S_b\, b^n + O(n)$, where S_b is a constant given by $(b^4 + 2b^3 + 2b^2 - 3b - 1 + b^{-1}) / (b^2 - 1)^2$.
(c) The search cost of a quarter-linear tree with n nodes has the form $Q_b\, b^n + O(n)$, where Q_b is a constant given by $(b^8 + 2b^7 + 4b^6 + b^5 + 6b^4 - 3b^3 - 5b^2 - b - 3 + b^{-1} + b^{-2}) / (b^4 - 1)^2$.
Proof. By formula (1) the search cost of linear search in an array of size n is $b^1(n) + b^2(n - 1) + \dots + b^n(1)$. Factoring out b^n and writing $x = 1/b$ leaves the sum $n\, x^{n-1} + (n - 1)\, x^{n-2} + \dots + 1$, which is the derivative with respect to x of the geometric sum $x^n + x^{n-1} + \dots + x + 1 = (x^{n+1} - 1) / (x - 1)$. It follows, after some computation, that the expected cost of linear search is

$$(b^{n+2} - b\,(b - 1)\,n - b^2) / (b - 1)^2 ,$$

from which the formula in part (a) follows immediately. In the semi-linear case, part of the proof involves summing a finite sum that begins $4b^{n-2} + 6b^{n-4} + 8b^{n-6} + \dots$. This can be accomplished by factoring out b^{n+1} and then writing $x = 1/b$ to produce the sum $4x^3 + 6x^5 + 8b^7 + \dots$, which is the derivative of a finite geometric sum. It is best to work out the case where n is odd separately from the case where n is even. The proof in the quarter-linear case is similar. We suppress the details, which are tedious but not difficult. □

We are now in a position to compare binary, linear, semi-linear, and quarter-linear search costs with each other and with optimal search costs. We begin by reporting that algebraic comparison of the expressions for L_b, S_b, and Q_b in Theorem 6 shows that Q_b is the smallest of the three when $1 < b < 1.6714$, that S_b is smallest when $1.6715 < b < 3.2143$, and that L_b is smallest when $b > 3.2144$. Thus we see that for all large values of n, quarter-linear search has smaller expected cost than semi-linear or liner search when $b < 1.7$ (roughly), semi-linear search is preferable when $1.7 < b < 3.2$, and linear search is better than the other two strategies when $b > 3.2$. How do these coefficients compare with the dominant coefficients for binary search derived in Theorems 4 and 5? When $1 < b < 1.182$ we have $\gamma_b < Q_b$, and thus in this range binary search is to be preferred over the three strategies of Proposition 4 when n is large. (The inequality $\gamma_b < Q_b$ was verified numerically rather than algebraically: γ_b and Q_b were computed for values of b spaced 0.001 apart; the value of M used to obtain γ_b was 128.) When

$b > 1.243$ we have $\alpha_b > Q_b$, and thus in this range binary search is always inferior to one of the three strategies of Theorem 6. When $1.183 < b < 1.242$ we have $\lambda_b < Q_b < \mu_b$, in the notation of Theorem 5. Thus, for each fixed b in this range there are always infinitely many values of n for which binary search is better than quarter-linear, and infinitely many values of n for which binary search is worse than quarter-linear.

It remains to compare these coefficients with the values of σ_b obtained from Theorem 3. The results of the relevant comparisons are as follows.

(a) If $1 < b < 1.182$, then $\gamma_b < 1.045\ \sigma_b$, so binary search cost exceeds optimal search cost by no more than 4.5 percent.

(b) If $1.183 < b < 1.242$, then $Q_b < 1.045\ \sigma_b$, so quarter-linear search cost exceeds optimal by at most 4.5 percent. Also, $\gamma_b < 1.047\ \sigma_b$, so binary search is competetive with quarter-linear search.

(c) If $1.243 < b < 1.6714$, then $Q_b < 1.021\ \sigma_b$, so quarter-linear search cost exceeds optimal by at most 2.1 percent. When $b = 1.67$ we have $\mu_b \approx 1.05\ \sigma_b$, which means that binary search is somewhat inferior.

(d) If $1.6715 < b < 3.2143$, then $S_b < 1.01\ \sigma_b$, so semi-linear search cost is within one percent of optimal. When $b = 3.21$ we have $\mu_b \approx 1.08\ \sigma_b$, which means that binary search is quite inferior.

(e) If $b > 3.2144$, then $L_b < 1.005\ \sigma_b$, so linear search cost is very nearly optimal.

4. Minimax-optimal strategies. Now consider the problem of finding strategies that minimize the maximum possible cost of an array search. The problem seems initially to be equivalent to finding search trees $T_{1,n}$ that minimize the quantity

$$\max \{ \Sigma_{k \in \Pi} P(k) \ : \ \Pi \text{ is a path in } T_{1,n} \}$$

(throughout this section, the word **path** in a search tree will refer to a list of node labels along a branch starting at the root and ending at a leaf). Actually, we demand more of a "minimax-optimal strategy": in the corresponding search tree, every subtree must be minimax-optimal. Thus, for example, if the minimax-optimal search tree has its most costly path in the right half of the tree, the left subtree of the root must nevertheless be a minimax-optimal search tree so that if an actual search leads into that part of the tree, it will not be unnecessarily expensive. As it turns out, minimax-optimal strategies in the cases we are considering are totally different from the strategies that minimize expected costs. Instead of probing near the left end of the array, one probes near the right end. For example, as we shall see, "reverse semi-linear" trees of the form shown in Figure 4 are optimal when $P(k)$ is a fast growing function. A **reverse semi-linear binary tree** with n nodes is defined recursively as follows: if $n = 0$, the tree is empty; if $n = 1$ the tree consists of a single node; if $n > 1$ the root of the tree has a right subtree of weight 1, and the left subtree is the reverse semi-linear tree with $n - 2$ nodes. In what follows, we use the letter R to denote reverse semi-linear binary trees; then $R_{1+t,n+t}$ denotes an n-node reverse semi-linear binary search tree with labels $1+t, \ldots, n+t$.

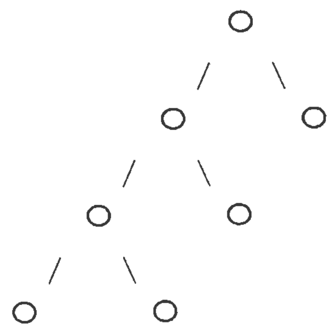


Figure 4. Reverse semi-linear binary tree with 7 nodes.

THEOREM 7. Let $P(k)$ be any function which grows so fast that $P(n) \geq P(n-2) + P(n-3)$ for all $n \geq 3$. (In particular, this applies to penalty functions of the form $P(k) = (k+t)!$, where t is a non-negative integer, and to penalty functions of the form $P(k) = b^k$, where $b \geq 1.325$, the root, approximately, of $b^3 = b + 1$.) Then for all n the tree $R_{1,n}$ is minimax-optimal, and if $n > 1$, then the path $(n-1, n)$ is a most costly path. If $P(n) > P(n-2) + P(n-3)$ for all $n \geq 3$, then these reverse semi-linear trees are the unique minimax-optimal trees.

Proof. We use induction. The theorem is easily seen to be true when $n = 0, 1, 2,$ or 3 . Now take any integer m such that the theorem is true for $n = 0,1,2, \ldots, m-1$, where $m \geq 4$. To see that the path $(m-1, m)$ is a most costly path in $R_{1,m}$, observe that the left subtree of $R_{1,m}$ is a reverse semi-linear tree with $m-2$ nodes and with a most costly path $(m-3, m-2)$, so it suffices to prove that the cost of the path $(m-1, m)$ in $R_{1,m}$ is at least as large as the cost of the path $(m-1, m-3, m-2)$. This is true because of the hypothesis $P(m) \geq P(m-2) + P(m-3)$ of the theorem. It now remains to prove that all other m-node search trees besides $R_{1,m}$ have paths whose costs are at least $P(m) + P(m-1)$. But this is obvious, because in every m-node search tree there is a path that contains nodes m and $m-1$. We leave the uniqueness claim in the final sentence of the theorem as a simple exercise. $\square$

Theorem 7 takes care of factorial penalty functions and many exponential ones. When $P(k) = b^k$ and b is a little smaller than 1.325, a reverse semi-linear tree is still minimax-optimal, as we shall see. However, the most costly path goes down to the left instead of to the right, although at the bottom it veers off to the right. For example, if the tree in Figure 4 is made into a search tree with labels $1, \ldots, 7$, then the most costly path would be $(6, 4, 2, 3)$.

THEOREM 8. Let $P(k) = b^k$, where $1.237 \le b < 1.325$. Then for all n the tree $R_{1,n}$ is minimax-optimal, and for $n \ge 4$ the path $(n - 1, n - 3, ..., 1+ (n \bmod 2), 2+ (n \bmod 2))$ is a most costly path. (The numbers 1.237 and 1.325 are the approximate roots of the equations $b^5 = b^3 + 1$ and $b^3 = b + 1$ respectively.)

Proof. We use induction, with base cases $n = 1, ..., 5$, for which the theorem is checked by examining all possibilities, many of which can be ruled out by use of the following three facts.
(a) When $n \ge 3$, the root of a minimax-optimal tree cannot be 1 (any search tree T with root 1 can be improved by moving the 1 to a position beneath 2 if 2 is not on a most costly path in T, or by moving 2 to the root position and putting 1 to its left if 2 is on a most costly path in T).
(b) When $n \ge 3$, the root of a minimax-optimal tree cannot be n (any search tree T with root n can be improved by lifting $n - 1$ to the root position and putting n to its right).
(c) If $1 \le b < 1.325$, then $b^3 - b - 1 < 0$; if $b \ge 1.237$, then $b^5 - b^3 - 1 \ge 0$.

Now suppose that the theorem holds for $n = 1,2, ..., m - 1$, where $m \ge 6$. We must prove that $R_{1,m}$ is minimax-optimal and that $(m - 1, m - 3, ..., 1 + (m \bmod 2), 2 + (m \bmod 2))$ is a most costly path. A routine computation involving a finite geometric sum shows that the cost of that path in $R_{1,m}$ is

$$\frac{b^{m+1} + (b^3 - b - 1)b^{1+(m \bmod 2)}}{b^2 - 1}.$$

Since $b^3 - b - 1 < 0$ because $b < 1.325$, it follows that this cost is less than $b^{m+1}/(b^2 - 1)$. The assertion concerning the most costly path is proved by noting that the path $(m - 1, m - 3, m - 2)$ in $R_{1,m}$ is more costly than the path $(m - 1, m)$ because $b^3 - b - 1 < 0$, and once we know that the most costly path enters the left subtree, induction does the rest. To prove that $R_{1,m}$ is minimax-optimal, take any *optimal* m-node search tree $T_{1,m}$ and denote its root by r. By observation (b) above, $r \neq m$. Suppose $r = m - 1$; then $T_{1,m}$ cannot be optimal unless its left subtree is optimal, so by induction its left subtree has the same minimax search cost as $R_{1,m-2}$, and thus $T_{1,m}$ has the same cost as $R_{1,m}$. Next, suppose $r = m - 2$; then it is easy to see that $T_{1,m}$ can be improved by moving that root down into the right subtree and replacing it by $m - 3$, which contradicts the optimality of $T_{1,m}$. Next, suppose $r = m - 3$; then $T_{1,m}$ contains the path $(m - 3, m - 1, m)$, and it is easy to check (using $b^5 - b^3 - 1 \ge 0$) that its cost is at least as large as the upper bound $b^{m+1}/(b^2 - 1)$ for the cost of the worst path in $R_{1,m}$, so $T_{1,m}$ cannot be strictly better than $R_{1,m}$. Finally, suppose $r \le m - 4$. Then the right subtree of $T_{1,m}$ is an optimal tree of weight $m - r \ge 4$, so by induction it can, without loss of generality, be assumed to be of the form $R_{r+1,m}$. Then the same argument that showed that r cannot be $m - 2$ shows that r cannot be $m - e$, where e is an even integer, so $m - r$ is odd. Then by induction the cost of the path in $T_{1,m}$ that starts at r and then follows the most costly path in the right subtree $R_{r+1,m}$ is given by

$$b^r + b^r \frac{b^{m-r+1} + (b^3 - b - 1)b^2}{b^2 - 1}$$

A little algebra shows that this is at least as large as the upper bound $b^{m+1}/(b^2 - 1)$ for the cost of the worst path in $R_{1,m}$; the verification uses the fact that $b^5 - b^3 - 1 \geq 0$. □

For still smaller values of b the minimax-optimal trees are, generally speaking, less predictable. We have proved, however, that if $1.089 \leq b \leq 1.203$, then the minimax-optimal trees of size $n \geq 8$ all have root $n - 3$, which makes them easy to construct recursively. (The numbers 1.089 and 1.203 are the roots of $b^7 + b^6 = b^3 + b^2 + 1$ and $b^7 = b^2 + b + 1$ respectively.) To shorten the paper we omit the proof, which runs along the same lines as that of Theorem 8. Details are available on request. We have not been able to prove any useful theorems about the exact minimax-optimal trees for $1.203 \leq b \leq 1.237$ or for $b \leq 1.089$

Acknowledgement. The authors thank Ed Reingold for relaying to them the problem encountered by Steiglitz and Parks.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Massachusetts, 1974.

[2] W. J. KNIGHT, *Search in an ordered array having variable probe cost*, SIAM J. Comput., vol. 17, no. 6, December 1988, pp. 1203-1214.

[3] E. M. REINGOLD AND W. J. HANSEN, *Data Structures*, Little and Brown, Boston, 1983.

[4] K. STEIGLITZ AND T. W. PARKS, *What is the Filter-Design Problem?*, Proc. 1986 Princeton Conference on Information Sciences and Systems, Princeton, NJ.