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

MARY V. CONNOLLY* and WILLIAM J. KNIGHT‡

Abstract. We consider 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 [3] have described a filter-design problem that turns out to be equivalent to the problem of searching an ordered array of n elements in which a probe into the k-th array location has an associated cost that increases as k increases. The equivalence of the two problems is explained in [1]. That paper confirmed a conjecture of Steiglitz and Parks to the effect that although binary search might be expected to perform less well, when measured by expected search cost, than search strategies that probe to the left of the center of the remaining array at each step of the search, 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 assume we confront the situation encountered by Steiglitz and Parks [3], 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

* 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.

Scan A6696

Connolly & Knight

Preprint

circa 1991

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 ways to decide whether one strategy is better than 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.

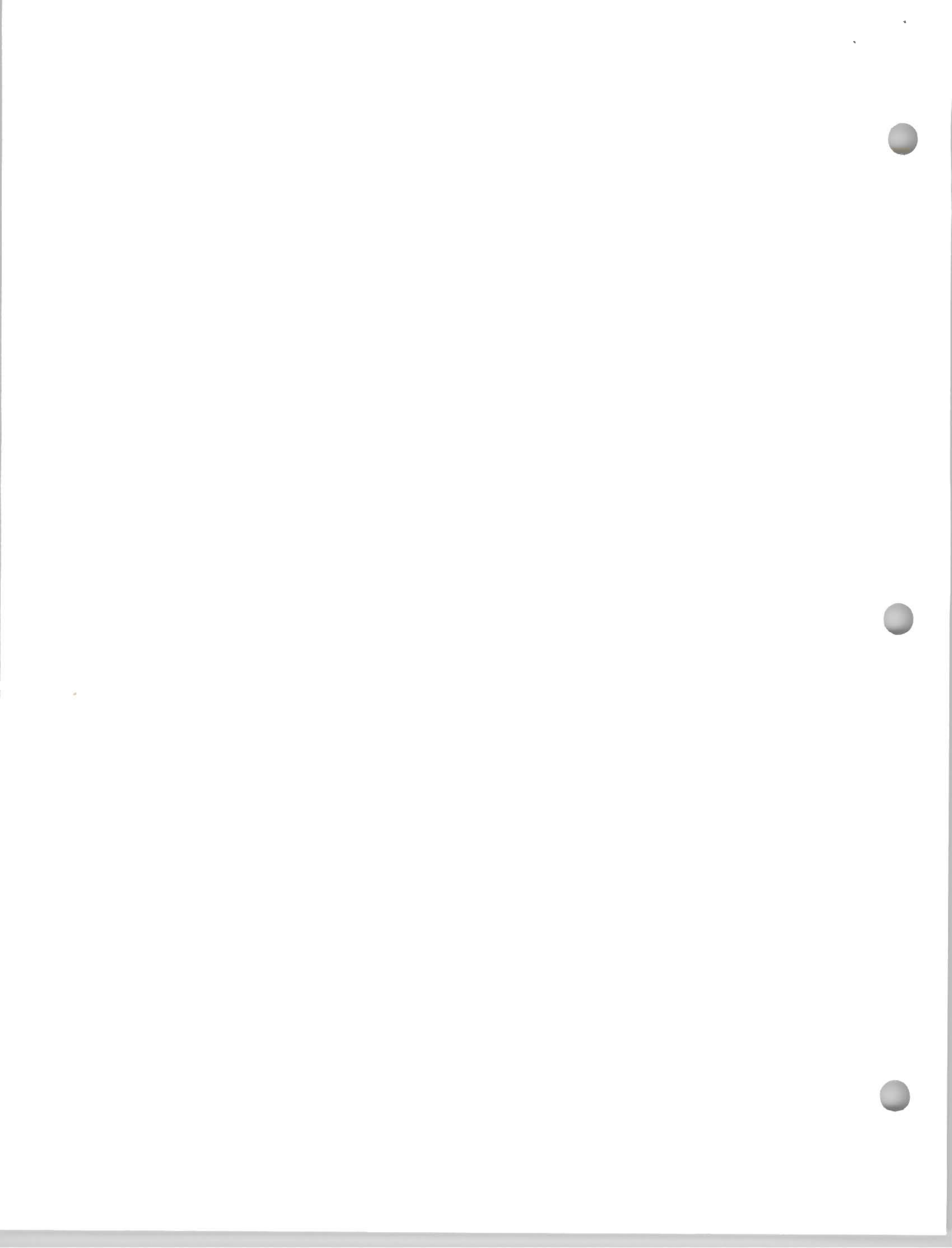
Every search strategy in an array of length n corresponds to a unique binary tree T_n with node labels $1, 2, ..., n$, placed so that the inorder traversal of T_n finds the labels in increasing order. Conversely, every such binary tree corresponds to exactly one search strategy. We shall call any such tree a search tree. The root of the tree gives the location of the first probe into the array. We go left if the probe results in a "yes", right if we get a "no", and repeat the process with the corresponding subtree. We ~~cannot stop~~ continue until we reach one of the $n + 1$ *external nodes* (see [2, p. 239]), at which point we can identify the location of the first "yes". Thus the search strategy prescribed by the tree corresponds to an *unsuccessful search* of a conventional array in which records with keys have been stored in increasing key order (see [2, p. 237]).

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 $1/(n + 1)$. It follows that the expected cost of an unsuccessful search using a search tree T_n is given by

$$(1) \qquad \sum_{j=1}^{n+1} \frac{1}{n+1} \left\{ \sum_{\text{nodes } k \text{ from root to j-th external node}} P(k) \right\}$$

where the inner sum in (1) is taken over all internal nodes k that lie on the path in T_n from the root down to the j-th external node. We can write (1) in a more useful form by letting $W_k(T_n)$ denote the number of internal nodes in that subtree of T_n whose root label is k. The number of internal nodes in an extended binary tree is one less than the number of external nodes. It follows that $W_k(T_n)$ is one less than the number of times that the term $P(k)$ will occur when the value of expression (1) is computed. Thus the expected cost of a search, using T_n, is equal to

$$\frac{1}{n+1} \sum_{k=1}^{n} P(k) [W_k(T_n) + 1] = \frac{1}{n+1} \sum_{k=1}^{n} P(k) W_k(T_n) + \frac{1}{n+1} \sum_{k=1}^{n} P(k) .$$



For computational convenience we discard the constant factor $1/(n+1)$ and the constant term $\Sigma\, P(k)$ and we call the remaining sum the *search cost of* T_n for $P(k)$, or sometimes the *search cost of the strategy corresponding to* T_n. We denote this by $S_{P(k)}(T_n)$. That is,

$$S_{P(k)}(T_n) = \sum_{k=1}^{n} P(k)\, W_k(T_n) \quad (2)$$

There is a recursion formula for $S_{P(k)}(T_n)$ that is essential both for proving theorems and for computer searches. Before deriving it, we find it useful to introduce a little more notation: if T_n denotes a search tree with n nodes, then $t + T_n$ will denote the same tree with all node labels increased by t (see Figure 1). Now observe

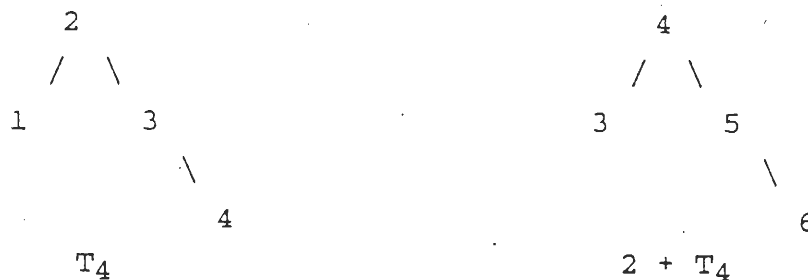


Figure 1

that if r denotes the root of T_n, then the left subtree at r is a search tree with $r - 1$ nodes, and we shall denote it by T_{r-1}, but the right subtree is a not a search tree. Instead, the right subtree has the form $r + T_{n-r}$, where T_{n-r} is a search tree with $n - r$ nodes. Then by splitting the sum in (2) into three pieces corresponding to the root, the left subtree, and the right subtree, we obtain the ~~perfectly general~~ recursion formula

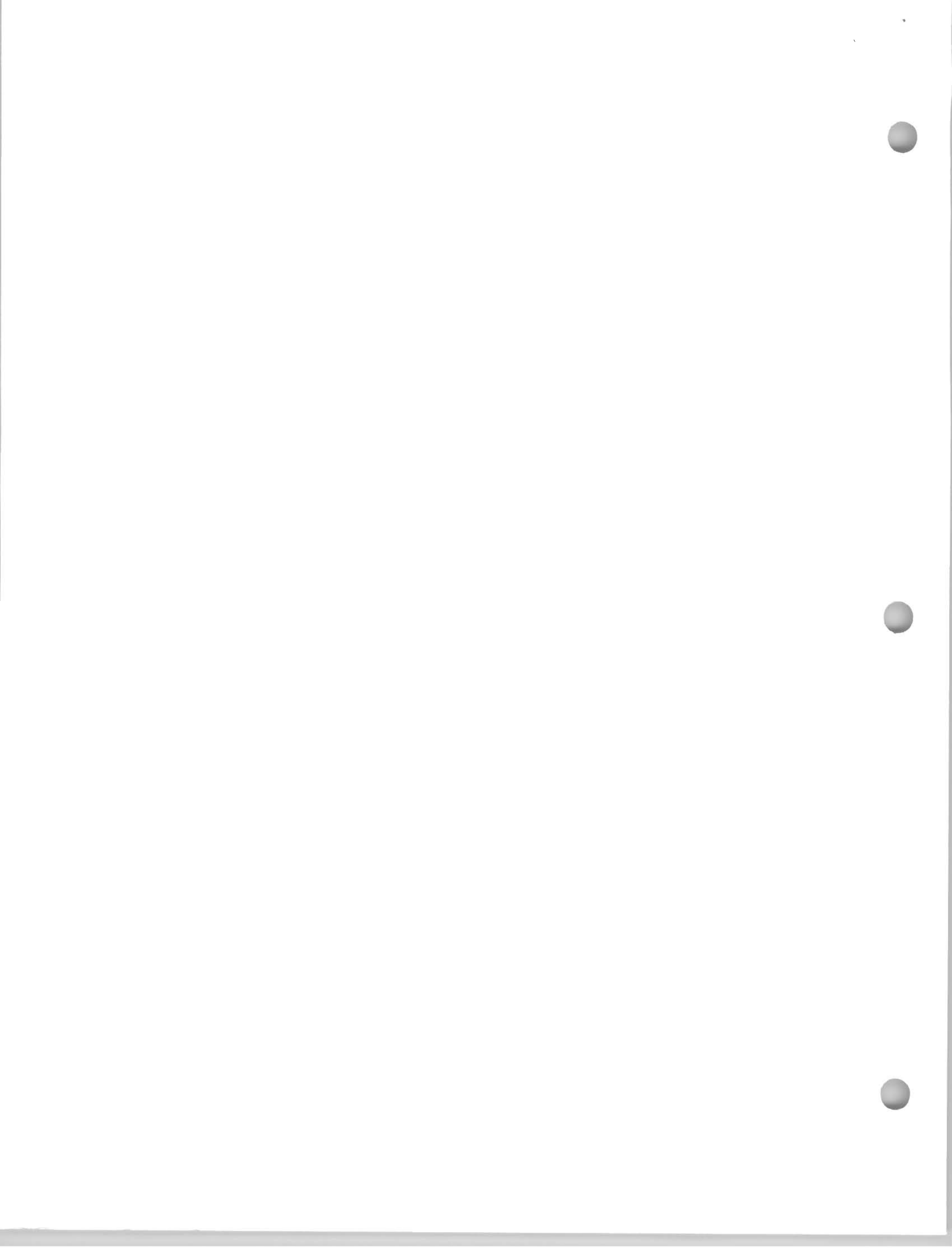
$$S_{P(k)}(T_n) = P(r)\, n + S_{P(k)}(T_{r-1}) + \sum_{k=r+1}^{n} P(k)\, W_k(r + T_{n-r}) \quad (3)$$

We can simplify the appearance of (3) by generalizing our notation slightly to allow a "translation parameter": if T is a search tree with m nodes, then for all non-negative integers t we write

$$S_{P(k)}(t + T) = \sum_{k=t+1}^{t+m} P(k)\, W_k(t + T) = \sum_{k=1}^{m} P(t+k)\, W_k(T) \quad (4)$$

Then (3) can be written as

$$S_{P(k)}(T_n) = P(r)\, n + S_{P(k)}(T_{r-1}) + S_{P(k)}(r + T_{n-r}) \quad (5)$$



and, more importantly, generalized in an obvious way to the form

$$(6) \qquad S_{P(k)}(t + T_n) = P(t+r)\, n + S_{P(k)}(t + T_{r-1}) + S_{P(k)}(t + r + T_{n-r}) \,.$$

If we let $S^*_{P(k)}(n, t)$ denote the minimal possible value of $S_{P(k)}(t + T_n)$ as T_n ranges over all search trees with n nodes, then it is easy to verify that (6) implies

$$(7) \qquad S^*_{P(k)}(n, t) = \min_{1 \le r \le n} \{ P(t+r)\, n + S^*_{P(k)}(r-1, t) + S^*_{P(k)}(n-r, t+r) \} \,,$$

and that the values of r that produce the minimum in (7) are the roots of optimal search trees for the prescribed values of n and t. Note that a search tree T_n may be optimal for one value of t, yet fail to be optimal for a different value of t. For example, when $P(k) = k!$ and $n = 4$ the tree T_4 shown in Figure 1 is optimal for $t = 0$ but not for $t = 2$. Or, to put it differently, when $P(k) = k!$ the tree T_4 is optimal among all 4-node search trees T, but $2 + T_4$ is not optimal among all 4-node trees $2 + T$. These assertions can be verified simply by examining all 14 possible 4-node search trees T. *why not give the optimal tree for 2+T ?*

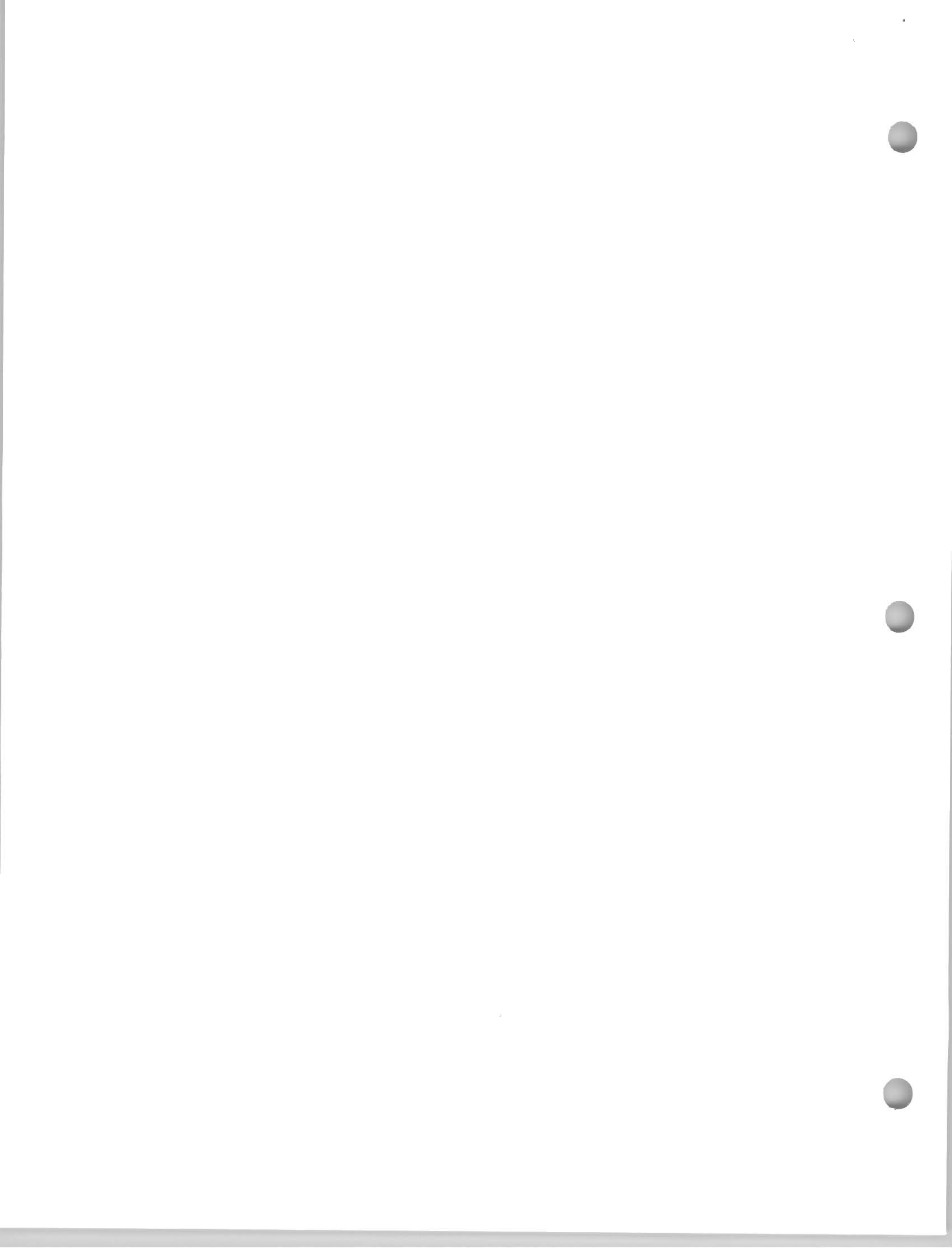
Generally, of course, what we really want is the number $S^*_{P(k)}(n, 0)$, which we shall abbreviate as $S^*_{P(k)}(n)$, for a prescribed P(k) and n, as well as complete information concerning at least one optimal search tree whose search cost is this optimal number. Recurrence relation (7), together with the fact that $S^*_{P(k)}(0, t) = 0$ for all t, makes it possible for us to compute all this information by computing a triangular array of optimal costs and roots for all pairs (k, t) satisfying $k \ge 0$, $t \ge 0$, $k + t \le n$. The following theorem drastically limits the number of values of r that must be examined in formula (7) for each n and t when P(k) is a fast growing function. *How about saying the words "dynamic programming" ?*

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 (7) must satisfy the inequality

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

Proof. Since most of the proofs in this paper are somewhat long and technical, we have chosen to place them all in a separate section at the end of the paper. Please see the last section.

not a good idea!



	$P(k) = 1.5^k$		$P(k) = 2^k$		$P(k) = 3^k$		$P(k) = k!$	
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

When it is deemed necessary, within a computer program, to have a search guided by an exact optimal search tree T for a prescribed integer n and penalty function $P(k)$, then T can be constructed by the following algorithm, which uses two triangular arrays $S(k, t)$ and $R(k, t)$, where $k \geq 0$, $t \geq 0$, $k + t \leq n$. $S(k, t)$ will be the search cost of an optimal tree with k nodes, and $R(k, t)$ will the the smallest root possible for such a tree.

A. For $t = 0, ..., n$, initialize $S(0, t)$ to zero. (The values of $R(0, t)$ are never used.)

B. For $k = 1, ..., n$ do

for $t = 0, ..., n - k$ do

$$S(k, t) \leftarrow \min \{ P(t+r)k + S(r-1, t) + S(n-r, t+r) \; : \; 1 \leq r \leq n, \; \frac{P(r+t)}{r} \leq \frac{n+1}{2} P(1+t) \};$$

$R(k, t) \leftarrow$ the smallest value of r that minimizes the expression above.

C. Invoke the following recursive procedure with T empty, $k = n$, and $t = 0$:

```
procedure BuildSearchTree (T, k, t)    {T : reference parameter; k , t : value parameters}
    if k > 0 then
        create a root node for T ;
        label the root node R(k, t);
        BuildSearchTree (Left (T), R(k, t) - 1, t) ;
        BuildSearchTree (Right (T), k - R(k, t), t + R(k, t)) .
```

We omit the induction proof that the resulting tree T is an n-node search tree and is optimal for $P(k)$. Note that the arrays $S(k, t)$ and $R(k, t)$ can be discarded after steps B and C respectively. Steps A and B were used to calculate the $S^*_{k!}(n)$ values shown in Table 1, but with the modification that $R(k, t)$ was made to be a list of *all* optimal roots.

Since it can be time and space consuming to construct exact optimal search trees to guide a search, it is reasonable to seek simple search trees (i.e. strategies) that can be proved to be nearly optimal. In this paper we have done this for penalty functions that grow exponentially or factorially. In particular, we find 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. These assertions are based on the following theorem.

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

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

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

(b) The search cost of a linear search in such an array is less than

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

(c) Only when n is 4 or 5 does the cost of linear search differ from the optimal by more than one percent. When $n > 10$, it differs by less than one one-hundredth of a percent.

(d) For infinitely many values of n the search cost of a binary search in an array of size n exceeds

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

Proof. See the last section.

Parts (b) and (d) show that the cost of binary search infinitely often has a significantly larger second-highest order term than the cost of linear search. Thus, although the highest order term 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.

Consider now 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 (4) that

$$S_{b^k}(t + T_n) = b^t S_{b^k}(T_n)\,. \tag{8}$$

This equation has the happy consequence that when $P(k) = b^k$ for some constant b, then a search tree T_n is optimal for one value of t if and only if it is optimal for all values of t (cf. the comments following equation (7)). Equation (5) now takes the form

$$S_{b^k}(T_n) = b^r n + S_{b^k}(T_{r-1}) + b^r S_{b^k}(T_{n-r})\,, \tag{9}$$

and (7) can be replaced by the simpler equation

$$S^*_{b^k}(n) = \min_{1 \le r \le n} \{ b^r n + S^*_{b^k}(r-1) + b^r S^*_{b^k}(n-r) \}\,. \tag{10}$$

Again, values of r that produce the minimum in (10) are roots of optimal n-node search trees.

It is now easy to write a program to find, for each n, the value of $S^*_{b^k}(n)$ and the roots of all optimal trees. Table 1 shows some of this information for three different values of b.

Table 2 shows the formulas for $S^*_{b^k}(n)$ for $n = 0,1,\ldots,7$ and all $b > 1$. Using (9) 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 $n = 7$.

as opposed to ?

Again we have investigated trees that can be shown to have search costs that are very nearly optimal. Here is a summary of our results for penalty functions of the form $P(k) = b^k$. Proofs of these assertions can be found in the last section of the paper.

(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	$S_b^*(n)$		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 you begin at the left and probe in every fourth location moving to the right until you find you 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.

4. Minimax strategies. Now consider the problem of finding strategies that minimize the maximum possible cost of an array search. Actually, we demand more of a "minimax strategy": in the corresponding search tree, every subtree must be optimal. Thus, for example, if the optimal search tree has its most costly path in the right half of the tree, the left subtree of the root must nevertheless be an optimal minimax 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, optimal minimax 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 2 are optimal when $P(k)$ is a fast growing function. A **reverse semi-linear 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 is $n - 1$, the right child of the root is n, and the left subtree of the root is the reverse semi-linear tree having $n - 2$ nodes.

THEOREM 3. 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 reverse semi-linear tree 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 the reverse semi-linear trees are the unique minimax optimal trees.

Proof. See the last section.

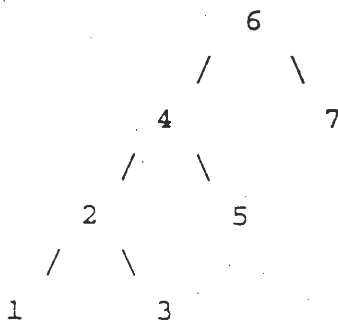


Figure 2. Reverse semi-linear tree for $n = 7$.

When $P(k) = b^k$ the reverse semi-linear trees are still optimal when b is a little smaller than 1.325, but as we shall see, the most costly path now goes down to the left instead of to the right, although at the bottom it veers off to the right. For example, in the tree in Figure 3 the most costly path would be (6, 4, 2, 3).

THEOREM 4. Let $P(k) = b^k$, where $1.237 \le b < 1.325$. Then for all n , the reverse semi-linear tree is minimax-optimal, and for $n \ge 4$ the path $(n - 1, n - 3, \ldots, 1 + (n \bmod 2), 2 + (n \bmod 2)$ 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. See the last section.

For still smaller values of b the minimax-optimal trees are, generally speaking, less predictable. It is possible to prove, however, that if $1.089 \le b \le 1.203$, then the minimax-optimal trees of size $n \ge 8$ 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.) We have not been able to prove any useful theorems about the exact minimax-optimal trees for $1.203 \le b \le 1.237$ or for $b \le 1.089$.

5. Proofs of the theorems.

Proof of Theorem 1. We use induction on n . If $n = 1$, then $r = 1$ and the inequality to be proved is trivial. Now suppose the theorem is true for $n = 1, 2, \ldots, m - 1$, where $m \ge 2$. We seek to prove that it is also true when $n = m$. It is trivially true if $r = 1$, so suppose $r > 1$. We know that r is the root of some optimal n-node search tree, say T_n^* , for the prescribed values of n and t . Let c denote the left child of r in T_n^*. Perform a simple tree rotation (see [2, p. 306]) to produce a new search tree having root c with right child r . Only two weights $W_k(T_n^*)$ have

changed: the weight on c in the new tree is m, whereas in T_n^* it was $r - 1$; and the new weight on r is $m - c$, whereas in T_n^* it was m. It follows that the net change in search cost in going from T_n^* to the new tree is

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

Since the new tree must have search cost at least as great as that of the optimal tree T_n^*, the expression above must be non-negative. From this we deduce the inequality

$$(11) \qquad P(r + t) \leq (m - r + 1) P(c + t) / c .$$

Now examine the subtree with root c in T_n^*. This subtree has $r - 1$ nodes, and it must itself be an optimal search tree, for if not then we could replace it in T_n^* by a search tree with lower search cost, and by (7) this would produce a better tree than the optimal T_n^*. By the induction hypothesis then, we must have $P(c + t) / c \leq ((r - 1) + 1) P(1 + t) / 2$. Combining this with (11) gives

$$P(r + t) \leq (m - r + 1) r P(1 + t) / 2 .$$

This can easily be seen to imply the inequality of the theorem. □

Before we prove Theorem 2 we state the following lemma.

LEMMA. For integers n we have the following three inequalities:

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

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

(14) $4(n - 4)! > 6(n - 5)! + 7(n - 6)! + ... + n(1)!$ when $n \geq 7$.

Proof. The induction proofs are straightforward. □

Proof of Theorem 2. Assume that $P(k) = k!$ and $n \geq 7$. Inequality (a) can be proved by proving it for all *optimal* trees. Take any optimal search tree T_n^* with $n \geq 7$ nodes. Then n must be a leaf of T_n^*, for otherwise we have the contradiction that the search cost of T_n^* is strictly greater than $2n!$, which by (12) would exceed linear search cost:

$$(15) \qquad 1(n)! + 2(n - 1)! + 3(n - 2)! + ... + n(1)! .$$

Then by the structure of a search tree, the only possible parent of n in T_n^* is $n - 1$. The left subtree on $n - 1$ must be empty, for if not then the weight of the subtree on $n - 1$ is at least 3,

and thus the cost of T_n^* is at least $n! + 3(n-1)! + (n-2)! + (n-3)! + \dots + 1!$, which by (12), with n replaced by $n-1$, exceeds linear search cost (15). Then by the structure of a search tree, the only possible parent of $n-1$ in T_n^* is $n-2$. The left subtree on $n-2$ must also be empty, for if not then the weight of the subtree on $n-2$ would be at least 4, and thus the cost of T_n^* would be at least

$$n! + 2(n-1)! + 4(n-2)! + (n-3)! + \dots + 1!,$$

which by (13) exceeds linear search cost (15). Then by the structure of a search tree, the only possible parent of $n-2$ is $n-3$, which therefore has weight at least 4. The weight of the subtree on $n-4$ is at least 1, so the search cost of T_n^* exceeds $n! + 2(n-1)! + 3(n-2)! + 4(n-3)! + (n-4)!$, as stated in Theorem 2. To prove part (b) of Theorem 2, we derive an upper bound on linear search cost (15) by using (14) in the obvious way. Parts (c) and (d) are proved by using Table 1 for $n \leq 20$ (say) and then proving algebraically that the difference between the upper bound for linear search cost and the lower bound for optimal search cost is less than one one-hundredth of a percent of the lower bound when $n > 20$. □

We now begin our detailed discussion of optimal and near optimal expected cost search trees when $P(k) = b^k$ for some constant $b > 1$. Along the way we shall state and prove several useful auxilliary propositions. The first gives us a useful lower bound for the search cost $S_{b^k}^*(n)$ in optimal trees with n nodes. It is a bit unusual because formula (18) below assumes that we have used recurrence relation (10) to calculate $S_{b^k}^*(n)$ for $n = 0, \dots, M-1$ for some integer M. The calculations can be numerical, for a prescribed value of b (cf. Table 1), or algebraic (cf. Table 2).

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

$$(16) \qquad S_{b^k}^*(n) \;\geq\; \sigma_b b^n - n - \sigma_b/b - \tau_b \;=\; \sigma_b b^n + O(n).$$

Values for σ_b and τ_b can be computed as follows: fix any positive integer M; then let

$$(17) \quad \tau_b = \min\{ r + (M - r + 1)/b^r \;:\; r = 1, \dots, M \},$$

$$(18) \quad \sigma_b = \min\{ (S_{b^k}^*(n) + n + \tau_b) / (b^n - 1/b) \;:\; n = 0, \dots, M-1 \}.$$

Proof. Let $b > 1$ be given. Fix M. Define τ_b and σ_b by (17) and (18). Then (18) implies that for $n = 0, \dots, M-1$,

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

We shall now show by induction that (19), and hence (16), holds for all $n \geq 0$. Take any integer $m \geq M$ for which (19) holds for all $n \leq m - 1$. To prove that (19) holds when $n = m$, note that by formula (10),

$$\begin{aligned} S^*_{bk}(m) &= \min_{1 \leq r \leq m} \{\ b^r\, m + S^*_{bk}(r-1) + b^r\, S^*_{bk}(m-r)\ \} \\ &\geq \min_{1 \leq r \leq m} \{\ b^r\, m + \sigma_b\,(b^{r-1} - 1/b) - (r-1) - \tau_b \\ &\qquad + b^r\, \sigma_b\,(b^{m-r} - 1/b) - b^r\,(m-r) - b^r\, \tau_b\ \} \\ &= \sigma_b\,(b^m - 1/b) - m - \tau_b + \min_{1 \leq r \leq m} \{\, m - r + 1 + b^r\,(r - \tau_b)\,\}\ . \end{aligned}$$

This shows that (19) will hold when $n = m$ provided that for all $m \geq M$,

$$\min_{1 \leq r \leq m} \{\, m - r + 1 + b^r\,(r - \tau_b)\,\} \ \geq\ 0\ .$$

For this it would suffice to prove that for $r = 1, \ldots, M$ we have

$$M - r + 1 + b^r\,(r - \tau_b) \ \geq\ 0\ .$$

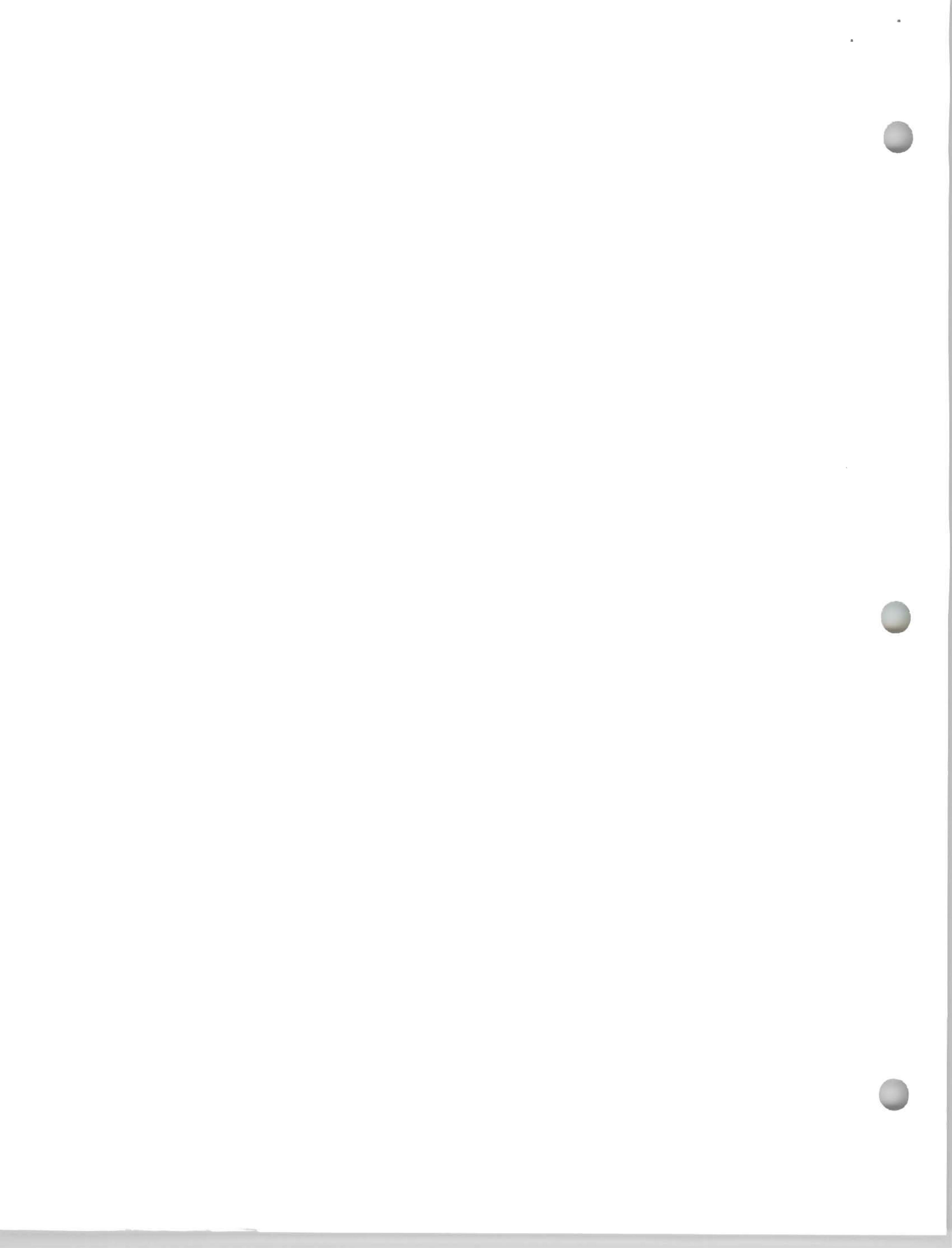
But this is true just by definition (17) of τ_b. This completes the proof that (19) holds for all n. □

Generally speaking, the larger the value one chooses for M in Proposition 1, the larger the value one is able to obtain for σ_b, although that is not invariably the case.

Next we estimate the cost of binary search. We shall let B_n denote the binary search tree corresponding to classical binary search. Its root, of course, is $\lfloor (1+n)/2 \rfloor$. The exact search costs $S_{bk}(B_n)$ 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 (8):

(20) $$S_{bk}(B_n) = \begin{cases} b^q\,(2q) + S_{bk}(B_{q-1}) + b^q\, S_{bk}(B_q) & \text{if } n = 2q\,, \\ \\ b^{q+1}\,(2q+1) + S_{bk}(B_q) + b^{q+1}\, S_{bk}(B_q) & \text{if } n = 2q+1\,. \end{cases}$$

Here we have used the fact that when $n = 2q$, the root of B_n is q and the left and right subtees are B_{q-1} and $q + B_q$, but when $n = 2q + 1$, the root is $q + 1$ and the subtrees are B_q and $q + B_q$. The next proposition assumes that (20) has been used to calculate $S_{bk}(B_n)$ for $n = 1, \ldots, 2M$ for some integer M.



PROPOSITION 2. 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 $n \geq 0$,

$$\alpha_b b^n - 2n - \beta_b \leq S_{b^k}(B_n) \leq \gamma_b b^n - 2n - \delta_b . \tag{21}$$

Values for α_b, β_b, γ_b, and δ_b can be computed as follows: fix any positive integer M and set

$$\alpha_b = \min \{ (S_{b^k}(B_n) + 2n) / (b^n - 1/b) : n = M, M+1, \ldots, 2M \} , \tag{22}$$

$$\varepsilon_b = \max \{ \alpha_b (b^n - 1/b) - 2n - S_{b^k}(B_n) : n = 0,1, \ldots, M-1 \} , \tag{23}$$

$$\beta_b = \alpha_b / b + \max \{ 0, \varepsilon_b \} , \tag{24}$$

$$\kappa_b = \max \{ 1 + 2/(e \ln b) , 2b/(e \ln b) \} , \tag{25}$$

$$\gamma_b = \max \{ (S_{b^k}(B_n) + 2n + \kappa_b) / (b^n - 1/b) : n = M, M+1, \ldots, 2M \} , \tag{26}$$

$$\zeta_b = \min \{ \gamma_b b^n - 2n - \gamma_b/b - \kappa_b - S_{b^k}(B_n) : n = 0,1, \ldots, M-1 \} , \tag{27}$$

$$\delta_b = \gamma_b/b + \kappa_b + \min \{ 0, \zeta_b \} . \tag{28}$$

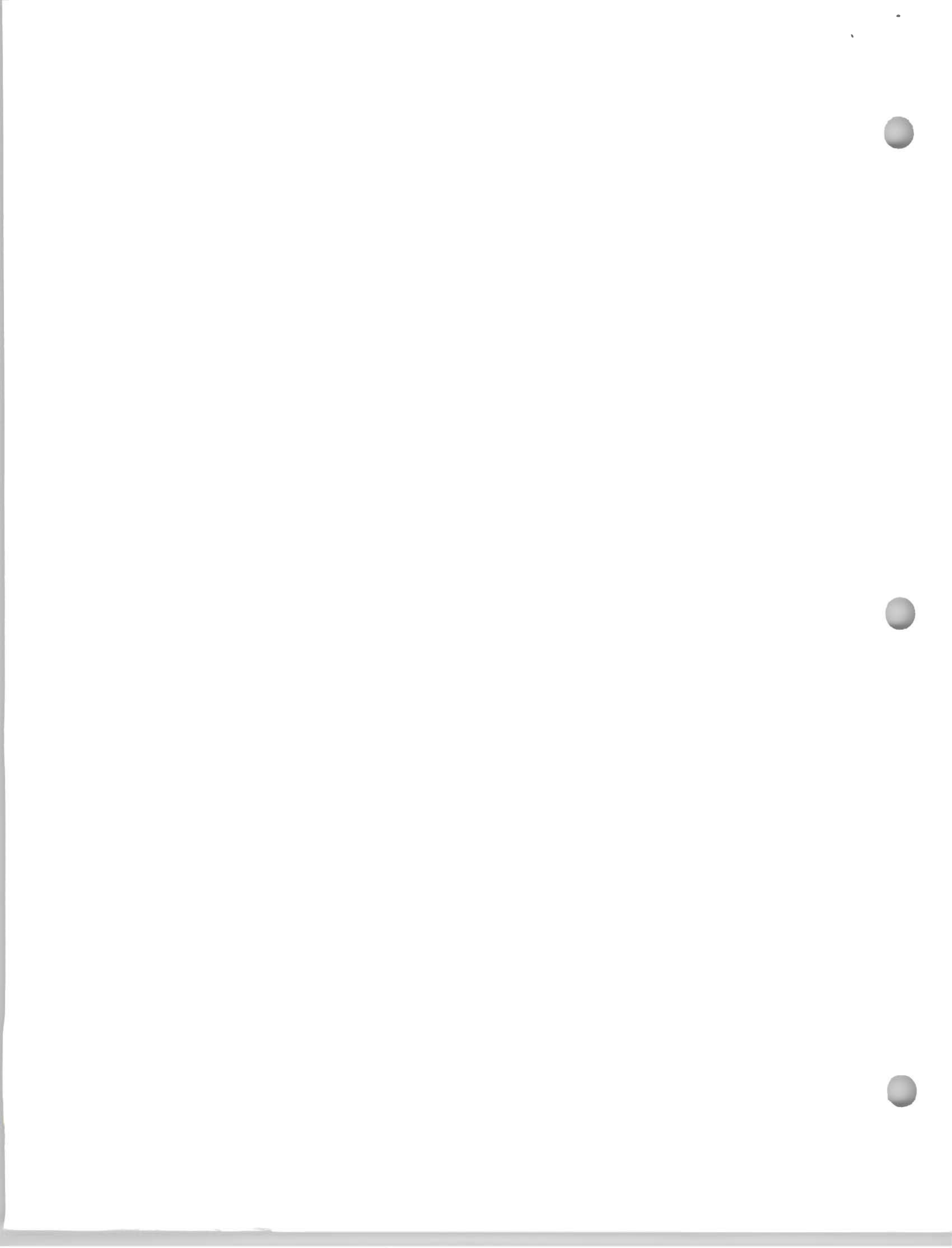
Proof. We begin by proving the right half of (21). Fix M, and let κ_b, γ_b, ζ_b, and δ_b be defined by (25), (26), (27), and (28). Then (26) implies that

$$S_{b^k}(B_n) \leq \gamma_b (b^n - 1/b) - 2n - \kappa_b \tag{29}$$

for all $n = M, M+1, \ldots, 2M$. We shall now show by induction that (29) holds for all $n \geq M$. The base cases are $n = M, M+1, \ldots, 2M$. Now take any integer $m > 2M$ such that (29) holds for $n = M, \ldots, m-1$. To prove that (29) holds for $n = m$, note that if m is odd, then by (20) and the inductive hypothesis we have

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

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 (20) and the inductive hypothesis,



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

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 T_m^*; the verification uses the fact that $b^5 - b^3 - 1 \geq 0$. □

In the interests of shortening the paper we omit the proof that when $1.089 \leq b \leq 1.203$ the minimax-optimal trees of size $n \geq 8$ all have root $n - 3$. The proof runs along the same lines as that of Theorem 4. Details are available on request.

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

REFERENCES

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

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

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