Search within a Page

H. R. STRONG

IBM Research Laboratory, San Jose, California

AND

G. MARKOWSKY AND A. K. CHANDRA

IBM Thomas J. Watson Research Center, Yorktown Heights, New York

ABSTRACT. Three families of strategies for organizing an index of ordered keys are investigated. It is assumed either that the index is small enough to fit in main memory or that some superstrategy organizes the index into pages and that search within a page is being studied. Examples of strategies within the three families are B-tree Search, Binary Search, and Square Root Search. The expected access times of these and other strategies are compared, and their relative merits in different indexing situations are discussed and conjectured on. Considering time and space costs and complexity of programming, it is concluded that a Binary Search strategy is generally preferable.

KEY WORDS AND PHRASES: B-tree, binary search, square root search, searching, index, access method

CR CATEGORIES: 3.71, 3.74, 5.29, 5.39

1. *Introduction*

In this paper we provide a theoretical framework allowing comparison of various strategies for indexing a data base. We investigate three families of strategies for organizing an index of ordered keys. Each strategy considered must support random accesses, inserts, and deletes, as well as order-dependent operations such as

RETRIEVE THE NEXT HIGHER KEY IN THE INDEX.

(In particular, this explicitly eliminates hashing strategies from consideration [3].) However, in this paper we concentrate almost exclusively on random access time. An *index entry* is assumed to be a pair consisting of a key and a pointer; but we make no assumption about other information which may be stored with an entry or about whether information in index entries is stored in a compressed form. We assume either that the index is small enough to fit in main memory or that some superstrategy organizes the index into pages and that we are studying search within a page. For example, the reader may assume that pages of the index are organized into a B-tree and that we are simply studying the organization of each page.

Our intention is to model each strategy sufficiently accurately to allow practical comparison while remaining machine and system independent. The level of precision we have chosen lies between counting key comparisons and counting machine instructions. We exhibit significant differences between strategies that do not show up at the level of key comparisons.

Authors' addresses: H. R. Strong, IBM Research Laboratory, San Jose, CA 95193; G. Markowsky and A. K. Chandra, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

We have named the families of strategies *Tree Search*, after the B-tree, *Root Search*, after Square Root Search, and *Binary Search*. Specific B-tree and Square Root Search strategies we consider are those used in VSAM (an IBM product: Virtual Storage Access Method). For this paper we use B-tree to refer to the VSAM modification of the B-tree strategy of [1] (called M-tree in [5]).

In the next section we describe our modeling technique by example. Then, in the following three sections we describe and analyze each of the three families of strategies. Section 5 also contains a brief discussion of radix or Patrician strategies [3, 10]. Section 6 is devoted to comparison with respect to random access time. Section 7 contains a discussion of space and other costs beyond the scope of the model presented in this paper. We conclude from this discussion that some form of Binary Search strategy is generally preferable to the strategies of the other two families for search within a page. This paper refines results of [2] together with those of [4].

2. *The Model*

We describe our modeling technique by presenting a paradigm for search strategies: *Sequential Search*. For Sequential Search, the index entries are totally ordered (from left to right) by increasing key order. Given a key, random access is achieved by searching each entry from left to right until one is found with a key greater than or equal to the given. Then its associated pointer is followed either to the data or to another page of the index. Random insertion (or deletion) into the index is achieved by locating the point of insert (delete) and moving the remaining entries to the right (left) a sufficient amount to allow the insert (delete). The index is ordered so that, once a particular entry has been located, locating the next higher key requires simply moving to the right.

Since we are only modeling random access in the index, the *program scheme* of Figure 1 and *organization graph* of Figure 2 serve to characterize our model of Sequential Search. Here we begin to introduce our notation. The program scheme is not to be thought of as the program implementing Sequential Search, but rather as a representation of the sequencing of operations which any program implementing Sequential Search must achieve. The nodes of the program scheme are commands, e.g., [FIND FIRST KEY], [COMPARE], [NEXT], or diamonds indicating a branch point. The arcs indicate sequencing. We often abbreviate commands by the first letters or their words. Thus, a typical sequence of operations for sequential search would be [FFK; C; N; C; N; C].

The program scheme represents a regular set of these expressions for sequences of operations. The organization graph offers an interpretation (semantics) for the commands and limits the number of possible sequences of operations. Since there are seven nodes (the *index size*) in Figure 2, the set of possible sequences corresponding to Figures 1 and 2 is {[FFK; C], [FFK; C; N; C], [FFK; C; N; C; N; C], [FFK; C; N; C; N; C; N; C], [FFK; C; N; C; N; C; N; C; N; C], [FFK; C; N; C; N; C; N; C; N; C; N; C], [FFK; C; N; C; N; C; N; C; N; C; N; C; N; C]}. In this paper we do not formally specify the interpre-

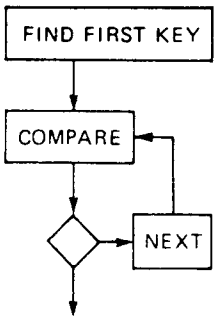


FIG. 1. Sequential Search program scheme

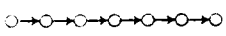


FIG. 2. Sequential Search organization graph

tation of a program scheme over an organization graph; we merely indicate that such a formal semantics is possible. We are correspondingly informal about the labeling of arcs and nodes in the organization graph, but we generally distinguish between two types of nodes: those corresponding to entries with associated pointers pointing out of the index (page), and those with local pointers. The former are called *goal nodes* and are represented by open circles; the latter are called *nongoal nodes* and are represented by solid circles. Sometimes the arcs are labeled to indicate to which commands they correspond. The *size* of an organization graph is the number of its goal nodes. This is the index size to which it corresponds. However, when the generalization is obvious, we use a single typical organization graph to indicate the set of all organizations (corresponding to all index sizes) for a particular strategy.

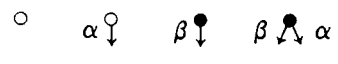
The basic *time unit* for time comparison of all of our strategies is the time taken to execute [N; C], the loop of Sequential Search. All other execution times are compared to this one. It turns out that every sequence of every strategy we study has [FFK; C] as a prefix. Moreover, it never appears except as a prefix. Since our purpose is performance comparison rather than absolute performance prediction, we allocate a time of 0 to the execution of [FFK; C]. We assume that *in a random access, each goal node is equally likely to be the goal of the search.* Thus, for an index of size N, the expected sequential search time is

$$SE(N) = \tfrac{1}{2}(N - 1). \tag{2.1}$$

Sequential Search is a specific strategy rather than a family of strategies. Its access time cost depends only on index size. It is, however, a degenerate case of each of the families of strategies we study, and its analysis forms a pattern for the other analyses.

3. *Tree Search*

The family of *Tree Search* strategies all possess the program scheme of Figure 3 and an organization graph which is a binary tree as in Figure 4. This family is distinguished from the other families we study by the existence of nongoal nodes in the organization graph. Each goal node has at most one child, this child being reachable by the [NEXT] command. A nongoal node may have two children, one reachable by [NEXT] and one by [FFKANL], which follows its local pointer. The loops of Figure 3 and corresponding arcs of Figure 4 have been labeled with α and β to show which loop follows which arcs. The four types of nodes of Tree Search organization graphs are as follows:



(i) (ii) (iii) (iv)

For the rest of this section, *tree* refers to a tree of nodes of types (i) through (iv). Nodes of type (i) are also called *leaves.*

We use the following notation:

t: a tree;
v: a node;
$s_\alpha(v)$: the α-successor of node v;
$s_\beta(v)$: the β-successor of node v;
$st_\alpha(v)$: the α-subtree of v;
$st_\beta(v)$: the β-subtree of v;
$v(v)$: the value of the key at v.

The final defining properties of the family of Tree Search strategies specify the order restrictions on keys associated with the organization graph:

$$v(v_1) < v(v_2) \quad \text{for } v_2 \text{ in } st_\alpha(v_1), \tag{3.1}$$
$$v(v_1) \geq v(v_2) \quad \text{for } v_2 \text{ in } st_\beta(v_1). \tag{3.2}$$

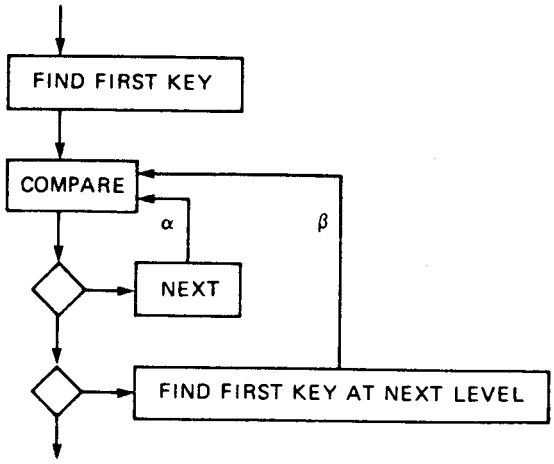


FIG. 3. Tree Search program scheme

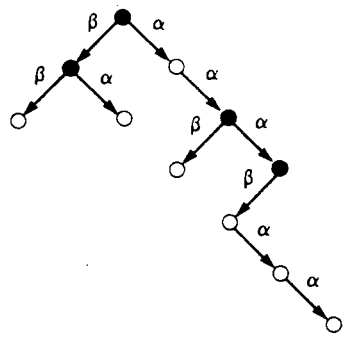


FIG. 4. Tree Search organization graph

With any organization tree satisfying (3.1) and (3.2), a decision to take the β-loop when the target key is less than or equal to the value of the key at the current node, and to take the α-loop otherwise, if possible, will leave the index (page) from the correct goal node. Random insertion is achieved by locating the point of insert and moving the remaining entries right as before. However, after such an insertion, local pointers must be updated. One efficient insertion method is to store the tree in the order of a depth first search with α-branches first. Then, for insertion, follow the algorithm for access and each time an α-branch is taken, update any corresponding β-pointer. Locating the next higher key requires keeping a trail during access so that the tree can be traversed to the next goal in the α-direction with backtracking as necessary.

In allocating time costs to Tree Search we depart from the usual count of key comparisons by allocating a cost of $\beta \geq 1$ to the execution of the β-loop [FFKANL; C]. The cost β is roughly the ratio of the cost of following a pointer to an entry and making a key comparison with the cost of reading the next entry and making a comparison. We make no assumption about β except that it is at least 1. We expect that for most implementations we will have $1 < \beta < 2$. For the purpose of computing an expected time cost for a Tree Search strategy, we define the *length* of a node to be the sum of the arc labels on the path from the root to the node, with $\alpha = 1$. The access time for a particular goal node is its length. The *path length* of a tree is the sum of the lengths of its goal nodes. Thus *the expected access time is just the path length divided by the size.*

Before discussing optimal Tree Search strategies we will analyze an important subfamily: the family of B-tree Search strategies. These are the strategies whose organization trees can be viewed as B-trees by grouping maximal α-chains together. We call these strategies B-tree strategies because they look like B-trees and some of the analysis here will apply to B-trees of pages, with, e.g., β as the expected cost of a page reference. However, these are not B-trees in the sense that they use a different insertion algorithm and free space is not distributed. We modify the requirements of B-trees slightly and require that all maximal α-chains be equal in length (except for the leaf chains), that such chains contain either all goal nodes or all nongoal nodes, and that the goal chains all be at a uniform *depth.* The fixed length of chain will be called the *order.* A B-tree of order 3 and depth 3 is displayed in Figure 5. A B-tree of order m and depth d partitions the goal nodes into m^{d-1} sections each to be searched sequentially. Specifying integers m (≥ 2) and d (≥ 1) fixes a unique strategy with expected random access time approximately

$$BT_{m,d}(N) = (d-1)SE(m) + SE(N/m^{d-1}) + (d-1)\beta, \tag{3.3}$$

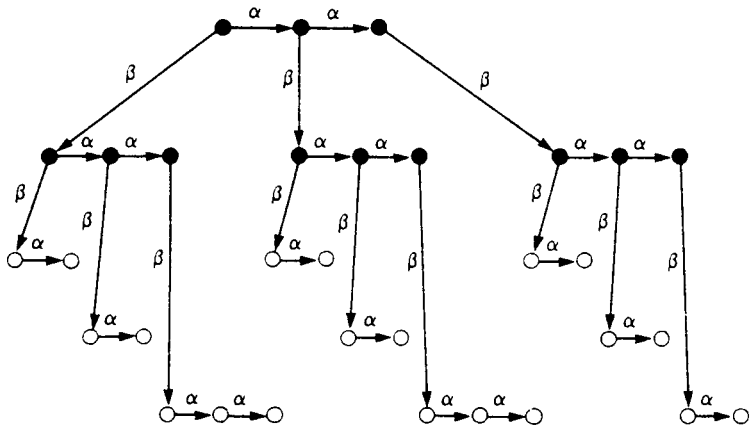


FIG. 5. B-tree of size 20, order 3, and depth 3

or

$$BT_{m,d}(N) = \tfrac{1}{2}(d-1)(m-1) + \tfrac{1}{2}((N/m^{d-1}) - 1) + (d-1)\beta. \tag{3.4}$$

This value is slightly lower than what would occur in practice since we assume an exactly even distribution to the leaf sections while we are not using the balancing properties of the insertion algorithm for real B-trees, but rather depending on the randomness of inserts for even distribution. But we will make similar evenness assumptions for the other strategies so that comparability is preserved.

Besides the evenness assumption, (3.3) and (3.4) involve another approximation, because actual α-chains must have integer lengths. It is convenient for analysis and comparison to use these approximate forms, and the results differ by less than one-fourth as is shown in the following lemma:

LEMMA 3.1. *If one of a number of sections of length m_1 or m_2 ($m_2 = m_1 + 1$) is to be searched sequentially and the probability of choosing a section is proportional to its length, then the difference between the expected search time and $SE(m)$, where m is the average length, is less than one-fourth.*

PROOF. Assume we have k_1 sections of length m_1 and k_2 sections of length $m_2 = m_1 + 1$, so that $N = k_1 m_1 + k_2 m_2$ and $k_1, k_2, m_1 \geq 1$. Let E be the expected search time. Then

$$E = SE(m_1)(k_1 m_1/N) + SE(m_2)(k_2 m_2/N),$$

while

$$SE(m) = SE(N/(k_1 + k_2)).$$

Expanding, we have

$$E = \tfrac{1}{2}[(k_1 m_1^2/N) + (k_2 m_2^2/N) - 1],$$

$$SE(m) = \tfrac{1}{2}[(k_1 m_1/(k_1 + k_2)) + (k_2 m_2/(k_1 + k_2)) - 1],$$

and

$$\begin{aligned} E - SE(m) &= \frac{1}{2}\frac{k_1 k_2}{N(k_1 + k_2)} \\ &= \frac{1}{2}\frac{k_1 k_2}{(k_1 m_1 + k_2 m_2)(k_1 + k_2)} \\ &= \frac{1}{2}\cdot\frac{\max(k_1, k_2)}{k_1 m_1 + k_2 m_2}\cdot\frac{\min(k_1, k_2)}{k_1 + k_2} < \frac{1}{4}. \end{aligned}$$

□

B-trees are far from optimal Tree Search organizations. Moreover, their use of relatively large numbers of redundant index entries makes them a bad choice for search within a page because the extra space requirement means fewer goal nodes per page. We study B-trees, and Tree Search strategies in general, more to provide a reference point for comparison of access times than to provide a serious alternative strategy. The uniform nature of the B-tree makes it relatively easy to program maintenance routines (insertion, deletion, etc). One might expect that this uniformity would also lead to optimal access times, but, as we show below, optimal organization trees are highly unbalanced.

In order to study optimal tree organizations, we use the following notation to facilitate making precise statements about trees:

$l(v)$: length of node v;
$n(t)$: number of goal nodes in t;
$k(t)$: number of nongoal nodes in t;
$P(t)$: path length of t;
$A(t)$: expected access time of $t = P(t) + n(t)$;
$opt(t)$: t is an optimal tree, i.e., $\forall t'.\ n(t') = n(t) \rightarrow P(t') \geq P(t)$;
feasible(N, k): there is an optimal tree with N goal and k nongoal nodes.

It is easy to see that we need only search a finite number of trees to find an optimal tree of size N. The following lemmas (given without proof) will be useful in describing the shape of optimal trees. Each can be proved easily by permuting nodes and subtrees.

LEMMA 3.2. *Optimal trees have no nodes of type*

(iii) β 

LEMMA 3.3 *In an optimal tree, the β-subtree of every nongoal node must have at least two goal nodes.*

LEMMA 3.4. *In an optimal tree, nongoal nodes cannot be at a greater length than goal nodes, i.e., $\forall t.\ \forall v_1$ in $t.\ \forall v_2$ in $t.\ opt(t) \wedge goal(v_1) \wedge nongoal(v_2) \rightarrow l(v_1) \geq l(v_2)$.*

Lemmas 3.2, 3.3, and 3.4 tell us that the optimal trees for N goal nodes must be constructed as follows: Consider an infinite binary tree as a skeleton, with each node having an α and a β successor. Fill this tree with k nongoal nodes for some $k < N$ so that these nodes occupy the positions as close to the root as possible, i.e., positions of minimal length. Each of the "leaves" of this tree of nongoal nodes has an α and a β successor that are goal nodes. Additional goal nodes are strung out in α-chains from these initial goal nodes. The goal nodes are added one by one, each at a minimal length available position. Ties for minimal length position are resolved arbitrarily for both goal and nongoal nodes. If we know k, this method will produce all possible optimal trees of size N. The choice of k need not be unique; see for example Figure 6 where $\beta = 5$, $N = 63$, and $k = 8$ or 9. As another example, for $\beta = 3$ and $N = 117$, there are optimal trees for $k = 18, 19, \ldots, 27$. Even for a fixed value of k, there can be several nonisomorphic optimal trees with k nongoal and N goal nodes (see Figure 6).

The following theorem will provide us with an efficient method for producing optimal trees. Its proof is somewhat lengthy and cumbersome, so we postpone it to the Appendix to this paper.

THEOREM 3.1. *Increasing the number of goal nodes by 1 either increases by 1 or leaves constant the number of nongoal nodes in an optimal tree, i.e.,* $[\forall N, k_1, k_2, k_3.\ k_1 \leq k_2 \leq k_3 \wedge feasible(N, k_1) \wedge feasible(N, k_3) \rightarrow feasible(N, k_2)] \wedge [\forall N, k.\ feasible(N, k) \rightarrow (feasible(N+1, k) \vee feasible(N+1, k+1)) \wedge (feasible(N-1, k) \vee feasible(N-1, k-1))]$.

PROOF. See the Appendix.

Given an optimal tree with N goal nodes, Theorem 3.1 gives the following procedure for obtaining an optimal tree with $N + 1$ goal nodes: Add a new goal node at a minimal length available position; compare the path length of this tree with that obtained from it by changing a minimal length goal node into a nongoal node and redistributing the goal

nodes so as to occupy the minimal length positions; and choose the tree with the smaller path length.

The next lemmas are proved by comparing the path length of the optimal tree with that of alternatives (Figure 7). We use them to bound the lengths of α-chains in optimal trees. (Note that the length of an α-chain is one less than the number of goal nodes in it.)

LEMMA 3.5. *Let c be the length of an α-chain which is an α-subtree of a nongoal node in an optimal tree. Then*

$$c \geq \beta - \tfrac{3}{2} + \tfrac{1}{2}\sqrt{4\beta + 1}.$$

PROOF. Assume t_1 of Figure 7 is optimal so that $P(t_1) \leq P(t_2)$ and $|(c + 1) - (d + \beta)| \leq 1$. Then

$$P(t_1) = \left(\sum_{i=\beta}^{\beta+d} i\right) + \left(\sum_{i=1}^{1+c} i\right),$$

and

$$P(t_2) = \sum_{i=0}^{c+d+1} i,$$

so that

$$P(t_2) - P(t_1) = d(c + 1) - \beta(d + 1) \geq 0.$$

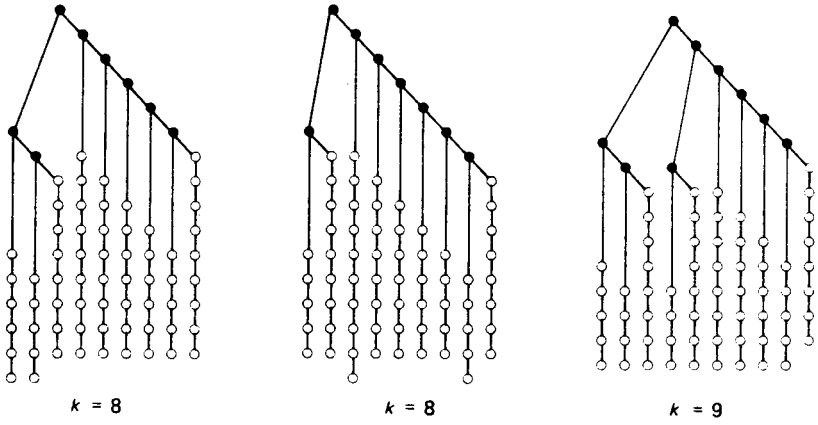


FIG. 6. Optimal trees with $\beta = 5$, $N = 63$, $k = 8$ or 9. Average access time = 11. β-arcs are indicated by extra length

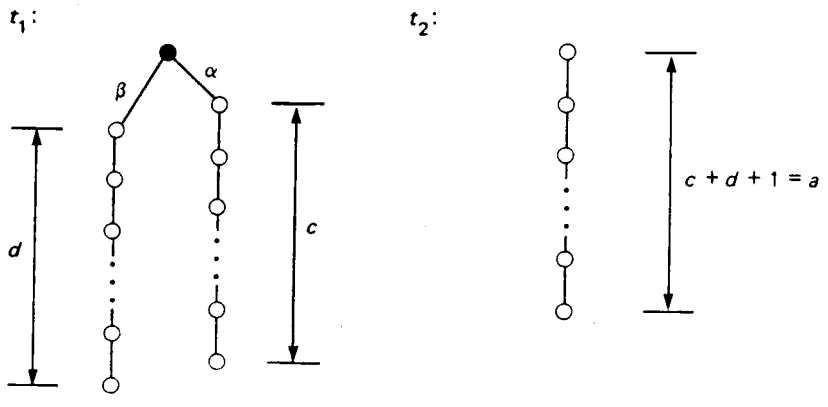


FIG. 7. Alternative possibilities for α-chains

Thus

$$d(c + 1 - \beta) - \beta \geq 0;$$

but

$$d \leq c + 2 - \beta,$$

so

$$c^2 + (3 - 2\beta)c + \beta^2 - 4\beta + 2 \geq 0.$$

If $c \leq \beta - \frac{3}{2} - \frac{1}{2}\sqrt{4\beta + 1}$, then $c < \beta - 2$ and $d < 0$. Thus $c \geq \beta - \frac{3}{2} + \frac{1}{2}\sqrt{4\beta + 1}$. □

LEMMA 3.6. *Let d be the length of an α-chain which is a β-subtree of a nongoal node in an optimal tree. Then*

$$d \geq -\tfrac{1}{2} + \tfrac{1}{2}\sqrt{4\beta + 1}.$$

PROOF. Assume t_1 of Figure 7 is optimal, as in the proof of Lemma 3.5, so that $|(c + 1) - (d + \beta)| \leq 1$ and $d(c + 1 - \beta) - \beta \geq 0$. Then $c + 1 - \beta \leq d + 1$, so

$$d^2 + d - \beta \geq 0.$$

Thus $d \geq -\frac{1}{2} + \frac{1}{2}\sqrt{4\beta + 1}$. □

LEMMA 3.7 *Let a be the length of any α-chain of an optimal tree. Then*

$$a < \beta + \sqrt{4\beta + 1}.$$

PROOF. Suppose t_2 of Figure 7 were optimal and $a \geq \beta + \sqrt{4\beta + 1}$. Then c and d could be chosen with $a = c + d + 1$ so that $c \geq \beta - \frac{3}{2} + \frac{1}{2}\sqrt{4\beta + 1}$ and $d \geq -\frac{1}{2} + \frac{1}{2}\sqrt{4\beta + 1}$ and at least one of the inequalities would be strict. Reversing the calculations of Lemmas 3.5 and 3.6, we would have $P(t_1) < P(t_2)$, contradicting the optimality of t_2. □

LEMMA 3.8. *Let t be an optimal tree with at least one nongoal node. Then*

$$l(t) - \tfrac{1}{2}(\beta + 2 + \sqrt{4\beta + 1}) < A(t) \leq l(t) - \tfrac{1}{4}(-1 + \sqrt{4\beta + 1}).$$

PROOF. The average of the lengths of nodes of an α-chain of t is its greatest length minus half the length of the chain. For each α-chain this average is greater than $l(t) - 1 - \frac{1}{2}(\beta + \sqrt{4\beta + 1})$ by Lemma 3.7. Thus $A(t)$ must be greater than this number. Similarly $A(t)$ must be less than or equal to $l(t) - \frac{1}{2}(-\frac{1}{2} + \frac{1}{2}\sqrt{4\beta + 1})$ by Lemmas 3.5 and 3.6. □

THEOREM 3.2. *Let t be an optimal tree. Then*

$$\log_r(n(t)) - \beta + 1 \leq l(t) \leq \log_r(n(t)) - 1 + \sqrt{4\beta + 1}$$

where r is the unique positive real root of $r^\beta = r^{\beta-1} + 1$.

PROOF. See the Appendix.

THEOREM 3.3. *Let $OT(N)$ be the expected random access time in an optimal tree with N goal nodes. Then*

$$\log_r(N) - \tfrac{1}{2}(3\beta + \sqrt{4\beta + 1}) \leq OT(N) \leq \log_r(N) - \tfrac{3}{4} + \tfrac{3}{4}\sqrt{4\beta + 1}$$

where $r^\beta = r^{\beta-1} + 1$.

PROOF. Substitute l from Theorem 3.2 in Lemma 3.8. □

The optimal Tree Search access times give us something to aim at with the other families of strategies. Before investigating strategies with no redundant entries, we introduce a subfamily of Tree Search strategies that attempts to approximate the skewness of optimal trees (viewed as trees of α-chains) while sacrificing very little of the uniformity of B-trees. We call the organization trees of this family *Promotion* trees or *P-trees* because each one results from applying a modifying procedure called promotion to a B-tree. *Promotion* is the process of replacing each node of type

(iii)

by the α-chain to which it points. Figure 8 presents the result of applying promotion to the B-tree of Figure 5. The *size, order,* and *depth* of a P-tree are those of its associated B-tree. Note that promotion preserves properties (3.1) and (3.2).

THEOREM 3.4. *The expected random access time for a P-tree with N goal nodes is* $PTm,d(N) = BTm,d(N) - \Delta\beta$, *where* $|\Delta - (d-1)/m| < (d-1)m^{d-2}/N$. (We will sometimes use a form with raised subscripts, as $PTm,d(N)$, in place of the more cumbersome subscripted form, $PT_{m,d}(N)$.)

Here we will make another evenness assumption about the distribution of lengths of promoted α-chains and approximate PT by

$$PTm,d(N) = BTm,d - ((d-1)/m)\beta, \tag{3.5}$$

or

$$PTm,d(N) = \tfrac{1}{2}(d-1)(m-1) + \tfrac{1}{2}((N/m^{d-1}) - 1) + ((d-1)(m-1)/m)\beta. \tag{3.6}$$

PROOF OF THEOREM 3.4. Let t be a B-tree and let p be its corresponding P-tree. Promotion reduces $P(t)$ by β at m^{d-2} leaf α-chains for each of the $d-1$ nonleaf levels. Thus,

$$(d-1)m^{d-2}\lfloor N/m^{d-1}\rfloor\beta \le P(t) - P(p) \le (d-1)m^{d-2}\lceil N/m^{d-1}\rceil\beta.$$

Let $\Delta = (P(t) - P(p))/N\beta$. Then $PTm,d(N) = BTm,d(N) - \Delta\beta$, and

$$((d-1)/m) - (d-1)m^{d-2}/N < \Delta < ((d-1)/m) + (d-1)m^{d-2}/N. \quad \square$$

We complete this section on Tree Search strategies with a summarizing result whose proof is immediate:

THEOREM 3.5 *Let* $BT(N) = min\{BTm,d(N)\}$ *and* $PT(N) = min\{PTm,d(N)\}$. *Then* $OT(N) \le PT(N) \le BT(N)$.

4. *Root Search*

The family of *Root Search* strategies uses the program scheme of Figure 9 and a totally ordered organization graph with no redundant, nongoal nodes, as in Figure 10. As with B-trees, Root Search strategies divide the goals evenly into m^{d-1} sections for sequential search. The correct section is found by the same method as is used with a B-Tree except that the largest goal node of each appropriate supersection is visited in place rather than having a copy visited at a higher level. This requires following a local pointer for the [NEXT AT THIS LEVEL] command. Thus, we charge β for the loop [NATL; C].

When $d = 2$ and m is approximately $\sqrt{N}$, the strategy is called *Square Root Search.* We also refer to the family of Square Root Search strategies for arbitrary m. When $d = 3$, we call the family *Cube Root Search,* etc. Figure 10 is an example of a Cube Root Search organization graph of size $N = 20$ and order $m = 3$. It corresponds to Figures 5 and 8 in size, order, and depth. The arcs followed by loops [N; C] and [NATL; C] are labeled α and β, respectively, but we have not labeled the arcs corresponding to loops [FFKANL; C] and

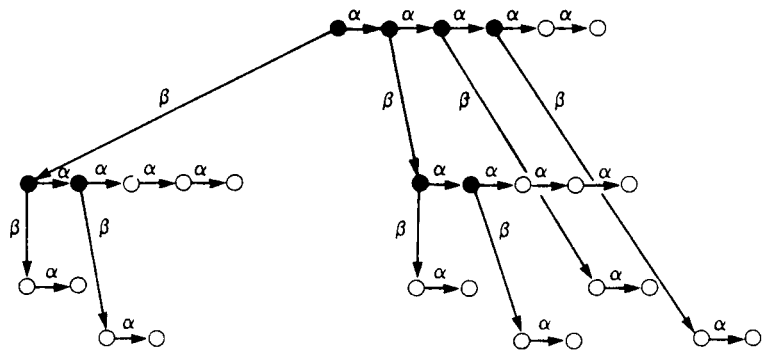


FIG. 8. P-tree of size 20, order 3, and depth 3. The result of applying promotion to the B-tree of Figure 5

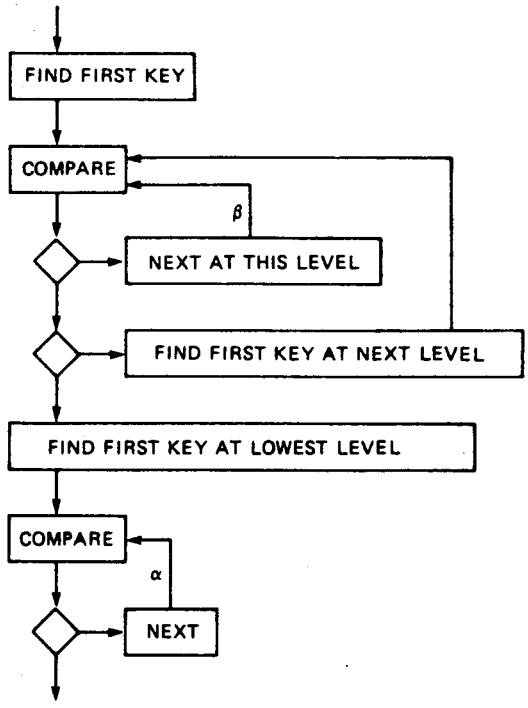


FIG. 9. Root Search program scheme

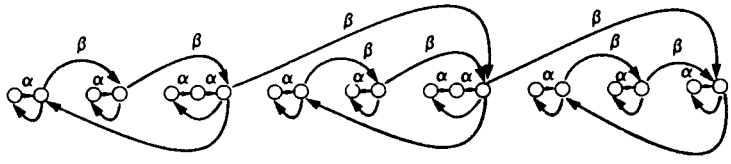


FIG. 10. Cube Root Search organization graph

[FFKALL; C]. Rather than fix a more precise implementation here, we simply postulate that their costs lie between 1 and β. Then, with the approximation justified by Lemma 3.1, we have the following bounds on expected random access time for Root Search:

$$d - 1 + SE(N/m^{d-1}) + (d-1)SE(m)\beta \leq BRm,d(N) \leq SE(N/m^{d-1}) + (d-1)(1 + SE(m))\beta, \quad (4.1)$$

or

$$d - 1 + \tfrac{1}{2}((N/m^{d-1}) - 1) + \tfrac{1}{2}(d-1)(m-1)\beta \leq BRm,d(N) \leq \tfrac{1}{2}((N/m^{d-1}) - 1) + \tfrac{1}{2}(d-1)(m+1)\beta. \quad (4.2)$$

For comparison purposes we will use the lower bound, particularly for Square Root Search where it is probably achievable. We define *Best Root Search* as the optimal Root Search:

$$BR(N) = \min\{BRm,d(N)\},$$

where

$$BRm,d(N) = d - 1 + \tfrac{1}{2}((N/m^{d-1}) - 1) + \tfrac{1}{2}(d-1)(m-1)\beta. \quad (4.3)$$

We also have $SR(N) = \min\{SRm(N)\}$, where

$$SRm(N) = \tfrac{1}{2}((N/m) + 1) + \tfrac{1}{2}(m-1)\beta, \quad (4.4)$$

for Square Root Search, and $CR(N) = \min\{CRm(N)\}$, where

$$CRm(N) = \tfrac{1}{2}((N/m^2) + 3) + (m - 1)\beta, \tag{4.5}$$

for Cube Root Search, etc. Note that the minimum value of $SRm(N)$ occurs with m approximately equal to $\sqrt{N/\beta}$ (m is of course an integer). Similarly, the minimum value of $CRm(N)$ occurs with m near $\sqrt[3]{N/\beta}$.

Square Root Search is used for search within a page in VSAM. It is simple to program and uses very little extra space for pointers. However, for N sufficiently large, Square Root Search is not the optimal Root Search. Cube Root Search is faster for $N \geq 27$. Square Root Search is generally (for most interesting cases) the slowest of all the strategies we consider. Thus it depends, for its justification, on the efficient use of space.

Insertion into the Square Root Search organization is extremely simple, since only the pointer to the highest key in the correct section (α-chain) need be updated (assuming relative addressing from the previous section).

Although all α-arcs are not filled in in Figure 10, the organization graphs for Root Search are laid out so that locating the next higher key is achieved by executing a [NEXT]. In the next section we study a family of strategies designed to run significantly faster than Root Search strategies while preserving most of their space and order advantages. Since the case $m = 2$ for Root Search strategies is just an inefficient implementation of this next family, *we assume $m \geq 3$ for all Root Search strategies.*

5. *Binary Search*

The family of *Binary Search* strategies uses the program scheme of Figure 11 and an organization graph similar to that of Figure 12. Like Root Search, Binary Search strategies use no redundant nongoal nodes. Binary Search strategies divide the goals as evenly as

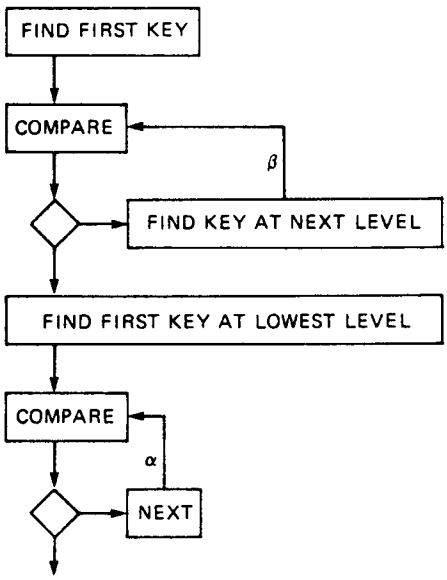


FIG. 11. Binary Search program scheme

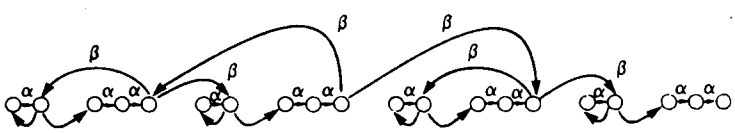


FIG. 12. Binary Search organization graph

possible into 2^d sections for sequential search. The time advantage for Binary Search comes from eliminating half the remaining candidates for correct section with each key comparison. The analysis is similar to that of Root Search; we charge β for the loop [FKANL; C] and postulate that the cost of [FFKALL; C] is between 1 and β with 1 being used for comparison purposes. Figure 12 is labeled the same way Figure 10 is: α for arcs followed by [N]; β for arcs followed by [FKANL]; and no label for arcs followed by [FFKALL]. It depicts a Binary Search organization of size $N = 20$ and depth $d = 3$.

Following the conventions of the previous sections, we approximate the expected random access time for Binary Search by

$$BId(N) = SE(N/2^d) + 1 + (d - 1)\beta, \tag{5.1}$$

or

$$BId(N) = \tfrac{1}{2}((N/2^d) + 1) + (d - 1)\beta. \tag{5.2}$$

We also define $BI(N) = \min\{BId(N)\}$. Binary Search strategies generally use a little more space and are a little more complicated than Square Root Search, but they often run significantly faster. Section 6 contains a detailed comparison.

Insertion for Binary Search organizations can be almost as simple and fast as for Root Search organizations. One method uses relative displacement pointers as in Square Root Search and updates a pointer followed only if the direction is reversed on the subsequent pointer followed. As with Root Search strategies, locating the next higher key requires merely executing [NEXT].

The radix partition tree is a binary tree of pointers set up to branch on bit values of the key rather than key comparisons. Radix partition trees can support the function NEXT HIGHER KEY by left to right tree traversal (as in B-trees), and they can support the function IS A PREFIX OF much more rapidly than any of the strategies we have studied. A study of radix partition tree strategies of [10] or Patrician strategies as presented in [3] is beyond the scope of this paper. However, under our uniformity assumptions, the radix partition tree is no faster and can be considerably slower than a full Binary Search (section size 1), which is in turn no faster than optimal Binary Search.

The next section presents the main results of this paper. It contains a somewhat exhaustive comparison of expected random access times for all the strategies introduced so far. We believe that some of the results are unexpected, and that it is not easy to determine which strategy runs faster by inspecting the formulas.

6. *Time Comparison Results*

We begin this section with a recapitulation of the approximations to expected random access time we will use for comparison. We assume N is an integer ≥ 5, m is an integer ≥ 2, and d is an integer ≥ 1. Recall that $\beta \geq 1$.

B-tree: $BT(N) = \min\{BTm,d(N)\}$, where

$$BTm,d(N) = \tfrac{1}{2}(d - 1)(m - 1) + \tfrac{1}{2}((N/m^{d-1}) - 1) + (d - 1)\beta. \tag{3.4}$$

P-tree: $PT(N) = \min\{PTm,d(N)\}$, where

$$PTm,d(N) = \tfrac{1}{2}(d - 1)(m - 1) + \tfrac{1}{2}((N/m^{d-1}) - 1) + ((d - 1)(m - 1)/m)\beta. \tag{3.6}$$

Best Root: $BR(N) = \min\{BRm,d(N) | m \geq 3\}$, where

$$BRm.d(N) = d - 1 + \tfrac{1}{2}((N/m^{d-1}) - 1) + \tfrac{1}{2}(d - 1)(m - 1)\beta. \tag{4.3}$$

Square Root: $SR(N) = \min\{SRm(N) | m \geq 3\}$, where

$$SRm(N) = \tfrac{1}{2}((N/m) + 1) + \tfrac{1}{2}(m - 1)\beta. \tag{4.4}$$

Binary: $BI(N) = \min\{BId(N)\}$, where

$$BId(N) = \tfrac{1}{2}((N/2^d) + 1) + (d - 1)\beta. \tag{5.2}$$

Optimal Tree: $\log_r(N) - \frac{1}{2}(3\beta + \sqrt{4\beta + 1}) \leq OT(N) \leq \log_r(N) - \frac{3}{4} + \frac{3}{4}\sqrt{4\beta + 1}$, where

$$r^{\beta} = r^{\beta-1} + 1.$$

Also recall from Theorem 3.5 $OT(N) \leq PT(N) \leq BT(N)$. And note that $BR(N) \leq SR(N)$.

We postpone the proofs of many of the following results to the appendix:

THEOREM 6.1. $BT(N) \leq BR(N)$.

PROOF. $BRm,d(N) - BTm,d(N) = (d - 1)(\beta - 1)(m - 3)/2$, and we have stipulated $m \geq 3$ for BR. □

THEOREM 6.2. *If* $\beta = 1$, *then* $PT(N) \leq BI(N)$.

PROOF. $PT_{2,d+1}(N) = \frac{1}{2}d + \frac{1}{2}(N/2^d) - \frac{1}{2} + \frac{1}{2}d = BI_d(N)$. □

THEOREM 6.3. *If* $\beta > 1$, *then for* $N \geq 16\beta$, $PT(N) < BI(N)$.

PROOF. Assume $N \geq 16\beta$. Then

$$BI_1(N) - BI_2(N) = (N/8) - \beta \geq 0 \quad \text{and} \quad BI_2(N) - BI_3(N) = (N/16) - \beta \geq 0,$$

so

$$BI(N) = \min\{BI_d(N) | d \geq 3\}.$$

Now

$$BI_d(N) - PT_{2,d+1}(N) = (\tfrac{1}{2}d - 1)(\beta - 1) > 0 \quad \text{for} \quad d \geq 3.$$

Thus $BI(N) > PT(N)$. □

THEOREM 6.4. *For* $1 \leq \beta \leq 3/2$ *and* $N \geq 8$, $BI(N) < BT(N)$.

PROOF. See the Appendix.

THEOREM 6.5. *For* $\beta > 3/2$ *and N sufficiently large*, $BT(N) < BI(N)$.

PROOF. See the Appendix.

THEOREM 6.6. *For* $N \geq 8$ *and* $1 \leq \beta \leq 1/(\log_2 3 - 1) \simeq 1.7095$, $BI(N) < BR(N)$.

PROOF. See the Appendix.

THEOREM 6.7. *For* $\beta > 1/(\log_2 3 - 1)$ *and N sufficiently large*, $BR(N) < BI(N)$.

PROOF. See the Appendix.

Figures 13 through 18 are graphs of mean access time for small values of N and β. The ranges are

$$10 \leq N \leq 500$$

and

$$1 \leq \beta \leq 2.$$

Binary Search (BI) is represented by small x characters. The other strategies are represented by solid lines and always obey the following order:

P-tree (PT) ≤ B-tree (BT) ≤ Best Root (BR) ≤ Square Root (SR).

For $\beta = 1$, $BI = PT$ and $BT = BR$.

7. *Space*

When we are searching pages from a large index, the cost of a page fault may be significantly greater than the cost of searching a page by any of our proposed strategies. Thus a strategy which minimizes the number of page references required is generally preferable. Sometimes packing more index entries onto a page will result in fewer page references. In this case we may even choose the strategy which requires the least amount of space—Sequential Search—in spite of its dismal time performance within the page. When the index entries are of fixed length, then the local pointers of Root Search and Binary Search strategies can be calculated rather than stored so that they too require no extra space. However, compression techniques used to pack more entries onto a page generally result in variable length entries. Thus the final choice of a strategy for search within the page often lies outside the context of the page and beyond the scope of this

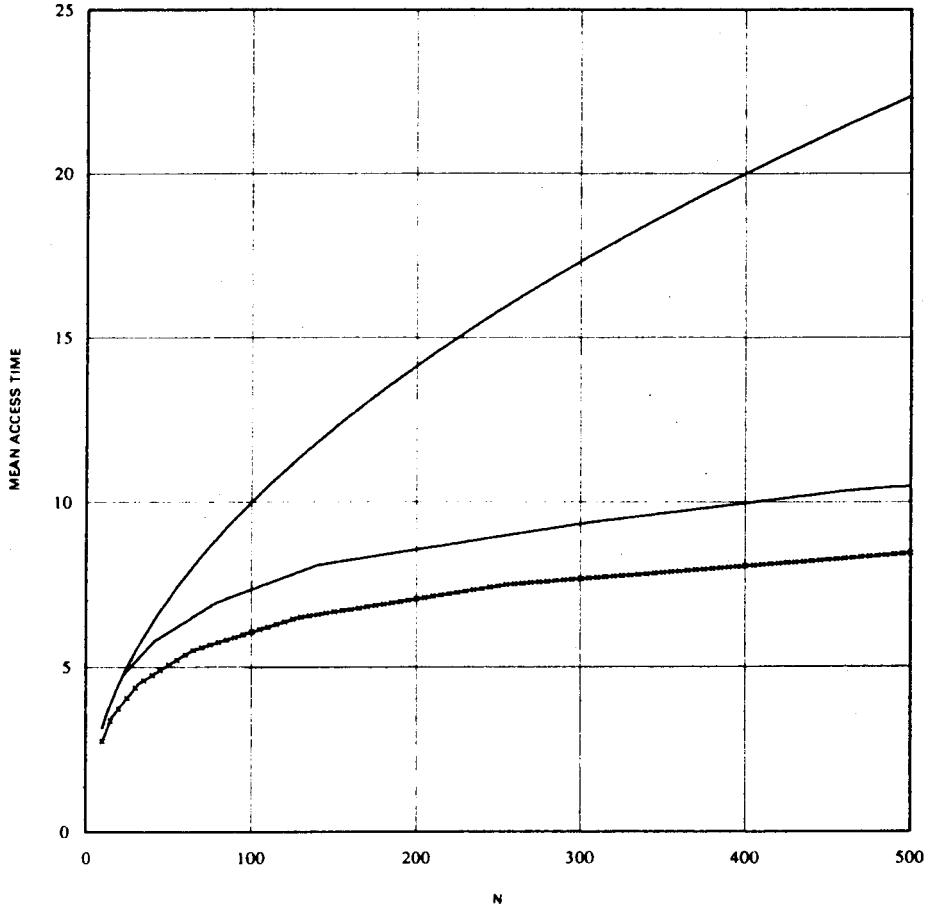


FIG. 13. $BI = PT < BT = BR \leq SR$

paper. What we do provide here is a comparison of the space requirements of the strategies so far presented.

The unit of space will be the average size of an index entry. We let δ be the space required for a local pointer: presumably $0 \leq \delta \leq 1$. For the nongoal entries of Tree Search organizations we charge a cost between δ and 1. We prefix the names of the access time functions with S to indicate the space functions. For example,

$$SSE(N) = N. \tag{7.1}$$

(Note that we do not charge for the space for any pointer to the first key of an organization.) The formulas we give are merely representative; they do not characterize every possible implementation of each strategy. For instance, Binary Search might use $2^d - 1$, 2^d, or $2^d + 1$ pointers; we simply choose 2^d. If they are calculated rather than stored, the space cost for each will be $\delta = 0$. Thus we have

$$SBI_d(N) = N + 2^d\delta. \tag{7.2}$$

We assume that the local pointers are kept in a contiguous part of the page for Root Search so that we require only $m^{d-1} - 1$ local pointers for Best Root. Thus

$$SBRm,\ d(N) = N + (m^{d-1} - 1)\delta. \tag{7.3}$$

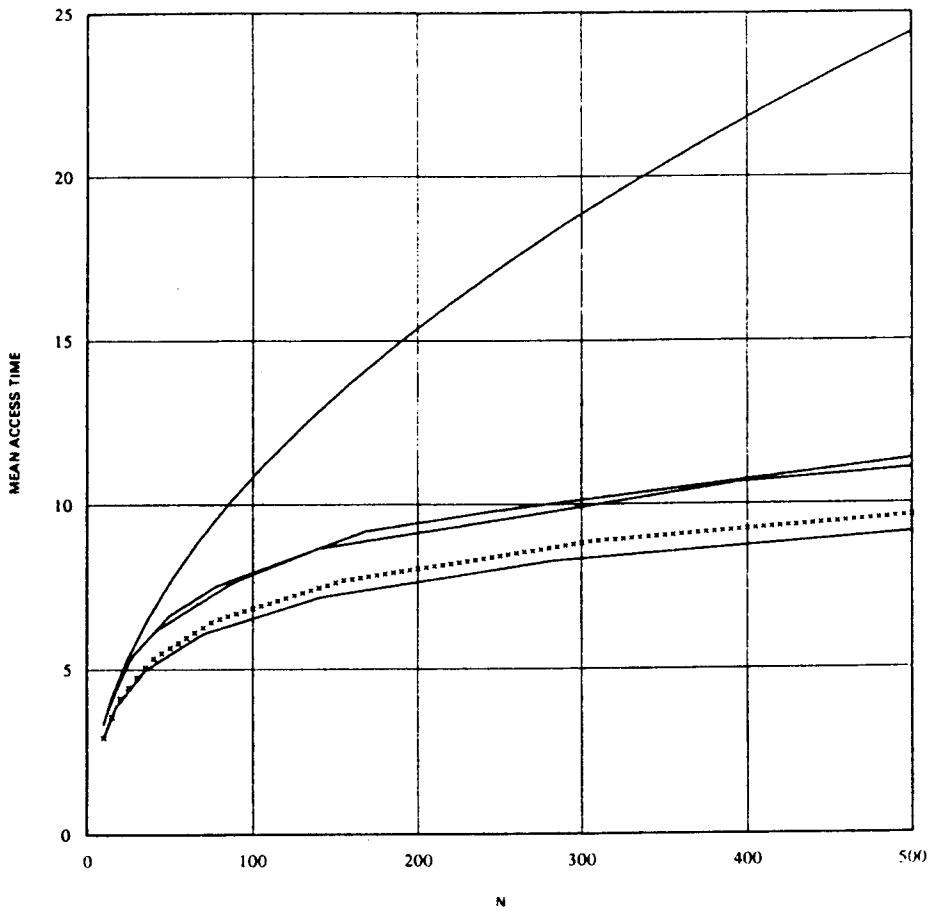


FIG. 14. $PT \leq BI < BT \leq BR \leq SR$

B-trees do not use separate local pointers, so we simply bound SBT by

$$N + ((m^d - m)/(m - 1))\delta \leq SBTm,\ d(N) \leq N + (m^d - m)/(m - 1). \tag{7.4}$$

P-trees use the same space as B-trees except for the elimination of $(m^{d-1} - 1)/(m - 1)$ type (iii) nodes. Thus we have

$$N + (m^{d-1} - 1)\delta \leq SPTm,\ d(N) \leq N + m^{d-1} - 1. \tag{7.5}$$

Inspection of the above expressions shows that if space is very important, the only real contenders are Square Root Search (Root Search with $d = 2$) and Binary Search. Now optimal Square Root Search has m close to $\sqrt{N/\beta}$ so that the space required for pointers is roughly $(\sqrt{N/\beta} - 1)\delta$. For optimal Binary Search, we have (see Lemma A4 in Appendix)

$$N/4\beta \leq 2^d < N/2\beta,$$

so that optimal Binary Search would require at least $(N/4\beta)\delta$ space for pointers. Thus optimal Binary Search is not space competitive.

However, some suboptimal depth Binary Search strategy generally competes favorably for combined time and space. In order to compare strategies simultaneously for time and space, we use a weighted sum: PERFORMANCE = TIME + SPACE · WEIGHT. Let x = (WEIGHT · δ)/β. Our region of interest is defined by

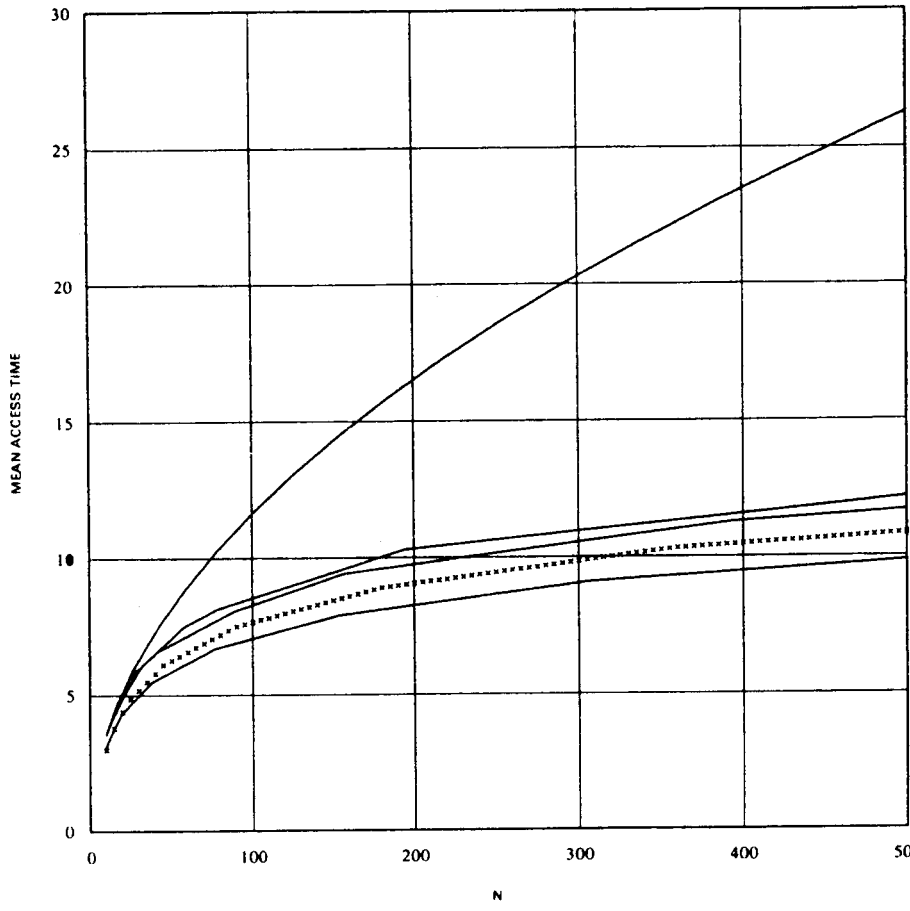


FIG. 15. $PT \leq BI < BT \leq BR \leq SR$

$$0 \leq x \leq \tfrac{1}{4}, \tag{7.6}$$

$$1 \leq \beta \leq 2, \tag{7.7}$$

and

$$50 \leq N. \tag{7.8}$$

Justification of these bounds is beyond the scope of this paper, but we believe they represent realistic bounds for real applications. Within this region we can make a strong statement about the dominance of Binary Search.

THEOREM 7.1. *For each N, β, and x within the region described by* (7.6), (7.7), *and* (7.8), *and for each $m \geq 3$, there is a d such that the PERFORMANCE of Binary Search with depth d is better than the PERFORMANCE of Square Root Search with m sections.*

SKETCH OF PROOF: Let $f(N, \beta, x, m, d)$ be the difference between the PERFORMANCE of Square Root Search with m sections and the PERFORMANCE of Binary Search with depth d. Then

$$f(N, \beta, x, m, d) = \tfrac{1}{2}N((1/m) - (1/2^d)) + \beta\,(\tfrac{1}{2}(m-1) - (d-1)) + \beta x(m - 1 - 2^d).$$

We will show that $\forall N, \beta, x, m.\ \exists d.\ f(N, \beta, x, m, d) > 0$.

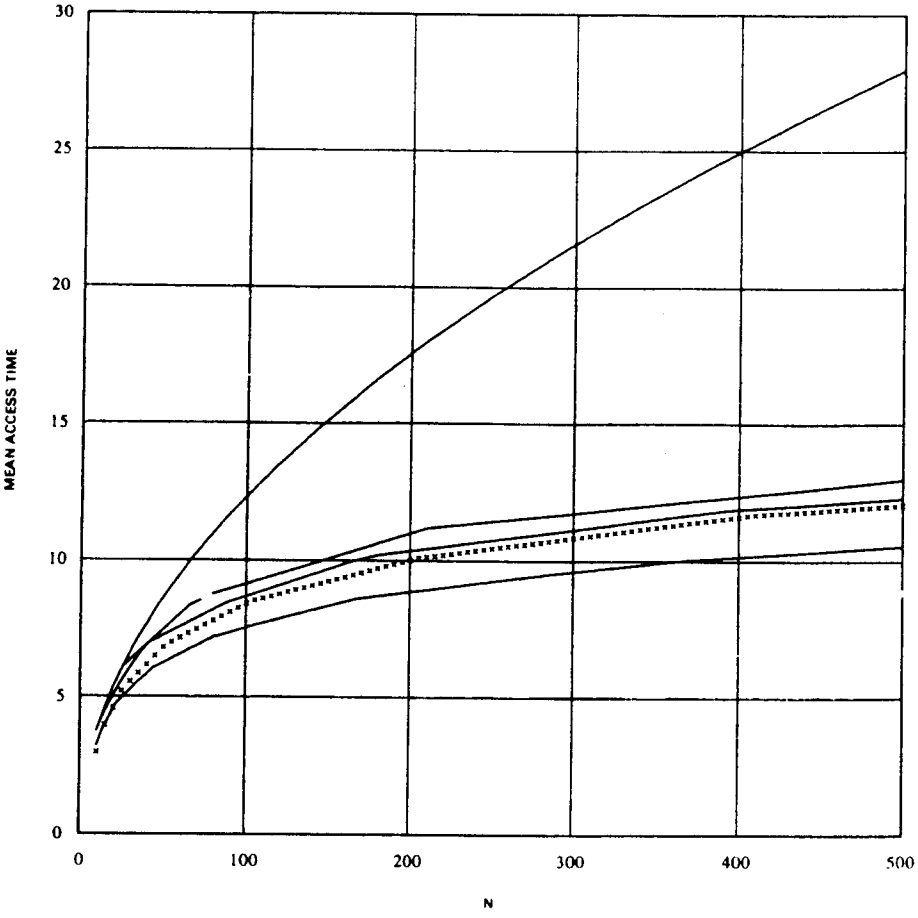


FIG. 16. $PT \leq BI < BT \leq BR \leq SR$

First we write $f = g - h$ where

$$g(N, \beta, x, m) = (N/2m) + (\beta m/2) + \beta x m,$$

and

$$h(N, \beta, x, d) = (N/2^{d+1}) + \beta(d - \tfrac{1}{2}) + \beta x(1 + 2^d).$$

It is easy to show that $g \geq \sqrt{N\beta(1 + 2x)}$. Solving $\sqrt{N\beta(1 + 2x)} > h$ for N, we get $L(N, \beta, x, d) < N < H(N, \beta, x, d)$ where

$$L(N, \beta, x, d) = \beta\left[A + Bx - 2^{d+1}\sqrt{C + Dx - 2^{d+2}x^2}\right],$$

$$H(N, \beta, x, d) = \beta\left[A + Bx + 2^{d+1}\sqrt{C + Dx - 2^{d+2}x^2}\right],$$

$$A = 2^{2d+1} - 2^{d+1}(d - \tfrac{1}{2}),$$

$$B = 2^{2d+1} - 2^{d+1},$$

$$C = 2^{2d} - 2^{d+1}(d - \tfrac{1}{2}),$$

$$D = 2^{2d+1} - 2^{d+2}d.$$

We can show for $d \geq 3$, $H(N, \beta, x, d) > L(N, \beta, x, d + 1)$, and $L(N, \beta, x, 3) < 25\beta \leq 50$.

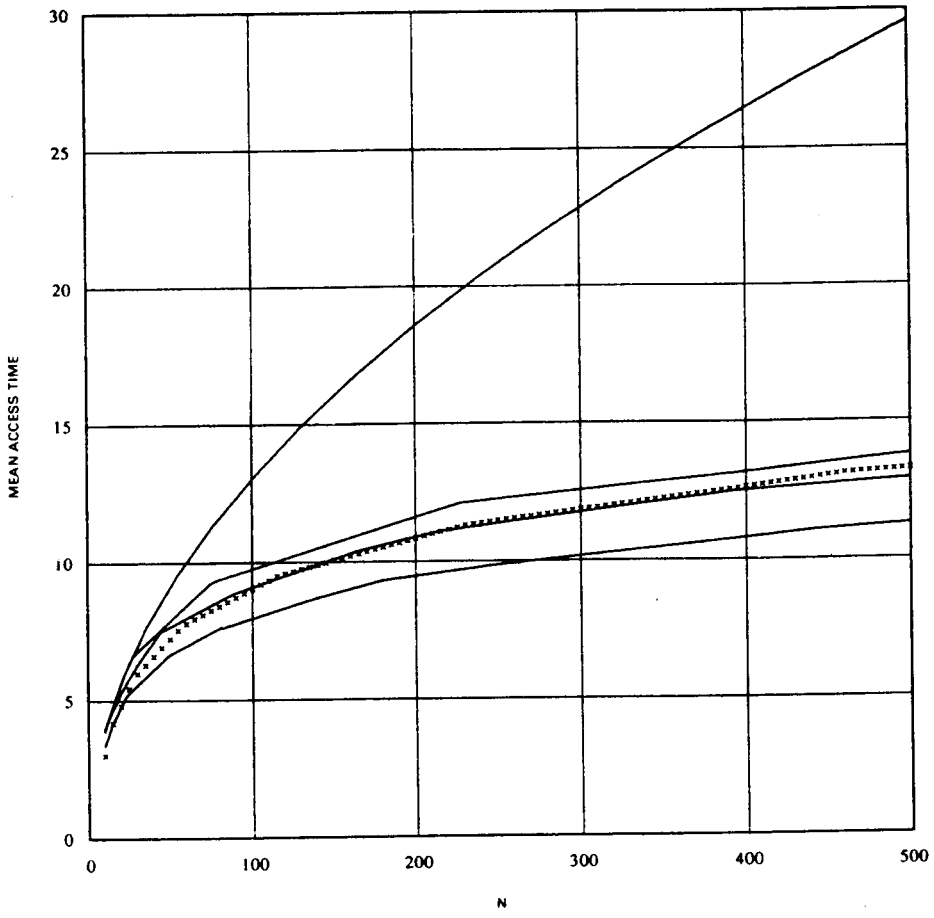


FIG. 17. $PT < BT \leq BR \leq SR$ & $PT \leq BI < BR$

Thus some $d \geq 3$ makes $f > 0$ for $N \geq 50$. This completes the proof sketch. □

It is an extremely rare case when even doubling the pointer space required causes a significant change in any performance factor. To cause an increase in the number of page references per access in a B-tree of pages, N must have been very close to an integer power of the page size. Thus we can argue that Binary Search is preferable.

When space is not important, the competing strategies are Binary Search and P-tree (assuming $\beta < 3/2$ as in Theorem 6.4). Unless β is fairly large we expect that the complexity of programming the implementation of the P-tree will outweigh its slight time advantage, and again Binary Search will be preferable.

When space is important but $\delta = 0$ because of fixed length index entries, Binary Search is far superior to the other strategies unless β is large. Notice that we must qualify all recommendations with a reference to β. In general we suggest using Binary Search but checking β to make sure it is small.

The following section is an Appendix in which we present the detailed and complex proofs from Sections 3 and 6.

Appendix

This Appendix contains the details of proofs we felt would interfere with the reading of

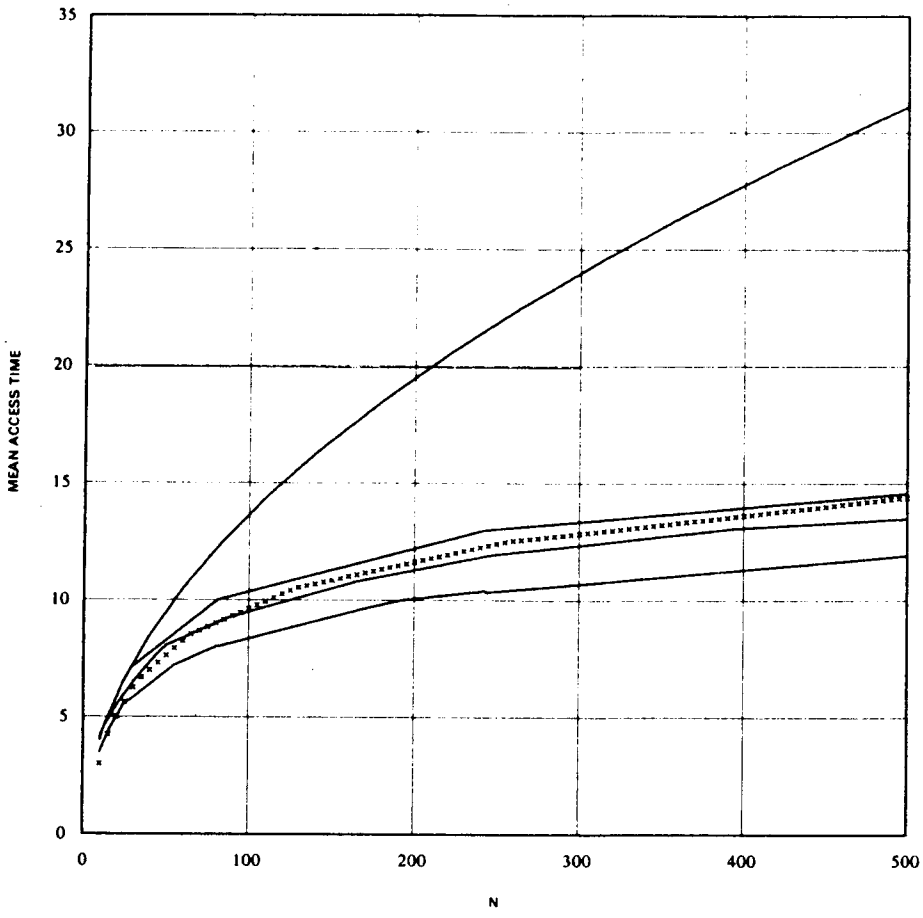


FIG. 18. $PT < BT \leq BR \leq SR$ & $PT \leq BI < BR$

the paper if included in the body. First we derive various bounds on the behavior of *BT*, *BI*, and *BR*.

For each integer $m \geq 2$ we define $BTm(N) = \min\{BTm, d(N) | d \geq 1, m^{d-1} \leq N\}$ so that $BT(N) = \min\{BTm(N) | m \geq 2\}$.

LEMMA A1. $BT_m(N) = BT_{m,d}(N)$ *where* $d = \lceil log_m((m-1)N/(m-1+2\beta))\rceil$.

PROOF: Let

$$f(x) = BT_{m,x+1}(N) - BT_{m,x}(N)$$
$$= \beta + \tfrac{1}{2}(m-1) - (m-1)N/(2m^x)$$

for integers $x \geq 1$. Then $f(x) \geq 0$ iff $m^x \geq (m-1)N/(m-1+2\beta)$. Thus $BT_{m,d}(N)$ attains its minimal value at $d = \lceil \log_m((m-1)N/(m-1+2\beta))\rceil$. □

LEMMA A2. $LBT_m(N) \leq BT_m(N) \leq HBT_m(N)$ *where*

$$LBT_m(N) = \frac{1}{2}(m-1+2\beta)\left(log_m\left(\frac{N\ ln\ m}{m-1+2\beta}\right) + \frac{1}{ln\ m}\right) - \frac{1}{2},$$

and

$$HBT_m(N) = \frac{1}{2}(m-1+2\beta)\left(log_m\left(\frac{(m-1)N}{m-1+2\beta}\right) + \frac{1}{m-1}\right) - \frac{1}{2}.$$

PROOF: By Lemma A1,

$$BT_m(N) = BT_{m,d}(N),$$

where

$$d = \log_m\left(\frac{(m-1)N}{m-1+2\beta}\right) + \theta \quad \text{and} \quad 0 \le \theta < 1.$$

Substituting d in (3.4) yields

$$BT_m(N) = \frac{1}{2}(m-1+2\beta)\left(\log_m\left(\frac{(m-1)N}{m-1+2\beta}\right) + \theta - 1 + \frac{m^{1-\theta}}{m-1}\right) - \frac{1}{2}. \tag{A1}$$

Let $g(\theta) = \theta - 1 + m^{1-\theta}/(m-1)$. Then $g'(\theta) = 1 - m^{1-\theta} \ln m/(m-1)$, $g'(0) < 0$, and $g'(1) > 0$. Thus a straightforward analysis shows that g attains its maximum at $\theta = 0$ and its minimum at $m^\theta = m \ln m/(m-1)$, and

$$\log_m(\ln m/(m-1)) + 1/\ln m \le g(\theta) \le 1/(m-1). \tag{A2}$$

Substituting (A2) in (A1) provides the upper and lower bounds of the lemma. □

LEMMA A3. *For $N \ge 6$, $1 \le \beta \le 3/2$, and all integers m with $2 \le m \le N$, $LBT_m(N) \ge min\{LBT_3(N), LBT_4(N), LBT_5(N)\}$.*

PROOF: Let $f(m) = LBT_m(N)$. We can write

$$f'(m) = A(m)D(m)/2 \ln m, \tag{A3}$$

where

$$A(m) = 1 - (m - 1 + 2\beta)/(m \ln m), \tag{A4}$$

and

$$D(m) = \ln(N \ln m) - \ln(m - 1 + 2\beta). \tag{A5}$$

The hypotheses of the lemma yield

$$A(m) < 0 \quad \text{for} \quad 2 \le m \le 3,$$

$$A(m) > 0 \quad \text{for} \quad 5 \le m \le N,$$

and

$$D(m) > 0 \quad \text{for} \quad 2 \le m \le N.$$

Thus $f(m) \ge \min\{f(3), f(4), f(5)\}$. □

LEMMA A4. *If $N \le 4\beta$, $BI(N) = BI_1(N) = N/4 + 1/2$. If $N > 4\beta$, $BI(N) = BI_d(N)$ where $d = \lceil log_2(N/\beta) \rceil - 2$.*

PROOF: $BI_{d+1}(N) - BI_d(N) = \beta - N/2^{d+2}$. If $N \le 4\beta$, then $BI_{d+1}(N) \ge BI_d(N)$ for all $d \ge 1$. If $N > 4\beta$, then $BI_d(N)$ is least at the first point where $2^{d+2} \ge N/\beta$. □

LEMMA A5. *For $N > 4\beta$,*

$$\beta\, log_2(N/\beta) + \tfrac{1}{2} + \beta(log_2(2\ ln\ 2) + 1/ln\ 2 - 3) \le BI(N) \le \beta\, log_2(N/\beta) + \tfrac{1}{2} - \beta.$$

PROOF: By Lemma A4, $BI(N) = BI_d(N)$ where $d = \log_2(N/\beta) + \theta - 2$ and $0 \le \theta < 1$. Substituting d in (5.2) yields

$$BI(N) = \beta \log_2(N/\beta) + \tfrac{1}{2} + \beta g(\theta), \tag{A6}$$

where

$$g(\theta) = 2^{1-\theta} + \theta - 3.$$

Now $g'(\theta) = 1 - 2^{1-\theta} \ln 2$, so a straightforward analysis shows that g attains its maximum at $\theta = 0$ and its minimum at $2^\theta = 2 \ln 2$. Thus

$$\log_2(2 \ln 2) + 1/\ln 2 - 3 \le g(\theta) \le -1. \tag{A7}$$

Substituting (A7) in (A6) provides the upper and lower bounds of the lemma. □

THEOREM 6.4. *For* $1 \le \beta \le 3/2$ *and* $N \ge 8$, $BI(N) < BT(N)$.

PROOF: Note that since $N \ge 8$ and $\beta \le 3/2$, $N > 4\beta$. We will show that

$$LBT_m(N) \ge \beta \log_2(N/\beta) + \tfrac{1}{2} - \beta, \quad \text{for} \quad m = 3, 4, 5,$$

from which our result follows by Lemmas A2, A3, and A5. □

We define $h(m, \beta, N) = LBT_m(N) - (\beta \log_2(N/\beta) + 1/2 - \beta)$. Thus

$$\begin{aligned} h(m, \beta, N) &= \frac{1}{2}(m - 1 + 2\beta)\left(\log_m\left(\frac{N \ln m}{m - 1 + 2\beta}\right) + \frac{1}{\ln m}\right) \\ &\quad - \beta\left(\log_2\left(\frac{N}{\beta}\right) - 1\right) - 1 \\ &= (\ln N)\left(\frac{m - 1 + 2\beta}{2 \ln m} - \frac{\beta}{\ln 2}\right) + (m - 1 + 2\beta) \\ &\quad \cdot\left(\frac{\ln(\ln m) + 1 - \ln(m - 1 + 2\beta)}{2 \ln m}\right) - 1 + \beta + \beta\frac{\ln \beta}{\ln 2}. \end{aligned} \tag{A8}$$

The derivative of (A8) with respect to β is

$$\begin{aligned} \frac{\partial h}{\partial \beta}(m, \beta, N) &= (\ln N)\left(\frac{1}{\ln m} - \frac{1}{\ln 2}\right) - \frac{\ln(m - 1 + 2\beta) - \ln(\ln m) - 1}{\ln m} \\ &\quad - \frac{1}{\ln m} + 1 + \frac{1}{\ln 2} + \frac{\ln \beta}{\ln 2}. \end{aligned} \tag{A9}$$

Differentiating (A9) with respect to β yields

$$\frac{\partial^2 h}{(\partial \beta)^2}(m, \beta, N) = \frac{(m - 1 + 2\beta) \ln m - 2\beta \ln 2}{\beta \ln 2(m - 1 + 2\beta) \ln m}. \tag{A10}$$

Since this second derivative is positive for $m \ge 2$, $\beta > 1$, and since

$$\frac{\partial h}{\partial \beta}(m, 1, e^{1.95}) > 0 \quad \text{for} \quad m = 3, 4, 5,$$

we have

$$\frac{\partial h}{\partial \beta}(m, \beta, e^{1.95}) > 0 \quad \text{for} \quad m = 3, 4, 5$$

and $1 \le \beta \le 3/2$. Thus

$$\begin{aligned} &\min\{h(m, \beta, N) | m = 3, 4, 5,\ 1 \le \beta \le 3/2,\ N = e^{1.95}\} \\ &\qquad = \min\{h(m, 1, e^{1.95}) | m = 3, 4, 5\} > 0. \end{aligned}$$

The coefficient of $\ln N$ is positive for $m \ge 2$ and $1 \le \beta \le 3/2$, so $h(m, \beta, N) > 0$ for $m = 3, 4, 5$, $1 \le \beta \le 3/2$, and $N \ge 8 > e^{1.95}$. □

THEOREM 6.5. *For* $\beta > 3/2$ *and* N *sufficiently large*, $BT(N) < BI(N)$.

PROOF: By Lemma A2,

$$BT(N) \le BT_4(N) \le HBT_4(N) = \tfrac{1}{2}(\tfrac{3}{2} + \beta)\log_2 N + c_1$$

where c_1 does not depend on N. By Lemma A5,

$$BI(N) \ge \beta \log_2 N + c_2$$

where c_2 does not depend on N. Since $\beta > 3/2$, $\beta > (1/2)(3/2 + \beta)$, so $BI(N) > BT(N)$ for sufficiently large N. □

For each integer $m \ge 3$, we define

$$BR_m(N) = \min\{BRm, d(N) | d \text{ integer} \ge 1,\ m^{d-1} \le N\}.$$

LEMMA A6. *For* $m \geq 3$, $BR_m(N) = BR_{m,d}(N)$ *where*

$$d = \left\lceil log_m \left(\frac{N(m-1)}{2 + \beta(m-1)} \right) \right\rceil.$$

PROOF: The proof is similar to Lemma A1. □

LEMMA A7. $LBR_m(N) \leq BR_m(N) \leq HBR_m(N)$ *where*

$$LBR_m(N) = \frac{1}{2}(2 + \beta(m-1)) \left(log_m \left(\frac{N \ ln \ m}{2 + \beta(m-1)} \right) + \frac{1}{ln \ m} \right) - \frac{1}{2},$$

and

$$HBR_m(N) = \frac{1}{2}(2 + \beta(m-1)) \left(log_m \left(\frac{(m-1)N}{2 + \beta(m-1)} \right) + \frac{1}{m-1} \right) - \frac{1}{2}.$$

PROOF: The proof is similar to Lemma A2. □

LEMMA A8. *For integers* $m \geq 3$, $1 \leq \beta \leq 1/(log_2 3 - 1) \simeq 1.7095$, *and integers* $N \geq max\{m, 6\}$, $LBR_m(N) \geq min\{LBR_3(N), LBR_4(N)\}$.

PROOF: We can write the derivative of $LBR_m(N)$ with respect to m as

$$\frac{\partial LBR_m(N)}{\partial m}(m, \beta, N) = \frac{1}{2m(\ln \ m)^2} A(m, \beta, N) \cdot D(m, \beta), \tag{A11}$$

where

$$A(m, \beta, N) = \ln \left(\frac{N \ \ln \ m}{2 + \beta(m-1)} \right)$$

and

$$D(m, \beta) = \beta(m \ \ln \ m - m + 1) - 2.$$

A straightforward analysis of A and D shows that $A(m, \beta, N) > 0$ for $m \geq 3$, $\beta \leq 1.71$, $N \geq \max\{m, 6\}$, and that $D(m, \beta) > 0$ for $m \geq 4$, $\beta \geq 1$. Thus (A11) is positive for $m \geq 4$, $\beta \leq 1.71$, $N \geq \max\{m, 6\}$, so that $LBR_m(N) \geq LBR_4(N)$ for $m \geq 4$. □

THEOREM 6.6. *For* $N \geq 8$ *and* $1 \leq \beta \leq 1/(log_2 3 - 1) \simeq 1.7095$, $BI(N) < BR(N)$.

PROOF: Note that since $N \geq 8$ and $\beta < 1.71$, $N > 4\beta$. We will show that

$$LBR_m(N) \geq \beta \log_2(N/\beta) + 1/2 - \beta \quad \text{for} \quad m = 3, 4,$$

from which our result follows by Lemmas A5, A7, and A8.

We define

$$h(m, \beta, N) = LBR_m(N) - (\beta \ \log_2(N/\beta) + \tfrac{1}{2} - \beta).$$

A straightforward analysis of

$$\frac{\partial h}{\partial \beta}(m, \beta, e^{1.8})$$

shows that $h(3, \beta, e^{1.8}) > 0$ and $h(4, \beta, e^{1.8}) > 0$ for $1 \leq \beta \leq 1.71$. (Note that $e^{1.8} \simeq 6.05 < 8$.) Also, it is easy to show that

$$\frac{\partial h}{\partial N}(4, \beta, N) > \frac{\partial h}{\partial N}(3, \beta, N)$$

and that

$$\frac{\partial h}{\partial N}(3, \beta, N) \geq 0 \quad \text{iff} \quad \beta \leq \frac{1}{\log_2 3 - 1}.$$

Thus for $N \geq e^{1.8}$, $m = 3, 4$, and $1 \leq \beta \leq 1/(\log_2 3 - 1)$, $h(m, \beta, N) > 0$. □

THEOREM 6.7. *For $\beta > 1/(\log_2 3 - 1)$ and N sufficiently large, $BR(N) < BI(N)$.*

PROOF: By Lemma A7, $BR(N) \leq BR_3(N) \leq HBR_3(N) = (1 + \beta)\log_3 N + c_1$ where c_1 does not depend on N. By Lemma A5, $BI(N) \geq \beta \log_2 N + c_2$ where c_2 does not depend on N. Since $\beta > 1/(\log_2 3 - 1)$, $(1 + \beta)/\log_2 3 > \beta$. Hence for sufficiently large N, $BR(N) < BI(N)$. □

Next we turn our attention to the details of the study of optimal trees in Section 3. We begin this study with further definitions and notation:

$l(t)$: maximum length of tree t ($l(t) = \max\{l(v) | v \text{ in } t\}$ unless t is empty, in which case $l(t) = -\infty$);

$K(t)$: the tree of nongoal nodes in t (containing exactly those nongoal nodes in t connected to the root via a path of only nongoal nodes);

form(t): t is formed by assigning $k(t)$ nongoal nodes to minimal length positions and then assigning $n(t)$ goal nodes to minimal length available positions;

$P(N)$: path length of an optimal tree with N goal nodes ($P(N) = \min\{P(t) | n(t) = N\}$).

The next three lemmas are offered without proof. They provide a review of the definitions and lemmas of Section 3 dealing with optimal trees:

LEMMA A9. $opt(t) \leftrightarrow form(t) \wedge feasible(n(t), k(t))$.

LEMMA A10. $form(t_1) \wedge form(t_2) \wedge n(t_1) = n(t_2) \wedge k(t_1) = k(t_2) \rightarrow l(t_1) = l(t_2) \wedge l(K(t_1)) = l(K(t_2)) \wedge P(t_1) = P(t_2)$.

LEMMA A11. $opt(t) \rightarrow l(t) \geq l(K(t)) + \beta + 1$.

LEMMA A12. $form(t_1) \wedge form(t_2) \wedge n(t_1) = n(t_2) \wedge k(t_1) \leq k(t_2) \wedge l(t_2) > l(K(t_2)) + \beta \rightarrow l(t_1) \geq l(t_2)$.

PROOF: Assume the hypotheses and, without loss of generality, assume $K(t_1)$ is a subset of $K(t_2)$ (if not, then there is a length preserving transformation of t_2 with the property). If *goal*(v) in t_1 but v is not a node of t_2, then $l(t_2) \leq l(v) \leq l(t_1)$ and we are done. Thus assume *goal*(v) in $t_1 \rightarrow v$ in t_2. We can now set up a 1–1 correspondence f between the goal nodes of t_1 and goal nodes of t_2 satisfying:

(a) *goal*(v) in $t_1 \wedge$ *goal*(v) in $t_2 \rightarrow f(v) = v$, and

(b) *goal*(v) in $t_1 \wedge$ *nongoal*(v) in $t_2 \rightarrow f(v)$ is a minimal length goal node of $st_\beta(v)$.

Property (b) can be satisfied because $st_\beta(v)$ is empty in t_1 but must contain a goal node in t_2 since $l(t_2) > l(K(t_2)) + \beta$. Then

$$l(t_2) = \max\{l(f(v)) | goal(v) \text{ in } t_1\}$$

and

$$goal(v) \text{ in } t_1 \rightarrow \text{either } l(f(v)) = l(v) \leq l(t_1) \text{ or } l(f(v)) \leq l(K(t_2)) + \beta < l(t_2).$$

Thus $l(t_2) \leq l(t_2)$. □

LEMMA A13. $k_1 \geq k_2 \wedge feasible(N, k_1) \wedge feasible(N + 1, k_2) \rightarrow feasible(N, k_2) \wedge feasible(N + 1, k_1)$.

PROOF: Suppose t_1 and t_2 are optimal trees with N and $N + 1$ goal nodes and k_1 and k_2 ($k_1 > k_2$) nongoal nodes, respectively. Let t_3 be the tree formed by adding one goal node to t_1 at a minimal length position so that *form*(t_3). Then $l(t_3) \geq l(t_1) \geq l(K(t_1)) + \beta + 1 = l(K(t_3)) + \beta + 1 > l(K(t_3)) + \beta$ (by Lemma A11), and hence (by Lemma A12), $l(t_3) \leq l(t_2)$.

Let t_4 be the tree obtained from t_2 by deleting one maximal length leaf. We have $P(t_2) - l(t_2) = P(t_4) \geq P(t_1)$. Thus

$$P(t_2) \geq P(t_1) + l(t_2) = P(t_3) - l(t_3) + l(t_2) \geq P(t_3)$$

and t_3 is optimal. Also

$$P(t_4) = P(t_2) - l(t_2) \leq P(t_3) - l(t_3) = P(t_1)$$

so t_4 is optimal. □

LEMMA A14. $k_2 \geq k_1 + 2 \wedge \mathit{feasible}(N, k_1) \wedge \mathit{feasible}(N+1, k_2) \rightarrow \mathit{feasible}(N, k_1 + 1) \wedge \mathit{feasible}(N+1, k_2 - 1)$.

PROOF: Let t_1 and t_4 be trees fulfilling the hypotheses:

$$\mathit{opt}(t_1) \wedge \mathit{opt}(t_4) \wedge n(t_1) = N \wedge k(t_1) = k_1 \wedge n(t_4) = N+1 \wedge k(t_4) = k_2 \wedge k_2 \geq k_1 + 2.$$

We construct trees t_2 and t_5 and establish that they fulfill the conclusion:

$$\mathit{opt}(t_2) \wedge \mathit{opt}(t_5) \wedge n(t_2) = N \wedge k(t_2) = k_1 + 1 \wedge n(t_5) = N+1 \wedge k(t_5) = k_2 - 1.$$

Let t_2 be obtained from t_1 by changing a minimal length goal node v_1 into a nongoal node, adding one goal node $s_\beta(v_1)$, and rearranging goal nodes so that $\mathit{form}(t_2)$. This rearranging is accomplished by moving r additional goals into $st_\beta(v_1)$ from maximal length positions where $r = \lfloor l(t_2) - l(v_1) - \beta \rfloor$. Let $d_1 = P(t_2) - P(t_1)$. Then

$$l(t_1) + (r-1)l(t_2) \leq \beta + r(l(v_1) + \beta) + r(r+1)/2 - d_1.$$

Let t_3 be obtained from t_2 by adding a goal node at a minimal length position. If $l(t_2) \geq l(v_1) + \beta + 1$, then $l(t_3) \leq l(t_1)$, since $r \geq 1$.

Let t_5 be obtained from t_4 by changing a maximal length nongoal node v_2 into a goal node and redistributing $r' = \lfloor l(t_4) - l(v_2) - \beta \rfloor$ goal nodes so that $\mathit{form}(t_5)$. (We assume without loss of generality that the maximum length node of $st_\beta(v_2)$ has length $> l(t_4) - 1$.) Note that $r' \geq 1$ since v_2 is a nongoal node of t_4. Let $d_2 = P(t_5) - P(t_4)$.

Let t_6 be obtained by eliminating one maximal length goal node from t_5. Then

$$l(t_5) + (r'-1)l(t_6) \geq \beta + r'(l(v_2) + \beta) + r'(r'+1)/2 + d_2.$$

Also

$$\mathit{form}(t_6) \wedge l(t_6) > l(K(t_6)) + \beta$$

so, by Lemma A12, $l(t_6) \leq l(t_2)$. Hence $l(t_4) \leq l(t_6) \leq l(t_2)$. Since $l(v_2) \geq l(v_1)$, we have

$$l(t_2) \geq l(t_4) \geq l(v_2) + \beta + 1 \geq l(v_1) + \beta + 1.$$

Thus $l(t_3) \leq l(t_1)$. By Lemma A12, $l(t_5) \leq l(t_3)$. Note also $r \geq r'$.

We can now rewrite our previous results as

$$-d_1 \geq l(t_1) - l(v_1) + (r'-1)(l(t_2) - l(v_1) - \beta) - 2\beta - \frac{r'(r'+1)}{2} + (r - r') \cdot \left(l(t_2) - l(v_1) - \beta - \frac{r + r' + 1}{2} \right)$$

$$\geq l(t_5) - l(v_2) + (r'-1)(l(t_6) - l(v_2) - \beta) - 2\beta - \frac{r'(r'+1)}{2} + (r - r') \cdot \left(l(t_2) - l(v_1) - \beta - \frac{r + r' + 1}{2} \right)$$

$$\geq d_2 + (r - r') \left(l(t_2) - l(v_1) - \beta - \frac{r + r' + 1}{2} \right)$$

$$\geq d_2, \quad \text{since } r \geq r' + 1 \rightarrow l(t_2) - l(v_1) - \beta \geq \frac{r + r' + 1}{2}.$$

But $d_1 \geq 0$ and $d_2 \geq 0$. Thus $d_1 = d_2 = 0$ and t_2 and t_5 are optimal. □

THEOREM 3.1 *Increasing the number of goal nodes by* 1 *either increases by* 1 *or leaves constant the number of nongoal nodes in an optimal tree.*

PROOF: The proof is immediate from Lemmas A13 and A14. □

Finally we develop the machinery for a proof of Theorem 3.2. For the rest of this Appendix, x ranges over the set $D_\beta = \{a + b\beta | a, b \text{ nonnegative integers}\}$. Let

$$p(x) = \max\{N | \exists t \cdot opt(t) \ \& \ N = n(t) \ \& \ l(t) \leq x\}.$$

Let

$$q(x) = \min\{N | \exists t \cdot opt(t) \ \& \ N = n(t) \ \& \ l(t) \geq x\}.$$

Let z_1 be any increasing function on D_β such that where defined

$$z_1(x) = z_1(x - 1) + z_1(x - \beta), \tag{A12}$$

and

$$z_1(x) \geq x + 1 \quad \text{for} \quad 0 \leq x, \ x \text{ an integer.} \tag{A13}$$

Let z_2 be any increasing function on D_β such that where defined

$$z_2(x) = z_2(x - 1) + z_2(x - \beta), \tag{A14}$$

and

$$z_2(x) \leq x + 1 \quad \text{for} \quad 0 \leq x < \beta + \sqrt{4\beta + 1}. \tag{A15}$$

LEMMA A15. *$p(x) \leq z_1(x)$ for x in D_β and $q(x) \geq z_2(x)$ for x in D_β.*

PROOF: The proof is by straightforward induction on x. □

Let r be the unique positive solution to $r^\beta = r^{\beta-1} + 1$.

LEMMA A16. *$z_1(x) = r^{x+\beta-1}$ satisfies* (A12) *and* (A13) *and $z_2(x) = r^{x+1-\sqrt{4\beta+1}}$ satisfies* (A14) *and* (A15).

PROOF: Any function z of the form $z(x) = r^{x+w}$ (where w is a constant) satisfies both (A12) and (A14). Thus we need only show that z_1 satisfies (A13) and z_2 satisfies (A15).

Now $z_1(0) = r^{\beta-1} > 1$, $z_1(1) = r^\beta = r^{\beta-1} + 1 > 2$, and $z_1(x + 1) - z_1(x) = r^{x+\beta} - r^{x+\beta-1} = r^{x-1} \geq 1$ for $x \geq 1$. Thus, by induction, $z_1(x) \geq x + 1$ for nonnegative integers.

Let $f(x) = x + 1 - z_2(x)$. A straightforward analysis of the derivative of f with respect to x shows that, if $f(0) \geq 0$ and $f(\beta + \sqrt{4\beta + 1}) \geq 0$, then $f(x) > 0$ for $0 < x < \beta + \sqrt{4\beta + 1}$. But (A15) is equivalent to $f(x) \geq 0$ for $0 \leq x < \beta + \sqrt{4\beta + 1}$. Thus we need only show $f(0) \geq 0$ and $f(\beta + \sqrt{4\beta + 1}) \geq 0$. Now $f(0) = 1 - r^{1-\sqrt{4\beta+1}} > 0$ and $f(\beta + \sqrt{4\beta + 1}) = \beta + \sqrt{4\beta + 1} + 1 - r^{\beta+1}$. Let $g(r) = \beta + \sqrt{4\beta + 1} + 1 - r^{\beta+1}$ where β is considered a function of r. It will be sufficient to show that $g(r) \geq 0$ in the range of r corresponding to $\beta \geq 1$.

By the definition of r, we can expand $r^{\beta+1}$ to $r + r^\beta$, to $r + 1 + r^{\beta-1}$, and by induction to $r + 1 + (r - 1)^{-1}$ or $r^2/(r - 1)$. Thus we can write $\beta = 1 - (\ln(r - 1))/\ln r$, showing incidentally that $1 < r \leq 2$ for $\beta \geq 1$. Let $\beta'(r)$ be the derivative of β with respect to r. Then $\beta'(r) < -((\ln r)(r - 1))^{-1} < 0$ for $1 < r < 2$. Let $g'(r)$ be the derivative of g with respect to r. Then $g'(r) < \beta'(r) + r(r - 2)/(r - 1)^2 < 0$ for $1 < r < 2$. But $g(2) = \sqrt{5} - 2 > 0$. Thus $g(r) > 0$ for $1 < r \leq 2$. This completes the proof of the lemma. □

THEOREM 3.2. *Let t be an optimal tree. Then*

$$\log_r(n(t)) - \beta + 1 \leq l(t) \leq \log_r(n(t)) - 1 + \sqrt{4\beta + 1},$$

where r is the unique positive real root of $r^\beta = r^{\beta-1} + 1$.

PROOF: Lemmas A15 and A16 show that $p(x) \leq r^{x+\beta-1}$ for x in D_β and that $q(x) \geq r^{x+1-\sqrt{4\beta+1}}$ for x in D_β. By definition, $q(l(t)) \leq n(t) \leq p(l(t))$. Thus

$$r^{l(t)+1-\sqrt{4\beta+1}} \leq n(t) \leq r^{l(t)+\beta-1},$$

from which the theorem follows by taking logs and rearranging. □

This theorem completes our Appendix.

ACKNOWLEDGMENTS. The authors would like to thank M. Auslander, W. Burge, R. Fagin, A. Heller, B. Rosen, D. Sacks, and L. Woodrum for their comments, criticisms, corrections, and other contributions to this and earlier versions of "Search Within a Page." They would like to thank the referee for a careful reading. They would also like to thank S. Dwan and M. Price for dealing with the complexity of typing this manuscript.

REFERENCES

(Note. References [6, 7, 8, 9] are not cited in the text.)

1. BAYER, R., AND MCCREIGHT, E. Organization and maintenance of large ordered indices. *Acta Inform. 1* (1972), 173–189.
2. CHANDRA, A.K., AND STRONG, H.R. A theory of index strategies. RC 5970, IBM T.J. Watson Res. Ctr., Yorktown Heights, N.Y., 1976.
3. KNUTH, D.E. *The Art of Computer Programming, Vol. 3: Sorting and Searching.* Addison-Wesley, Reading, Mass., 1973.
4. MARKOWSKY, G. An analysis of indexing strategies. RC 6038, IBM T.J. Watson Res. Ctr., Yorktown Heights, N.Y., 1976.
5. MARUYAMA, K. Index structures for virtual memory–comparison between B-trees and M-trees." RC 5258, IBM T.J. Watson Res. Ctr., Yorktown Heights, N.Y., 1975.
6. MARUYAMA, K., AND SMITH, S.E. Analysis of design alternatives for virtual memory indexes. RC 5087, IBM T.J. Watson Res. Ctr., Yorktown Heights, N.Y., 1975.
7. OS/VS2 virtual storage access method (VSAM) logic release 3. SY26-3825-0, IBM.
8. Planning for enhanced VSAM under OS/VS. GC26-3842-0, IBM.
9. STANFEL, L.E. Tree structures for optimal searching. *J. ACM 17*, 3 (July 1970), 508–517.
10. WOODRUM, L.J. Radix partition trees. IBM Dept. 075, East Fishkill, N.Y., 1977 (to be published).

RECEIVED OCTOBER 1978; REVISED JANUARY 1979