SLAC-PUB-1679
November 1975

AN ALMOST OPTIMAL ALGORITHM FOR UNBOUNDED SEARCHING

Jon Louis Bentley*
Department of Computer Science
University of North Carolina
Chapel Hill, North Carolina 27514

and

Stanford Linear Accelerator Center
Stanford University, Stanford, California 94305**

Andrew Chi-Chih Yao
Department of Mathematics
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

ABSTRACT

Unbounded searching can be viewed as searching an ordered table of infinite size. In this paper we present an algorithm for unbounded searching and derive a lower bound on the number of comparisons required which shows that the algorithm is very nearly optimal.

(Submitted to Information Processing Letters)

---

* Work supported in part by U. S. Energy Research and Development Administration under contract E(04-3)515 and in part by a National Science Foundation Graduate Fellowship.

** Work was done while this author was at the Stanford Linear Accelerator Center.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences, under Contract No. DE-AC02-76SF00515.

1. Introduction

Many search methods based on comparisons of keys are described by Knuth in [6, section 6.2]. His work, however, deals exclusively with searching in a table of finite size (that is, searching in a bounded key space). In this paper we will consider the problem of comparison searching in an unbounded key space.

To give an exact formulation of the problem to be attacked we define N^+ to be the set of positive integers and specify the function $F:N^+ \rightarrow \{X,Y\}$ as

$$F(j) = \begin{cases} X \text{ for } j < n \\ Y \text{ for } j \geq n \end{cases}$$

where n is an integer that uniquely defines F. The problem of unbounded searching is the following: give an algorithm to determine n, using as primitive operations only comparisons of F(i) to X for a sequence of $i \in N^+$ chosen by the algorithm. That is, the algorithm must determine the unique n such that F(n) = Y and F(n-1) = X by testing different values of F(i). We say that n is the <u>solution</u> to the unbounded searching problem.

For a given algorithm A let us define the cost function $C_A:N^+ \rightarrow N^+$ as $C_A(n) = m$ iff algorithm A uses m evaluations of F to determine that n is the solution to the unbounded search problem. If we were given a probability distribution of n we could compare two unbounded searching algorithms by comparing their average search times for the two distributions. For lack of such knowledge we will consider only asymptotic performance and say that algorithm A is better

than (equal to, worse than) algorithm B iff $\lim_{n\to\infty} (C_A(n) - C_B(n))$ is negative (zero, positive).

It is easy to see an isomorphism between the problem of unbounded searching and the problem of table lookup in an ordered table of infinite size. Given $S = S_1, S_2, S_3, \ldots$, a strictly increasing infinite sequence of reals ($S_k \in R$ and $S_{k+1} > S_k$ for all $k \in N^+$), suppose that we are asked to find the unique first element in S that is greater than or equal to a fixed $z \in R$ using as a primitive operation only the question "Is $S_i < x$?" for $i \in N^+$. To solve this table lookup problem, use $F(i) = X$ iff $S_i < z$; the desired element of S is then S_n where n is the solution to the isomorphic problem of unbounded searching.

It is also easy to see that for any deterministic unbounded search algorithm A there is a corresponding binary encoding of the integers, constructed as follows: for every $i \in N^+$, the codeword representing i is $S_i = a_1 a_2 \ldots a_{C_A(i)}$, where $a_m = 1$ iff the mth evaluation of F is Y when using algorithm A to find i as the solution to the unbounded search problem. Notice that the length of the codeword S_i representing i is $C_A(i)$. Clearly the set $\{S_i\}$ is a prefix set (i.e., S_i is not a prefix of S_j for $j \neq 1$); if it were not a prefix set then algorithm A would not be able to terminate accurately for the i in violation of the prefix definition.

The problem of unbounded searching arises in many diverse areas. Suppose that one wants to find the zero of function $G:N^+ \to R$ that is known to cross the x-axis only once; this can be viewed as an unbounded searching problem if one ignores such (possibly misleading)

properties as the derivatives of G. Testing a system for a breaking point might involve an unbounded search; one knows that the system functions with zero workload and wishes to find that workload at which the system no longer functions. Unbounded searching could be used in searching an extremely large ordered table if one wanted to pay a search cost proportional to the item's distance from the front of the table. We have already seen that every unbounded search strategy yields a uniquely decipherable prefix encoding of the integers; therefore, good search strategies can have important applications in information theory.

We investigate a number of algorithms for solving the unbounded searching problem in section 2. In section 3 we will show a lower bound for the cost function $C_A(n)$ for any unbounded searching algorithm A that is almost attained by one of the algorithms given in section 2. Possible topics for further work in this area are mentioned in section 4.

2. Unbounded searching algorithms

In this section we will examine a number of algorithms for unbounded searching.

Algorithm B_0 (Unary Search)

The most straightforward algorithm for unbounded searching is to test F(1), F(2), ..., until F(n) = B. It is easy to see that the cost of this algorithm is $C_{B_0}(n) = n$.

Algorithm B_1 (Binary Search)

The next algorithm suggested is the standard bounded binary search

algorithm, using the "gambler's strategy" of doubling successive guesses to provide the upper bound needed for the binary search. More precisely, the first stage of the algorithm determines $m = \lfloor \lg n \rfloor + 1$ by successively evaluating $F(2^i-1)$ for $i = 1,2,\ldots$ until $F(2^m-1) = B$, at which time we know that $2^{m-1} \le n \le 2^m-1$. The second stage then uses a standard bounded binary search on those 2^{m-1} elements to determine the exact value of n. The first stage will require $m = \lfloor \lg n \rfloor + 1$ evaluations of F and the second stage will require $\lg 2^{m-1} = m-1 = \lfloor \lg n \rfloor$ evaluations of F, so the total cost is $C_{B_1}(n) = 2 \lfloor \lg n \rfloor + 1$. It is helpful to view an unbounded search algorithm as a decision tree in which the label i on an internal node represents the evaluation of F(i), a left (right) branch corresponds to the outcome of the evaluation being Y (X), and external nodes represent solutions to the problem. Figure 1 is the decision tree representation of algorithm B_1.

Algorithm B_2 (Double Binary Search)

The first stage of algorithm B_1 essentially uses unary search (algorithm B_0) to determine $m = \lfloor \lg n \rfloor + 1$ by successively evaluating $F(2^i-1)$. We could, however, make use of algorithm B_1 to find m by replacing every occurrence of the expression F(j) in stage one of B_1 with the expression $F(2^j-1)$. The cost of the second stage of the resulting algorithm B_2 will be the same as the second stage of B_1 (that is, $m-1 = \lfloor \lg n \rfloor$), but the cost of the first stage will be reduced from $C_{B_0}(m) = m$ to $C_{B_1}(m) = 2 \lfloor \lg(m) \rfloor + 1$, so the total cost of the algorithm (substituting $m = \lfloor \lg n \rfloor + 1$) is

$$C_{B_2}(n) = \lfloor \lg n \rfloor + 2 \lfloor \lg(\lfloor \lg n \rfloor + 1) \rfloor + 1.$$

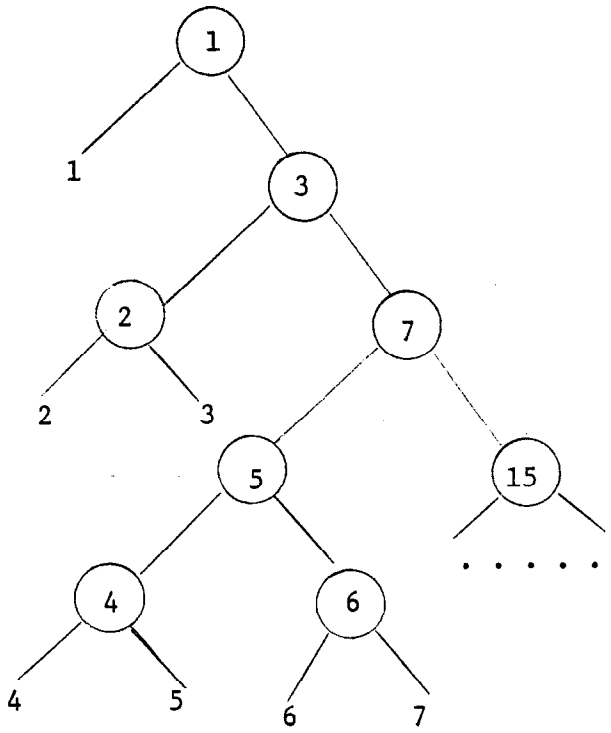

Figure 1. Decision tree representation of algorithm B_1

Algorithm B_k (k-nested Binary Search)

To synthesize algorithm B_2 from B_1 we replaced a unary search by a binary search. This same technique could be used to create a new algorithm B_3 from B_2; we would use algorithm B_1 to determine $\lfloor \lg(\lfloor \lg n \rfloor + 1) \rfloor + 1$ by rewriting F(i) in algorithm B_1 to be $F(2^{2^i-1}-1)$; (algorithm B_2 uses a unary search to find $\lfloor \lg(\lfloor \lg n \rfloor + 1) \rfloor + 1$.) In general this technique could be applied to algorithm B_{k-1} to yield a new algorithm B_k.

To analyze algorithm B_k we define $\ell^j(n)$ recursively by

$\ell^0(n) = n$, and

$\ell^{j+1}(n) = \lfloor \lg \ell^j(n) \rfloor + 1$.

We define $L^j(n) = \ell^j(n) - 1$ for $j \in N$. It is easy to prove by induction that

$$C_{B_k}(n) = L^1(n) + L^2(n) + \ldots + L^{k-1}(n) + 2L^k(n) + 1$$
$$= \sum_{1 \le i \le k} L^i(n) + L^k(n) + 1. \qquad (1)$$

Our discussion of algorithm B_1 provides a basis for the induction. We will not give a detailed form of the inductive part here but the spirit of that proof is that the $L^k(n) + 1$ in (1) represents the cost of a unary search and the cost of the corresponding binary search is $2L^{k+1}(n) + 1$. Since $C_{B_0}(n) = n = L^0(n) + 1$, (1) holds for any $k \in N$.

It is helpful to view the algorithm $B_0, B_1, B_2, \ldots$ as a progression. To do this we have represented the first few algorithms in Figure 2. Each box in the figure corresponds to what might be thought of as a subroutine, and the tree structure represents the calling hierarchy. Rounded boxes call further routines; rectangular boxes represent

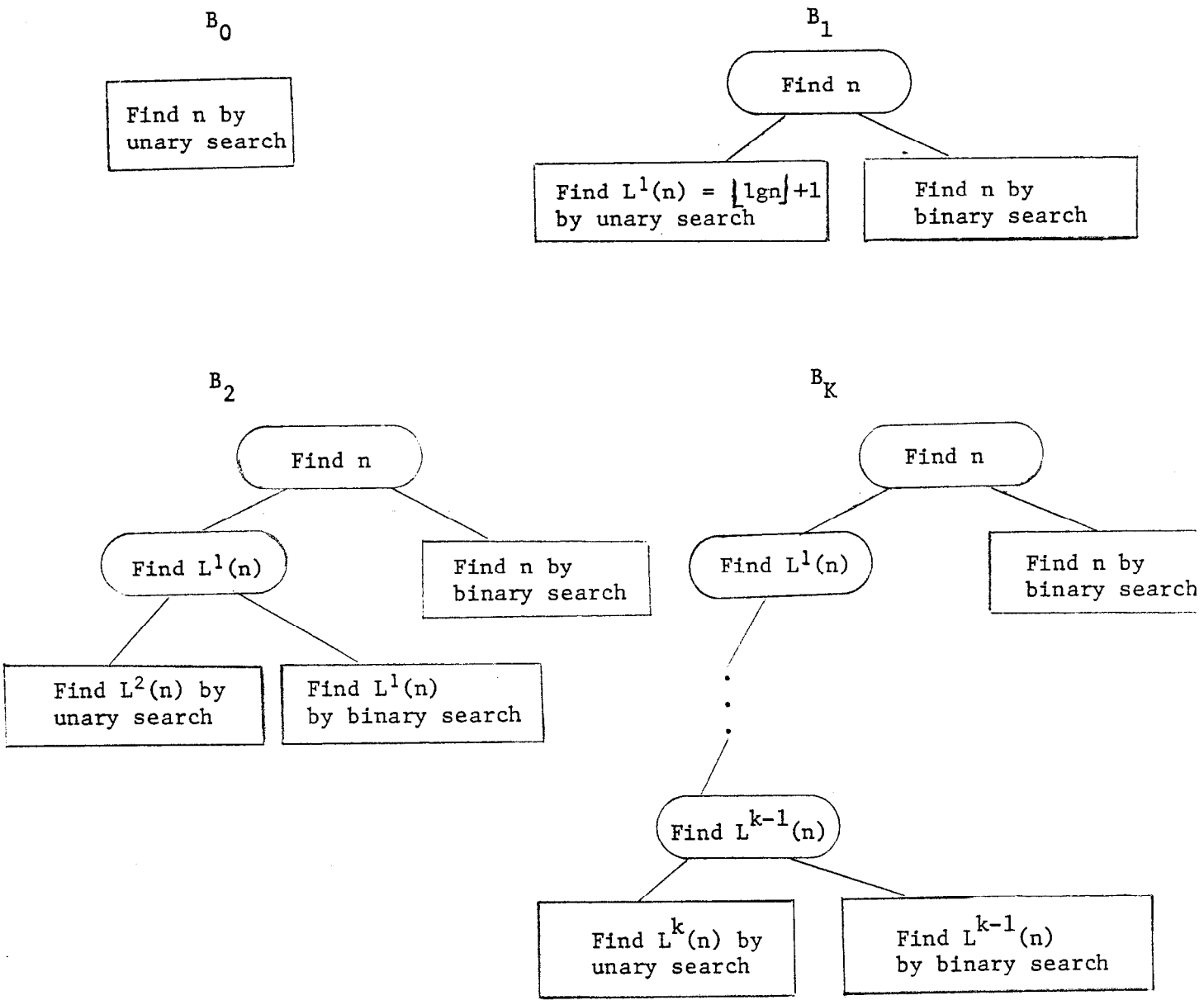

Figure 2. Succession of algorithm $B_0, B_1, B_2, \ldots, B_k, \ldots$

basic operations. The left "subroutine" of a round box is called before the "right" subroutine.

Algorithm U (The Ultimate Algorithm)

Now that we have at our disposal an infinite number of unbounded searching algorithms, which one shall we use for a particular search? If we choose B_k for a fixed k we can fall into either of two traps: if k is small and n is large, we are paying the high cost of $2L^k(n)$ when we could be paying only $L^k(n) + 2L^{k+1}(n)$, which is quite a significant difference for some values of n and k. On the other hand, if k is large and n is small, we could be using "too sophisticated" an algorithm and therefore paying a lot of comparisons for very little information. These two examples hint to us that we should choose k as a function of n.

We therefore propose that algorithm U consist of two parts: the first stage will choose an appropriate value of k and the second stage will then use algorithm B_k to solve the problem. The above two examples suggest that to avoid both traps we should choose the least k such that $L^k(n)$ is constant. In particular, we propose to choose $k = L^*(n) = \min j$ such that $L^j(n) = 1$. Since L is defined by ℓ, it will be easier to work with $L^*(n) = \ell^*(n) = \min j$ such that $\ell^j(n) = 2$. By the definition of $\ell^j(n)$ we can see that $\ell^*(n)$ is the least $j \in I$ such that $g(j) \geq n$ where g is defined recursively as

$g(0) = 2$, and

$g(j+1) = 2^{g(j)}-1$.

(Notice that $L^*(n)$ has behavior similar to $\lg^*(n)$.) Thus the first

stage of algorithm U will determine k by testing F(g(0)), F(g(1)), ..., until F(g(k)) = B, and then use B_k to determine n.

The first stage of algorithm U will require $k+1 = L^*(n) + 1$ evaluations of F to find $L^*(n)$. Our analysis of algorithm B_k tells us the cost of the second stage. Thus the total cost of algorithm U is

$$C_U(n) = [1 + L^*(n)] + [C_{B_{L^*(n)}}(n)]$$

$$= [1 + L^*(n)] + [\sum_{1 \le i \le L^*(n)} L^i(n) + L^{L^*(n)}(n) + 1]$$

$$= 4 + L^*(n) + \sum_{1 \le i \le L^*(n)-1} L^i(n).$$

3. A lower bound

In this section we shall prove a lower bound for the cost function of any correct unbounded searching algorithm which shows that algorithm U given in section 2 is very nearly optimal. We demonstrated in section 1 that any unbounded searching algorithm yields an encoding of the integers. For a given encoding of the integers, let f(n) represent the number of bits used to represent the integer n. By the mapping of search algorithms to codes, any lower bound for f(n) also is a lower bound for the cost function $C_A(n)$, for any correct unbounded search algorithm A.

We saw in section 1 that the codes induced by search algorithms are prefix codes, which implies Kraft's inequality (see [3]):

$$\sum_{j \ge 1} \frac{1}{2^{f(j)}} \le 1. \qquad (2)$$

We shall presently show that (2) implies the following theorem:

Theorem A: For infinitely many n, $f(n) > \lg n + \lg^{(2)} n + \ldots + \lg^{(\lg^* n)} n - 2(\lg^* n)$

Proof: Define, for positive integers ℓ and n,

$$k_\ell = 2^{2^{\left.\right\}\ell}} - \ell - 2$$

$$n_\ell = 2^{2^{\left.\right\}k_\ell+\ell+1}} + 1$$

$$n'_\ell = 2^{2^{\left.\right\}k_\ell+\ell+2}}$$

and $K(n) = \lg^* n - \lg^*(\lg^* n) - 2$.

The following facts are easy to verify:

Fact 1: $K(n) = k_\ell$ if $n_\ell \le n \le n'_\ell$

$$\lg^{(k_\ell+1)} n'_\ell > 2^{k_\ell}$$

$$\lg^{(k_\ell+1)} n_\ell < k_\ell + \ell + 3$$

Fact 2: $$\int \frac{dx}{x(\lg x)(\lg^{(2)} x) \ldots (\lg^{(k)} x)} = (\ell n 2)^k \, \ell g^{(k+1)} x$$

Lemma B: Let $A = \{n \mid n_\ell \le n \le n'_\ell \quad \ell = 3,4,\ldots\}$. Then there exists infinitely many $n \in A$ such that $2^{f(n)} > n(\lg n)(\lg^{(2)} n) \ldots (\lg^{(k(n))} n)$

Proof of Lemma B: Suppose the lemma is false. Then there exists ℓ_0 such that $2^{f(n)} \le n(\lg n)(\lg^{(2)} n) \ldots (\lg^{(k(n))} n)$ if $n_\ell \le n \le n'_\ell$ for some $\ell \ge \ell_0$.

This implies, for $\ell \ge \ell_0$,

$$\sum_{j=1}^{\infty} \frac{1}{2^{f(j)}} > \sum_{n_\ell \le n \le n_\ell} \frac{1}{n \lg n \lg^{(2)} n \ldots \lg^{(k(n))} n}$$

$$\geq \int_{n_\ell}^{n'_\ell} \frac{dx}{x(\lg x)(\lg^{(2)} x) \ldots \lg^{(k_\ell)} x}$$

$$= (\ell n 2)^{k_\ell} \, {}_\ell g^{(k_\ell+1)} x \Big|_{x = n_\ell}^{x = n'_\ell}$$

$$> (\ell n 2)^{k_\ell} (2^{k_\ell} - k_\ell - \ell - 3)$$

$$= (2\ell n 2)^{k_\ell} (1 + 0(1)) \qquad (3)$$

where Facts 1 and 2 are used in the derivation of (3).

Since $2\ell n 2 > 1$, (2) implies $\sum_{j=1}^{\infty} \frac{1}{2^{f(j)}} = \infty$. This contradicts (1) and Lemma B is proved.

Thus, we have shown that for infinitely many $n \in A$,

$$f(n) > \ell g\, n + \ell g^{(2)}\, n + \ldots + \ell g^{(k(n)+1)}\, n$$
$$= \ell g\, n + \ell g^{(2)}\, n + \ldots + \ell g^{(\ell g^* n)}\, n - g(n) \qquad (4)$$

where $g(n) = \ell g^{(k(n)+2)}\, n + \ldots + \ell g^{(\ell g^* n)}\, n$.

Observe that, for $n_\ell \leq n \leq n_\ell^1$, $\ell g^{(k(n)+2)}\, n \leq \ell g^*\, n$. Therefore,

$$g(n) \leq \ell g^*\, n + \ell g(\ell g^*\, n) + \ldots + \ell g^{(\ell g^*(\ell g^*\, n))}(\ell g^*\, n)$$
$$< 2\ell g^*\, n \qquad \text{for all sufficiently large } \ell. \qquad (5)$$

Theorem A follows from (4) and (5) immediately.

4. Areas for further work

There are many ways in which the unbounded search problem can be extended. To model a multicomparator system, one might consider unbounded searching with primitive operations consisting of testing k different values of F. Notice that the outcome of such a test has k+1 different possibilities and hence can be described by lg (k+1) bits.

This problem is similar to Karp and Miranker's generalization of finding a maximum of a function [5] (see also Linn [4]). We considered the cost of evaluating F to be independent of the outcome of the evaluation; in certain applications, however, it might be much more expensive to have tested (for example) a Y rather than an X (perhaps in locating a breaking point of a system). In general, given costs x and y of evaluating F to X and Y, what is the optimal algorithm to use? Knuth describes in [6,section 6.2.1] a Fibonaccian search for a finite ordered table that has a better average search time (though worse worst-case time) than straightforward binary search [6, section 6.2.1]. Is the corresponding unbounded Fibonaccian search interesting?

It would be nice to close the gap between the running time of algorithm U and the lower bound derived in section 3.

It was demonstrated in section 1 that every unbounded search strategy suggests a prefix encoding of the integers. Indeed, Elias [1] has studied codes that are isomorphic to each of the search strategies in section 2, and Even and Rodeh [2] have studied a code similar to algorithm U. Conversely, does there exist a search strategy corresponding to every prefix code for the integers? Does the framework of unbounded searching provide any insight into problems in information theory? What are the implciations of the lower bound derived in section 3 to Elias' work?

References

1. Elias, P. "Universal codeword sets and representation of the integers." IEEE Trans. on Information Theory, vol. IT-21, no. 2, March 1975, pp. 194-203.

2. Even, S. and M. Rodeh. "Economical encoding of commas between strings." TECHNION Technical Report No. 54, July 1975, Haifa, Israel.

3. Gallager, P. E. Information Theory and Reliable Communication, Wiley, New York, 1968.

4. Linn, J. "General methods for parallel searching." Technical Report No. 61, Digital Systems Laboratory, Stanford Electronics Laboratories, Stanford University, May 1973.

5. Karp, R. M. and W. L. Mirankier. "Parallel minimax search for a maximum." J. of Comb. Theory 4, pp. 19-35.

6. Knuth, D. E. The Art of Computer Programming, volume 3 (Sorting and Searching), Addison-Wesley, 1975.