February 1988

UILU-ENG-88-2210
ACT-89

COORDINATED SCIENCE LABORATORY
College of Engineering
Applied Computation Theory

SEARCHING ON A TAPE

Scot W. Hornick
Sanjeev R. Maddila
Ernst P. Mücke
Harald Rosenberger
Steven S. Skiena
Ioannis G. Tollis

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-88-2210 ACT #89	5. MONITORING ORGANIZATION REPORT NUMBER(S) N/A
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION National Science Foundation and Amoco Foundation for Faculty Development
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Avenue Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) 1800 G Street, N.W. Washington, D.C. 20552
8a. NAME OF FUNDING/SPONSORING ORGANIZATION National Science Foundation & Amoco Foundation 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER NSF - ECS-84-10902 Amoco - CS 1-6-44862 & NSF - CCR-87-14565
8c. ADDRESS (City, State, and ZIP Code) 1800 G. Street, N.W. Washington, D.C. 20552	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

Searching on a Tape

12. PERSONAL AUTHOR(S)
Hornick, S.W., Maddila, S. R., Mücke, E. P., Rosenberger, H., Skiena, S. S., and Tollis, I. G.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM ______ TO ______	1988 February	18

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	comparison complexity, computational complexity, head movement complexity, searching algorithms, sequential access machine model, tape searching

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

In this paper, we consider the problem of minimizing the average time to search for a key in a sorted list of keys in a *sequential access machine model*, where both the time to access a key in the list and the time to compare it with the given search key are taken into account, e.g., a magnetic tape. The time to access a key in the list is proportional to the distance the head moves from its current location to reach the key. The time to read and compare the key with the search key is taken to be constant. We analyze two classes of algorithms and present a matching lower bound on the average search time. These results answer an open problem posed by Nishihara and Nishino [NN] regarding the optimal search algorithm for such a model. We then show that the organization of the input data is crucial in determining the SAM complexity of the search problem.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR   83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Searching on a Tape[†]

February 1, 1988

Scot W. Hornick, Sanjeev R. Maddila, Ernst P. Mücke,

Harald Rosenberger, Steven S. Skiena, and Ioannis G. Tollis

Abstract: In this paper, we consider the problem of minimizing the average time to search for a key in a sorted list of keys in a *sequential access machine model,* where both the time to access a key in the list and the time to compare it with the given search key are taken into account, e.g., a magnetic tape. The time to access a key in the list is proportional to the distance the head moves from its current location to reach the key. The time to read and compare the key with the search key is taken to be constant. We analyze two classes of algorithms and present a matching lower bound on the average search time. These results answer an open problem posed by Nishihara and Nishino [NN] regarding the optimal search algorithm for such a model. We then show that the organization of the input data is crucial in determining the SAM complexity of the search problem.

Index Terms: comparison complexity, computational complexity, head movement complexity, searching algorithms, sequential access machine model, tape searching.

[†]This research was supported in part by the National Science Foundation under grant ECS-84-10902, the Amoco Foundation for Faculty Development under grant CS 1-6-44862, and the National Science Foundation under grant CCR-87-14565.

S. W. Hornick and S. R. Maddila are with the Coordinated Science Laboratory and Department of Electrical and Computer Engineering, University of Illinois, Urbana, IL 61801.

E. P. Mücke, H. Rosenberger, and S. S. Skiena are with the Digital Computer Laboratory and Department of Computer Science, University of Illinois, Urbana, IL 61801.

I. G. Tollis was with the Coordinated Science Laboratory and Department of Computer Science, University of Illinois, Urbana, IL 61801. He is currently with the Department of Computer Science, University of Texas at Dallas, Richardson, TX 75083.

1. Introduction

Searching for a given key in a sorted list is a fundamental problem in computer programming [Knu]. This problem has been studied extensively in various models of computation by a number of researchers: the random-access machine (RAM) model [Knu,O,F], the hierarchical-memory machine (HMM) model [AACS] and the similar variable probe cost model of [Kni], and most recently, a *sequential access machine* (SAM) model [NN]. In this last model, the keys are stored in sorted order in a linear array (the *tape*), the tape head is initially to the left of the tape, and the cost of accessing key i on the tape, when the head is at position j, is proportional to the amount of head movement, i.e., $|i - j|$. Furthermore, the head has a small memory of its own in which it can store $O(1)$ tape position indices (e.g., the indices of the first and last keys of a subinterval on the tape) and the value of the search key. This model differs from the HMM model (using a linear cost function) and the variable probe cost model of [Kni] in that the access cost is a function of the *relative* location of the data to be accessed instead of the *absolute* location of the data.

Under the assumption that each key is equally likely to be accessed, [NN] showed that a variant of the classical Fibonacci search has less average head movement than the conventional binary search ($\approx .809n - o(n)$ vs. $n - o(n)$, where n is the number of keys in the list). However, if average head movement is the only cost function, then it is clear that the optimal algorithm is simply linear search, which achieves an average head movement of $n/2$. Of course, linear search does require, on average, a linear number of comparisons, as opposed to the logarithmic number required by either binary search or Fibonacci search. This suggests the inclusion of both the average number of comparisons and the average head movement in the cost function for the SAM complexity of searching; including both gives a more interesting and realistic model of computation since stopping the tape to read and compare each key may significantly slow head movement.

Here, we propose a searching complexity measure for the SAM model that captures both average number of comparisons and average head movement by combining them in a measure of time complexity. Let us denote by n the length of the sorted list and by k the index of a particular key position in the list. We can distinguish two cases of this problem: searching for a key that is present in the list and searching for a key that is absent from the list. We only analyze the time complexity of the latter case since it serves as an upper bound for the former. In this context, the index k refers to the index of the list element that would be the immediate successor of the search key if the search key was in the list, so $k \in [1,n]$ (we assume that the key cannot succeed the last list element). Let T_M be the time that is required to move the tape head one record forward or backward, where a *record* is a key and its associated information, and let T_C be the time that is required to read a key from the tape and compare it with the search key.[1] Then the total time required by an algorithm A to find the kth key in a sorted list of n elements can be expressed as a function:

$$S_A(n,k,T_M,T_C) \triangleq T_M M_A(n,k) + T_C C_A(n,k), \tag{1}$$

where $M_A(n,k)$ is the amount of head movement and $C_A(n,k)$ is the number of comparisons. Since we are interested in average case performance, we wish to find the expected value of these functions under the assumption that all n possible values of k are equally likely. Therefore, we define

$$\overline{S}_A(n,T_M,T_C) \triangleq T_M \overline{M}_A(n) + T_C \overline{C}_A(n), \tag{2}$$

where "$\overline{\ }$" denotes the expectation over all k. (For notational convenience, we will remove the explicit reference to the independent variables T_M and T_C from the average total time functions $\overline{S}_A(\cdot)$). Thus, the SAM complexities of binary (denoted by subscript B) and linear (denoted by subscript L) search are roughly

$$\overline{S}_B(n) \approx T_M n + T_C \log n \tag{3}$$

and

$$\overline{S}_L(n) \approx (T_M + T_C)n/2. \tag{4}$$

[1]Current magnetic tapes have densities in the range of 1600 to 6250 bpi/track, seven to nine tracks, and speeds around 120 inches/sec, so T_M is in the range of $1.3R$ to $5.2R$ µsec for an R character record, while T_C is typically in the range of 1 to 20 msec [G].

Using this model, we analyze two general classes of algorithms, one of which includes binary and Fibonacci search as special cases, and the other of which is simpler conceptually and slightly better in terms of performance. We then prove a lower bound that indicates that these classes of algorithms are asymptotically optimal within a constant factor. Finally, we make some remarks regarding improved list organizations for tape searching.

2. Fixed-Ratio Search

Binary search has long been known to minimize the average number of comparisons required to find a key in a sorted list. However, in [NN], the average head movement for binary search was shown to be $n - o(n)$, while a variant of the classical Fibonacci search has average head movement $\approx .809n - o(n)$. These two algorithms belong to the class of algorithms referred to as *fixed-ratio searches*. Such algorithms are parametrized by a constant ratio $r \in (0,1)$. The algorithm moves the head to the rnth key and compares it with the search key; if the two keys match, then the algorithm halts, otherwise, the algorithm recurs on the left or right sublist, depending on the result of the comparison (Figure 1). The algorithm is given concisely below. Initially, it is called with list L, search key σ, and pointers $h = 0$ and $t = n$; it returns the index k of the successor key in L.

```
FIXED_RATIO_SEARCH($L$, $\sigma$, $h$, $t$, $k$)
    **if** ($|t - h| = 1$) **then**
        $k := \max(h, t)$;
        **return**;
    **else**
        $p := \lceil r \cdot (t-h) \rceil$;
        **if** ($\sigma < L[h+p]$) **then**        /* This requires time $T_M |p| + T_C$. */
            FIXED_RATIO_SEARCH($L$, $\sigma$, $h+p$, $\min(h, t)$, $k$);
        **else**
            FIXED_RATIO_SEARCH($L$, $\sigma$, $h+p$, $\max(h, t)$, $k$);
        **endif**
        **return**;
    **endif**
```

For binary search, $r = \frac{1}{2}$, and, for the modified Fibonacci search of [NN], $r = 1 - \phi^{-1} = \frac{1}{2}(3 - \sqrt{5}) \approx .382$, where ϕ is the "golden ratio." [NN] posed the open problem of determining which r minimizes average

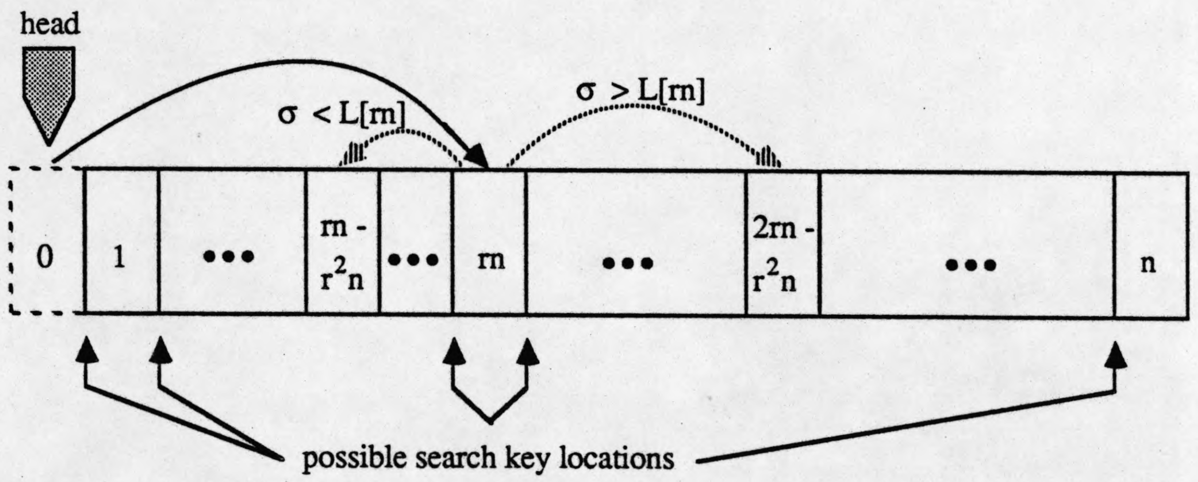

Figure 1. A recursive step of fixed-ratio search

head movement. As we pointed out earlier, though, if average head movement is the only cost function, then the optimal algorithm is simply linear search. On the other hand, if we adopt the model proposed above, which takes both head movement and comparisons into account, this question becomes interesting.

In this section, we will determine separately the average amount of head movement and average number of comparisons for a fixed-ratio search with parameter r. The recurrence equation for the average amount of head movement is

$$\overline{M}_{FR(r)}(n) = r\overline{M}_{FR(r)}(rn) + (1-r)\overline{M}_{FR(r)}((1-r)n) + rn, \tag{5}$$

where the algorithmic subscript $FR(r)$ denotes a fixed-ratio search algorithm with ratio r, and with the simplifying assumption that $\overline{M}_{FR(r)}(\cdot)$ is a continuous function of a real variable instead of a discrete function of an integer variable. The first term of the recurrence represents the probability that σ lies between the initial head position and the probe position times the average head movement while searching in that region, the second term represents the probability that σ lies between the probe position and the far end of the tape (as measured from the initial head position) times the average head movement while searching in that region. The last term represents the head movement required to reach the probe position in order to discriminate between the two cases. One can verify that Equation (5) has as its solution

$$\overline{M}_{FR(r)}(n) = \frac{n}{2(1-r)}. \tag{6}$$

A similar recurrence can be written for the average number of comparisons:

$$\overline{C}_{FR(r)}(n) = r\overline{C}_{FR(r)}(rn) + (1-r)\overline{C}_{FR(r)}((1-r)n) + 1, \tag{7}$$

which has as its solution

$$\overline{C}_{FR(r)}(n) = \frac{\log n}{H_2(r)}, \tag{8}$$

where $H_2(r) = -r\log r - (1-r)\log(1-r)$ is the binary entropy function and log is taken with respect to base two. Thus, the average total time for a fixed-ratio search is

$$\overline{S}_{FR(r)}(n) = T_M\overline{M}_{FR(r)}(n) + T_C\overline{C}_{FR(r)}(n) = \frac{T_M n}{2(1-r)} + \frac{T_C \log n}{H_2(r)}. \tag{9}$$

Note that, as r decreases (for $r \le ½$), the first term of (9) decreases, while the second increases. This tradeoff is exhibited in Figure 2.

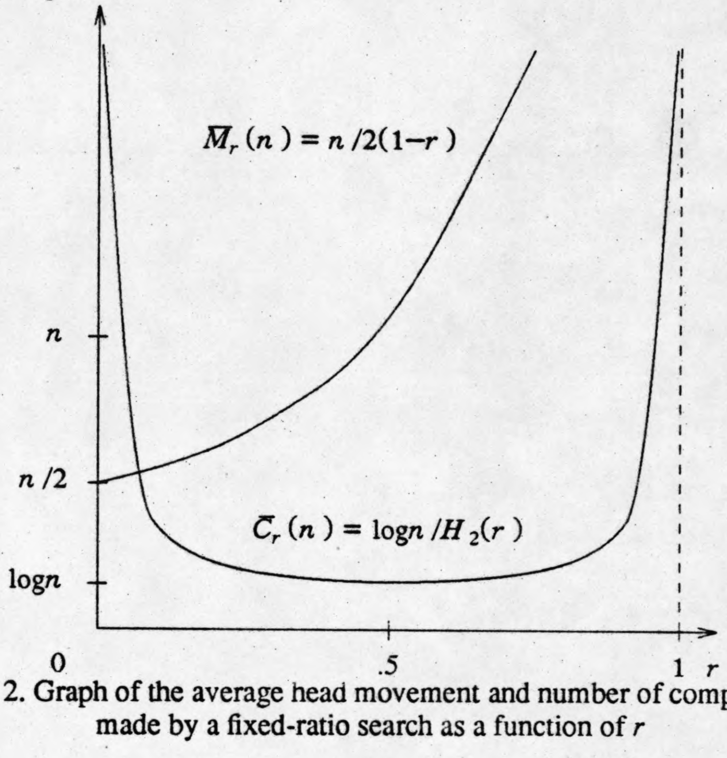

Figure 2. Graph of the average head movement and number of comparisons made by a fixed-ratio search as a function of r

To optimize a fixed-ratio search algorithm, r must be chosen so that

$$\frac{d}{dr}\bar{S}_{FR(r)}(n) = 0. \tag{10}$$

(Since both $\bar{M}_{FR(r)}(\cdot)$ and $\bar{C}_{FR(r)}(\cdot)$ are convex $\cup$ functions of r this will only occur at a minimum of $\bar{S}_{FR(r)}(\cdot)$.) Taking the derivative with respect to r in Equation (9), reveals that, for the optimal r, r^*,

$$\frac{2(1-r^*)^2\log((1-r^*)/r^*)}{(H_2(r^*))^2} = \frac{T_M n}{T_C \log n}. \tag{11}$$

While we cannot solve this equation for r^* explicitly, we can determine the asymptotic behavior of r^*, i.e., the behavior of the optimal r as n increases without bound for fixed values of T_M and T_C.

Increasing n (or increasing T_M/T_C) increases the right-hand side of Equation (11). On the other hand, decreasing r^* increases the left-hand side of (11). Therefore, r^* should be considered a very small number $\ll 1$ as n becomes sufficiently large. Neglecting the asymptotically insignificant terms of Equation (11), we obtain:

$$\frac{2\log(1/r^*)}{(r^*\log(1/r^*))^2} \approx \frac{T_M n}{T_C \log n}$$

$$\frac{4/r^{*2}}{\log(1/r^{*2})} \approx \frac{T_M n}{T_C \log n}$$

$$\frac{4/r^{*2}}{\log(4T_C/(T_M r^{*2}))} \approx \frac{T_M n}{T_C \log n},$$

which is clearly satisfied by

$$r^* \approx \sqrt{\frac{4T_C}{T_M n}} = 2\sqrt{\frac{T_C}{T_M n}}. \tag{12}$$

Therefore, contrary to the apparent expectations of [NN], the asymptotically optimal ratio for fixed-ratio search is not a constant but, rather, is inversely proportional to the square root of the length of the sorted list. The average total time for the resulting fixed-ratio search is

$$\bar{S}_{FR(r^*)}(n) \approx \frac{T_M n}{2(1-2\sqrt{T_C/(T_M n)})} + \frac{T_C \log n}{H_2(2\sqrt{T_C/(T_M n)})}.$$

Manipulating this equation and, again, neglecting insignificant terms, we find, for large n,

$$\bar{S}_{FR(r^*)}(n) \approx T_M \left[\frac{n}{2} + \sqrt{\frac{T_C n}{T_M}} \right] + T_C \sqrt{\frac{T_M n}{T_C}} = T_M \frac{n}{2} + 2\sqrt{T_M T_C n}\,, \tag{13}$$

which results in the following theorem.

Theorem 1: $\bar{S}_{FR(r^*)}(n) = T_M \frac{n}{2} + 2\sqrt{T_M T_C n} + o(\sqrt{n}\,)$.

In Equation (13), it is interesting to note that the term contributed by the moves in excess of the $n/2$ lower bound balances the term contributed by the comparisons in excess of the (asymptotically negligible) $\log n$ lower bound, i.e., both are $\sqrt{T_M T_C n}$. Also observe that, although we can optimize r, fixed-ratio search itself is suboptimal because the value of r is chosen initially and remains fixed and, therefore, cannot be optimal for the smaller recursive subproblems. We will expound on this idea in Section 4, where we discuss lower bounds.

3. Block Searching Algorithms

We now consider an alternate class of algorithms for searching on a tape, the *block searching algorithms*. We will partition the data into blocks of size rn for some r, and sequentially compare the last element of each block with the search key to ensure that we do not move the head past the block containing the search key. Thus, in this first stage, we never incur excess head movement greater than $2rn$ while we perform at most $1/r$ comparisons (Figure 3). Once we have found the block containing the key, we can either recur on that block or use almost any reasonable algorithm to find the key within the block. In this section, we will consider just two variants of this approach: the first, *block-binary search*, uses binary search to locate the key within its block, while the second, *block-linear search*, uses linear search.

```
BLOCK_BINARY_SEARCH($L, n, \sigma, k$)
    $p := 0$
    repeat
        $p := p + r \cdot n$
    until ($\sigma < L[p]$)    /* This requires time $T_M rn + T_C$. */
    BINARY_SEARCH($L[p - r \cdot n : p]$, $\sigma$, $k$)    /* This requires time $T_M(rn-1) + T_C \log(rn)$. */
    return
```

```
BLOCK_LINEAR_SEARCH($L, n, \sigma, k$)
    $p := 0$
    repeat
        $p := p + r \cdot n$
    until ($\sigma < L[p]$)    /* This requires time $T_M rn + T_C$. */
    LINEAR_SEARCH($L[p - rn : p]$, $\sigma$, $k$)    /* This requires average time $(T_M + T_C) rn/2$. */
    return
```

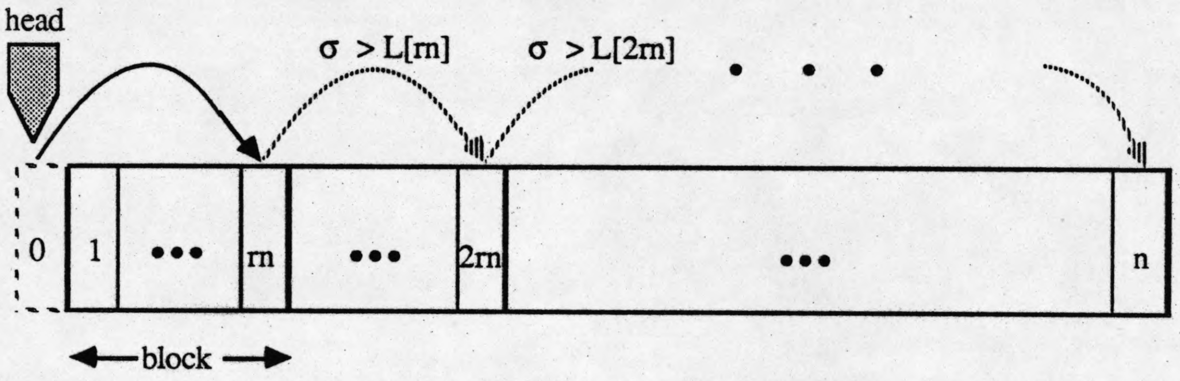

Figure 3. First stage of a block search algorithm

The performance of these seemingly ad hoc algorithms depends upon the value chosen for r, and, for block-binary search, can vary between linear search if $r = 1/n$ and binary search if $r = 1$. In order to calculate the optimal value of r, we must determine the average head movement and average number of comparisons for each algorithm. The average amount of head movement is the sum of the averages of each of the two stages. For the first stage of either algorithm, each key in block $i = \lceil k/(rn) \rceil$ requires moving rni positions. In the second stage, the head movement required to find the key within the block is $rn-1$ in the case of binary search and $rni - k$ in the case of linear search. Thus:

$$\overline{M}_{BB(r)}(n) = \frac{1}{n}\left[\sum_{i=1}^{1/r}(rn)(rni) + n(rn-1)\right] = \frac{n}{2} + \frac{3rn}{2} - 1 \qquad (14)$$

$$\overline{M}_{BL(r)}(n) = \frac{1}{n}\left[\sum_{i=1}^{1/r}(rn)(rni) + \frac{1}{r}\sum_{j=1}^{rn} j\right] = \frac{n}{2} + rn + \frac{1}{2} \tag{15}$$

The average number of comparisons can be similarly determined:

$$\overline{C}_{BB(r)}(n) = \frac{1}{n}\left[\sum_{i=1}^{1/r}(rn)i + n\log(rn)\right] = \frac{1}{2r} + \frac{1}{2} + \log(rn) \tag{16}$$

$$\overline{C}_{BL(r)}(n) = \frac{1}{n}\left[\sum_{i=1}^{1/r}(rn)i + \frac{1}{r}\sum_{j=1}^{rn} j\right] = \frac{1}{2r} + \frac{1}{2} + \frac{rn+1}{2} \tag{17}$$

Here, the algorithmic subscripts $BB(r)$ and $BL(r)$ respectively denote block-binary and block-linear search algorithms with block size rn. Note that, for both algorithms, $\overline{M}(\cdot)$ and $\overline{C}(\cdot)$ can be reduced by treating the rn elements of the last block separately and not making the $(1/r)$th comparison. This modification complicates the analysis, but does not effect the asymptotic performance of the algorithms, and, hence, is ignored.

The average total time of block-binary search is therefore

$$\overline{S}_{BB(r)}(n) = T_M\left[\frac{n}{2} + \frac{3rn}{2} - 1\right] + T_C\left[\frac{1}{2r} + \frac{1}{2} + \log(rn)\right], \tag{18}$$

while that of block-linear search is

$$\overline{S}_{BL(r)}(n) = T_M\left[\frac{n}{2} + rn + \frac{1}{2}\right] + T_C\left[\frac{1}{2r} + \frac{1}{2} + \frac{rn+1}{2}\right]. \tag{19}$$

Taking the derivative of (18) with respect to r and setting it equal to zero, we find that the optimal r for block-binary search is

$$r^* = \frac{\sqrt{T_C^2\log^2 e + 3T_M T_C n} - T_C \log e}{3T_M n}, \tag{20}$$

which asymptotically approaches

$$r^* \approx \sqrt{\frac{T_C}{3T_M n}} \tag{21}$$

as $n \rightarrow \infty$. The average total time for the resulting block-binary search is

$$\bar{S}_{BB(r^*)}(n) \approx T_M\left[\frac{n}{2}+\frac{3n}{2}\sqrt{\frac{T_C}{3T_M n}}\right]+T_C\left[\frac{1}{2}\sqrt{\frac{3T_M n}{T_C}}+\log(rn)\right].$$

Simplifying and discarding asymptotically insignificant terms, we find

$$\bar{S}_{BB(r^*)}(n) \approx T_M\left[\frac{n}{2}+\frac{1}{2}\sqrt{\frac{3T_C n}{T_M}}\right]+\frac{T_C}{2}\sqrt{\frac{3T_M n}{T_C}}=T_M\frac{n}{2}+\sqrt{3T_M T_C n}\,, \tag{22}$$

which results in the following theorem.

Theorem 2: $\bar{S}_{BB(r^*)}(n)=T_M\frac{n}{2}+\sqrt{3T_M T_C n}+o(\sqrt{n})$.

This theorem shows that block-binary search is somewhat better than fixed-ratio search.

Now, for block-linear search, taking the derivative of (19) with respect to r and setting it equal to zero, we find the optimal r is

$$r^*=\sqrt{\frac{T_C}{(2T_M+T_C)n}}. \tag{23}$$

The average total time for the resulting block-linear search is

$$\bar{S}_{BL(r^*)}(n)=T_M\left[\frac{n}{2}+n\sqrt{\frac{T_C}{(2T_M+T_C)n}}+\frac{1}{2}\right]+T_C\left[\frac{n}{2}\sqrt{\frac{T_C}{(2T_M+T_C)n}}+1+\frac{1}{2}\sqrt{\frac{(2T_M+T_C)n}{T_C}}\right].$$

Simplifying, we obtain the following theorem.

Theorem 3: $\bar{S}_{BL(r^*)}(n)=T_M\frac{n}{2}+\sqrt{T_C(2T_M+T_C)n}+\frac{T_M}{2}+T_C$.

This shows that block-linear search is asymptotically better than block-binary search if and only if $T_C < T_M$, but it can never achieve average total time less than

$$T_M\frac{n}{2}+\sqrt{2T_M T_C n}.$$

4. Lower Bounds

All three of the algorithms analyzed thus far have average time complexity $\approx T_M\frac{n}{2}+\sqrt{cT_M T_C n}$, for some constant c. However, the only obvious lower bound on the average time complexity is

$\bar{S}(n) \geq T_M \frac{n-1}{2} + T_C \log n$, which follows directly from independent lower bounds on the average amount of head movement and the average number of comparisons. In particular, if the correct index of the search key σ is k, then the head must move at least k times just to compare σ with $L[k]$. Thus, $\bar{M}_A(n) \geq (\sum_{i=1}^{n-1} i + n-1)/n = (n-1)/2 + 1 - 1/n$ for any algorithm A. Information theoretic arguments allow one to conclude that $\bar{C}_A(n) \geq \log n$ for any A. Here, we strengthen this lower bound to show that the algorithms proposed above are nearly optimal in a well defined sense.

It is useful to begin by considering the possible structure of an optimal search algorithm. The following lemma shows that we may restrict our attention to *varying-ratio search* algorithms, i.e., ratio search algorithms where the parameter r is no longer fixed for a given problem instance (as it was in Section 2) but rather is a function $r(n)$ dependent on the length of the current sublist.

Lemma 4.1: There exists a varying-ratio search that achieves the optimal search time.

Proof: Every optimal algorithm starts its search by moving the head to an initial list key position and comparing this key with the search key, after which it continues its search in one of the two pieces of the list, either to the left or right of the initial probe location. Since the only information available to the algorithm is the length of the list, this initial probe location is a function $p(n)$ of the length n of the list, and the varying ratio itself is the function $r(n) = p(n)/n$. Since the algorithm is optimal, the search in either the left or right sublist must also be optimal, and, therefore, may be effected by a recursive call to the same optimal algorithm (even though there may be alternative optimal algorithms). □

The preceding lemma indicates that an optimal algorithm can be written as follows:

```
OPTIMAL_SEARCH($L$, $\sigma$, $h$, $t$, $k$)
    if ($|t-h| = 1$) then
        $k := \max(h,t)$;
        return;
    else
        $p := r(|t-h|) \cdot (t-h)$;
        if ($\sigma < L[h+p]$) then
            OPTIMAL_SEARCH($L$, $\sigma$, $h+p$, $\min(h,t)$, $k$);
        else
            OPTIMAL_SEARCH($L$, $\sigma$, $h+p$, $\max(h,t)$, $k$);
        endif
        return;
    endif
```

The complexity of this algorithm is determined by the recurrence

$$\bar{S}_{OPT}(n) = r(n)\bar{S}_{OPT}(r(n)n) + (1-r(n))\bar{S}_{OPT}((1-r(n))n) + T_M r(n)n + T_C, \tag{24}$$

which, unfortunately, is difficult to analyze exactly without specific knowledge of $r(n)$. Dynamic programming allows one to compute tables of values of $r(n)$ that can be used to provide insight into its asymptotic behavior, but it is quite likely that $r(n)$ will not have a closed form for most values of T_M and T_C.

On the other hand, we can determine lower bounds on the average total search time $\bar{S}_{OPT}(n)$ required by the varying-ratio optimal algorithm. The following theorem gives the desired lower bound on the solution of Equation (24).

Theorem 4: $\bar{S}_{OPT}(n) \geq T_M n/2 + \sqrt{T_M T_C n} + T_C - \sqrt{2T_M T_C}$.

Proof (by induction on n):

Basis ($n = 2$): $\bar{S}_{OPT}(2) = T_M + T_C = T_M 2/2 + \sqrt{T_M T_C 2} + T_C - \sqrt{2T_M T_C}$.

Inductive hypothesis: Assume that the theorem is true for all integers $n' < n$; now we must prove it for n.

Replacing the terms in Equation (24) with the bounds given by the inductive hypothesis, we obtain

$$\bar{S}_{OPT}(n) \ge T_M r^2(n) n/2 + \sqrt{T_M T_C}\, r^{3/2}(n)\sqrt{n} + (T_C - \sqrt{2T_M T_C})r(n) +$$
$$T_M(1-r(n))^2 n/2 + \sqrt{T_M T_C}(1-r(n))^{3/2}\sqrt{n} + (T_C - \sqrt{2T_M T_C})(1-r(n)) + T_M r(n) n + T_C,$$

which can be rewritten as

$$\bar{S}_{OPT}(n) \ge T_M n/2 + T_M r^2(n) n + \sqrt{T_M T_C}[r^{3/2}(n) + (1-r(n))^{3/2}]\sqrt{n} + 2T_C - \sqrt{2T_M T_C}. \tag{25}$$

Since $0 < 1-r(n) < 1$ (in fact, $1/2 \le 1-r(n)$), $(1-r(n))^{3/2} > (1-r(n))^2$. Therefore, we can replace the factor $[r^{3/2}(n) + (1-r(n))^{3/2}]$ in Equation (25) with $(1-r(n))^2$, which yields

$$\begin{aligned}
\bar{S}_{OPT}(n) &\ge T_M n/2 + T_M r^2(n) n + \sqrt{T_M T_C}(1-r(n))^2\sqrt{n} + 2T_C - \sqrt{2T_M T_C} \\
&= T_M n/2 + [T_M r^2(n) n + \sqrt{T_M T_C n}\,(1 - 2r(n) + r^2(n)) + T_C] + T_C - \sqrt{2T_M T_C} \\
&= T_M n/2 + [T_M r^2(n) n - 2\sqrt{T_M T_C n}\, r(n) + T_C + \sqrt{T_M T_C n}\,(1 + r^2(n))] + T_C - \sqrt{2T_M T_C} \\
&= T_M n/2 + [(\sqrt{T_M n}\, r(n) - \sqrt{T_C})^2 + \sqrt{T_M T_C n}\, r^2(n))] + \sqrt{T_M T_C n} + T_C - \sqrt{2T_M T_C}.
\end{aligned} \tag{26}$$

This must hold for any choice of $r(n)$; in particular, it must hold for the $r(n)$ that minimizes the right-hand side of Equation (26). Taking the derivative of the right-hand side with respect to r and equating it to zero, we see that it is minimized for

$$r(n) = r^*(n) = \frac{\sqrt{T_C}}{\sqrt{T_M n} + \sqrt{T_C}}. \tag{27}$$

Substituting back into Equation (26):

$$\begin{aligned}
\bar{S}_{OPT}(n) &\ge T_M n/2 + \frac{T_C^2}{(\sqrt{T_M n} + \sqrt{T_C})^2} + \frac{T_C\sqrt{T_M T_C n}}{(\sqrt{T_M n} + \sqrt{T_C})^2} + \sqrt{T_M T_C n} + T_C - \sqrt{2T_M T_C} \\
&= T_M n/2 + \frac{T_C^{3/2}}{\sqrt{T_M n} + \sqrt{T_C}} + \sqrt{T_M T_C n} + T_C - \sqrt{2T_M T_C} \\
&> T_M n/2 + \sqrt{T_M T_C n} + T_C - \sqrt{2T_M T_C}. \ \square
\end{aligned}$$

This theorem shows that the upper bounds obtained previously are very nearly optimal, asymptotically differing from the lower bound only by a constant factor in the "surplus" term (the $O(\sqrt{n})$ term beyond the "obvious" $T_M n/2$).

5. Conclusion and Remarks

In this paper, we have shown that the obvious algorithms for searching on a tape are not necessarily the best when the cost of both head movement and comparisons is taken into account. Binary search is asymptotically suboptimal by a factor of 2 (see Equation (3)), while linear search is asymptotically suboptimal by a factor of $(T_M + T_C)/T_M$ (see Equation (4)). Fixed-ratio search and block search, however, are optimal with respect to the constant of the linear term and come within a constant factor of the $O(\sqrt{n})$ term.

The SAM model of computation is also interesting for other search problems, e.g., *batch-mode* search [MKB], in which several search keys are to be located on the tape. A reasonable solution for the SAM model is to sort the search keys in ascending order and proceed in one pass from head to tail. We leave as an open problem the analysis of this or a better algorithm. Another avenue for further research is the analysis of a broader class of problems in the SAM model analogous to the analysis of sorting, matrix multiplication, and DFT in the HMM model [AACS]. In these problems, special attention should be devoted to determining an appropriate organization of the input data on the tape, since this is a crucial aspect of the analysis. Indeed, the "natural" sorted order is not optimal for the case of searching on a tape.

Throughout this paper, we have assumed that the list of elements is stored in sorted order on the tape, which seems to be the "natural" approach. However, if we need this ordering only for fast search then there is a far better strategy. Recall that simple linear search is optimal with respect to average head movement, whereas binary search is optimal with respect to number of comparisons, both on the average and in the worst case. Now, if we can organize the list on the tape in a different manner, the advantages of both approaches can be combined in a solution that is optimal with respect to both average head movement and average number of comparisons.

The special organization $\Pi(L)$ of a sorted list L that achieves these desirable properties is called a *preorder organization* and can be recursively defined as follows:

$$\Pi(L)[1] = L[\lceil n/2 \rceil]$$

$$\Pi(L)[2{:}\lceil n/2 \rceil] = \Pi(L[1{:}\lceil n/2 \rceil - 1])$$

$$\Pi(L)[\lceil n/2 \rceil + 1{:}n] = \Pi(L[\lceil n/2 \rceil + 1{:}n])$$

In other words, the median of L appears first, followed by the first half of L (recursively organized), and then followed by the second half of L (also recursively organized) (Figure 4).

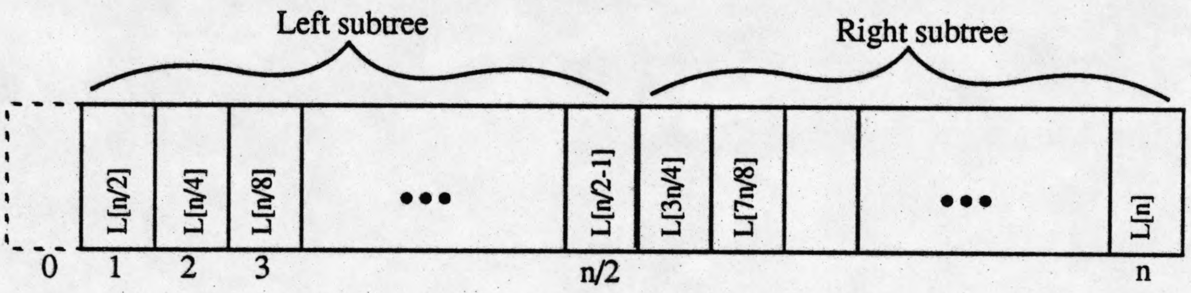

Figure 4. Preorder organization of keys on a tape

The origin of the term "preorder" should now be clear; this organization corresponds to a preorder embedding of the binary search tree on the tape. (The list could be converted to a preorder organization by using a second tape along with an external sorting algorithm [Knu].) The search procedure itself is also straight-forward; to search for σ in Π, we move the head to the first tape cell and then call the following algorithm with $h = 1$, $t = n$, and $k = n$:

```
PREORDER_SEARCH($\Pi$, $\sigma$, $h$, $t$, $k$)
    if ($\sigma < \Pi[h]$) then        /* This requires time $T_C$. */
        $k := h$;
        $t := \lfloor (t+h)/2 \rfloor$;
        $h := h + 1$;        /* This requires time $T_M$. */
        if ($t \geq h$) then
            PREORDER_SEARCH($\Pi$, $\sigma$, $h$, $t$, $k$);
        else
            return;
        endif
    else
        $h := \lfloor (t+h)/2 \rfloor + 1$;        /* This requires time $T_M(\lfloor (t-h)/2 \rfloor + 1)$. */
        if ($t \geq h$) then
            PREORDER_SEARCH($\Pi$, $\sigma$, $h$, $t$, $k$);
        else
            return;
        endif
    endif
```

The index of the successor key to σ in Π is returned in k.

The average total time required by preorder search is

$$T_M \frac{n}{2} + (\frac{T_M}{2} + T_C)\log n + o(\log n),$$

but, if we assume that the search key is in the list, this complexity is further reduced to roughly

$$T_M \frac{n}{2} + T_C \log n,$$

which matches the obvious lower bound discussed in the introduction to Section 4.

REFERENCES

[AACS] A. Aggarwal, B. Alpern, A. K. Chandra and M. Snir, "A Model for Hierarchical Memory," *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, New York, NY, May 1987, pp. 305-314.

[F] D. E. Ferguson, "Fibonaccian Searching," *Communications of the ACM*, vol. 3, no. 12, Dec. 1960, pp. 648.

[G] C. W. Gear, *Computer Organization and Programming*, third edition, McGraw Hill Book Company, New York, NY, 1980.

[Kni] W. J. Knight, "Search in an Ordered Table Having Variable Probe Cost," masters thesis, Department of Computer Science, University of Illinois, Urbana, IL, 1986.

[Knu] D. E. Knuth, *The Art of Computer Programming*, vol. 3, Addison-Wesley, Reading, MA, 1973.

[MKB] Y. P. Manolopoulos, J. (Y.) G. Kollias, and F. W. Burton, "Batched Interpolation Search," *The Computer Journal*, vol. 30, no. 6, Dec. 1987, pp. 565-568.

[NN] S. Nishihara and H. Nishino, "Binary Search Revisited: Another Advantage of Fibonacci Search," *IEEE Transactions on Computers*, vol. C-36, no. 9, Sep. 1987, pp. 1132-1135.

[O] K. J. Overholt, "Efficiency of the Fibonacci Search Method," *BIT*, vol. 13, no. 1, Jan. 1973, pp. 92-96.