SIAM J. COMPUT.
Vol. 11, No. 2, May 1982

0097-5397/82/1102-0007 $01.00/0

A TIME-SPACE TRADEOFF FOR SORTING ON A GENERAL SEQUENTIAL MODEL OF COMPUTATION

A. BORODIN† AND S. COOK†

Abstract. In a general sequential model of computation, no restrictions are placed on the way in which the computation may proceed, except that parallel operations are not allowed. We show that in such an unrestricted environment TIME·SPACE $= \Omega(N^2/\log N)$ in order to sort N integers, each in the range $[1, N^2]$.

Key words. time-space tradeoffs, conputational complexity, sorting, time lower bounds, space lower bounds

1. Introduction. Within the field of computational complexity, our inability to establish lower bounds on the complexity of "natural problems" stands in marked contrast to the progress that has been made in algorithmic design and analysis, and the progress in characterizing the central issues. To be fair, there are the following important exceptions:

1. Relative to an appropriate reducibility, a problem can be shown "hard" for an entire complexity class. Then diagonalization can be used to infer a corresponding complexity lower bound. For example, see the discussion in Aho, Hopcroft and Ullman [1, Chapt. 11].

2. For certain natural but "structured" models of computation, we have a number of interesting lower bounds. We use "structured" in the sense of Pippenger and Valiant's [2] use of "conservative" to mean that the computation can only proceed within a fixed mathematical structure (e.g., a partial order for comparison based models, a ring or field for algebraic complexity) and only uses the relations and functions within that structure for the computation (see also Borodin [3]). For example, using comparison trees it is well known that sorting n elements requires at least $n \log n + O(n)$ comparisons.

3. On certain nonstructured but restricted models of computation we have a few results. For example, to recognize the set $\{w \# w^R\}$ on a *one-tape* Turing machine requires $\Omega(n^2)$ steps.

A *general* sequential model of computation can be viewed as a string processing machine. While the input string may arise as the encoding of a set of mathematical objects, there is no obligation to process these objects in ways prescribed by the mathematical structure. In this context complexity is measured as a function of the input (plus output) length. If we ignore "diagonalization based results", the following barriers are well recognized:

a. To establish a nonlinear lower bound on time.

b. To establish a nonlogarithmic lower bound on space.[1]

c. To establish a nonlogarithmic lower bound on depth (= parallel time).

Having recognized these barriers, it might seem wise to see if we can at least show that for some problem we cannot simultaneously achieve (say) linear time and logarithmic space. Such a result already appears in Cobham [4], where he shows that for recognizing the set of perfect squares (or for recognizing $\{w \$ w^R\}$) we must have $T \cdot S = \Omega(n^2)$ for any computational device (including a multitape T · M.) having a

* Received by the editors July 28, 1980, and in final form May 4, 1981.

† Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A1.

[1] That is, prove space is not $O(\log n)$.

separate *one head-read only input tape.* Here T = number of steps, S = "capacity" = $\log_2$ (number of configurations the machine enters when processing all strings of length n). The concept of "capacity" introduced by Cobham seems to capture just that property of space which lends itself to lower bound analysis. But whereas we accept a capacity lower bound on one of Cobham's general machines to be an "intrinsic" lower bound (i.e., independent of the choice of any reasonable computational models) on space requirements, we cannot say that a $T \cdot S = \Omega(n^2)$ lower bound has the same intrinsic quality, because of the restriction of having only one input head. More specifically, by easily adapting Cobham's argument (based on Hennie's [5] crossing sequence technique), Tompa [6] shows that sorting m numbers, each of length $\log m$ bits (hence $n = m \log m$), requires $T \cdot S = \Omega(n^2)$. But the proof literally states and shows that merging two lists of m sorted numbers would require the same lower bound. But for merging, the use of (say) two input heads would trivially (via a linear merge) permit a simultaneous linear time and logarithmic space merge. We are then led to the following question: Given k "random access" input heads, can we sort (say on a multitape T · M. or unit cost RAM) in simultaneous linear time and logarithmic space? The main result of this paper shows that indeed this is not possible. In fact we will establish a lower bound analogous to (and based upon) the lower bound of $T \cdot S = \Omega(n^2)$ established for sorting in the structured context of "branching programs" by Borodin, et al. [7]. Specifically we show $T \cdot S = \Omega(N^2/\log N)$, N the number of inputs and $N = \Omega(n/\log n)$ where n is the input length. To the best of our knowledge this is a unique result in that it establishes a lower bound (without diagonalization) on a completely unrestricted general model of computation. Unfortunately, we have not yet been able to establish a similar bound for a set recognition problem and we should also note that our methods do not appear applicable to Knuth's [8] problem of in situ sorting.

2. The formal model and an outline of the proof. In a general model of computation, we might be able to solve a given problem by processing the input string in a manner which is completely outside the mathematical domain within which the problem has been defined. For example, consider solving for the existence of a path on a graph by using Strassen's matrix multiplication algorithm and modular arithmetic (see Fischer and Meyer [9]). It seems almost impossible to make sense out of the individual bit operations in terms of the original problems.

The "fortunate" fact for sorting is that such a problem, with its explicit requirement for "ongoing progress" (in the sense of having to output ranks) allows us to enjoy a structured view of the computation even though we are working within a general computational model. Indeed we shall try to mimic the proof for the structured case [7]. That proof was based on the following intuitive idea: if we don't compare many elements, then we can't know the ranks of many elements for many input permutations. We will need a somewhat more involved argument to show an analogous statement for the general model.

Before discussing the model, we should define the problem formally. We consider an input of the form $x_1 \# x_2 \# \cdots \# x_N$ where each x_i is an integer in $[1, N^2]$ and is coded in binary. Hence the total length of the input is $O(N \log N)$. The sorting problem is to output a sequence of distinct pairs $i_1, r_1; i_2, r_2; \cdots; i_N, r_N$ such that x_{i_j} has rank r_j. (Without loss of generality we can assume that x_i's are distinct.) As in Borodin et al. [7] we can define the k-ranking subproblem; namely, output a sequence $i_1, r_1; \cdots; i_l, r_l$, $l \geqq k$, which correctly represents the ranks of l of the x_{i_j}'s. (The l indices i_j which are assigned ranks may be different for different input values.)

This definition of sorting is not standard. Usually, one requires outputting the x_i values in sorted order. However, for the model we are considering, any algorithm which sorts in this usual manner can be adapted to one which outputs pairs $\langle i_j, r_j \rangle$ by assuming that the index i has been concatenated onto x_i as the low order bits. It will follow that a TIME · SPACE lower bound in our framework for inputs in the range $[1, N^2]$ will imply the same lower bound in the usual setting for inputs in the range $[1, N^3]$.

Our formal models are as follows:

DEFINITION. An *R-way integer tree program* is an R-ary tree (hereafter called an R tree), where each branch is labelled by elements of $[1, R]$, and each internal node is labelled by some index i (referring to x_i). The interpretation is that if the computation has proceeded to an internal node labelled by "x_i" then it will continue to proceed along edge u iff $x_i = u$. Output takes place at the leaves. In particular, it should now be clear to say how a computation tree solves the k-ranking problem, or more generally how a computation tree solves the k-ranking problem for some subset I of the possible inputs. The time complexity of a computation tree is its depth; that is, the maximum number of times inputs are accessed in a computation. Since we assume all x_i are distinct, any branch which has two edges with the same label u for distinct x_i will be inaccessible. We assume these inaccessible paths have been pruned.

An R-way integer branching program (hereafter called an R branching program) is the nonstructured analogue of a comparison branching program [6]. Namely, it is a directed acyclic rooted graph with each nonsink node having out-degree R, with the R out edges labelled $1, 2, \cdots, R$. Without loss of generality, we can assume that the graph is in levels and that an edge out of a node at level l is directed to a node at level $l+1$. (See Tompa [6] for a discussion of the analogous assumption for comparison branching programs.) Outputs can now occur on any edge. The time complexity is again the depth and space = capacity = $\log_2$ (number of nodes in the graph). We can now state:

MAIN THEOREM. *Let τ be an R branching program for sorting N integers and let $R = R(N) = N^2$. Then $T \cdot S = \Omega(N^2/\log N)$ where T and S denote, respectively, the time and space complexity of τ.*

Before proceeding to the proof, we should comment briefly on the generality of the model. Suppose we have a general computational machine with k read-only, "random access" heads. It should be clear that by assuming $k = 1$ we will only slow down the machine by at most a constant factor (i.e., k). Our tree and branching programs assume that we will know an entire input x_i if we access any bit of that input. Hence, we are willing to ignore the $\log N$ factor it might cost to look at a given input. Each node of the computation graph represents a distinct state of the computation. Like Cobham [4], it is profitable for us to ignore completely how (and if) the storage can be represented and manipulated. Again, we are willing to ignore the time spent manipulating the storage between accesses of the input. We thus argue that our model and the time and space measures are sufficiently general that any lower bounds do reflect an intrinsic property of the function (sorting) being computed.

Having presented and justified the model, we can now informally sketch the proof. To do so, it is helpful to review the proof for the structured case [7]. The basic lemma in that proof states that a $\{<, >\}$ comparison tree program on n inputs of depth (time) t can solve the k ranking problem for at most $(t+1)^k(n-k)!$ input permutations. This lemma is applied with $k = S$ = space. Thus for any $c > 1$, by making $t = \alpha n$ for α sufficiently small, we can say that the S ranking problem has been solved for at most a fraction $(1/c)^S$ of the $n!$ possible input permutations. Now if τ is a

$\{<, >\}$ branching program for sorting, we consider the computation at the ith "stage" $= (i \cdot t)$th step, $i \leqq n/S$. In going from stage i to stage $i+1$, we can only correctly calculate S more ranks for at most $n!2^S \cdot (1/c)^S$ input permutations, since there are at most 2^S nodes at the ith stage, each of which can be considered the root of a tree program. Thus by an appropriate choice of $t = \Omega(n)$ we have $c > 2$ so that in going from stage i to stage $i+1$ we have computed more than S new ranks for at most a fraction $(1/d)^S$ of all possible input permutations. It follows that we will need at least $i = n/S$ stages to complete the computation, and hence $T = \Omega(n^2/S)$.

We want to establish the analog of the basic lemma, after which the rest of the proof follows exactly as before. We will show that for any $c > 1$, we can find suitable α such that any R-tree program ($R = N^2$) of depth $t = \alpha N$ can solve the $S \log N$ ranking problem for at most a fraction $(1/c)^S$ of the the possible inputs. In our case, that are $N!\binom{R}{N}$ possible input sequences $\langle x_1, \cdots, x_N \rangle$ since we are assuming distinct $\{x_i\}$.

In viewing the proof of the structured case, we can observe that *every* path in a computation tree can successfully solve the k ranking problem for at most a fraction $(t+1)^k/[n \cdot (n-1) \cdots (n-k+1)]$ of the permutations following that path. In our case, we can see that some short paths can be very successful; indeed if we discover that some $x_i = 1$ (or $x_i = N^2$) on a given path, then we know the smallest (respectively, largest) element for every input sequence on that path. Moreover, if we find some $x_i = 2$ (and no x_j seen so far is equal to 1) we still have a pretty good chance if we guess that x_i is the smallest element. But, we can also see intuitively that our chance of guessing correctly as to which is the smallest element starts to decrease if we have only seen a few not so small numbers.

So this will be our approach for establishing the analogous main lemma: We assert that, with sufficiently high probability, at a leaf of an R-tree program the elements that we have seen on this path will be "spread out" in such a way that there is only a small probability (i.e., for only a small fraction of all possible input sequences) that we will correctly output S ranks.

3. The proof of the main lemma. Throughout this section we will be considering R tree programs τ such that each leaf θ of τ is labelled with a ranking sequence $i_1, r_1; \cdots; i_l, r_l$, where $l = l_\theta$ may depend on the leaf θ. We say τ *solves the m ranking problem for an input sequence* $\langle x_1, \cdots, x_N \rangle$ provided this input leads to a leaf θ for which $l_\theta \geqq m$, and all l_θ ranks are correctly specified (i.e., x_{i_j} is the r_jth smallest input, $1 \leqq j \leqq l_\theta$). The following notation will be maintained: $t \leqq \frac{1}{2}N$ is the depth of the R tree program τ (we may asssume all paths in τ have length t by extending shorter ones if necessary), $N \geqq 2$ is the number of input elements, k is a positive integer satisfying $2f(k) \leqq N$, and $f(k)$ stands for $k\lceil \log N \rceil$. We will see that $R = R(N) = N^2$ is sufficiently large for our purposes, and since all results hold a fortiori for larger R, we will assume $R = N^2$. Our proofs will be formulated in the language of probability theory and we will speak of a random input in the sense that any of the $N!\binom{R}{N}$ possible input sequences are considered to be equally likely.

We are now ready to state the main lemma, which says that any sufficiently shallow R tree program (regardless of its capacity) cannot output many ranks correctly.

LEMMA 1. *For all $c > 0$ there is an $\alpha > 0$ such that for all τ with $t \leqq \alpha N$ and N sufficiently large, and for all k with $f(k) \leqq t$, the set I of inputs for which τ solves the $2f(k)$ ranking problem satisfies $\#I/(N!\binom{R}{N}) \leqq (1/c)^k$. Restated: with probability at most $(1/c)^k$, τ correctly outputs $2f(k)$ or more ranks for a random input.*

DEFINITION. A set $S=\{x_{i_1}, \cdots, x_{i_t}\}$ of inputs is $\langle \rho, k\rangle$ *spread out* if for every subset $S' \subseteq S$ with $\#S' = f(k)$ there is a subset $\{y_1, \cdots, y_k\}$ (listed in increasing order) of S' such that $y_{j+1} - y_j - 1 \geqq \rho$, for $0 \leqq j \leqq k$. (Here $y_0 = 0$ and $y_{k+1} = R+1$.)

LEMMA 2. *For all integers $\beta > 0$ there is an $\alpha > 0$ such that, for all τ with $t \leqq \alpha N$ and all k with $f(k) \leqq t$, $P[\tau, k, \beta] \leqq (1/N)^k$, where $P[\tau, k, \beta]$ is the probability that a random input $\langle x_1, \cdots, x_N\rangle$ to τ will follow a path along which the accessed input elements are not $\langle \beta R/N, k\rangle$ spread out.*

LEMMA 3. *For all $d > 0$ there is an integer $\beta > 0$ such that for all τ with N sufficiently large and for all k with $2f(k) \leqq N$ if the accessed input elements $\langle x_{i_1}, \cdots, x_{i_t}\rangle$ at a leaf θ of τ are $\langle \beta R/N, k\rangle$ spread out and the ranking sequence labelling θ contains at least $2f(k)$ ranks, then the fraction of those inputs leading to θ that are correctly ranked is at most $(1/d)^k$.*

Lemma 1 follows from Lemmas 2 and 3 as follows. Choose $d \geqq 2c$ in Lemma 3 to get β and apply Lemma 2 to get α. By Lemma 2 it does no harm to assume all leaves whose accessed inputs are not spread out always correctly solve the $2f(k)$ ranking problem, and the remaining leaves either output fewer than $2f(k)$ ranks or (by Lemma 3) are correct for too few inputs.

Proof of Lemma 2. Every leaf θ of τ uniquely determines a t-tuple $(x_{i_1}, \cdots, x_{i_t})$ of accessed elements, written in the order in which they are accessed on the path to θ. Conversely, every t-tuple of distinct integers in the interval $[1, R]$ uniquely determines a leaf. Thus there is a one to one correspondence between leaves and t-tuples, and exactly a $t!$ to one correspondence between leaves and sets of t distinct integers from $[1, R]$. Further, any two leaves have the same number of input sequences $(x_1, \cdots, x_N)$ leading to them. Therefore $P[\tau, k, \beta]$ is just that fraction of sets of t distinct integers from $[1, R]$ which are not $\langle \beta R/N, k\rangle$ spread out.

Divide the interval $[1, R]$ into N equal subintervals called *bins* of length N each (recall $R = N^2$). Let $\tilde{P}[t, N, k, \delta]$ be the probability that, when t balls are drawn (without replacement) from an urn of R balls numbered $1, 2, \cdots, R$, there exists some set of $f(k)$ of the drawn balls which lie in at most δ bins[2]. We claim that $\tilde{P}[t, N, k, \delta]$ is an upper bound on $P[\tau, k, \beta]$ where $\delta = k(\beta+1)+\beta$. For, if S is the set of t drawn balls and if every subset $S' \subseteq S$ of $f(k)$ balls lies in $\delta+1$ or more bins $B_1, \cdots, B_{\delta+1}$ (listed in the order in which these intervals occur in $[1, R]$), then we can choose one ball from each of the k bins $B_{j(\beta+1)}$, $1 \leqq j \leqq k$, to form the required subset $\{y_1, \cdots, y_k\}$ in the definition of $(\beta R/N, k) = (\beta N, k)$ spread out. This is because at least β bins lie entirely to the left of y_1, at least β bins lie entirely to the right of y_k, and any two adjacent y_j's are separated by at least β bins (β bins equals βN elements).

To estimate $\tilde{P}[t, N, k, \delta]$, let $p(b_i)$ be the probability that a particular bin B_i has at least b_i balls (after t are drawn). We claim that $\prod_{i=1}^{\delta} p(b_i)$ is an overestimate of the probability that a particular set of δ bins $B_1, \cdots, B_\delta$ get packed (respectively) with at least $b_1, \cdots, b_\delta$ balls. This is because the condition that a set of bins has some minimum number of elements can only decrease the probability that a particular bin has at least b_i elements. Hence

$$\tilde{P}[t, N, k, \delta] \leqq \sum_{\substack{(b_1, \cdots, b_\delta \\ \sum b_i = f(k) \\ b_i \geqq 0}} \binom{N}{\delta} \prod_{i=1}^{\delta} p(b_i).$$

[2] Here is the essential place that the $\log N$ factor in our main result $T \cdot S = \Omega(N^2/\log N)$ enters the proof. Specifically, we cannot assert that $\tilde{P}$ would be sufficiently small if $f(k)$ were $O(k)$ rather than $k \log N$.

Here $\binom{N}{\delta}$ gives the number of ways to choose a set of crowded bins, and the summation represents the number of ways to pack a set of crowded bins.

We claim that for all $c \geqq 1$ there is $\alpha > 0$ such that $p(b) \leqq (1/c)^b$ (where $t \leqq \alpha N$).

Proof. The probability that a particular bin has exactly l balls is given by

$$\frac{\binom{N}{l}\binom{N^2-N}{t-l}}{\binom{N^2}{t}} \leqq \frac{N^l(N^2-N)^{t-l}}{(N^2-t)^t} \cdot \frac{t!}{l!(t-l)!} \leqq \left(\frac{2}{N}\right)^l \cdot t^l = \left(\frac{2t}{N}\right)^l \leqq (2\alpha)^l.$$

Thus

$$p(b) \leqq \sum_{l=b}^{t} (2\alpha)^l < \frac{(2\alpha)^b}{1-2\alpha} \leqq \left(\frac{1}{c}\right)^b \quad \text{for } \alpha = \frac{1}{4c},$$

assuming $b \geqq 1$. The claim is obvious if $b = 0$.

We thus have

$$\begin{aligned}
\tilde{P}[t, N, k\delta] &\leqq \sum_{b_1+\cdots+b_\delta = f(k)} \binom{N}{\delta} \prod_{i=1}^{\delta} p(b_i) \\
&\leqq (f(k)+1)^\delta N^\delta \left(\frac{1}{c}\right)^{f(k)} \\
\text{(since } f(k) < N, f(k) = k\lceil \log N \rceil) \quad &\leqq N^{2\delta} \left(\frac{1}{c}\right)^{k\lceil \log N \rceil} \\
\text{(for } \delta = k(\beta+1)+\beta) \quad &\leqq N^{2k(\beta+1)+2\beta} \left(\frac{1}{N^{\log c}}\right)^k \\
&\leqq \left(\frac{1}{N}\right)^k \qquad \text{for sufficiently large } c. \qquad \square
\end{aligned}$$

Proof of Lemma 3. Let $\{x_{i_1}, \cdots, x_{i_t}\}$ be the input elements accessed on the path to θ. Suppose at θ the labels assert that x_{i_ν} has rank r_ν for $1 \leqq \nu \leqq 2f(k)$. Note that we are not necessarily implying that any $x_{i_\nu} \in \{x_{i_1}, \cdots, x_{i_t}\}$ but, intuitively, one would expect a better chance at "guessing" the rank of an element which has been seen. Suppose that fewer than half of the indices for which θ assigns ranks are among the set $\{i_1, \cdots, i_t\}$. Then there is a set S of $u \geqq k\lceil \log N \rceil$ indices i for which θ assigns a rank and whose corresponding value x_i can be anything in the set $\{1, 2, \cdots, R\} - \{x_{i_1}, \cdots, x_{i_t}\}$. In particular, all $u!$ possible orderings of the set $\{x_i \mid i \in S\}$ are possible and equally likely, and a necessary condition that θ rank them properly is that they be in the right order. Hence at most a fraction $1/u!$ of the inputs leading to θ are correctly ranked, and $1/u! \leqq (1/d)^k$ for sufficiently large N, since $u \geqq k \log N$.

The remaining case is that half or more (that is at least $k\lceil \log N \rceil = f(k)$) of the indices for which θ assigns ranks are among the set $\{i_1, \cdots, i_t\}$. Let S' be the set of inputs at these indices (so $\#S' \geqq f(k)$). Since $\{x_{i_1}, \cdots, x_{i_t}\}$ is $\langle \beta R/N, k \rangle$ spread out, there is a subset $\{y_1, \cdots, y_k\}$ of S' such that $y_{j+1} - y_j - 1 \geqq \beta R/N$, $0 \leqq j \leqq k$, where $y_0 = 0$, $y_{k+1} = R+1$. Let θ output the assertions that y_j has rank r_j, $1 \leqq j \leqq k$. These assertions are equivalent to saying that exactly $r_j - r_{j-1} - 1$ of the inputs lie in the open interval (y_{j-1}, y_j), $1 \leqq j \leqq k+1$, where we understand that $r_0 = 0$ and $r_{k+1} = N+1$. This in turn is equivalent to saying that exactly k_j of the inputs which θ *does not access* lie in the set $C_j = (y_{j-1}, y_j) - \{x_{i_1}, \cdots, x_{i_t}\}$, where $k_j = r_j - r_{j-1} - 1 - u_j$, and $u_j = \#(y_{j-1}, y_j) \cap \{x_{i_1}, \cdots, x_{i_t}\}$ (i.e., u_j is the number of inputs which θ knows to lie between y_{j-1} and y_j).

We have thus reduced our problem to a more traditional probability setting, namely that of the hypergeometric distribution (see Feller [10, p. 43]). We have a population of size $n = R - t$, made up of n_i elements of "color i" (i.e., member of the set C_i), $1 \leqq i \leqq l = k+1$. We seek an upper bound on the probability

$$p_{k_1 \cdots k_l} = \frac{\binom{n_1}{k_1}\binom{n_2}{k_2} \cdots \binom{n_l}{k_l}}{\binom{n}{r}} \tag{1}$$

that a sample (without replacement) of size $r = N - t = \sum_{i=1}^{l} k_i$ will contain *exactly* k_i elements of color, i, $1 \leqq i \leqq l$. The required bound is given by Lemma 4 below. For our application we have $rn_i/n = (N-t)(\#C_i)/(R-t) \geqq (N-t)(\beta R/N - t)/(R-t)$. But $t \leqq \frac{1}{2}N$ and furthermore[3] $R = N^2$ so that $\beta R/N - t \geqq \frac{1}{2}\beta R/N$ for $\beta \geqq 1$. Thus $rn_i/n \geqq \beta/4$ since $N - t \geqq \frac{1}{2}N$. Further $l - 1 = k \leqq N/\log N$. Hence the constraints on r, n, n_i and l for Lemma 4 will be satisfied for sufficiently large N. Lemma 3 now follows from the following:

LEMMA 4. *For all $d > 0$ there exists a $\beta > 0$ such that if $rn_i/n \geqq \beta$ for $1 \leqq i \leqq l$, $r \geqq 2l\beta$, and $n \geqq 2r$, then for all $k_1, \cdots, k_l$ the hypergeometric distribution satisfies*

$$p_{k_1 \cdots k_l} \leqq \left(\frac{1}{d}\right)^l.$$

We need the following two lemmas to prove Lemma 4. Note that Lemma 5 states that the value of k_i for which the hypergeometric distribution is maximal is close to the expected value rp_i of the number of elements of color i obtained in r draws. If this optimal value were exactly rp_i, the proof of Lemma 4 would be substantially simpler.

LEMMA 5[4]. *The values of k_i in the maximal term of the hypergeometric distribution $p_{k_1 \cdots k_l}$ satisfy*

$$\frac{rp_i}{1 + l/n} - 1 < k_i < rp_i + (l-1)p_i + 1, \tag{2}$$

where $p_i = n_i/n$, $1 \leqq i \leqq l$.

Proof. For any pair (i, j) of distinct indices we calculate the ratio

$$\frac{p_{\cdots k_i+1, \cdots, k_j-1 \cdots}}{p_{k_1, \cdots, k_l}} = \frac{(n_i - k_i)k_j}{(k_i+1)(n_j - k_j + 1)}.$$

A necessary condition for $p_{k_1, \cdots, k_l}$ to be maximal is that the numerator does not exceed the denominator, or $(n_i - k_i)k_j \leqq (k_i+1)(n_j - k_j + 1)$. If we divide by n and rearrange this becomes

$$p_i k_j \leqq p_j k_i + p_j + \frac{k_i - k_j + 1}{n}. \tag{3}$$

If we sum (3) over all $j \neq i$ and use the identities $\sum p_i = 1$ and $\sum k_i = r$, then we obtain the left half of (2). Similarly, if we sum (3) over all $i \neq j$ we obtain the right-hand side of (2). □

[3] This is the only place we need assume that R is as large as N^2.

[4] Feller [10, p. 171, Exercise 28] states a similar result for the multinomial distribution. Our proof is suggested by Feller's hints.

LEMMA 6. *For all $\varepsilon > 0$ there is a z_ε such that for all θ and for all $z \geqq z_\varepsilon|\theta|$*

$$\left(1+\frac{\theta}{z}\right)^z \geqq (1+\varepsilon)^{-|\theta|} e^\theta.$$

Note that z_ε does not depend on θ.

Proof. From elementary calculus we have $\lim_{z\to\infty}(1+\theta/z)^z = e^\theta$. Setting $\theta = 1$ and -1 we conclude $(1+1/z)^z \geqq (1+\varepsilon)^{-1} e$ and $(1-1/z)^z \geqq (1+\varepsilon)^{-1} e^{-1}$ for $z \geqq z_\varepsilon$. Setting $z = z'|\theta|$ we have $(1+\theta/z)^z = (1+\theta/(z'|\theta|))^{z'|\theta|} \geqq (1+\varepsilon)^{-|\theta|} e^\theta$ for $z' \geqq z_\varepsilon$; i.e., for $z \geqq z_\varepsilon|\theta|$. □

Proof of Lemma 4. We have

$$p_{k_1\cdots k_l} = (\textstyle\prod n_i!)r!(n-r)!/[n!\prod(k_i!(n_i-k_i)!)].$$

Stirling's approximation implies that $1/C_0 \leqq \sqrt{2\pi m}(m/e)^m/m! \leqq C_0$ for some constant $C_0 \geqq 1$ and all $m \geqq 1$. Using this approximation for each factorial, and substituting $rp_i + \theta_i$ for k_i, $1 \leqq i \leqq l$, where $p_i = n_i/n$ and θ_i has been chosen to maximize (1), we obtain

$$p_{k_1\cdots k_l} \leqq ABC_0^{3l+3}, \tag{4}$$

where

$$A = \sqrt{\frac{(\prod n_i)r(n-r)}{(2\pi)^{l-1} n \prod (rp_i+\theta_i)((n-r)p_i-\theta_i)}} \tag{5}$$

and

$$B = \frac{(\prod n_i^{n_i}) r^r (n-r)^{n-r}}{n^n \prod [(rp_i+\theta_i)^{rp_i+\theta_i}((n-r)p_i-\theta_i)^{(n-r)p_i-\theta_i}]}. \tag{6}$$

For (5) and (6) we have used the identity $n_i - k_i = (n-r)p_i - \theta_i$. Notice that all occurrences of e cancel, since $\sum n_i = n$.

Since $\sum p_i = 1$ and $\sum k_i = r$, it follows that $\sum \theta_i = 0$. This fact can be used to verify that if B' is the number obtained by substituting 0 for the two occurrences of θ_i in the denominator (but not in the exponents) in the expression for B, then $B' = 1$. Thus if we multiply and divide the denominator of (6) by $\prod [(rp_i)^{rp_i+\theta_i}((n-r)p_i)^{(n-r)p_i-\theta_i}]$ we can simplify and obtain

$$B = \left[\prod_{i=1}^{l}\left(\left(1+\frac{\theta_i}{rp_i}\right)^{rp_i+\theta_i}\left(1-\frac{\theta_i}{(n-r)p_i}\right)^{(n-r)p_i-\theta_i}\right)\right]^{-1}.$$

Now we apply Lemma 6 and use the fact that $(1+\theta/z)^\theta \geqq 1$ for all $z > 0$ and all θ to obtain $B \leqq (1+\varepsilon)^{2\sum|\theta_i|}$, provided

$$rp_i \geqq z_\varepsilon|\theta_i| \quad \text{and} \quad (n-r)p_i \geqq z_\varepsilon|\theta_i|, \qquad 1 \leqq i \leqq l. \tag{7}$$

By Lemma 5, we have $|\theta_i| \leqq lrp_i/(n+l) + (l-1)p_i + 2$, so $|\theta_i| \leqq 2lp_i + 2$. By assumption, we have $rp_i \geqq \beta$, $(n-r) \geqq r$ and $r \geqq 2l\beta$. Hence

$$(n-r)p_i \geqq rp_i \geqq \frac{\beta|\theta_i|}{4}, \tag{8}$$

so the provisos (7) are satisfied for $\beta \geqq 4z_\varepsilon$. Now summing the bound $|\theta_i| \leqq 2lp_i + 2$, we obtain $\sum|\theta_i| \leqq 4l$, so

$$B \leqq (1+\varepsilon)^{8l}. \tag{9}$$

It remains to estimate A from (5). We rewrite the product $\prod$ in the denominator as the product of five factors:

$$\prod (rp_i) \cdot \prod \left(1+\frac{\theta_i}{rp_i}\right) \cdot \prod n_i \cdot \left(1-\frac{r}{n}\right)^l \cdot \prod \left(1-\frac{\theta_i}{(n-r)p_i}\right).$$

To estimate the first factor $\prod rp_i$, notice that $\sum rp_i = r$, and each $rp_i \geqq \beta$ by assumption. With these constraints, this product obtains its minimum when all but one of the factors are as small as possible (namely β). Thus

$$\prod (rp_i) \geqq \beta^{l-1}(r-(l-1)\beta) > \tfrac{1}{2}r\beta^{l-1}.$$

From (8), we have $1+\theta_i/(rp_i) \geqq \frac{1}{2}$ for $\beta \geqq 8$, so $\prod (1+\theta_i/(rp_i)) \geqq 2^{-l}$. The same bound applies to the fifth factor and (since $n \geqq 2r$) to the fourth. The third factor cancels with the numerator. Thus

$$A \leqq [(2\pi)^{l-1}\beta^{l-1}2^{-3l+1}]^{-1/2} \leqq \left(\frac{1}{c}\right)^l$$

for any c and $\beta \geqq \beta_c$. Lemma 4 follows from this, (4) and (9). □

4. Proof of the main theorem. As indicated earlier in the paper, we will follow the general argument used in the structured case. As in § 3, we again assume $R = R(N) = N^2$. We let T denote the time (that is, the depth) of a branching program, and let S denote the space (that is, the capacity = $\log_2$ # nodes). Since we must clearly (by the simplest adversary argument) have $T \geqq N$ and $S \geqq \log_2 T$, we have $S \geqq \log_2 N$. Let us restate the main theorem.

THEOREM. *Let τ be an R branching program for sorting N integers. Then $T \cdot S = \Omega(N^2/\log N)$.*

Proof. Letting $c = 4$, use Lemma 1 to obtain α for N sufficiently large. We will now consider τ in stages, where every stage represents $t = \lfloor \alpha N \rfloor$ steps.

For $1 \leqq i \leqq N/(2f(S))$, let P_i be the fraction of input sequences for which τ has output at least $2if(S)$ ranks by the end of the ith stage. We shall now prove

$$(*) \qquad P_i \leqq i(\tfrac{1}{2})^S.$$

For each node θ on the $(i \cdot t)$th level (= end of stage i) let $P_{i,\theta}$ be the fraction of input sequences which lead to θ and for which τ outputs at least $2f(S)$ ranks during the $i+1$st stage. If we expand the part of the $(i+1)$st stage that is rooted at θ into an R tree, we see by Lemma 1 that (regardless of what has happened in earlier stages) $P_{i,\theta} \leqq (\frac{1}{4})^S$. Since there are at most 2^S nodes θ at level $i \cdot t$, we have $P_{i+1} \leqq P_i + \sum_\theta P_{i,\theta} \leqq P_i + 2^S \cdot (\frac{1}{4})^S$, or $P_{i+1} \leqq P_i + (\frac{1}{2})^S$. This inequality holds for $0 \leqq i \leqq N/(2(f(S))$, if we define $P_0 = 0$. The inequality (*) now follows by induction on i.

Recall $f(S) = S\lceil \log N \rceil$. If $2f(S) > N$ then $S > N/2\lceil \log N \rceil$, so $ST = \Omega(N^2/\log N)$ in this case. If $2f(S) \leqq N$, then we can set $i = i_0 = \lfloor N/(2f(S)) \rfloor$ in (*) to obtain (since $S \geqq \log N$) $P_{i_0} \leqq (N/(2f(S))) \cdot 1/N = 1/(2f(S)) < 1$. Hence at stage i_0 for some input, τ has output fewer than $2i_0 f(S) \leqq N$ ranks, so $T \geqq i_0 t = \lfloor N/(2f(S)) \rfloor \cdot \lfloor \alpha N \rfloor = \Omega(N^2/(S \log N))$ steps. □

5. Conclusion. In order to better appreciate the application of the main theorem, we offer the following example.

Let M be any machine (say, a unit cost RAM or vector machine with operations $+, -, \times, \div, \uparrow$) whose inputs are accessed from a random access *read-only* input device. We only insist that there is a bound on the number of inputs accessible on a given computation step. Choose any "fair" definition of space, e.g., space =

$\max_j \sum_{i=1}^{t} \lceil \log (r_i^j + 1) \rceil$ where r_i^j is the contents of register i at time j and t is the largest register used. For such a machine the theorem yields $T \cdot S = \Omega(N^2/\log N)$. And, of course, the same result holds for multidimensional Turing machines, etc.

Although the lower bound $T \cdot S = \Omega(N^2/\log N)$ established in this paper for a general model of computation differs by a log factor from the lower bound for the structured case [7], the upper bounds for the structured case apply unchanged. This is because a "structured algorithm" is a $\{<\ ,\ >\}$ branching program, and a comparison $x_i < x_j$ over the domain $[1, R]$ can be carried out on an R branching program in two time steps and $R+2$ nodes. However, in order to be sure that the time and space of the simulating program are of the same order as the time and space of the original program, it is necessary to assume $R = O(N^k)$ for some k. Under this assumption, the upper bound $T \cdot S = O(N^2 \log N)$, for $\Omega(\log N) \leqq S \leqq O(N)$ recently established by Frederickson [16] for a unit cost "structured" random access machine with suitable instructions applies to an R branching program. (Frederickson's bound generalizes the one in [7]). It is worth noting that for "unstructured" (i.e., general) random access machines, the upper bound can be extended, using radix sort, to the case $T = O(N)$ and $S = O(N \log N)$.

We thus have a $\log^2 N$ discrepancy in the upper and lower bounds. We note that we can improve on the upper bounds when $R = N + O(N)$, say by finding the missing elements.

It seems to us, however, that the discrepancy in the bounds is far less important than the need to establish analogous results for a set-recognition problem; for example, determining if $X \cap Y = \varnothing$. At the present time such a time-space result has not yet been established for the structured comparison model. We believe that our results suggest that proofs for the structured model may provide a framework for the general model. However, it must be noted that the less constructive variant of branching programs for "silent sorting" mentioned in Borodin et al. [7, Conclusion] becomes trivial in the general setting.

In retrospect, we can see that our methods are quite "brute-force". In particular, we do not make an essential use of an adversary. Rather what we have is basically a counting argument. Moreover, we do not make full use of the fact that space is limited throughout the computation; we only use the fact that it is restricted at certain points of the computation. We suspect that the set recognition problems will entail a more sophisticated argument.

A more general view of time-space complexity is captured in Cook's class SC [11], [12] (formerly PLOPS); that is, those problems for which there exist algorithms which run *simultaneously* in polynomial (sequential) time and $\log^k$ (for some k) space. Obviously, any problem (e.g., sorting, $X \cap Y = \varnothing$?, etc.) which is in log space, is also in SC. A central issue for computational complexity is to establish the conjecture (assuming it is true) that $\mathrm{P} \cap (\bigcup_k \mathrm{DSPACE}(\log^k)) \not\supseteq \mathrm{SC}$. Cook and Tompa (see Tompa [6]) show that the structured branching program model (with either $\{=, \neq\}$ or $\{<, =, >\}$ as the allowable comparisons) may provide a sufficiently general setting for this conjecture.

Another important direction for future work lies in the related (but apparently different) question of size vs. depth. The recent work of Pippenger [13], Ruzzo [14], and Dymond and Cook [15], has focused attention on the stability and importance of the class NC; that is, those problems for which there are algorithms which run *simultaneously* in polynomial size (= sequential operations) and $\log^k$ depth (= parallel time). Again, it is a central issue in complexity to establish the conjecture $P \cap (\bigcup_k$ parallel time $(\log^k)) \not\supseteq \mathrm{NC}$.

Motivated by the results of this paper, we would like to find a problem for which (say) size · depth $= \Omega(N^2)$. Sorting will not suffice since we can sort simultaneously in $\log^2$ depth and $N \log^2 N$ size using a Batcher sorting network. However, one is tempted to conjecture that any Boolean circuit for sorting which uses only $k \log N$ depth requires $cN^{1+\varepsilon}$ size where c and ε will depend upon k. The class of problems which are computable by a log depth, $N \log^k N$ size circuit is a class of practical importance. We suspect that it will be difficult to prove that a given problem does not belong to this class.

Acknowledgment. We sincerely thank Romas Aleliunas and Patrick Dymond for their many helpful suggestions.

REFERENCES

[1] A. AHO, J. HOPCROFT AND J. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] N. PIPPENGER AND L. G. VALIANT, *Shifting graphs and their applications*, J. Assoc. Comput. Mach. 23 (1976), pp. 423–432.

[3] A. BORODIN, *Structured vs general models in computational complexity*, presented at Internationales Symposium über Logik und Algorithmik zu Ehren von Professor Ernst Specker, Feb., 1980, Zurich, L'Enseignement Mathematique, to appear.

[4] A. COBHAM, *The recognition problem for the set of perfect squares*, Conference Record, IEEE 7th Annual Symposium on Switching and Automata Theory, 1966, pp. 78–87.

[5] F. HENNIE, *Crossing sequences and off-line Turing machine computations*, Conference Record IEEE Symposium on Switching Circuit Theory and Logical Design, 1965, pp. 179–190.

[6] M. TOMPA, *Time-space tradeoffs for straight-line and branching programs*, Tech. Rep. 122/78, Dept. Computer Science, Univ. of Toronto, July 1978.

[7] A. BORODIN, M. J. FISCHER, D. KIRKPATRICK, N. LYNCH, M. TOMPA, *A time-space tradeoff for sorting on non-oblivious machines*, Proc. IEEE 20th Annual Symposium on Foundations of Computer Science, Puerto Rico, Oct. 1979.

[8] D. E. KNUTH, *Mathematical analysis of algorithms*, Proc. of IFIP Congress 71, C. V. Freeman, ed., vol. 1, North-Holland, Amsterdam, 1972, pp. 19–27.

[9] M. FISCHER AND A. MEYER, *Boolean matrix multiplication and transitive closure*, Conference Record, IEEE 12th Annual Symposium on Switching and Automata Theory, 1971, pp. 129–131.

[10] W. FELLER, *An Introduction to Probability Theory and its Applications*, I, John Wiley, New York, 1968.

[11] S. COOK, *Deterministic CFL's are accepted simultaneously in polynomial time and log squared space*, Proc. ACM Symposium on Theory of Computing, 1979, pp. 338–345.

[12] ———, *Towards a complexity theory of synchronous parallel computation*, presented at Internationales Symposium über Logik und Algorithmik zu Ehren von Professor Ernst Specker, Feb. 1980, Zurich, L'Enseignement Mathematique, to appear.

[13] N. PIPPENGER, *On simultaneous resource bounds*, Proc. IEEE 20th Annual Symposium on Foundations of Computer Science, Puerto Rico, Oct. 1979, pp. 307–311.

[14] L. RUZZO, *On uniform circuit complexity*, Proc. IEEE 20th Annual Symposium on Foundations of Computer Science, Puerto Rico, Oct. 1979, pp. 312–318.

[15] P. DYMOND AND S. COOK, *Hardware complexity and parallel computation*, Proc IEEE 21st Annual Symposium on Foundations of Computer Science, Oct. 1980, pp. 360–372.

[16] G. N. FREDERICKSON, *Upper bounds for time-space trade-offs in sorting and selection*, Tech. Rep. CS-80-3, Dept. Computer Science, Pennsylvania State University, January 1980.