Samplesort: A Sampling Approach to Minimal Storage Tree Sorting

W. D. FRAZER*

IBM Corporation,† Yorktown Heights, New York

AND

A. C. McKELLAR‡

Princeton University,§ Princeton, New Jersey

ABSTRACT. The methods currently in use and previously proposed for the choice of a root in minimal storage tree sorting are in reality methods for making inefficient statistical estimates of the median of the sequence to be sorted. By making efficient use of the information in a random sample chosen during input of the sequence to be sorted, significant improvements over ordinary minimal storage tree sorting can be made.

A procedure is proposed which is a generalization of minimal storage tree sorting and which has the following three properties: (a) There is a significant improvement (over ordinary minimal storage tree sorting) in the expected number of comparisons required to sort the input sequence. (b) The procedure is statistically insensitive to bias in the input sequence. (c) The expected number of comparisons required by the procedure approaches (slowly) the information-theoretic lower bound on the number of comparisons required. The procedure is, therefore, "asymptotically optimal."

KEY WORDS AND PHRASES: algorithms, comparison, minimal storage, probability, Quicksort, random sample, sorting, tree sorting

CR CATEGORIES: 5.31

Introduction

General purpose sorting routines commonly approach the problem of sorting a very large file, one whose size greatly exceeds available high speed storage capacity, by breaking the sorting process into two or more "phases" [2]. In the initial, "internal sorting" phase, subsets of the file which can be accommodated in the available memory are sorted, and the resulting sorted sequences transferred to backup storage. The subsequent phase(s) attend to the merging of several such sorted sequences to produce a sorted file. Significant economies can be effected by making the size of the subsets sorted in the initial phase as large as possible, provided this can be done in an efficient way. A corollary of this is the requirement that auxiliary "working" storage required by an internal sorting algorithm (in excess of the storage necessary to hold the items being sorted) be minimized.

* This work was done while the author, on leave from IBM Corporation, was with the Department of Electrical Engineering, Princeton University, Princeton, N. J.
† Thomas J. Watson Research Center.
‡ Present address: Polytechnic Institute of Brooklyn, Brooklyn, N. Y.
§ Department of Electrical Engineering.

A measure of the working efficiency of an internal sorting algorithm is provided by matching the expected number of comparisons it requires against the information-theoretic lower bound of $\log_2 (n!)$ comparisons [1, 8], provided comparisons constitute the bulk of the algorithm's operations.

The two most efficient methods for minimal storage internal sorting are (a) that of Shell [9] as modified by Frank and Lazarus [3], or by Hibbard [4], and (b) the Quicksort tree sorting procedure of Hoare [6], modified to minimal storage form by Hibbard [4, 5]. Although the former requires essentially no auxiliary storage space and makes approximately $n \times \log_2 (n)$ comparisons [3, 5], the additional work involved, aside from comparisons, seems in practice to make it less efficient than minimal storage Quicksort for large n. For small n, however, it is quite attractive [5].

Quicksort constructs a binary tree whose vertices correspond to the items in the input set, and whose formation is governed by the following rule, applied recursively: if item x_0 is assigned to vertex i of the tree under construction, then any item, x, which is assigned to the subtree rooted at vertex i will be assigned to the left subtree of vertex i if $x < x_0$ and to the right subtree of vertex x_0 if $x > x_0$. (For convenience in analysis, the elements in the input set are assumed to be distinct.) Under the assumption that all permutations of the input set are equally likely, it has been shown [4, 11] that the expected number of comparisons, $E(C_q)$, using minimal storage Quicksort on an input sequence of size n is given by:

$$\begin{aligned} E(C_q) &= 2(n+1)\sum_{1}^{n}\frac{1}{i+1} - 2n \\ &\doteq 1.39(n+1)\log_2 (n) - 2.85n + 2.15. \end{aligned} \tag{1}$$

The choice of a root is critical, for the more nearly balanced the tree resulting from the algorithm, the fewer the comparisons required to complete the sort. Although it makes no difference under the assumption above, Hoare proposed choosing an element at random from the input sequence as the root of the sort tree, in order to circumvent bias which may in practice be present. He further realized, but did not pursue, the fact that one might obtain a more balanced expected sorting tree by choosing the median of a small sample as the root of each sort tree. Hoare's proposal amounts to making an estimate of the median of the input sequence from a sample of size one, and the idea he did not pursue to making an estimate from a sample of larger size. We demonstrate that by estimating instead the cumulative distribution function of the input sequence from a sample, and by choosing the appropriate subtree roots on the basis of this estimate, one can arrive at a minimal storage tree sorting procedure (a) which yields significant improvement over Quicksort in the expected number of comparisons (1) over the range of n of practical interest, even without the assumption of an unbiased input sequence, and (b) for which the expected number of comparisons approaches (slowly) the information-theoretic lower bound, and which is, therefore, asymptotically optimal.

Description and Implementation of the Procedure

The procedure we propose is a generalization of minimal storage Quicksort. For purposes of description, its operation is best divided into three distinct phases, although the minimal storage requirement forces the three phases to be intermingled in implementation. In the first phase, a sample chosen at random from the input

sequence is sorted. In the second phase, this sample is "inserted" (literally or figuratively, depending upon implementation) into the input sequence to form the apex of the sort tree. In the final phase, the remaining subsequences are sorted, one at a time. The manner in which the sample is sorted in the first phase, and that in which the subsequences are sorted in the final phase, are not critical to the success of the procedure; however, we assume for convenience of analysis that minimal storage Quicksort is used throughout.

Remarks concerning the generation and sorting of the random sample required for the first phase of our procedure are deferred until later. Assuming, however, that we have a random sample sorted by some method, we come to the second phase of the procedure, wherein the sample is used to form the apex of the sort tree. We describe two approaches to implementation of this "insertion" phase of the procedure; other approaches are possible, however.

Examination of the description of minimal storage tree sorting reveals that the root need not be chosen from the input sequence; the root, y, need only be comparable with the items of the input sequence X. It is inefficient not to choose y from the input sequence, but it is unnecessary to insert y physically into the sorted sequence, X, immediately. The simplest implementation of the method we propose takes advantage of this fact.

The root for the initial partition of the "remainder sequence" (i.e. the input sequence less the sample) is chosen to be the median of the (sorted) sample. The root for the next subtree sort is chosen to be the lower or upper quartile point of the sample, depending upon whether the initial or terminal segment, respectively, resulting from the first partition was shorter. One continues in this way, deciding which subsequence to partition next on the basis of the two minimal storage rules of Hibbard [4], and choosing as root the appropriate percentile point of the sample, until no more such percentile points of the sample are available. One then sorts the remaining subsequences by means of, say, ordinary minimal storage Quicksort, choosing the remaining roots as usual from the subsequences being sorted, and inserting them at once. At any stage in this process one has in auxiliary storage a list of pointers indicating which subsequences remain to be sorted; as with ordinary minimal storage Quicksort, the size of this list is bounded by $\log n$. Depending upon its level, each such subsequence will be sorted on the basis of a root chosen either from the sample or the subsequence itself.

At the conclusion of this process, one has two separate sorted sequences: the sample and the remainder sequence. These are then merged during output to yield a sorted sequence. If one has tagged those items of the remainder sequence which immediately precede insertion points for items of the sample, few comparisons need be made during the merging. This is referred to subsequently as method I.

The second implementation which we discuss inserts the sample physically into the remainder sequence during construction of the sort tree. Although this alleviates the need for merging at the time of output, it necessitates a few additional data transfers during the course of the procedure. For this procedure, we assume that the sorted sample is separated initially into two portions, one half stored in locations preceding the remainder sequence and the other half following it. The procedure itself follows closely the pattern of operations of ordinary minimal storage Quicksort.

Assume, without loss of generality, that the median of the sample is stored ad-

jacent to the initial item of the remainder sequence. The median, y_m, is used as the root of the sort tree; it is transferred to a temporary location and compared, as before, with the items of the remainder sequence beginning with the last and working forward. When the first item, in location j_1, is found which is less than y_m, x_{j_1} is copied into the location at the beginning of the input sequence formerly occupied by y_m. y_m is next compared with the items of the input sequence, beginning with the first, until an item is encountered which is smaller; this item is then copied from its location, i_1, into location j_1. The process is now repeated, beginning with location $(j_1 - 1)$ and working backward to the next item, in location j_2, less than y_m. This item is then copied into location i_1, and so on. At the end of this process, when the i and j pointers converge to adjacent locations, y_m is copied into the location from which the last item was transferred. Those items in locations following that of y_m now have values greater than y_m, and those in locations preceding that of y_m have values less than y_m. Suppose that y_m lies before the midpoint location of the remainder sequence; then, as required by minimal storage protocol, those items in the second quartile of the sample are exchanged with the items of the remainder sequence which immediately precede y_m. The initial subsequence of the remainder sequence now looks just like the entire remainder sequence did at the beginning of the process: it is preceded by a sorted sequence containing one half of the sample elements to be inserted into it, and followed by a sorted sequence containing the other half. The lower quartile point is now inserted just as the median was previously. The case in which the final subsequence is the shorter is handled analogously.

One continues in this way, choosing the subsequence to be processed next according to the rules described previously, and surrounding a subsequence with the items to be inserted into it before making any insertions, until no more sample items remain to be inserted. Each such resulting subsequence is then sorted by means of, say, ordinary minimal storage Quicksort. At the conclusion of this process, one has a completely sorted sequence and no further operations, other than output, are necessary. This will be referred to in the sequel as method II.

There is a variant of method II which does not require the added data transfers. The transfers are eliminated by moving the items of the sample only when they are required as roots of some (sub-) tree. The bookkeeping for this method is quite complex in the latter stages, however, for items of an unsorted subsequence, rather than being in consecutive memory locations, are interspersed with items of the sample sequence. Further, an additional log $(n - l)$ locations are required for pointers to enable one to find these items.

Analysis

Let C be the number of comparisons required to sort n items by the method proposed. Then the expected value of C is given by:

$$E(C) = E(C_1) + E(C_2) + E(C_3), \tag{2}$$

where C_1 is the number of comparisons required to sort the sample, C_2 is the number of comparisons required to insert the sample, and C_3 is the number of comparisons required to sort the segments of the remainder sequence. In order to compute the expected values of C_1, C_2, and C_3, we introduce the following notation.

Let $X = \{x_1, \cdots, x_n\}$ be the set of input items with items numbered such that

$x_i < x_{i+1}$. Let $Y = \{y_1, \cdots, y_l\}$ be the randomly chosen subset of X also numbered such that $y_i < y_{i+1}$. The insertion process, described previously, partitions the set $X - Y$ into $l + 1$ subsets, $X_0, \cdots, X_l$, where:

$$X_0 = \{x : x < y_1\},$$
$$X_i = \{x : y_i < x < y_{i+1}\}, \qquad 1 \leq i < l,$$
$$X_l = \{x : y_l < x\}.$$

Let n_i be the number of elements in X_i, $0 \leq i \leq l$.

In order to compute the expected values of C_2 and C_3 we need a variety of probability density functions. Let $q_i(j)$ denote the probability that $y_i = x_j$, i.e. that y_i is the jth element of the sorted set. It is a routine matter to show that:

$$q_i(j) = \binom{j-1}{i-1}\binom{n-j}{l-i}\Big/\binom{n}{l} \tag{3}$$

The assumption that Y is a random sample is implicit in (3) but no assumptions are made about the distribution of items in X, nor about the order in which they appear in the input sequence. With the usual definition of the binomial coefficient, $q_i(j)$ is zero except for $i \leq j \leq n - l + i$ as required.

Let $p_i(j)$ be the probability that $n_i = j$. Lemma 1 shows that $p_i(j)$ is independent of i, i.e. the n_i are identically distributed random variables, although they are not statistically independent. Hence, in the sequel, $p_i(j)$ is denoted by $p(j)$.

LEMMA 1.

$$p_i(j) = \binom{n-j-1}{l-1}\Big/\binom{n}{l}. \tag{4}$$

PROOF. For $i = 0$ and $i = l$, the lemma follows immediately from (3). For $0 < i < l$, we have:

$$p_i(j) = \sum_{t=i}^{n-l-j+i} q_i(t)q_{i+1}(t + j + 1 \mid y_i = x_t), \tag{5}$$

which follows from consideration of the joint density function for the position of y_i and y_{i+1} in X. Observing that the conditional probability in (5) is just $q_1(j + 1)$ for a sample of size $(l - i)$ from a set of size $(n - t)$, we have, on substitution from (3),

$$p_i(j) = \sum_{t=1}^{n-l-j+i} \binom{t-1}{i-1}\binom{n-t-j-1}{l-i-1}\Big/\binom{n}{l}. \tag{6}$$

Equation (4) can be derived from (6) by binomial coefficient identities or induction on i. Hence the lemma is proven.

Not surprisingly, it follows from (4) that the expected value of n_i is $(n - l)/(l + 1)$.

When l is a number of the form $2^k - 1$, C_2 is $(n - l)\log_2(l + 1)$. For other values of l, C_2 becomes a random variable. Lemma 2 shows that $(n - l)\log_2(l + 1)$ is a reasonable approximation for the expected value of C_2 for all values of l.

LEMMA 2.

$$(n - l)\log_2(l + 1) \leq E(C_2) < (n - l)[0.0861 + \log_2(l + 1)]. \tag{7}$$

PROOF. Let

$$l = 2^k - 1 + m, \qquad 0 \leq m < 2^k. \tag{8}$$

It follows that

$$E(C_2) = (n - l)k + 2(n - 1)m/(l + 1), \tag{9}$$

since $(n - l)k$ is the number of comparisons required to insert $2^k - 1$ items, and the remaining m items are inserted into disjoint subsets of $X - Y$ whose expected size is $2E(n_i)$. It is easily verified that

$$f(m) = E(C_2)/(n - l) - \log_2 (l + 1) \tag{10}$$

has a maximum at

$$m = (2 \ln 2 - 1)2^k. \tag{11}$$

Since the value of this maximum is less than 0.0861, the lemma follows.

We now address the problem of evaluating the expected value of C_3. Toward this end, we need:

LEMMA 3.

$$E(C_3) = (l + 1) \sum_{j=0}^{n-l} p(j)E(c(j)), \tag{12}$$

where $E(c(j))$ is the expected number of comparisons required to sort a set of size j.

PROOF. C_3 is the sum of the random variables w_i, $0 \leq i \leq l$, where w_i is the number of comparisons required to sort the ith segment of the remainder sequence. Using the fact that the expectation of a sum is the sum of the expectations,

$$E(C_3) = \sum_{i=0}^{l} E(w_i). \tag{13}$$

Now, expressing $E(w_i)$ as

$$E(w_i) = \sum_{j=0}^{n-l} E(w_i \mid n_i = j)p(j), \tag{14}$$

we obtain, after interchanging the order of summation,

$$E(C_3) = \sum_{j=0}^{n-l} \sum_{i=0}^{l} E(w_i \mid n_i = j)p(j). \tag{15}$$

But $E(w_i \mid n_i = j)$ is just $E(c(j))$; hence the lemma follows.

Equation (12) is valid regardless of the method used to sort the $(l + 1)$ subsets. In this analysis, we postulate the use of minimal storage Quicksort for which $E(c(j))$ is given by (1).

The form of our next result is much more instructive if we make use of the following clever identity, suggested by D. E. Knuth [7]:

LEMMA 4.

$$\sum_{k=0}^{n} \binom{k}{m} \frac{1}{n + 1 - k} = \binom{n + 1}{m} \sum_{k=m+1}^{n+1} \frac{1}{k}. \tag{16}$$

PROOF. Equation (16) is easily verified for $m = 0$ and for $m \geq n$. Its validity for $0 < m < n$ can be established by an inductive argument. We show that validity of the identity for $m \leq n \leq (N - 1)$ implies validity for $m \leq n = N$; together with

the previously established validity for $m = 0$ and $m = n$, this implies that the identity is valid for all m, n. Assume that (16) holds for $m \leq n \leq (N-1)$:

$$\sum_{k=0}^{N} \binom{k}{m} \frac{1}{N+1-k} = \sum_{k=m}^{N} \frac{1}{N+1-k} \sum_{i=m-1}^{k-1} \binom{i}{m-1} \tag{17}$$

$$= \sum_{i=m-1}^{N-1} \binom{i}{m-1} \sum_{k=i+1}^{N} \frac{1}{N+1-k} \tag{18}$$

$$= \sum_{i=m-1}^{N-2} \binom{i}{m-1} \sum_{k=1}^{N-i-1} \frac{1}{k} + \sum_{i=m-1}^{N-2} \binom{i}{m-1} \frac{1}{N-i} + \binom{N-1}{m-1}. \tag{19}$$

Combining the last two terms and invoking the induction hypothesis yields:

$$\sum_{k=0}^{N} \binom{k}{m} \frac{1}{N+1-k} = \binom{N}{m} \sum_{m+1}^{N} \frac{1}{k} + \binom{N}{m-1} \sum_{m}^{N} \frac{1}{k}. \tag{20}$$

Combining these terms, we arrive at the desired result:

$$\sum_{k=0}^{N} \binom{k}{m} \frac{1}{N+1-k} = \binom{N+1}{m} \sum_{m+1}^{N+1} \frac{1}{k}, \tag{21}$$

and (16) is true for $m \leq n = N$, as required.

We next address the question of evaluating $E(C_3)$.

THEOREM 1. *If Quicksort is used to sort the $(l+1)$ subsets, then*

$$E(C_3) = 2(n+1) \sum_{i=l+1}^{n} \frac{1}{i+1} - 2(n-l) \tag{22a}$$

$$\leq 2(n+1)\, ln\,(n/l) - 2\left[(n-l) + \frac{n+1}{l+1} - 2\right]. \tag{22b}$$

PROOF. Substitution of (1) and (4) in (12) yields

$$E(C_3) = \frac{l+1}{\binom{n}{l}} \sum_{j=1}^{n-l} \binom{n-j-1}{l-1} \left[2(j+1) \sum_{i=1}^{j} \frac{1}{i+1} - 2j\right]. \tag{23}$$

Interchanging the order of summation, (23) can be cast as

$$E(C_3) = \frac{l+1}{\binom{n}{l}} \sum_{i=1}^{n-l} \sum_{j=1}^{n-l} 2\binom{n-j-1}{l-1} \frac{j+1}{i+1} - \frac{l+1}{\binom{n}{l}} \sum_{j=1}^{n-l} 2j\binom{n-j-1}{l-1}. \tag{24}$$

Using the identity

$$\sum_{j=i}^{n-l} j\binom{n-j-1}{l-1} = \binom{n-1}{l+1} + i\binom{n-i}{l}, \tag{25}$$

(24) can be written as

$$E(C_3) = \frac{l+1}{\binom{n}{l}} \sum_{i=1}^{n-l-1} \frac{2}{i+1} \binom{n-i}{l+1}. \tag{26}$$

Making use of Lemma 4, we obtain, as required:

$$E(C_3) = \frac{2(l+1)}{\binom{n}{l}}\left[\binom{n+1}{l+1}\sum_{i=l+1}^{n}\frac{1}{i+1} - \binom{n}{l+1}\right] \tag{27}$$

$$= 2(n+1)\sum_{i=l+1}^{n}\frac{1}{i+1} - 2(n-l). \qquad (\cdots 22a)$$

Equation (22b) comes from the usual approximation for partial sums of the harmonic series [4, 11].

It is instructive to compare (22a) with (1); the latter may be viewed as a special case of the former, with sample size $l = 0$.

In view of (1), (2), (7), and (22b) we have:

COROLLARY 1. *If n, l, and $(n - l)$ are large,*

$$E(C) \doteq \left[(n+1)\ln(n+1) - (n-l)\ln(l+1) + \gamma(l+1) - (n+l+1) - \left(\frac{n-l}{2}\right)\log_2(l+1)\right] \tag{28a}$$

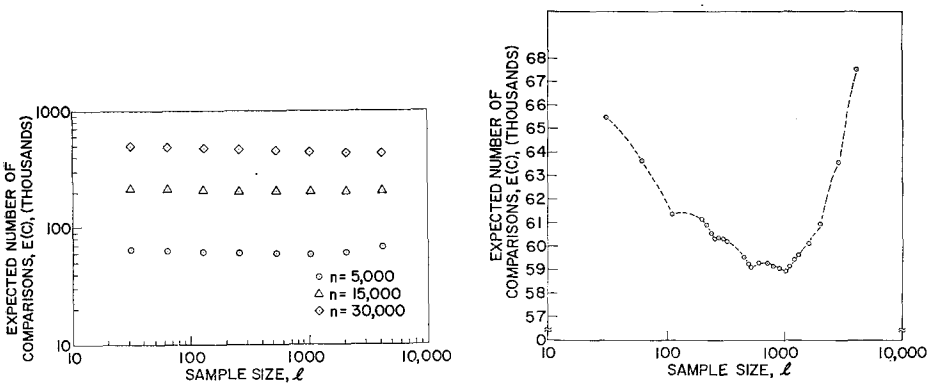
$$\doteq 1.386n\log_2 n - 0.386(n-l)\log_2 l - 2n - 0.846l. \tag{28b}$$

If l is not a number of the form $2^k - 1$, the coefficients of n and l should be modified according to the upper bound of Lemma 2. For values of n in the range of interest, n, l, and $n - l$ are sufficiently large that the approximations made in arriving at (28) are valid. If method I is used without tags and if the final merging process cannot be overlapped with output, $E(C)$ should be increased by n.

Approximating the partial sums of the harmonic series in the usual way [4, 11], approximating $E(C_2)$ by $(n - l)\log_2(l + 1)$, and equating to zero the partial derivative of $E(C)$ with respect to l, we have, on neglecting small terms:

$$n \doteq (l+1)\left[\ln(l+1) + \frac{2\gamma\ln 2 - 1}{2\ln 2 - 1}\right] \doteq (l+1)[\ln(l+1) - 0.51]. \tag{29}$$

Figure 1(a) shows $E(C)$ as a function of l for several values of n. These curves were obtained by computer from the exact formulas for $E(C)$. Figure 1(b), for $n = 5000$,



(a) (b) $n = 5000$ (enlarged)

FIG. 1. Expected comparisons as function of sample size

has an expanded scale to show the shape of the curves in the neighborhood of the optimum value of l. The discontinuities in the slope which occur whenever $l = 2^k - 1$ are of course due to the properties of $E(C_2)$ as given by (9). For all cases examined in detail the optimum choice of l was a number of the form $2^k - 1$ and the following discussion provides evidence which indicates strongly that this is true at least for $n \leq 50{,}000$.

For $l \neq 2^k - 1$, we consider the following measure of the curvature of $E(C(l))$:

$$E(C(l)) - \frac{1}{2}[E(C(l+1)) + E(C(l-1))] = \frac{(n+1)(2^k + 1 - m) - l(l+2)}{l(l+1)(l+2)}, \quad (30)$$

where $l = 2^k - 1 + m$; $0 < m < 2^k$. This is easily seen to be a monotone decreasing function of l. Thus the shape of Figure 1 (b) between $l = 511$ and $l = 1023$ is typical. For a given value of n, suppose that of the numbers of the form $2^k - 1$, $l = 2^r - 1$ is the one which minimizes $E(C)$. It follows that if $2^r - 1$ is not the optimum choice for l, then $l = 2^r - 2$ must yield a smaller expected number of comparisons. Furthermore, it is reasonable to suppose that the critical values of n are those with the property that for $n - 1$ items to be sorted, $l = 2^{r-1} - 1$ is the optimum choice of the form $2^k - 1$. All such points in the range $1000 \leq n \leq 50{,}000$ were tested by computer. The results showed that for this range of n, our procedure is optimized by setting $l = 2^k - 1$ and taking the optimum choice of k. Table I indicates the range of n for which each choice of k is appropriate.

While it is apparently possible that there exists a value of n for which our procedure is not optimized by an l of the form $2^k - 1$, this would be of academic interest only. Any saving achieved by allowing arbitrary choice of l would be small and would probably be nullified by the resulting complication to the procedure.

One can gain further insight into the source of the improvement achieved by the method we propose from an examination of the difference between $E(C_q)$ and $E(C)$. This difference has the approximate form:

$$E(C_q) - E(C) \doteq (n - l)[(2 \ln (l + 1) + 2\gamma) - (\log_2 (l + 1) + 2)]. \quad (31)$$

The two terms of (31) have interesting interpretations: the first represents the expected number of comparisons required to insert the l sample items into the remainder sequence in a random manner, while the second represents the expected number required to insert them systematically (i.e. median first, etc.). It is also interesting to observe that (31) is easily shown to have the same functional form as $E(C_2)$.

TABLE I. OPTIMUM CHOICE OF k FOR $1000 \leq n \leq 50{,}000$

k	$2^k - 1$	*Range of n*	
		From	*To*
8	255	5,000	2,008
9	511	2,009	4,521
10	1023	4,522	10,058
11	2047	10,059	22,154
12	4095	22,155	48,392
13	8191	48,393	50,000

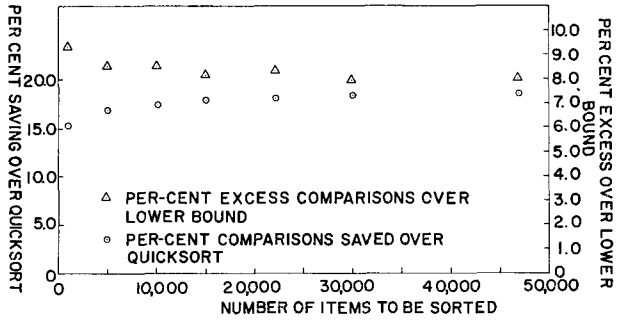


FIG 2. Performance of Samplesort in range of practical interest

Note that the value of l which maximizes this difference (31) is the same as that given by (29). Figure 2 shows this difference as a percentage of $E(C_q)$ for the best choice of l of the form $2^k - 1$. For current memory sizes, we can expect an improvement of 15–20 percent. Further, we note that our procedure becomes more attractive as the size of internal storage and (hence n) increases.

We now proceed to compare $E(C)$ with the information theoretic lower bound of $\log_2 (n!)$. For this purpose, we have:

LEMMA 5. *For* $n > 1000$,

$$\frac{E(C) - log_2 (n!)}{log_2 (n!)} < \frac{0.387\ [ln\ (ln\ l) + 0.15]}{ln\ (l) + ln\ (ln\ (l)) - 1}. \tag{32}$$

PROOF. Using an upper bound for $E(C)$ derived from (1), (7), and (22b) and the lower bound for $\log_2 (n!)$ given by:

$$\log_2 (n!) > \log_2 e(n \ln n - n), \tag{33}$$

we have

$$E(C) - \log_2 (n!) < (2 - \log_2 e)(n \ln n - n \ln l + l \ln l - n) + 0.0861n - .932l - \frac{2n}{l} + \left(2 - \frac{1}{2}\log_2 e\right) \ln n + 8. \tag{34}$$

Choosing l by $n = l \ln l$ and eliminating n, we have:

$$\frac{E(C) - \log_2 (n!)}{\log_2 (n!)} \leq \frac{\left[(2 - \log_2 e)[l \ln l\ (\ln \ln l) + 0.15l \ln l] + [-.932l - 2 \ln l + \left(2 - \frac{1}{2}\log_2 e\right)(\ln l + \ln \ln l) + 8]\right]}{\log_2 e(l \ln l \ln l + l \ln l \ln \ln l - l \ln l)}. \tag{35}$$

It is readily verified that for $n > 1000$ and hence $l > 100$, the second term in the numerator of (34) is negative. Thus the lemma follows.

Lemma 5 is also valid for much smaller values of n but that fact is of no interest. As a corollary of Lemma 5, we have:

THEOREM 2.

$$\lim_{n \to \infty} \frac{E(C)}{log_2 (n!)} = 1. \tag{36}$$

Thus in the limit, as n (and hence l) tends to infinity, $E(C)$ approaches the lower bound, and our procedure is asymptotically optimal. Figure 2 indicates the rate at

which this convergence takes place and shows that for practical storage size Theorem 2 is of academic interest only. Figure 2 was obtained by using Stirling approximation for $n!$ and (28b) for $E(C)$. The corresponding improvement ove Quicksort approaches 28 percent in the limit.

For values of n in the range of interest (i.e. current core storage sizes) the optimu value of l is roughly $0.1\,n$. This validates the conditions of Corollary 1. In addition, i suggests that one might consider sorting the sample using our procedure rather tha Quicksort. This would reduce $E(C)$ only by about 2 percent in the range of practic interest, however; hence it is unlikely to be worthwhile.

Of more importance is the effect that the size of l has on the generation of th random sample. Although the generation of such a random sample adds complexit to the overall sorting algorithm, it is of little consequence in practice, since the sele tion of the sample can normally be concurrent with input of the sequence to be sorte Since the optimal value of l is a significant fraction of n, it seems reasonable to a sume that for most applications one could generate the sample by choosing ever (n/l)-th item from the input sequence. An even simpler (but riskier) heuristic woul be to choose the first l items from the input sequence. The objective, of course, is t generate a sample whose cumulative distribution function provides an unbiased est mate for the cumulative distribution function of the input sequence and thus ensu that the probability distribution of (3) is the appropriate one. The selection of simple heuristic to accomplish this depends upon the nature of the bias present in th input sequence for a particular application.

Summary and Remarks

We have presented here a new method for minimal storage internal sorting, one whic can provide both an increase in efficiency and a decrease in sensitivity to bias in th input sequence.

At the outset of this investigation, we thought in terms of a small sample. Th well-known Kolmogorov-Smirnov [10] result concerning the maximum difference b tween the cumulative distribution functions between two samples from an arbitrar continuous distribution suggests that for a fixed sample size, l, this maximum di ference should be nearly independent of n. The surprising fact is, however, that a one continues to increase n, the efficiency decreases unless l is also increased.

Perhaps the most striking feature of this sorting procedure, apart from the im provement over minimal storage Quicksort shown in (31) and Figure 2, is that thi procedure should be almost insensitive to bias which may be present in the input se quence. In fact, this insensitivity to bias would be guaranteed if one were actually t choose a random sample, but, in view of the size of sample that is required, it woul seem more practical to employ one of the coarse approximations to random samplin which were suggested previously.

Another interesting (and reassuring) feature of the procedure we propose is it asymptotic optimality. Although convergence to the absolute minimum is slow room for further improvement is small even in the range of current practical interes

There are several ways in which one can achieve additional small improvements i efficiency. As mentioned above, our method rather than Quicksort could be used t sort the sample, thus reducing the number of comparisons by roughly 2 percent ove the range of interest (in the limit, the percentage improvement gained by this tacti

tends to zero). Since the expected size of the subsequences which remain after the insertion process is roughly 10 for most existing computers, Shell's method [9], "shuttle sort," or some other procedure might be used to sort them. However, the additional storage required for the program would reduce the size of the input sequence which could be accommodated, and hence it is an open question as to whether or not the efficiency of the total sorting process could be improved in this way.

REFERENCES

1. BELL, D. A. The principles of sorting. *Computer J. 1*, 1 (1958), 71–77.
2. GOTLIEB, C. C. Sorting on computers. *Comm. ACM 6*, 5 (May 1963), 194–201.
3. FRANK, R. M., AND LAZARUS, R. B. A high-speed sorting procedure. *Comm ACM 3*, 1 (Jan. 1960), 20–22.
4. HIBBARD, T. N. Some combinatorial properties of certain trees with applications to searching and sorting. *J. ACM 9*, 1 (Jan. 1962), 13–28.
5. ——. An empirical study of minimal storage sorting. *Comm. ACM 6*, 5 (May 1963), 206–213.
6. HOARE, C. A. R. Quicksort. *Computer J. 5* (1962), 10–15.
7. KNUTH, D. E. Personal communication.
8. MORRIS, R. Some theorems on sorting. *SIAM J. Appl. Math. 17*, 1 (Jan. 1969), 1–6.
9. SHELL, D. L. A high-speed sorting procedure. *Comm. ACM 2*, 7 (July 1959), 30–32.
10. SIEGEL, S. *Nonparametric Statistics*. McGraw-Hill, New York, 1956.
11. WINDLEY, P. F. Trees, forests and rearranging. *Computer J. 3*, 2 (1960), 84–88.

RECEIVED DECEMBER, 1968; REVISED JULY, 1969