SELECTION AND SORTING WITH LIMITED STORAGE

J.I. Munro *

Dept. of Computer Science
University of Waterloo
Ontario, Canada.

M.S. Paterson

Dept. of Computer Science
University of Warwick
Coventry, U.K.

Abstract

When selecting from,or sorting, a file stored on a read-only tape and the internal storage is rather limited, several passes of the input tape may be required. We study the relation between the amount of internal storage available and the number of passes required to select the K^{th} highest of N inputs. We show, for example, that to find the median in two passes requires at least $\Omega(N^{\frac{1}{2}})$ and at most $O(N^{\frac{1}{2}} \log N)$ internal storage. For probabilistic methods, $\Theta(N^{\frac{1}{2}})$ internal storage is necessary and sufficient for a single pass method which finds the median with arbitrarily high probability.

1. Introduction

As a paradigmatic study of effects of internal storage limitations on large-scale data-processing tasks, we consider problems of searching and sorting in data stored on a one-way read-only tape when the amount of random access working space is severely constrained. We shall quantify rather closely the relation between the number of passes over the input file which are required for these tasks and the amount of storage available for a given size of the file. In several cases the upper bounds are demonstrated by new sampling algorithms of some practical interest.

In our computational model the data is a sequence of N distinct elements stored on a one-way read-only tape. An element from the tape can be read into one of S locations of random-access storage. The elements are from some totally ordered set (for example the real numbers) and a binary comparison can be made at any time between any two elements within the random-access storage. Initially the storage is empty and the tape is placed with the reading head at the beginning. After each pass the tape is rewound to this position with no reading permitted.

* This author was partially supported by a Senior Visiting Fellowship from the Science Research Council while visiting the University of Warwick.

Notational note.

For functions of several arguments we shall write $f(\underset{\sim}{X}) = O(g(\underset{\sim}{X}))$ when $\exists c > 0$ such that $|f(\underset{\sim}{X})| < c.g(\underset{\sim}{X})$ for all $\underset{\sim}{X}$ except those naturally or explicitly excluded. We also use $f = \Theta(g)$ for $g = O(f)$; and we use $f = \Omega(g)$ for $f = O(g)$ and $g = O(f)$.

In section 2 we present results concerning the problem of sorting the data, where, in view of the limitations imposed by our model, this must be considered as the determination of the sorted order rather than any actual rearrangement. For P-pass algorithms we show that $\Theta(N/P)$ storage locations are necessary and sufficient.

The greater part of this paper is occupied with the selection problem of retrieving for some given K, the K^{th} highest among N input elements. For clarity and convenience we adopt a terminology of altitude in respect of the ordering, e.g. we use terms such as "highest", "below", "lower than". The most interesting special case of this is finding the median (i.e. when $K = \lceil N/2 \rceil$). By symmetry we may always assume that $K \leqslant \lceil N/2 \rceil$.

It is easy to show that K + 1 locations are necessary and sufficient to retrieve the K^{th} highest element ($1 \leqslant K \leqslant \lceil N/2 \rceil$) in a single pass. Algorithms using this minimal storage are studied in [1], where it is shown that for the median only $\Theta(N)$ comparisons are needed, whereas for $K \sim \alpha N$, α fixed, $0 < \alpha < \frac{1}{2}$, the retrieval of the K^{th} highest requires $\Theta(N.\log N)$ comparisons.

In contrast, a two-pass probabilistic method using only $\Theta(N^{2/3})$ storage and $3N/2 + o(N)$ comparisons is presented in [2]. Making use of an internal randomizer, it finds the K^{th} highest element with an arbitrarily great probability, which is independent of the order of the inputs.

The principal results obtained in this paper are upper and lower bounds which show the amount of storage required by a P-pass deterministic selection algorithm to be roughly $N^{1/P}$. Other results are that under the rather strong assumption that all input orderings are equally likely, for a single-pass algorithm with a high expectation of selecting the median, $\Theta(N^{\frac{1}{2}})$ locations are necessary and sufficient.

2. Elementary results

Here, as throughout the paper, S denotes the number of storage locations available. Since comparisons can only be made within these locations, we will assume always that $S \geqslant 2$. A naive sorting algorithm determines in its first pass the highest S-1 elements of the input, and their relative order, and then in successive passes it ignores any elements ranked in previous passes in order to determine the ranks of the next S-1 highest elements. This algorithm requires only $\lceil (N-1)/(S-1) \rceil$ passes. We note that to "ignore" previously ranked elements requires the retention of a large amount of information by the program. A large "program memory" is inconsistent with our storage limitation for any practical application. If we suppose that the ranking may be output as it is being determined then an algorithm with very small program memory may be obtained at the cost of just one extra storage location. This location is to hold the lowest element ranked so far and each new data element is compared with this to determine whether or not it should be ignored. Nearly all the algorithms to be described will use this technique in order to remain within the domain of practicality.

We give a simple lower bound argument to establish the following result.

Theorem 1. The least storage required by any P-pass sorting algorithm for N elements is $\Theta(N/P)$.

Proof. In view of the algorithm given above we require only a lower bound. Suppose that the ordering of the data is such that 1st, 3rd, 5th, ... highest elements are in the first half of the tape, whereas the 2nd, 4th, 6th, ... are in the second half. Since a valid algorithm must at some time make a direct comparison between the $(2r-1)^{st}$ and $(2r)^{th}$ elements for $r = 1, \ldots, \lfloor N/2 \rfloor$, either the odd-ranked element must be carried in storage at some forward transition across the midpoint of the tape or the even-ranked element must be retained during some intermediate rewind. If P passes are used by an algorithm for this case we can argue that

$$(2P-1)S \geqslant \lfloor N/2 \rfloor$$

Hence $S > N/4P$ $\square$

3. Multi-pass algorithms for selection

When S is more than about $(\log N)^2$ an efficient algorithm may be designed as follows. At the beginning of each pass a pair of elements, filters, between which the required element is guaranteed to lie, is retained in the storage, though their precise ranks may so far be undetermined. At the start of the algorithm we may pretend that "ideal" elements representing $\pm\infty$ fulfil this role. During the pass any elements not between the filters are used merely to establish the exact ranks of the filters. From the remainder a suitably constructed sample is retained from which a new pair of filters is selected.

For the initial pass the number of elements between the filters is N, and for the final pass this is to be reduced to at most S-2 so that all such elements can be retained for a final selection. With the details of the algorithm we shall establish the following relation.

Lemma 1. If at most n elements lie between the filters at the beginning of a pass then for the following pass this number is $O(n(\log n)^2/S)$.

A simple estimation from this lemma yields the next upper bound.

Theorem 2. A P-pass algorithm which selects the K^{th} highest of N elements requires storage at most $O(N^{1/P}.(\log N)^2)$, and only $O(N^{\frac{1}{2}}.\log N)$ for $P = 2$.

Outline of the algorithm.

For some fixed s, a sample at level i will be a sorted set of s elements chosen from a "population" of $2^i.s$ elements according to the following scheme.

A sample at level 0 consists of the whole population of $2^o.s$ elements in sorted order. A sample at level $i + 1$ is formed by taking two samples at level i from the first and second halves of the population, "thinning" each by retaining only the second, fourth, sixth, ... elements from the top in each, and then merging the two subsamples to form one sorted sample.

We shall show that a sample deserves its name in that it contains a reasonably well spaced selection from the total order of its population. To this end consider the j^{th} element from the top in a sample at level i. We denote by L_{ij}, M_{ij} respectively the least, and most numbers of elements from its corresponding population which can appear strictly above it in the total order.

Lemma 2.
$$L_{ij} = j.2^i - 1$$
$$M_{ij} = (i + j - 1).2^i$$

Proof. Clearly, for $1 \leqslant j \leqslant s$, $L_{oj} = M_{oj} = j - 1$. We use the convention that $L_{io} = -1$ for all $i \geqslant 0$. For $i \geqslant 1$, $j \geqslant 1$, we may then verify that

$$L_{ij} = \min_{\substack{p+q=j\\ p>o,\ q\geqslant o}} \{L_{i-1,2p} + L_{i-1,2q} + 1\}$$

and

$$M_{ij} = \max_{\substack{p+q=j\\ p>o,\ q\geqslant o}} \{M_{i-1,2p} + M_{i-1,\ 2q+2}\}$$

From these equations the result may be proved inductively. □

For a population of size at most $2^r.s$ from which we wish to select the k^{th} highest we shall choose as new filters, the u^{th} and v^{th} elements of the final sample at level r, where u,v are the greatest and least integers respectively such that

$$k - 1 \geqslant M_{ru} = (r + u - 1)2^r$$

and

$$k - 1 \leqslant L_{rv} = v.2^r - 1$$

The k^{th} element must then be one of these elements or lie between them in the order. The number of elements between them is at most

$$\begin{aligned}M_{rv} - L_{ru} - 1 &= (L_{rv}+1+(r-1)2^r)-(M_{ru}-1-(r-1)2^r)-1\\ &= 2(r-1)2^r + (L_{rv} - M_{ru} + 1)\\ &\leqslant (2^r-1)2^r\end{aligned}$$

since $L_{rv} + 1 \leqslant M_{ru} + 2^r$ by the extremality of u,v.

The maximum storage required is for a subsample (consisting of even-positioned elements of a sample) for each level below the r^{th}, for one "working sample" and for the pair of filters. This is at most $rs/2 + s + 2$. We choose $s = \lceil S/\log n\rceil$ and $r = \lceil \log(n/s)\rceil$ so that $n \leqslant 2^r.s$ and the storage required is at most S, when S is sufficiently large. (We can assume $S \geqslant \Omega((\log n)^2)$.) The population size n' for the next pass satisfies

$$n' \leqslant (2r-1)2^r \leqslant 4rn/s = O(n(\log n)^2/S)$$

and we have justified Lemma 1. The storage requirement of the algorithm can be reduced by a constant factor if samples are combined five at a time instead of two at a time.

Very small storage

It is clear that the above algorithm requires $S \geqslant \Omega((\log N)^2)$. For smaller values of S, one might employ the more practical of the "sorting" algorithms and terminate it after $\lceil(K-1)/(S-2)\rceil$ passes. This is the only algorithm we know for very small storage which does not require extensive program memory. If we disregard practical limitations and allow an algorithm to remember an arbitrary amount of information about previous comparisons, we can prove the following upper bound.

Theorem 3. For $2 \leqslant S \leqslant O((\log N)^2)$, there is a class of selection algorithms which use at most $O((\log N)^3/S)$ passes.

Proof. The algorithms simulate each pass of the algorithm of Theorem 2 by several passes with smaller storage. The comparisons performed in one pass of the original algorithm can be understood in correspondence with a binary tree of height r. At the leaves are 2^r level 0 samples of size $s = \lceil \log n \rceil$. At successive levels of the tree pairs of adjacent samples are thinned and merged until the final sample at level r is reached.

With storage S equal to s, all the operations at one level of the tree can be carried out in one pass, whereas with $S > s$, it is possible to execute $\Theta(^S/s)$ levels at once. When $S < s$, a single level can be completed in $\Theta(^s/S)$ passes. The sorting and merging operations are done by the naive multi-pass sorting algorithm described in section 2, applied simultaneously to each sample. The memory required by the program to record the partial progress during such an operation would be intolerable in practice. However in all cases where $2 \leqslant S < O((\log N)^2)$ the total number of passes to simulate one pass before is $\Theta\left(\frac{(\log N)^2}{S}\right)$. The total of passes for the selection problem is therefore $O((\log N)^3/S)$. □

4. Lower bounds for multi-pass selection

To show that the upper bounds derived in the previous section are close to optimal we here present corresponding lower bounds. Our main proof uses the idea of the "Adversary" who, knowing the innermost workings of our algorithm, devises an ordering of the input to confound it. He may also supply us with any extra information whatsoever, which cannot of course adversely affect the performance of the algorithm but is designed to facilitate the proof.

Theorem 4. Any P-pass algorithm to determine the median (or Kth highest for $N/2 \geqslant K = \Omega(N)$) of N elements requires at least $\Omega(N^{1/P})$ storage locations.

Corollary 1. The minimum storage S for a two-pass algorithm satisfies

$$\Omega(N^{\frac{1}{2}}) \leqslant S \leqslant O(N^{\frac{1}{2}}.\log N)$$

Proof. Immediate from Lemma 1 and Theorem 4. □

Corollary 2. Provided $\log S \geqslant \Omega((\log N.\log\log N)^{\frac{1}{2}})$, the maximum number of passes required is $\log N/\log S + O(1)$, while for $\log\log N = o(\log S)$ we have $P \sim \log N/\log S$.

Proof. Immediate from Theorems 2 and 4. □

The proof of Theorem 4 follows at once from the following Lemma which establishes that after one pass of any (median-finding) algorithm using S locations there remains to be done a computation at least as hard as finding the median for an input of size approximately N/2S.

Lemma 3. For any S-location algorithm on N input elements there is an ordering of the input tape so that after the first pass there is a set X of inputs with the following properties:

(i) no element of X remains in storage,

(ii) no orderings between elements of X are known,

(iii) the median of the original set is the median of X,

and (iv) X contains at least $\lfloor N/(2S-1) \rfloor$ elements.

Proof of Lemma. Without loss of generality the algorithm reads the first S inputs into storage and decides which one to discard as the (S+1)st input is read. The Adversary ensures that this (S+1)st element stands in the same relative ordering with respect to the remaining S-1 elements in storage as the one it replaces. This strategy for the Adversary is followed repeatedly, replacing each discarded element by a new element which is effectively indistinguishable. For $x = \lfloor N/(2S-1) \rfloor$, as the (Sx + 1)st element is about to be read, at least one of the storage locations has had discarded from it a set X of at least x elements between which no comparisons have been made and no orderings can yet be deduced. It may be verified that the relative ordering of the remaining N - Sx elements may be designed so that the median element is the median of X. □

Whilst the asymptotic constant of $\frac{1}{2}$ in this lemma can be raised to ln 2, and even higher, by a more refined argument, an upper limit of this approach is marked by the trivial algorithm, which inputs and discards S at a time and leaves "incomparable" sets of size at most $\lceil N/S \rceil$.

5. Selection algorithms that "nearly always" succeed

If we make the assumption (not required in [2]) that all input orderings are equally likely and we are willing to tolerate some small probability, say 10^{-6}, of failure then the amount of storage required can be much reduced. For example a single-pass median algorithm finds $\Theta(N^{\frac{1}{2}})$ storage necessary and sufficient.

Probabilistic algorithms for selecting the median.

For a suitable choice of storage size S, the algorithm maintains in storage for as long as it can S-1 elements whose ranks among those read thus far are consecutive and as close to the current median as possible. To this end it keeps two counts H and L, both initially zero, of the numbers of elements which have so far been discarded above and below, respectively, the consecutive segment retained. Under our assumption of equal likelihood, the probability that a new element read lies above all those retained is precisely $(H + 1)/(H + S + L)$. In this case the element must be discarded and H incremented by one. The case where it lies below the retained segment is similar. With probability $(S - 2)/(H + S + L)$, the new element can be inserted strictly within the segment and either the highest or lowest of those retained is chosen for discarding according as $H < L$ or $H \geqslant L$ respectively.

At the end of the tape the median has been retained and determined provided $H + 1 \leqslant \lceil N/2 \rceil \leqslant N - L$. We have only to estimate the size of S required to guarantee this result with high probability. The progress of the algorithm can be viewed as a random walk of the integer variable $D = H-L$ starting from the origin. It can be verified that for $0 < |D| < S-1$ the probability that $|D|$ is reduced at the next step is at least one half, and furthermore the condition that the median is found is equivalent to $|D| < S-1$.

For any $\epsilon > 0$, there is a constant C such that during the first CS^2 steps of a random walk about the origin, with equal probabilities of a step to the right or left, the probability of the random variable attaining magnitude S-1 is at most ϵ (see [3]). Since the random walk for our algorithm is more biased towards the origin over the region of interest the same result holds.

The algorithm described can be used as the basis of a multi-pass algorithm in the following way. For suitably chosen constants C_1, C_2, depending on ϵ, the probability that the median of the whole input set lies between the extreme elements of the segment retained after C_1S^2 steps is very high. From this point on, for the remainder of the pass, the same S-1 elements are retained in storage and their ranks are found by comparisons with the rest of the input. If one of the retained elements is the median, the algorithm terminates; if not, the number of elements sharing the same "gap" as the median with respect to the stored elements can be shown to be at most C_2N/S^2 with high probability. This set of elements satisfy the same assumption as to randomness as the initial set and so the same procedure may be used for further passes. Hence:

Theorem 5. For any $\epsilon > 0$, $P \geqslant 1$ there is a P-pass median-finding algorithm with probability of failure at most ϵ which uses only $O(N^{1/2P})$ storage.

Lower bound for probabilistic algorithms.

Theorem 6. There is an $\epsilon > 0$, such that any one-pass algorithm which finds the median with probability of failure less than ϵ requires at least $\Omega(N^{\frac{1}{2}})$ storage.

Proof. Consider the situation after $\lceil N/2 \rceil$ elements have been read. The probability is at least half that the median is one of these, but only S of them can have been retained. The most likely candidates are towards the middle but the straightforward estimation of a hypergeometric distribution [3] shows that for a subset of size S of these elements to contain the median with probability above one quarter requires $S \geqslant \Omega(N^{\frac{1}{2}})$. □

Corollary. For a single-pass algorithm which nearly always finds the median, $\Theta(N^{\frac{1}{2}})$ locations are necessary and sufficient.

6. Conclusions

Our aim has been to determine the precise computational requirements for specific tasks of selecting from, or sorting, data presented on a read only input tape under a regime of limited internal storage. We present new algorithms of some practical interest as well as lower bound proofs which exploit the joint constraints on internal storage and access to input data.

Our main algorithm for selection uses a novel sampling technique and can be implemented easily to require only about $N(3P/2 + \log S)$ comparisons in all. The upper and lower bounds on storage differ only by a factor of order $(\log N)^2$ and give a clear idea of the trade-off relation between the number of passes and the amount of storage required.

The picture we have in the probabilistic case is much less complete. Theorem 6 is readily extensible to give a lower bound of $\log\log N - \log\log S - O(1)$ passes if we require that the only information retained from one pass to the next is a pair of filters and their ranks. It seems likely that the upper bound may be reduced to about this value but analysis of the algorithms we considered has so far proved intractable.

References

[1] Dobkin, D.P. and J.I. Munro, "Time and space bounds for selection problems", Proc. 5th Int. Colloq. on Automata, Languages and Programming, July 1978, Udine, Italy.

[2] Eisenstat, S.C., R.J. Lipton and J.I. Munro, "Probabilistic algorithms", in preparation.

[3] Feller, W., "An Introduction to Probability Theory and Its Applications : Vol.I" 3rd edition. (J. Wiley 1968).