Discrete Math. Appl., Vol. 16, No. 2, pp. 175–180 (2006)

A random algorithm for multiselection

M. H. ALSUWAIYEL

Abstract — Given a set S of n elements drawn from a linearly ordered set and a set $K = \{k_1, k_2, \ldots, k_r\}$ of positive integers between 1 and n, the multiselection problem is to select the k_ith smallest element for all values of i, $1 \leq i \leq r$. We present an efficient randomised algorithm to solve this problem in time $O(n \log r)$ with probability at least $1 - cn^{-1}$, where c is a positive constant.

1. INTRODUCTION

Let S be a set of n elements drawn from a linearly ordered set, and let $K = \{k_1, k_2, \ldots, k_r\}$ be a sorted list of positive integers between 1 and n, that is, a set of ranks. The multiselection problem is to select the k_ith smallest element for all values of i, $1 \leq i \leq r$. If $r = 1$, then we have the classical selection problem. On the other hand, if $r = n$, then the problem is tantamount to the problem of sorting.

It appears that finding efficient algorithms for the multiselection problem did not receive as much attention in the sequential environment as in the parallel invironment. The classical and simple sequential algorithm in [6] remains the only one and is a base on which several parallel algorithms were developed. It seems that the first parallelisation of the multiselection problem was that of Shen [8], in which he presented an optimal parallel algorithm that runs in time $O(n^{\varepsilon} \log r)$ on the EREW PRAM with $n^{1-\varepsilon}$ processors, $0 < \varepsilon < 1$. In the special case where $\varepsilon = \log(\log n \log^* n)/\log n$, his algorithm runs in time $O(\log n \log^* n \log r)$. He presented a general framework, which is basically a parallelisation of an optimal sequential algorithm that first finds the element x with rank $k_{r/2}$, partitions S into two groups: the elements smaller than x and the elements greater than x, which induces two subproblems that are solved by applying the algorithm recursively. Another algorithm with the same running time and number of processors can be found in [2]. This algorithm is a result of a simple modification of the parallel QUICKSORT algorithm in [1]. In [3], an optimally efficient parallel algorithm was presented. It runs in time $O\left(((n/p) + t_s(p))(\lg r + \lg(n/p))\right)$ on the EREW PRAM with p processors, $r \leq p < n$, where $t_s(p)$ is the time needed to sort p elements using p processors. This algorithm implies an efficient parallelisation of quicksort with multiple number of pivots on an EREW PRAM with p processors. More on the work of Shen on parallel algorithms for the multiselection problem on interconnection networks (e.g., the hypercube, the mesh and multidimensional meshes) can be found in [9, 10, 11, 12].

Originally published in *Diskretnaya Matematika* (2006) **18**, No. 2 (in Russian).
Received May 13, 2004.

In this research, we attempt to switch to the sequential environment and exhibit not only an efficient, but also a fast and practical randomised algorithm that is simple to describe. Moreover, the idea behind it is intuitive and the analysis of the algorithm is fairly simple.

2. DETERMINISTIC MULTISELECTION

The algorithm in [6], which we will refer to as MULTISELECT, is straightforward. Find the $\lceil k/2 \rceil$th smallest element a, partition the input set S into two sets S_1 and S_2 of elements, respectively, smaller and larger than a and make two recursive calls: one with S_1 and $\{k_1, k_2, \ldots, \lceil k/2 \rceil - 1\}$ and another with S_2 and $\{\lceil k/2 \rceil + 1, \lceil k/2 \rceil + 2, \ldots, k_r\}$.

Algorithm MULTISELECT (S, K).

(1) If $|K| > 0$ do Step 2 to 6.

(2) Set $k = k_{\lceil r/2 \rceil}$. Use SELECT to find s, the kth smallest element in S. Output s.

(3) By comparing s with the elements in S, determine the two sets S_1 and S_2 of elements less of equal to s and greater than s, respectively.

(4) Let $K_1 = \{k_1, k_2, \ldots, k_{\lceil r/2 \rceil - 1}\}$, $K_2 = \{k_{\lceil r/2 \rceil + 1}, k_{\lceil r/2 \rceil + 2}, \ldots, k_r\}$.

(5) Recursively call MULTISELECT on (S_1, K_1).

(6) Recursively call MULTISELECT on (S_2, K_2).

In Step 2, SELECT is a deterministic $\Theta(n)$ time algorithm for selection. Obviously, the algorithm solves the multiselection problem in time $\Theta(n \log r)$, as the recursion depth is $\log r$ and the work done in each level of the recursion tree is $\Theta(n)$. As to the lower bound for multiselection, suppose that it is $o(n \log r)$. Then, by letting $r = n$, we would be able to sort n elements in $o(n \log n)$ time, contradicting the $\Omega(n \log n)$ lower bound for comparison-based sorting on the decision tree model of computation. This lower bound has been previously established in [6]. It follows that the multiselection problem is $\Omega(n \log r)$, and hence the algorithm given above is optimal.

It appears that the deterministic multiselection algorithm above, as well as any other deterministic algorithm, are typical of the well-known classical selection algorithm (see [4]). Algorithm MULTISELECT is impractical, especially for small and moderate values of n. This impracticality is inherited, and indeed compounded, by the classical sequential multiselection algorithm. To see this, consider the case $K = \{1, 2, 3\}$. The algorithm first calls Algorithm SELECT with the input set S to find the 2nd smallest element. In a subsequent call to Algorithm SELECT, $n - 2$ elements will be reprocessed to find the 3rd smallest element. In general, it can be shown by referring to the recursion tree that if $K = \{1, 2, \ldots, r\}$, then the algorithm will call Algorithm SELECT $O(\log \log r)$ times with at least $n - r - 1$ elements.

Hoare's FIND algorithm[7], which is also referred to in the literature as Algorithm QUICKSELECT, is a very popular deterministic selection algorithm due to its simplicity and good average performance in spite of its $O(n^2)$ worst case behaviour. It seems that this algorithm is the best candidate to be used in conjunction with Algorithm MULTISELECT.

An obvious alternative for improving the efficiency of the algorithm is to resort to randomisation. A straighforward approach is to use a randomised version of Algorithm QUICKSELECT as a replacement of Algorithm SELECT in Step 2. However, the crucial issue of ranks being clustered in one or more regions, especially at the two extremes, as exemplified above, remains to be resolved. If, for instance, the ranks are clustered at the beginning, e.g., $K = \{1, 2, \ldots, r\}$, it would be desirable to get rid of as many unwanted large elements as possible.

3. THE ALGORITHM

In this section, we propose a simple and efficient algorithm which is tailor-made for the problem of multiselection. Randomised QUICKSORT is a very powerful algorithm, and as it turns out, a slight modification of the algorithm solves the multiselection problem efficiently. The idea is so simple and straightforward. Call the elements sought by the multiselection problem targets. For example if $j \in K$, then the jth smallest element in S is a target. Pick an element $s \in S$ uniformly at random, and partition the elements in S around s into small and large elements. If both small and large elements contain targets, let QUICKSORT continue normally. Otherwise, if only the small (large) elements contain targets, then discard the large (small) elements and recurse on the small (large) elements only. So, the algorithm is a hybrid of both QUICKSORT and QUICKSELECT algorithms. Note that by QUICKSORT we mean the randomised version of the algorithm.

In the algorithm description, we will use the following (invariably standard) notation to repeatedly partition S into smaller subsets. Let $y \in S$ with rank $k_y \in K$. Partition S into two subsets $S_{\leq} = \{x \in S \mid x \leq y\}$ and $S_{>} = \{x \in S \mid x > y\}$. Since k_y denotes the rank of y, this partitioning of S induces the bipartitioning of K

$$K_{\leq} = \{k \in K \mid k \leq k_y\}, \qquad K_{>} = \{k - k_y \mid k \in K,\ k > k_y\}.$$

The two pairs $(S_{\leq}, K_{\leq})$ and $(S_{>}, K_{>})$ will be called selection pairs. A selection pair (S, K), as well as the sets S and K, will be called active if $|K| > 0$; otherwise they will be called inactive. A more formal description of the algorithm is given below.

Algorithm QUICK-MULTISELECT (S, K)**.**

(1) If $|K| > 0$ do Step 2 to 6.

(2) If $S = \{a\}$ and $|K| = 1$, then output a.

(3) Let s be an element chosen from S uniformly at random.

(4) By comparing s with the elements in S, determine the two sets $S_{\leq}$ and $S_{>}$ of elements less of equal to s and greater than s, respectively. At the same time, compute $r(s)$, the rank of s in S. Use $r(s)$ to partition K into $K_{\leq}$ and $K_{>}$.

(5) If $|K_{\leq}| > 0$, call QUICK-MULTISELECT recursively on $(S_{\leq}, K_{\leq})$.

(6) If $|K_{>}| > 0$, call QUICK-MULTISELECT recursively on $(S_{>}, K_{>})$.

Clearly, in Step 2 of the algorithm, recursion should be halted when the input size become sufficiently small. It was stated this way only for the sake of simplifying the analysis of the algorithm and to make it more general (so that it will degenerate to QUICKSORT when $r = n$).

4. ANALYSIS OF THE ALGORITHM

Now we analyse the running time of the algorithm. First, we show that the recursion depth is $O(\log n)$ with high probability. Next, we show that its running time is $O(n \log r)$ with high probability too.

Fix a target element $t \in S$, and let the intervals containing t throughout the execution of the algorithm be $I_0^t, I_1^t, I_2^t, \ldots$ of sizes $n = n_0^t, n_1^t, n_2^t, \ldots$ Henceforth, we will drop the superscript t, and it should be understood from the context. In the jth partitioning step, a pivot v_j chosen randomly partitions the interval I_j into two intervals, one of which is I_{j+1}. Assume without loss of generality that $n \equiv 1 \pmod 4$). Then, $n_{j+1} \leq 3n_j/4$ if and only if v_j is within a distance at most $(n-1)/4$ from the median. Hence, the probability that $n_{j+1} \leq 3n_j/4$ is

$$\frac{1 + 2(n-1)/4}{n} = \frac{n+1}{2n} > \frac{1}{2}.$$

Let $d = 16\ln(4/3) + 4$. For clarity, we will write $\lg x$ in place of $\log_{4/3} x$.

Lemma 1. *For the sequence of intervals $I_0, I_1, I_2, \ldots$, after dm partitioning steps, $|I_{dm}| \leq (3/4)^m n$ with probability at least $1 - c_1((4/3)^{-2m})$, where c_1 is a positive constant. Consequently, the algorithm will terminate after $d \lg n$ partitioning steps with probability at least $1 - c_2(n^{-1})$, where c_2 is a positive constant.*

Proof. A partitioning step is called successful if it decreases the size of each induced interval to at least $3/4$ of the initial size. Since each successful partitioning decreases an interval size to $3/4$ or less of the the initial size, the number of successful splittings needed to reduce the size of I_0 to at most $(3/4)^m n$ is at most m. Therefore, it suffices to show that the number of failures exceeds $dm - m$ with probability $O((4/3)^{-2m})$.

Define the indicator variable X_j, $0 \leq j < dm$, to be 1 if $n_{j+1} > 3n_j/4$ and 0 if $n_{j+1} \leq 3n_j/4$. Let

$$X = \sum_{j=0}^{dm-1} X_j,$$

so X counts the number of failures. Clearly, as shown above, $X_1, X_2, \ldots$ are independent with $\mathbf{P}\{X_j = 1\} \leq 1/2$, $j = 0, 1, \ldots, dm-1$. Hence,

$$\mu = \mathbf{E}[X] = \sum_{j=0}^{dm-1} \mathbf{E}[X_j] = \sum_{j=0}^{dm-1} \mathbf{P}[X_j = 1] \leq \frac{dm}{2}.$$

We apply the Chernoff inequality

$$\mathbf{P}\{X \geq (1+\delta)\mu\} \leq \exp(-\mu\delta^2/4), \qquad 0 < \delta < 2e - 1,$$

to derive an upper bound for the number of failures. Namely, we estimate the probability $\mathbf{P}\{X \geq dm - m\}$ as follows:

$$\begin{aligned}\mathbf{P}\{X \geq dm - m\} &= \mathbf{P}\{X \geq (2 - 2/d)(dm/2)\} = \mathbf{P}\{X \geq (1 + (1 - 2/d))(dm/2)\} \\ &\leq \exp\left(\frac{-(dm/2)(1 - 2/d)^2}{4}\right) = \exp\left(\frac{-m(d - 4 + 4/d)}{8}\right) \\ &\leq \exp\left(\frac{-m(d-4)}{8}\right) = \exp\left(\frac{-m(16\ln(4/3))}{8}\right) \\ &= e^{-2m\ln(4/3)} = (4/3)^{-2m}.\end{aligned}$$

Consequently,

$$\mathbf{P}\{|I_{dm}| \leq (3/4)^m n\} \geq \mathbf{P}\{X < dm - m\} \geq 1 - (4/3)^{-2m}.$$

Since the algorithm will terminate when the sizes of all active intervals becomes 1, setting $m = \lg n$, we obtain

$$\begin{aligned}\mathbf{P}\{|I_{d\lg n}| \leq 1\} &= \mathbf{P}\{|I_{d\lg n}| \leq (3/4)^{\lg n}\, n\} \\ &\geq \mathbf{P}\{X < d\lg n - \lg n\} \geq 1 - (4/3)^{-2\lg n} = 1 - n^{-2}.\end{aligned}$$

Since the number of targets (and hence intervals) can be as large as $\Omega(n)$, using Boole's inequality, we conclude that the algorithm will terminate after $d\lg n$ partitioning steps with probability at least $1 - cn^{-1}$, where c is a positive constant.

Theorem 1. *The running time of the algorithm is $O(n \log r)$ with probability at least $1 - c/n$, where c is a positive constant.*

Proof. The algorithm will go through two phases: the first phase consists of the first $\log r$ iterations and the remaining iterations constitute the second phase. The first phase consists mostly of the first $\log r$ iterations of Algorithm QUICKSORT, while the second phase is mostly an execution of Algorithm QUICKSELECT. At the end of the first phase, the number of intervals will be $r + 1$, with at most r being active. Throughout the second phase, the number of active intervals will also be at most r. In each iteration, including those in the first phase, an active interval I is split into two intervals. If both intervals are active, then they will be retained, otherwise one will be discarded. Thus, for $q \geq 0$, after $2^q \log r$ iterations, $O(r^q)$ intervals will have been discarded, and at most r will have been retained.

Clearly, the time needed for partitioning the set S in the first phase of the algorithm is $O(n \log r)$. As to partitioning the set K of ranks, which is sorted, binary search can be employed after each partitioning of S. Since $|K| = r$, binary search will be applied at most $r - 1$ times for a total of $O(r \log r)$ extra steps.

Now we use Lemma 1 to bound the running time required for the second phase. In this phase, with high probability, there are at most $d\lg n - \log r$ iterations with at most r intervals, whose total number of elements is less than n at the beginning of the second phase. By Lemma 1, it follows that, with high probability, the number of comparisons in the second phase is upperbounded by

$$\sum_{t=1}^{r} \sum_{j=1}^{d\lg n - \log r} \left(\frac{3}{4}\right)^j |I^t_{\log r}| \sum_{t=1}^{r} |I^t_{\log r}| \sum_{i=1}^{d\lg n - \log r} \left(\frac{3}{4}\right)^j < n \sum_{i=0}^{\infty} \left(\frac{3}{4}\right)^j = 4n.$$

Consequently, the time taken by the second phase is $O(n)$. As a result, the overall time taken by the algorithm is $O(n \log r)$ with probability at least $1 - c/n$, where c is a positive constant.

5. CONCLUSION

In this research, we have presented an efficient randomised algorithm for the multiselection problem that runs in time $O(n \log r)$ with probability at least $1 - c/n$, where c is a positive constant. Algorithm QUICK-MULTISELECT can be viewed of as a unifying approach to randomised selection, multiselection and sorting, as it degenerates to Algorithm QUICKSELECT when $r = 1$ and to Algorithm QUICKSORT when $r = n$. Obviously, in practice, the algorithm should be used only for multiselection with the condition that r is not too large, that is, r should be in order of $O(n^{\varepsilon})$ for a sufficiently small ε as any other algorithm for multiselection.

ACKNOWLEDGEMENT

The author is grateful to King Fahd University of Petroleum and Minerals for their continual support. Thanks to an anonymous reviewer for his valuable comments.

REFERENCES

1. S. G. Akl, *The Design and Analysis of Parallel Algorithms.* Prentice Hall, Englewood Cliffs, New Jersey, 1989.
2. M. H. Alsuwaiyel, An optimal parallel algorithm for the multiselection problem. *Parallel Computing* (2001) **27**, 861–865.
3. M. H. Alsuwaiyel, An efficient and adaptive algorithm for multiselection on the PRAM. *Proc. ACIS 2nd Int. Conf. on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing* (2001) 140–143.
4. M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan, Time bounds for selection. *J. Comput. Syst. Sci.* (1973) **7**, 448–461.
5. R. W. Floyd and R. L. Rivest, Expected time bounds for selection. *Commun. ACM* (1975) **18**, 165–172.
6. M. L. Fredman and T. H. Spencer, Refined complexity analysis for heap operations. *J. Comput. Syst. Sci.* (1987), 269–284.
7. C. A. R. Hoare, FIND(Algorithm 65). *Commun. ACM* (1961) **4**, 321–322.
8. H. Shen, Optimal parallel multiselection on EREW PRAM. *Parallel Computing* (1997) **23**, 1987–1992.
9. H. Shen, Efficient parallel multiselection on hypercubes. In: *Proc. 1997 Intern. Symp. on Parallel Architectures, Algorithms and Networks.* IEEE CS Press, 1997, pp. 338–342.
10. H. Shen, Optimal multiselection in hypercubes. *Parallel Algorithms and Appl.* (2000) **14**, 203–212.
11. H. Shen, Y. Han, Y. Pan, and D. J. Evans, Optimal parallel algorithms for multiselection on mesh-connected computers. *Intern. J. Comput. Math.* (2003) **80**, 165–179.
12. H. Shen and F. Chin, Selection and multiselection on multi-dimensional meshes. *Proc. Intern. Conf. on Parallel and Distributed Processing Techniques and Appl.* 2002, 899–906.