Faster, Space-Efficient Selection Algorithms in Read-Only Memory for Integers

Timothy M. Chan[1], J. Ian Munro[1], and Venkatesh Raman[2]

[1] Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada, {tmchan,imunro}@uwaterloo.ca
[2] The Institute of Mathematical sciences, Chennai 600 113, India, vraman@imsc.res.in

Abstract. Starting with Munro and Paterson (1980), the selection or median-finding problem has been extensively studied in the read-only memory model and in streaming models. Munro and Paterson's deterministic algorithm and its subsequent refinements require at least polylogarithmic or logarithmic space, whereas the algorithms by Munro and Raman (1996) and Raman and Ramnath (1999) can be made to use just $O(1)$ storage cells but take $O(n^{1+\varepsilon})$ time for an arbitrarily small constant $\varepsilon > 0$.

In this paper, we show that faster selection algorithms in read-only memory are possible if the input is a sequence of integers. For example, one algorithm uses $O(1)$ storage cells and takes $O(n \lg U)$ time where U is the universe size. Another algorithm uses $O(1)$ storage cells and takes $O(n \lg n \lg \lg U)$ time. We also describe an $O(n)$-time algorithm for finding an approximate median using $O(\lg^{\varepsilon} U)$ storage cells.

All our algorithms are simple and deterministic. Interestingly, one of our algorithms is inspired by 'centroids' of binary trees and finds an approximate median by repeatedly invoking a textbook algorithm for the 'majority' problem. This technique could be of independent interest.

1 Introduction

The topic of this paper is the classical *selection* problem, where we want to find the k-th smallest element of an input sequence of n elements for a given number k. (The *median* corresponds to the $k = \lceil n/2 \rceil$ case.) Specifically, we are interested in space-efficient algorithms in the *read-only memory* model, where the input is a read-only array and we want to minimize the amount of extra space needed in addition to the input array.

Selection in read-only memory has been studied since Munro and Paterson's pioneering work on streaming algorithms [13]. Their deterministic algorithm runs in $O(n \lg_s n + n \lg s)$ time using $O(s)$ storage cells, for any given $s = \Omega(\lg^2 n)$. Frederickson [9] refined the running time to $O(n \lg_s n + n \lg^* s)$. (This bound has been recently improved [8] slightly in the extreme end when s approaches $n/\lg n$.) All these algorithms work by finding an 'approximate median' in one pass, which is used to find the exact median or the k-th smallest element over several passes.

The space of one-pass streaming algorithms for approximate medians has been improved to $O(\lg n)$ cells by Greenwald and Khanna [11]; this could potentially extend the range of allowed values for s to $\Omega(\lg n)$.

However, for small, sublogarithmic space, the best time bound increases dramatically. Munro and Raman [14] gave an $O(2^s s n^{1+1/s})$-time algorithm, and this was improved to $O(s n^{1+1/s} \lg n)$ by Raman and Ramnath [15], for any s from 1 to $\lg n$.

If randomization is allowed, much better results are possible. Chan [6] provided a matching upper and lower bound of $\Theta(n \lg \lg_s n)$ on the expected running time for all s from 1 to n. Our focus here, however, will be on deterministic algorithms only.

All the above results are in the comparison model. In this paper, we study the selection problem in the setting where the input elements are integers in $[U] := \{1, 2, \ldots, U\}$. Our model of computation is the word RAM model where standard (arithmetic, shift and bitwise logical) word operations can be performed in constant time. We assume that each word is w bits long, and w is at least $\lg n$ and $\lg U$, so that a pointer or input integer can fit in a word. We assume that each memory cell can store a word.

The case of integer input is standard when studying general lower bounds, for example, for sorting in the read-only memory model [1, 3], and randomized selection in the streaming model [5]; these lower bounds often match or nearly match their corresponding upper bounds in the comparison model. In contrast, we show that the integer assumption makes a big difference for the deterministic selection problem in read-only memory with small space. Our results are the following:

1. a selection algorithm using $O(n \lg_s U)$ time and $O(s)$ words of space for any s from 1 to n;
2. a selection algorithm using $O(n \lg n \lg_s \lg U)$ time and $O(s)$ words of space for any s from 1 to $\lg U$.

The first algorithm, presented in Section 2, is very simple and works well when the universe size U is polynomially bounded. For example, for $s = 1$ and $U = n^{O(1)}$, the running time is $O(n \lg n)$, which is significantly faster than those of Munro and Raman or Raman and Ramnath for constant s. For $s = \lg^{\varepsilon} n$ and $U = n^{O(1)}$, the running time is $O(n \lg n / \lg \lg n)$, where ε denotes an arbitrarily small positive constant. For $s = n^{\varepsilon}$ and $U = n^{O(1)}$, we get linear running time, eliminating the iterated logarithmic factor from Frederickson's result.

The second algorithm, presented in Section 3, is less sensitive to U and beats previous algorithms for a wider range of universe sizes. On our way to obtaining this second result, we also give

3. an approximate median algorithm using $O(n \lg_s \lg U)$ time and $O(s)$ words of space for any s from 1 to $\lg U$.

For $s = 1$, the time bound is $O(n \lg \lg U)$. For $s = \lg^{\varepsilon} U$, it is linear. The algorithm makes a clever use of a well-known algorithm, first published in [4] (discovered in 1980) to find the *majority* of a sequence of bits. To speed up this algorithm, we give a procedure to find the majority at several prefix lengths of the given sequence simultaneously in linear time. The resulting linear-time algorithm for approximate median is relatively simple and self-contained, and immediately implies a new deterministic linear-time algorithm for selection in the traditional, non-space-efficient setting, when the input elements are integers. This may be of independent interest, as the standard deterministic linear-time algorithm for selection [2] requires a doubly recursive structure, which the new algorithm manages to avoid.

When the two algorithms are combined, it is possible to obtain, for example, a selection algorithm that uses $O(1)$ words of space and whose running time is guaranteed to be at most $O(n \lg^{1+\varepsilon} n)$ regardless of the universe size U. This consequence is noted in Section 4.

We should mention that at least one prior work on approximate medians has also addressed upper bounds for the integer case: Shrivastava et al. [16] gave one-pass streaming algorithms for maintaining approximate quantiles using $O(\lg U)$ words of space. There are some similarities, but our approach is simpler, besides being more space-efficient (they maintained information about the trie induced by the binary representations of the numbers, whereas we maintain just one path in the trie, since we do not need all approximate quantiles). Like many other works in streaming algorithms, they were less interested in analyzing the total running time. It is conceivable that some of the previous streaming algorithms on approximate medians could be sped up in the integer case by using advanced data structures such as the fusion tree [10]. Our algorithms however do not require such complicated data structures.

2 An $O(n \lg_s U)$-time algorithm

We assume that the input integers are in the range $[U]$, and the aim is to find the k-th smallest element in the sequence of n integers. We give a very simple selection algorithm in read-only memory that takes $O(n \lg U)$ time using $O(1)$ space.

The algorithm goes through $\lg U$ stages where in stage i, we determine the i-th bit of the solution. At the first stage, we simply count the number of input integers with leading bit 0 and leading bit 1 and based on k, we determine the first bit of the answer, and update k to be the new rank of the element to be found among those input integers with the leading bit same as that of the answer.

We repeat the same procedure in the i-th stage, for $i > 1$ (having computed the $i-1$ bits of the answer) by counting how many integers, among those matching the first $i-1$ bits of the answer, have the i-th bit 0 and how many have the i-th bit 1, updating k and recursing appropriately. Thus we immediately obtain the following result:

Theorem 1. *Given a sequence of n integers in the range $[U]$, we can find the k-th smallest element of the sequence using $O(\lg U)$ passes, in $O(n \lg U)$ time using $O(1)$ words of space, in read-only memory.*

If we have $s = 2^b$ extra cells available, then we can read b bits of the input in each pass counting the number of inputs with each b-bit value using the 2^b counters, and finding b bits of the answer at each stage. The number of stages is thus reduced to $O((\lg U)/b)$. To ensure $O(n)$ run time for each pass, we limit the number of counters s to n. This results in the following theorem.

Theorem 2. *Given a sequence of n integers in the range $[U]$, we can find the k-th smallest element of the sequence using $O(\lg_s U)$ passes in $O(n \lg_s U)$ time using $O(s)$ words of space, in read-only memory for any $s \leq n$.*

3 An $O(n \lg n \lg_s \lg U)$-time algorithm

Our next approach solves the selection problem by designing better algorithms for finding an approximate median—specifically, an element whose rank is between $n/4$ and $3n/4$. The connection between exact selection and approximate median is well known. Given an algorithm for the latter problem, we can apply it to find an approximate median m for the subsequence of all input elements inside a current interval $[\ell, r]$ that contains the answer; we can then compute the rank of m in one additional pass, and then shrink $[\ell, r]$ to either $[\ell, m]$ or $[m, r]$, depending on whether the rank of m is greater or less than k. After about $\lg_{4/3} n$ iterations, the interval will be shrunk to a single point. If the given approximate median algorithm makes P passes over the input, has running time proportional to the cost of the P linear scans, and uses $O(s)$ space, then the overall algorithm has running time $O(nP \lg n)$ and uses $O(s)$ space.

We first present a simple algorithm to find an approximate median using $O(\lg \lg U)$ passes and $O(1)$ space. Our idea is inspired by the standard construction of a 'centroid' of a binary tree (namely, the trie formed by the binary representations of the input integers).

Theorem 3. *An approximate median from a range of $[U]$ can be found in $O(\lg \lg U)$ passes and $O(n \lg \lg U)$ time in read-only memory using $O(1)$ words of space. Hence the k-th smallest element can be found in $O(n \lg n \lg \lg U)$ time in read-only memory using $O(1)$ words of space.*

Proof. The idea is to find the longest prefix p such that the number of integers in the input sequence whose binary representations starts with p is more than $n/2$. Let m_0 and m_1 be the smallest input integer with prefix $p0$ and prefix $p1$ respectively, and let m_2 be the largest input integer with prefix $p1$. Let r_0, r_1, r_2 be the ranks of m_0, m_1, m_2 respectively. As $r_1 - r_0 \leq n/2$, $r_2 - r_1 \leq n/2$, and $r_2 - r_0 \geq n/2$ (due to the choice of p), it is easy to see that at least one of r_0, r_1, r_2 must lie in $[n/4, 3n/4]$. So, we can report one of m_0, m_1, m_2 as an approximate median.

Now, to find this longest prefix p, we use a standard binary-search over the lengths. We maintain an interval containing the desired prefix length (which is the entire length of $\lg U$ to start with). We take the midpoint of the interval, and if we find that this length has no element that occurs more than $n/2$ times (i.e., has no *majority* element), then we replace the interval with its lower half, else we replace the interval with its upper half. The number of iterations is $O(\lg\lg U)$.

At each prefix length, we need to decide whether a majority element exists in of a list of n elements. There is a well-known 'textbook' algorithm that solves the majority problem in $O(n)$ time, in two passes. We include a brief description, as it will be useful later. In the first pass, the pseudocode below finds a possible candidate p for the majority of $A[1], \ldots, A[n]$:

```
1.  initialize $c = 0$
2.  for $t = 1, \ldots, n$:
3.      if $c = 0$ then set $p = A[t]$
4.      if $A[t] = p$ then increment $c$ else decrement $c$
```

The claim is that after each iteration t,

$(*)$ if the majority of $A[1], \ldots, A[t]$ exists, then it must be p.

This can be seen from the following invariants: Let t' be the last value such that $c = 0$ at the end of iteration t'. Then

(a) $A[1], \ldots, A[t']$ do not have a majority; and
(b) $c = [\#\text{ of times } p \text{ occurs in } A[t'+1], \ldots, A[t]] -$
$[\#\text{ of times } p \text{ does not occur in } A[t'+1], \ldots, A[t]] \geq 0.$

By (b), the majority of $A[t'+1], \ldots, A[t]$ is p if $c > 0$, and does not exist if $c = 0$. Together with (a), this implies that $(*)$, because of the following property: the majority of the concatenation of two lists, if it exists, must be the majority of one of the two lists.

After computing the candidate majority p, we can in a second pass compute its frequency and then check whether it is more than $n/2$. Once we find the longest prefix p, we find m_0, m_1 and m_2 and their ranks r_0, r_1 and r_2 respectively in two additional passes in $O(n)$ time. □

We show how to speed up the preceding algorithm when we allow a little more space:

Theorem 4. *An approximate median can be found in $O(n \lg_s \lg U)$ time using $O(s)$ words of space in read-only memory for any $s \leq \lg U$. Hence the k-th smallest element can be found in $O(n \lg n \lg_s \lg U)$ time using $O(s)$ words of space in read-only memory.*

Proof. We speed up the preceding binary search with an 's-ary search', which reduces the number of iterations to $O(\lg_s \lg U)$. This requires extending the majority algorithm to compute majority candidates for s length values simultaneously, in two passes. More precisely, let $A[1], \ldots, A[n]$ be the input array, let $\ell_1, \ldots, \ell_s$ be s given lengths in increasing order, and let $\pi_i(p)$ denote the length-ℓ_i prefix of the binary representation of p. We want to compute a candidate for the majority of $\pi_i(A[1]), \ldots, \pi_i(A[n])$ for every $i = 1, \ldots, s$.

The pseudocode for the modified majority algorithm is given below. The key is to observe that we can make the s majority candidates to be prefixes of one common string p. This requires one subtle twist in the algorithm, where a counter may in one particular case stay at zero without being incremented or decremented:

```
0. initialize $c_1, \ldots, c_s$ to 0, $p$ to 0
1. for $t = 1, \ldots, n$:
2.     find the smallest $j$ with $\pi_j(A[t]) \neq \pi_j(p)$
3.     if $c_j = 0$ (or $j$ does not exist) then
4.         set $p = A[t]$ and increment $c_1, \ldots, c_s$
5.     else increment $c_1, \ldots, c_{j-1}$
6.         decrement the *nonzero* entries of $c_j, \ldots, c_s$
```

Correctness. The claim is that after each iteration t, for each i,

($*$) if the majority of $\pi_i(A[1]), \ldots, \pi_i(A[t])$ exists, then it is $\pi_i(p)$, and $c_i \neq 0$.

Let t_i be the last value such that $c_i = 0$ at the end of iteration t_i. Then the following three invariants are true:

(a) $\pi_i(A[1]), \ldots, \pi_i(A[t_i])$ do not have a majority;

(b) $c_i = [\#\text{ of times } \pi_i(p) \text{ occurs in } \pi_i(A[t_i+1]), \ldots, \pi_i(A[t])] -$
$[\#\text{ of times } \pi_i(p) \text{ does not occur in } \pi_i(A[t_i+1]), \ldots, \pi_i(A[t])] \geq 0$;

(c) $c_1 \geq c_2 \geq \cdots \geq c_s$.

First we argue that ($*$) follows from the three invariants. By (b), the majority of $\pi_i(A[t_i+1]), \ldots, \pi_i(A[t])$ is $\pi_i(p)$ if $c_i > 0$, and does not exist if $c_i = 0$. Together with (a), this implies ($*$).

It is straightforward to verify that the invariants are preserved in the case when lines 3–4 are executed (here, $c_j = \cdots = c_b = 0$ by (c)). So consider the $c_j \neq 0$ case instead when lines 5–6 are executed. It is straightforward to see that the invariants are preserved for any index i where c_i is incremented or decremented. The remaining subcase is when $i > j$ and c_i is zero, and is not decremented, but stays at zero. Then a majority of $\pi_i(A[1]), \ldots, \pi_i(A[t-1])$ does not exist. To maintain (a), we need to confirm that a majority of $\pi_i(A[1]), \ldots, \pi_i(A[t])$ does not exist. Assume otherwise. Then this majority must be $\pi_i(A[t])$. This would imply that the majority of $\pi_j(A[1]), \ldots, \pi_j(A[t])$ is $\pi_j(A[t])$, but this majority can only be $\pi_j(p)$: a contradiction with $\pi_j(A[t]) \neq \pi_j(p)$.

Implementation. Line 2 can be performed in $O(1)$ time by taking the exclusive-or of $A[t]$ and p and identifying the most significant 1-bit (a commonly encountered operation, which is known to be reducible to $O(1)$ standard word operations [10]). Lines 4, 5, and 6 require increment/decrement operations on length-s vectors, which can be done using a known data structure for dynamic counting by Dietz [7]. Because s is small, however, the data structure can be greatly simplified and we can give a short description.

The idea is to express $(c_s, \ldots, c_1)$ as a sum $(c'_s, \ldots, c'_1)+(\delta_s, \ldots, \delta_1)$, where the first vector does not change often and the second vector is kept small. Specifically, after a round of $s-1$ iterations, we reset $c'_i = \max\{c_i - s, 0\}$ and $\delta_i = c_i - c'_i \geq 0$; the amortized cost of the reset is $O(1)$. During a round, updates are applied to the δ_i's. Then $\delta_i < 2s$ at all times, so $(\delta_s, \ldots, \delta_1)$ can be packed in one word provided that $s \lg s = o(\lg U)$. If $s \lg s = \Omega(\lg U)$, we can change s to $\sqrt{\lg U}$, for example, and the result is the same because $O(n \lg_s \lg U)$ is the same as $O(n)$. Testing whether $c_i = 0$ is equivalent to testing whether $\delta_i = 0$. Line 4 corresponds to adding a constant $(1, \ldots, 1)$ to $(\delta_s, \ldots, \delta_1)$ and lines 5–6 correspond to adding a vector of the form $(0, \ldots, 0, 1, \ldots, 1)$ and subtracting a vector of the form $(0, \ldots, 0, 1, \ldots, 1, 0, \ldots, 0)$. These reduce to $O(1)$ standard arithmetic operations on words. Before we decrement, we need to identify the smallest i with $\delta_i = 0$; this reduces to finding the most significant 1-bit in $(\delta_s, \ldots, \delta_1)$.

Thus, the algorithm can be implemented to run in $O(n)$ time. It uses $O(s)$ words of space for storing p and $(c_s, \ldots, c_1)$.

After computing the candidate majorities $\pi_i(p)$, we can in a second pass compute their frequencies f_i: initialize $f_1, \ldots, f_s$ to 0, and for each $t = 1, \ldots, n$, find the smallest j with $\pi_j(A[t]) \neq \pi_j(p)$ and increment $f_1, \ldots, f_{j-1}$. By the same approach, the second pass takes $O(n)$ time as well. We can then check which of the frequencies are more than $n/2$. □

4 Final Remarks

We have shown that the selection problem for integers in read-only memory can be solved deterministically in $O(\min\{n\lceil\lg_s U\rceil,\ n \lg n \lceil\lg_s \lg U\rceil)$ time with $O(s)$ storage cells. Actually in all our algorithms, all but a constant number of cells in the extra memory store $O(\lg n)$-bit pointers or counters rather than $O(\lg U)$-bit integers. Thus, the space used in bits is $O(\lg U + s \lg n)$.

For example, we can set $s = \lg U / \lg n$ to ensure $O(\lg U)$ bits of space, and a time bound of $O(\min\left\{n\frac{\lg U}{\lg(\lg U/\lg n)},\ n \lg n \frac{\lg\lg U}{\lg(\lg U/\lg n)}\right\})$. Notice that the second term is $O(n \lg n)$ when $\lg U \geq \lg^{1+\varepsilon} n$. On the other hand, when $\lg U \leq \lg^{1+\varepsilon} n$, the first term is at most $O(n \lg^{1+\varepsilon} n)$. So, a combination of our two approaches yields an algorithm that uses $O(1)$ words of space and always runs in at most $O(n \lg^{1+\varepsilon} n)$ time—a bound independent of U.

A remaining question is whether there is a deterministic selection algorithm for integers in read-only memory that uses $O(1)$ words of space and runs in $O(n \lg n)$ time (or better) for all U. Another question is to what extent can our approximate median algorithm be improved; for example, is there a deterministic linear-time algorithm using $O(1)$ words of space in read-only memory? Also, in the comparison model, the best deterministic algorithm with $O(1)$ space currently requires $O(n^{1+\varepsilon})$ time in read-only memory [14, 15]. Can one prove a matching lower bound? This question appears difficult.

Finally, as was mentioned in the introduction, if randomization is allowed, $\Theta(n \lg \lg_s n)$ expected time algorithm is possible using $O(s)$ storage cells for general input. Can one extend the $\Omega(n \lg \lg_s n)$ lower bound proof [6] to a non-comparison model in the case when the input is a sequence of integers?

References

1. P. Beame, ‘A general sequential time-space tradeoff for finding unique elements’, SIAM Journal on Computing 20(2): 270-277 (1991).
2. M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest and R. E. Tarjan, ‘Time bounds for selection’, Journal of Computer and System Sciences 7: 448-461 (1973).
3. A. Borodin and S. A. Cook, ‘A time-space tradeoff for sorting on a general sequential model of computation’, SIAM Journal on Computing 11: 287-297 (1982).
4. R. S. Boyer and J. S. Moore, ‘MJRTY - A fast majority vote algorithm’, Automated Reasoning: Essays in Honor of Woody Bledsoe (R. S. Boyer, Ed.), Automated Reasoning Series, Kluwer, 105-117 (1991).
5. A. Chakrabarti, T. S. Jayram and M. Pătraşcu, ‘Tight lower bound for selection in randomly ordered streams’, Proceedings of the 19th ACM-SIAM Symposium on Discrete Algorithms (SODA): 720-729 (2008).
6. T. M. Chan, ‘Comparison-based time-space lower bounds for selection’, ACM Transactions on Algorithms 6(2): 26 (2010).
7. P. F. Dietz, ‘Optimal algorithms for list indexing and subset rank’, Proceedings of the First Workshop on Algorithms and Data Structures (WADS): 39-46 (1989).
8. A. Elmasry, D. Dahl Juhl, J. Katajainen and S. Rao Satti, ‘Selection from read-only memory with limited work space’, Proceedings of COCOON 2013: 147-157 (2013).
9. G. Frederickson, ‘Upper bounds for time-space trade-offs in sorting and selection’, Journal of Computer and System Sciences 34(1): 19-26 (1987).
10. M. L. Fredman and D. E. Willard, ‘Surpassing the information theoretic bound with fusion trees’, Journal of Computer and System Sciences, 47:424-436 (1993).
11. M. Greenwald and S. Khanna, ‘Space-efficient online computation of quantile summaries’, Proceedings of ACM SIGMOD: 58-66 (2001).
12. G. S. Manku, S. Rajagopalan and B. G. Lindsay, ‘Approximate medians and other quantiles in one pass with limited memory’, Proceedings of ACM SIGMOD 27(2) 426-435 (1998).
13. J. I. Munro and M. Paterson, ‘Selection and sorting with limited storage’, Theoretical Computer Science 12: 315-323 (1980).
14. J. I. Munro and V. Raman, ‘Selection from read-only memory and sorting with optimum data movement’, Theoretical Computer Science 165(2): 311-323 (1996).
15. V. Raman and S. Ramnath, ‘Improved upper bounds for time-space tradeoffs for selection’, Nordic Journal of Computing, 6 (2): 162-180 (1999).
16. N. Shrivastava, C. Buragohain, D. Agrawal, S. Suri, ‘Medians and beyond: new aggregation techniques for sensor networks’, Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems: 239-249 (2004).