ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 5, Issue 6, June 2016

Optimal median selection under memory access constraints

By Mirza Abdulla
AMA International University

Abstract
We consider the problem of selecting the element whose order is the kth element of a list of n unsorted elements but with access cost to an address that is a function of the address, in deviation from the widely used RAM model. We study the problem under various access cost functions and the general case of a continuous monotonically increasing function and obtain optimal bounds under these constraints.

Index Terms—**median, selection, time complexity, memory access.**

I. Introduction

Kth order statistics or the selection of the *k*th element is the problem of given a list of *n* elements, we are required to find the element that has rank *k* or has *k*-1 elements not greater than it. When k is equal to 1 we call the element the minimum, maximum when *k* is *n* and median when *k* = *n*/2. This problem can be solved in *O*(*n*) time in the worst case [1]. The solution is optimal, since one has to read all data to decide correctly if the chosen element is indeed the *k*th element. However, if the data was preprocessed then it is possible to get better times. For example, if the data is sorted prior to the request for the *k*th element then the required element can be found *O*(1). One can also use a data structure like the binary heap[2]. The construction of the heap can be completed in *O*(*n*) time as a preprocessing step. However, each extractMin would require *O*(lg*n*) time which may imply that the median would need to be found in *O*(*n*lg*n*) time.
A number of implementations of solutions to selection problem exist. An efficient solution on the average is QuickSelect [3], whose operates like the QuickSort algorithm, but would only recur on one of the lists instead of both as in quicksort. Unfortunately, the worst case performance of QuickSelect is $O(n^2)$, exceeding even the worst cost of sorting all the data by a good sorting algorithm. However, the first algorithm to achieve *O*(*n*) time in the worst case for the Selection problem is due to [1] which is referred to as the median of medians.

The complexity computation of these algorithms is based on the RAM model of computation, which ignores memory hierarchy by treating the memory as composed of registers only. Such a model is unrealistic, and can lead to many implementations of what can provably be optimal algorithm on the RAM model but not necessarily optimal in real life computers due to the factor played by memory hierarchy. The cost factor caused by memory hierarchy is usually offset by bringing data from higher levels of memory to lower levels of memory in blocks. Moreover, effort is usually made to make use of both temporal and spatial localities, which needn't be catered for on Random Access Machine model.

In this paper, we shall consider a model of computation that takes care of memory hierarchy and block transfer, yet remaining with the simplicity and ease of the RAM model.

II. Model of Computation

The Hierarchical Memory Machine (HMM) of [4] is a machine similar to the Radom Access Machine (RAM), in that it consists of infinite number of registers, R1, R2, R3, …. with each register having room for one integer. The operations on the HMM are the same as those for the RAM. However, the cost of memory access is different. While memory access on the RAM model costs a single unit for every location accessed, on the HMM, on the other hand, the access to location *n* is a function $f(n)$ of the location of *n*. The function $f(n)$ is assumed to be monotonically increasing function. To cater for the block transfer of data from higher memory levels to lower memory levels as is the case in practice, the cost to move a block of length *l*, of data from location [*x*-*l*, *x*] to location [*y*-*l*, *y*] is $f(x) + f(y) + l$. So in effect the cost to move a block of memory from one place in me of memory to another is determined by the cost to access the farthest memory location plus a cost of one for every other member in the block to be copied.

Definition [4]: HMM

One can make modifications to the memory hierarchy to make it appear as layers of memory in the hierarchy, where the access cost to any location in a layer is fixed according to the access cost function. As follows:

1. When the access cost function, f, is sub-linear monotonically increasing function. All memory locations from $1 + kf(n)$ to kn, for some suitably chosen integer constant $k \geq 1$, to have access cost $f(n)$. Similarly all memory locations from $1 + kf(f(n))$ to $kf(n)$ to have access cost $f(f(n))$, and so on. In this

modification it is possible to access any block of size $f(n)$ in the layer whose locations are from $1+kf(n)$ to kn and bring it to lower memory locations at a constant cost per element. Similarly, one can define the access cost of higher memory locations such as locations $1+kn$ to $kf^{-1}(n)$ to have access cost n.

2. When the access cost function is linear or super-linear monotonically increasing function. All memory locations from $1+\ kn/2$ to kn to have access cost$f(n)$. Similarly all memory locations from $1+\ kn/4$ to $kn/2$ to have access cost$f(f(n))$, and so on. Similarly, one can define the access cost of higher memory locations $1+kn$ to $2kn$ to be n.

3. We assume that only the first $f^i(n)-\ f^{i+1}(n)$ locations of the layer whose access cost is $f^i(n)$ holds data and the other locations are can be used as "buffer" area.

It is obvious from the definition of the multi-layer memory (MLM) machine that it acts exactly the same as the HMM machine with the exception that each layer of memory has a uniform memory access cost for all the addresses within that layer. Moreover, the access cost to an address on the MLM machine is not asymptotically less than the corresponding address in the HMM machine, and thus a lower bound on the HMM machine can serve as a lower bound on the corresponding MLM machine. Similarly, an upper bound on the MLM machine can serve as an upper bound on the HMM machine. However, if we modify the access cost for addresses on the various layers in the MLM machine from $f^i(n)$ to $f^{i+1}(n)$ then the MLM machine can simulate the corresponding HMM machine and a lower bound on this modified MLM machine can serve as a lower bound for an implementation on the corresponding HMM machine. We shall see an application of this idea later when we seek to obtain lower bounds on reading *n* memory locations.

Definition. We say that *n* items were ***read*** in ***t***time if each of these items was moved to location 1 during the time ***t***.

Lemma 1. The number of layers , $h(f,n)$, is a function that is asymptotically equal to the number of times we compose the function f on n values with itself before we get a value less than a constant of our choice that is greater than or equal 1, and it grows asymptotically not more than $O(lgn)$ for linear or sub-linear access cost functions.

Proof.

Given an input of size n in the first n locations, then according to the definition of the memory layers, one can move all the data to locations starting from$1+(k-1)n$ in one block transfer. Now consider moving the last item of the data until it reaches the first location of memory but without skipping any layer in between. Then we have a cost of access as follows:

$$f(n)+f(f(n))+f\left(f(f(n))\right)+\cdots+f^i(n)$$

For f sub-linear function $f^{i+1}(n)$is asymptitcally not more than $f^i(n)/2$. It follows, therefore, that the total access cost is:

$$f(n)+\frac{f(n)}{2}+\frac{f(n)}{4}+\cdots+\frac{f(n)}{n} = \frac{f(n)}{2^0}+\frac{f(n)}{2^1}+\frac{f(n)}{2^2}+\cdots+\frac{f(n)}{2^{lgn}}$$

The exponent of the denominator for each term is the number of layers we have moved the data item to so far. It follows, therefore, that the number of layers is $\leq 1+lgn$. ■

Corollary 2. The number of layers for access costs lgn, n^α for constant $0<\alpha\leq 1$, and n is asymptotically not more than clg^*n , $cloglogn$, for $0<\alpha<1$, and $clogn$, for $\alpha=1$, respectively, for some constant *c*> 0.

Proof.

For access cost lg the number of layers is lg^*n. This follows from the fact that access cost to the uppermost layer is lgn, and when a block of that size is copied into the next layer the cost at that layer to each element is $loglogn$, and so. Thus the cost of transfer is at most $O(1)$ per transfer of data from one layer to the next. The transfer of data to lower memory layers is stopped when the size of a block reaches a constant > 0. It follows that the number of layers is lg^*n which follows from the definition of lg^*n , and the recurrence $T(n)=T(logn)+1,$ and $T(1)=1,$ for the maximum number of different layers a data item is moved.

For access cost n^α, $0<\alpha<1$, we can move a block of size n^α from the *n* input to the next lower layer at a cost of n^α, which is *O*(1) per item moved. Similarly we can move a block of size $(n^\alpha)^\alpha=n^{\alpha^2}$ from the second memory layer to the immediately lower layer at a cost of *O*(1) per element moved, and so on until the cost of the move of the block is itself is constant. This can be described by the recurrence for the maximum number of different layers of:

$T(n)=T(n^\alpha)+1,$ and $T(1)=1$

In this case it must be the case that after ***i*** block moves n^{α^i} became a constant.

Now let $\beta=\frac{1}{\alpha}$, a constant >1. Thus $n^{\alpha^i}=n^{(\frac{1}{\beta})^i}=n^{\frac{1}{\beta^i}}=2^{lgn.\frac{1}{\beta^i}}=2^{\frac{lgn}{\beta^i}}$ must be a constant.

Therefore, $\frac{lgn}{\beta^i}=\frac{2^{llgn}}{2^{i\log_\beta 2}}=2^{\log_2\log_2 n-i\log_\beta 2}$ must be a constant. It follows, therefore, that the number of layers is $i=\frac{\log_2\log_2 n}{\log_\beta 2}=O(loglogn)$.

For access cost n the number of layers is ***i***, where is $n/2^i$ is constant, from which it follows that $i=logn.$ ■

Corollary 3.*n* items can be read in time $O(nlog^*n)$, $O(nloglogn)$, and $O(nlogn)$, on the multi-memory layers machine with access cost $logn,\ n^\alpha,$ and *n*, for constant 0$<\alpha<1$ respectively. ■

Corollary 4.*n* operations can be performed on the following structures in time $O(nlog^*n)$, $O(nloglogn)$, and $O(nlogn)$, on the multi-memory layers machine with access cost $logn,\ n^\alpha,$ and *n*, for constant 0$<\alpha<1$ respectively:

a) Push/Pop stack operations when starting from an empty stack.

b) Enqueue/Dequeue queue operations when starting from an empty queue.

While corollary 2 and 3 provide upper time bounds on reading or stack and queue operations, we need to obtain a lower bound on these operations to prove the asymptotic optimality of our solution. To this end we will make a minor modification to the multi-layer memory model introduced earlier as follows:

Definition. A specially blocked multi-layer memory machine is a multi-layer memory machine with the exception that:

a) Access to a location in a memory layer in the specially blocked machine costs $f^{i+1}(x)$ if the same location in the corresponding multilayer memory machine costs $f^{i}(x)$.

b) Data movement in the specially blocked machine can be made between immediately adjacent layers ***only***.

Next we will show that any sequence of data movement that requires time ***T*** in made in the multi-layered memory machine can be simulated in $O(T)$ time in the specially blocked MLM machine.

Proposition 5. Given a function f a sub-linear monotonically increasing continuous function, then $f^{i+1}(n) < \frac{f^{i}(n)}{c}$, where $n > n_0$, for constant $n_0 \geq 0$, and any constant $c > 0$.

Proof. Follows from the fact that $\lim_{n\to\infty} \frac{f^{i+1}(n)}{f^{i}(n)} = 0 < \frac{1}{c}$, since f is a sub-linear function. ■

Proposition 6. The size of any layer, ***i***, on MLM machine, is not less than the sum of size of all layers of memory from location 1 up to but not including the layer ***i***.

Proof.

Let's consider first the case when access cost of the MLM machine is linear or super-linear. In this case any layer with the access cost $f^{i}(n)$ contains memory locations from $1+\frac{kn}{2^{i+1}}$ to $\frac{kn}{2^{i}}$. The size of this layer is $\frac{kn}{2^{i}} - \left(1+\frac{kn}{2^{i+1}}\right) + 1 = \frac{kn}{2^{i+1}}$ which is the sum of all memory locations from 1 to $\frac{kn}{2^{i+1}}$.

The argument made earlier together with the proposition 5 take care of the case when the access cost of the MLM is sub-linear. ■

Lemma 7. A sequence of data movements in an algorithm on the MLM machine with access cost *f* that run in time *T* can be simulated in $O(T)$ time on a specially blocked MLM machine.

Proof.

Let's consider the block data movement from [*x*-*l*..*x*] to [*y*-*l* .. *y*] on the MLM machine, and without loss of generality, let's assume that $x > y$. Thus the cost of data transfer on the MLM machine is $< 2\,f^{i}(n) + l$, where $f^{i}(n)$ is the access cost to the layer which location **x**resides.

Now consider the corresponding specially blocked MLM machine. The access cost to the layer (say $\boldsymbol{d}$) in which location $\boldsymbol{x}$ resides is $f^{i+1}(n)$ by definition. The cost of transfer would be: $f^{i+2}(n) + l$ for moving the block to the immediately adjacent layer $d-1$. Some of the data may remain in the same layer d, and others may be moved to the buffer area of layer $d-1$. Again some of these data may be moved to the data area in layer $d-1$, while the others may be moved to the buffer area of layer $d-2$, and so on. The size of data moved in layer d is less than $f^{i}(n)$, and the rest were moved to layer $d-1$. The size of data moved in layer $d-1$ is no more than $f^{i+1}(n)$, and the rest would be moved to layer $d-2$. it follows that data transferred within layer d were moved only once, while data transferred within layer $d-1$ were moved twice; once from level d to level $d-1$ and once within level $d-1$. Similarly data transferred within layer $d-2$ were moved at most 3 times from layer d to layer $d-1$ to layer $d-2$ and then within layer $d-2$. Thus the overall cost of the data transfer is:

$$f^{i+1}(n) + l + \sum_{j=2}^{h(f,n)-i} j.f^{i+j} = f^{i+1}(n) + l + O(f^{i+1}) = O(l + f^{i+1}) = O(l + f^{i}) = O(T)$$

. ■

Corollary 8. A lower bound on the data movement for an algorithm implemented on a specially blocked MLM machine is also a lower bound on the MLM machine and the HMM machine with block transfer.

Proof. The corollary follows from lemma 7 and the fact that the access cost to a layer on the specially blocked MLM machine is asymptotically less than the access cost corresponding MLM machine. Moreover, the MLM machine was defined so that access cost to any locations is less than or equal to the same data location on the HMM with block transfer. ■

Corollary 9. Reading *n* data elements located in the first *n* locations of HMM with block transfer, MLM machine or specially blocked MLM machine can be performed in $\theta(n.h(f,n))$ time.

Proof.

The number of layers on specially blocked MLM machine is $h(f,n)$, and for reading the elements in the top layer where the access cost is $f^{2}(n)$, and the size of the layer is $n - f(n) < cn$, for constant $c > 0$, the number of layers each data element will pass through until it reaches location 1 is $h(f.n)$. Thus the overall cost is $\Omega(nh(f.n))$.

The data can be read by recursively transferring $f^{i}(n)$ data items from the layer whose access cost is $f^{i}(n)$ to the layer below it (i.e. one with access cost $f^{i+1}(n)$). Thus $T(n) = \frac{n}{f(n)} T(f(n)) + f(n).$ by definition of the number of layers this is equal to $O(nh(f.n))$. ■

Corollary 10.*n* items can be read in time $\theta(n\,log^{*}(n))$, $\theta(n.loglogn)$, and $\theta(n.logn)$ by MLM machine with access cost $f(n) = \log n$, $f(n) = n^{\alpha}$, constant $0 < \alpha < 1$, and $f(n) = n$respectively. ■

III. The Selection Algorithm

Step	
	SELECT(*i*, *n*){
1	Divide the *n* elements into groups of 5. Find the median of each 5-element group by rote.
2	Recursively SELECT the median *x* of the $\lceil \frac{n}{5} \rceil$ group medians to be the pivot.
3	Partition around the pivot *x*. Let *k* = rank(*x*).
4	**if** *i* = *k* **then return** *x* **elseif** *i* < **then** recursively SELECT the *i* th smallest element in the lower part **else** recursively SELECT the (*i*–*k*)th smallest element in the upper part
	}

Analysis of the algorithm

Access cost lgn.

The algorithm follows the step of the [1] technique in finding the *k*th element. In step 1, the whole data is read and the median of each group of 5 elements is found. Reading all the data costs $cnlg^*n$ time for some constant $c > 0$. Once a group is in lowest memory locations, the access cost is constant, and therefore finding the median of a constant number of elements is constant. A copy of the median can be added to a queue at a cost of $O(lg^*n)$ time per element. Thus the whole step can be performed in $O(nlg^*n)$ time.

Step 2 would recursively apply select method to the list of medians obtained during step 1. If the worst case running time of the method is T_n , then this step would require at most $T_{n/5}$ in the worst case.

Step 3 partitions the list of *n* elements around the partition found in step 2. Since reading or writing n elements can be performed in $O(lg^*n)$ time in the worst case, it follows that this step can be performed in $O(lg^*n)$ time in the worst case.

Step 4 recurs on either the lower or the upper partition obtained in step3. The maximum size of such a partition can be obtained as follows:

Each of the medians found in step 1 is greater than or equal to at least 3 of the 5 elements in that partition. It follows that the median of these elements, found in step 2, which by definition of median is already greater than or equal to half the list of medians found in step 1. Therefore, such a median must be greater than or equal to at least: $3\left(\frac{1}{2}\left(\frac{n}{5}\right)\right) = \frac{3n}{10}$ of the items of the original list. Thus the largest set to be searched recursively in step is of size at most $\frac{7n}{10}$. Therefore, it follows that the running time of the select method is at most

$T_n = cnlg^*n + T_{n/5} + T_{7n/10} = O(nlg^*n)$.

Given that reading *n* items requires $\Omega(nlg^*n)$, it follows that this bound is optimal for the given memory access cost.

Access cost: n^{α} *for constant* $0 < \alpha \leq 1$

In step 1, the whole data is read and the median of each group of 5 elements is found. Since there are at most $O(loglogn)$ layers, and copying a block from higher layer to a lower layer can be performed at a cost of $O(1)$ per element, it follows that this step can be performed in at most $O(nloglogn)$.

Step 2 would recursively apply select method to the list of medians obtained during step 1. If the worst case running time of the method is T_n , then this step would require at most $T_{n/5}$ in the worst case.

Step 3 can be performed in $O(loglogn)$ time in the worst case, following the same argument made for the case when the access cost is $logn$.

Step 4 as stated earlier recurs on either the lower or the upper partition obtained in step3. The argument for the largest of these partitions being of size not more than $\frac{7n}{10}$ remains valid irrespective of the access cost. Therefore, it follows that the running time of the select method is at most

$T_n = cnloglogn + T_{n/5} + T_{7n/10} = O(nloglogn)$.

Reading *n* on a HMM machine with the given access cost items requires $\Omega(nloglogn)$. It follows that this bound is optimal for the given memory access cost.

General case for the access cost $f(n) \leq n$.

Now we can generalize the argument made in obtaining the asymptotic time complexity of the selection algorithm as follows:

Step 1 can be performed in $O(nh(f,n))$ time by corollary 9.

Step 2 can be performed in time $T(n/5)$ where the running time for the select is $T(n)$.

The partitioning of the list in step 3 can be viewed as a process of reading or push/pop operations and can be performed in $O(nh(f,n))$ time by corollary 9.

Step 4 can be performed in the order of $T(3n/10)$ operations.

Thus the time recurrence is

$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{3n}{10}\right) + cnh(f,n)$, for some constant $c > 0$ and

$T(1) = 1$.

Solving this recurrence we get $T(n) = O(nh(f,n))$

Thus we have:

Theorem 11. Given a set of n numbers, the element of rank *k*, can be found in $\Theta(\max(f(n), nh(f,n))$ in the worst case on a HMM machine with access cost $f(n) \leq n$.

■

IV. Conclusion

In this paper we examined the performance of the selection of the *k*th element of *n* elements on a hierarchical memory machine with access cost that is a continuous monotonically increasing function. We proved lower bounds for the reading of data on such a machine in the general case, where f can be a sub-linear, linear or superliner function. We used this lower bound together with the upper bound obtained on reading to obtain optimal time bounds on the selection of the *k*th element problem, when the access cost to an address is not constant.

REFERENCES

[1] Blum, M.; Floyd, R. W.; Pratt, V. R.; Rivest, R. L.; Tarjan, R. E. *(August 1973). "Time bounds for selection" (PDF). Journal of Computer and System Sciences 7 (4): 448–461.*

[2] Atkinson, M.D., J.-R. Sack, N. Santoro, and T. Strothotte (1 October 1986). *"Min-max heaps and generalized priority queues." Programming techniques and Data structures. Comm. ACM, 29(10): 996–1000.*

[3] Hoare, C. A. R. (1961). *"Algorithm 65: Find". Comm. ACM 4 (7): 321–322.*

[4] Alok Aggarwal ; Ashok K. Chandra ; Marc Snir, *Hierarchical memory with block transfer,28th Annual Symposium on Foundations of Computer Science, 1987., PP.204 – 216*

[5] A. Aggarwal , B. Alpern , A. K. Chandra and M. Snir, *"A Model for Hierarchical Memory"*, Proc. of the 19th Annual ACM Symposium on the Theory of Computing, *pp. 305-314, 1987.*

[6] J. W. Hong and H. T. Kung, "I/O Complexity: The Red-Blue Pebble Game", Proc. of the 13th Ann. ACM Symposium on Theory of Computing, pp. 326-333, 1981

[7] A Schonhage, A Nonlinear Lower Bound for Random Access Machines Under Logarithmic Cost, 1984

[8] Alok Aggarwal, Jeffrey, S. Vitter, The input/output complexity of sorting and related problems, Comm. ACM 31(9): 1116-1127, 1988.

[9] C. A. R. Hoare, Quicksort, Computer Journal, 5:1015, 1962.

[10] D. E. Knuth, Selected Papers on Analysis of Algorithms, CLSI Publications, 2000.

[11] C. Martinez, Partial Quicksort, Partial quicksort, Proc. of the 6th ACM-SIAM Workshop on Algorithm Engineering and Experiments and the 1st ACM-SIAM Workshop on Analytic Algorithmics and Combinatorics , pp. 224-228, 2004