International Journal of
Science and Engineering Investigations
vol. 5, issue 56, September 2016
ISSN: 2251-8843

Quicksort Using Median of Medians as Pivot

Aviral Khattar
HCL Technologies
(aviral92@gmail.com)

Abstract- Median of Median is an algorithm for selecting the kth largest element in an unordered list, having worst case linear time complexity [1]. It is based on the Hoare's selection algorithm also called quickselect algorithm [6]. The Median of Median algorithm uses an asymptotically optimal approximate median selection algorithm to make an asymptotically optimal general search algorithm. It finds the approximate median in linear time which is then used as pivot in the quickselect algorithm. This approximate median can be used as pivot in Quicksort, giving an optimal sorting algorithm that has worst-case complexity O(n log n) [2].

This paper proposes variation in the above mentioned implementation technique for the median selection problem to find the Median of Medians and the value obtained is further used to guarantee a good pivot for the Quicksort algorithm. The result of the experiment performed show that the strategy proposed has worst-case complexity O(n log n) for Quicksort. Graphs were also plotted for the usual Quicksort function and the Quicksort function that uses the proposed median of median (PMOM) as pivot for relatively large values of size of array and results were compared. These confirm that proposed algorithm indeed has worst-case complexity O(n log n) for Quicksort.

Keywords- *Median of Median, Quicksort, Partition, Median Selection*

I. INTRODUCTION

Median is the middle value in a data set. Median selection is a problem that can be considered a special case of selecting the ith smallest element in an ordered set of n elements, when $i=\lceil n/2 \rceil$. An approach to solve this problem could be to sort the list and then choose the ith element. This could be using any sorting algorithm such as - Heapsort that has the worst case upper bound as O(n log n), Quicksort that has an expected running time O(n log n) though its running time is $O(n^2)$ in the worst case. Once the data values are sorted, it takes O(1) time to find the ith order statistics. Using an optimal sorting algorithm, the aforesaid approach gives complexity of O(n log n) as upper bound for selecting i^{th}order statistics [2].

The problem of selecting the i^{th}smallest element from an unsorted list of n elements has been solved in linear time by algorithms such as Quickselect [6], BFPRT (also called Median of Median Algorithm) [1], Introselect [7] and using Softheaps [8]. The ith order statistics selection implies- given a set of n unordered numbers we find the ith smallest number where (i is an integer between 1 and n). An interesting application of these selection algorithms is to select the median and then use it as pivot for balanced Quicksort. For instance, a good pivot is chosen using the BFPRT algorithm and used as pivot for partitioning in Quicksort resulting in worst-case O(n log n) run time rather than the usual $O(n^2)$.

II. STATE OF ART

The technique of finding the median of medians in [1] and strives to achieve the goal of finding the median of a given list in O(n) in the worst case.

The idea behind i^{th} order statistics selection is in Fig.1. It is a divide-and-conquer approach to solve the selection problem.

1. Any one element is selected as the pivot.
2. The list is divided into two sublists, the first containing all elements smaller than the pivot element and the second containing all elements greater than the pivot.
3. When searching for ith order statistics. Let the index of the pivot in this partitioned list be k. If element at position k=i, then pivot is returned.
4. If i<k, recurse on the sub-list of elements smaller than the pivot, looking for the ith smallest element.
5. If i>k, recurse on the sub-list of elements larger than the pivot, looking for the $(i-k-1)^{th}$ smallest element.

Figure 1. i^{th} Order Statistics Selection

In quicksort, if the chosen pivot is the largest or the smallest element in the list (in each iteration), it results as worst case performance of $O(n^2)$. When any random element is chosen as pivot, it results in expected linear time performance, but a worst case scenario of quadratic time is possible [2].

An important part of the algorithm in Fig. 1 is the choice of pivot element. To guarantee the linear running time of O(n), the strategy used for choosing the pivot must guarantee that the chosen pivot will partition the list into two sublists of relatively comparable size. Median of the values in the list could be the optimal choice. Thus, if median can be found in linear time then it is possible to have an optimal solution to the general selection problem. The Median-of-Medians algorithm chooses its pivot in the following intelligent manner [1] [2].

1. Divide the entire list into sublists each of length five. (The last sublist may have a length less than five.)
2. Sort each of the above sublists and determine their median.
3. Use the Median-of-Medians algorithm to recursively determine the median of the set of all medians from step 2.
4. Use the median from step 3 as the pivot to divide the list for finding the ith order statistics as this guarantees a good split.

Figure 2. Median-of-Medians for i^{th} order statistics

To visualize why the algorithm in Fig. 2 [1-2] works consider Fig. 3, 4 and 5.

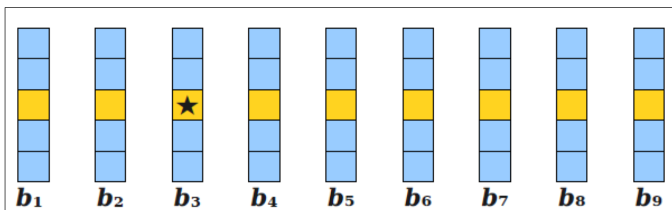

Figure 3.

Let b_1 through b_9 be the groups of 5 elements each and their medians after sorting be in at yellow positions in Fig.3[3]. Sorting within the groups, all elements above the median are greater than the median and the ones below it are less than the median as shown in Fig. 4[3]. The Median-of-Medians is marked as a ★ in the Fig. 3, 4 and 5.

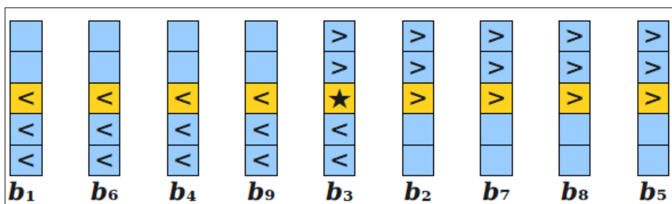

Figure 4.

Thereafter the medians in yellow positions in groups b_1 through b_9 are sorted and the entire groups are repositioned accordingly as shown in Fig 4.

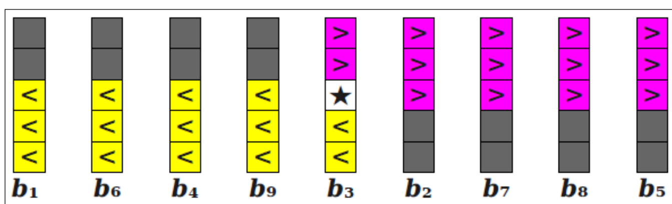

Figure 5.

Logically interpreting the arrangement in Fig. 4 leads to Fig. 5[3] where the elements in the yellow positions to the left of the median of medians (MOM), are certainly less than the MOM and the elements in the pink positions to the right of the MOM are certainly greater than the MOM. Nothing can clearly be said about the grey elements.

So, MOM (★) is larger than 3/5 of the elements from (roughly) the first half of the blocks. And thus, ★ larger than about 3/10 of the total elements. On the same lines ★ is smaller than about 3/10 of the total elements. So the scheme guarantees a 30:70 split [2].

1. Split the input into blocks of size 5 in time $\Theta(n)$.
2. Compute the median of each block non-recursively.
3. Takes time $\Theta(n)$, since there are about n/5 blocks.
4. Recursively invoke the algorithm on this list of n/5 blocks to get a pivot.
5. Partition using that pivot in time $\Theta(n)$.
6. Make up to one recursive call on an input of size at most (n - 3n/10) = 7n/10 elements.

Figure 6. Median-of-Medians Algorithm

Analyzing the algorithm in Fig. 6[2], mathematically, there are $\lceil n/5 \rceil$ groups, including the leftover elements. Half of these, that is, $\lceil \lceil n/5 \rceil /2 \rceil$ groups have 3 elements greater than or equal to the MOM(★). The group containing MOM and the last group (that may or may not have 5 elements) are special cases. Hence each of the $\lceil \lceil n/5 \rceil /2 \rceil - 2$ groups; except the last and the one containing the median; contributes three elements greater than MOM(★).

Let X be the number of elements greater than MOM(★), then,

$$X \geqslant 3(\lceil \lceil n/5 \rceil /2 \rceil - 2)$$

$$\geqslant 3(n/10 - 2)$$

$$= 3n/10 - 6$$

Our recursive call can be on a sub array of size

$$n\text{-}X \leqslant n\text{-}(3n/10\text{-}6)$$

$$\leqslant 7n/10 + 6$$

Thus the recurrence relation for the above algorithm is:

$$T(n) \leqslant c \text{ if } n \leqslant 100 \qquad (1)$$

$$T(n) \leqslant T(\lceil n/5 \rceil) + T(\lceil 7n/10 + 6 \rceil) + cn \text{ otherwise}$$

The solution to this can be proved to be O(n), [2].

The above linear-time selection algorithm runs in time O(n), but there is a huge constant factor hidden in it. This has basically two reasons: (i) work done by each call is large that is, finding the median of each block requires nontrivial work; (ii) problem size decays slowly across levels; each layer is

roughly only 10% smaller than its predecessor. The first non-trivial lower bound for the problem was presented, in 1973, by Blum et al.[1] using an adversary argument. This result was improved in 1976 by Schnhage, Paterson, and Pippinger [4] who presented an algorithm that uses only 3n+o(n) comparisons. This remained the best algorithm for almost 20 years, until Dor and Zwick [5] reduced the number of comparisons a bit further to 2.95n+o(n).

The main objective of the proposed work is to solve the problem in O(n) behavior without a large constant factor. The proposed paper presents a variant of MOM algorithm hereafter called the **PMOM** Algorithm by considering more sophisticated approaches to this problem in this paper.

III. Proposed Algorithm

The proposed variation in this paper, aims at reducing the time to find the Median of Medians, and then the value obtained is further used to guarantee a good pivot for the Quicksort algorithm where an important part of the algorithm the choice of pivot for partition. To guarantee the linear running time of O(n), the strategy used for choosing the pivot must guarantee that the chosen pivot will partition the list into two sublists of relatively comparable size. Thus, Median of the values in the list could be the optimal choice. The proposed algorithm finds the median for each group of 5 elements, and then moves these medians into a block of values atthe beginning of the array. Putting the medians in a block simplifies further operations and increases their locality of reference hence improving the constant factor. Thereafter, the proposed algorithm uses select it to identify the median of these median-of-5's.

Functions used in the proposed approach are as in Fig. 7 through Fig. 9. The select algorithm in Fig. 7 permutes the array to place the kth largest value in a[k]. It is provably worst-case linear. As in MOM, instead of relying on fate, the proposed algorithm seeks to find a good pivot, a[MOMIdx], which always makes each partition at least 3/10 the size of the array. To do this, the proposed algorithm first divides the array up into small groups of 5 elements each. It then uses the median5 function described in Fig. 8 to find the median for each, and then moves these medians into a block of values at the beginning of the array. As already stated putting the medians in a block simplifies further operations and increases their locality of reference hence improving the constant factor. Having done this, the proposed algorithm uses select itself to identify the median of these median-of-5's, and in the proposed paper this will be the pivot in Quicksort.

```
void select(int a[ ],  int size,  int k)

{

if (size < 5)   //insertion sort

        {

        for ( i=0; i<size; i++)
        for ( j=i+1; j<size; j++)
        if (a[j] < a[i])
                swap(&a[i],  &a[j]);
        return;

        }
else
{
intgroupNum = 0;

int* group = a;

for(;groupNum*5<=size-5;group+=5,groupNum++)

{
 // gets medians of all groups to beginning of array
   swap(group[median5(group)],  a[groupNum]);
}

intnumMedians = size/5;  // total number of groups or medians

intMOMIdx = numMedians/2;// index of median of medians

select(a,  numMedians,  MOMIdx);

intnewMOMIdx = partition(a,  size,  MOMIdx);

if (k != newMOMIdx)
{
         if (k <newMOMIdx)
        {
              select(a,  newMOMIdx+1,  k);
        }
        else/* if (k >newMOMIdx) */
 {

select(a+newMOMIdx+1,size-newMOMIdx-1,k-newMOMIdx -1);

}
}
}
}
```

Figure 7. Functions used in the proposed approach

```
int median5(T* a)

{
// Load array values for 5 elements in CPU registers
        registerint a0 = a[0];
        registerint a1 = a[1];
        registerint a2 = a[2];
        registerint a3 = a[3];
        registerint a4 = a[4];

/*Perform insertion sort on the five registers, Gives median
value in a2 */

        if (a1 < a0)
                swap(a0,  a1);
        if (a2 < a0)
                swap(a0,  a2);
        if (a3 < a0)
                swap(a0,  a3);
        if (a4 < a0)
                swap(a0,  a4);
        if (a2 < a1)
                swap(a1,  a2);
        if (a3 < a1)
                swap(a1,  a3);
        if (a4 < a1)
                swap(a1,  a4);
        if (a3 < a2)
                swap(a2,  a3);
        if (a4 < a2)
                swap(a2,  a4);
/* Find and return index of median of the 5 elements */
        if (a2 == a[0])
                return 0;
        if (a2 == a[1])
                return 1;
        if (a2 == a[2])
                return 2;
        if (a2 == a[3])
                return 3;
        if (a2 == a[4])
                return 4;}
```

Figure 8. Fast Median-of-5 Function

The function in Fig. 8 locates the median of 5 values. The function loads the five array values into five register variables. This function is called several times. Ideally, if this function is written in assembly language the constant factors can be further improved.

The partition algorithm in Fig. 9 works by positioning two pointers at the beginning of the array, the load pointer and the store pointer. The load pointer advances through the array, finding all values less than the pivot value and swapping them into the initial segment of the array.

```
int partition(T a[],  int size,  int pivot)
 {
        intpivotValue = a[pivot];
        swap(a[pivot],  a[size-1]);
        intstorePos = 0;
        for(intloadPos=0; loadPos< size-1; loadPos++)
        {
                if (a[loadPos] <pivotValue)
                {
                        swap(a[loadPos],  a[storePos]);
                        storePos++;
                }
        }
        swap(a[storePos],  a[size-1]);
        returnstorePos; }
```

Figure 9.

The proposed algorithm uses the proposed Median of Median algorithm (PMOM) in Fig. 6[2] to find a good pivot in Quicksort. Hence, when the Quicksort as in Chapter 7[2] uses PMOM to find pivot for very large array sizes is able to guarantee a worst case running time of O(n log n).

IV. OBSERVATIONS

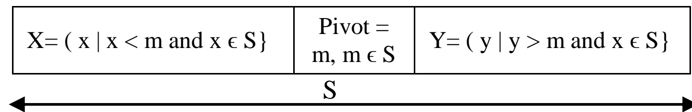

Figure 10.

Even when any other element from S is chosen as the pivot, the algorithm works correctly, but looking at it closely, it can be found that the worst-case running time depends on |X|: the size of the set X and |Y| : the size of the set Y. Let T(|S|) denote the worst-case running time of the algorithm on the list S, then

$$T(|S|) = T(|S|/5) + O(|S|) + \max\{T(|X|), T(|Y|)\} \quad (2)$$

As discussed in the section above, (refer Equation 1) that if m the pivot is the "median of medians", both |X| and |Y| are at most 3|S|/4.

$$T(n)=T(n/5) + O(n) + T(3n/4) \quad (3)$$

From (1)

$$T(n) = O(n) + T(n/5) +T(7n/10)$$

Solving the above using substitution, Guess $T(n) < C*n$

$\Rightarrow$ $T(n)= p*n + T(n/5) + T(7n/10)$
$\Rightarrow$ $C*n >= T(n/5) +T(7n/10) + p*n$
$\Rightarrow$ $C>= C*n/5+ C*7*n/10 + p*n$
$\Rightarrow$ $C>= 9*C/10 +p$
$\Rightarrow$ $C/10 >= p$
$\Rightarrow$ $C>= 10*p$

Since it is possible to find such a constant p it implies

$$T(n) = O(n)$$

Using a similar logic it can be proved that T(n)=O(n) for (3).

The experiment was done using three types of lists say L1, L2, L3 where a list of type L1 has all elements arranged in descending order, list of type L2 has all elements arranged in ascending order and list of type L3 has elements in random order.

Three sets of lists of type L1, L2, L3 were used as input for the proposed experiment. Exactly the same sets of lists of type L1, L2, L3 were given as input to

(i) Quicksort algorithm using Hoare's partition [2] to find pivot and sort the list. Hoare's partition uses the first element of the list as the pivot

(ii) Quicksort algorithm using PMOM to find the pivot for partition and sort the lists.

The time taken by the two variants for Quicksort discussed above was measured by the system clock in terms difference between the start time and finish time for the program while no other programs were running on the system. Let the difference between the two time values be x, then it is termed as x ticks.

Readings of execution time were recorded for the two variants of Quicksort for lists of type L1, L2 and L3 and graphs were plotted to study the variation in execution time.

Case 1: **List of type L1**: Elements of set S are arranged in descending order. The data table for the readings for Case 1 is depicted in Fig. 11. The results for Quicksort using Hoare's partition are plotted as a red line and the results of Quicksort using PMOM to find the pivot for partition are plotted as a blue line graph in Fig. 12. The blue line graph shows an O(n log n) curve. The red line graph shows an O(n^2) curve which is the worst case for Quick sort.

Number of Elements	PMOM	Hoare
500	0	1
1000	1	2
2000	2	16
3000	4	32
4000	6	48
5000	8	63
10000	18	190
20000	37	703
25000	45	1121
30000	51	1551
40000	68	2749
45000	73	3472
47000	75	3809

Figure 11.

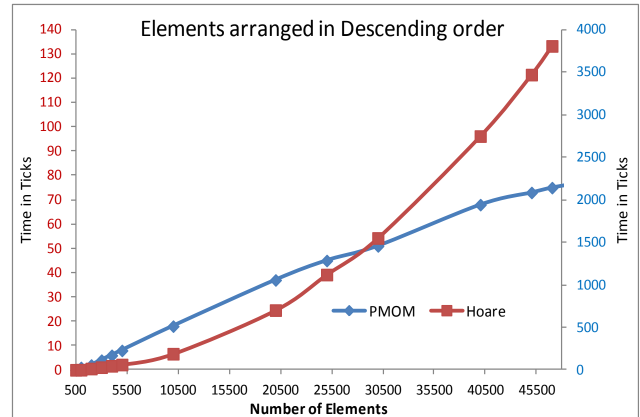

Figure 12.

Case 2: List of type L2: Elements of set S are arranged in ascending order. The data table for the readings for Case 2 is depicted in Fig. 13. The blue line in Fig. 14 shows results of Quicksort using PMOM while the red line graph shows Quicksort using Hoare's partition. The blue line graph shows an O(n log n) curve. The red line graph shows an O(n^2) curve which is the worst case for Quicksort.

Number of Elements	PMOM	Hoare
1000	1	4
2000	2	13
3000	3	31
4000	4	43
5000	8	47
10000	16	153
20000	31	677
25000	39	968
30000	43	1433
40000	58	2637
45000	61	3334
47000	63	3380

Figure 13.

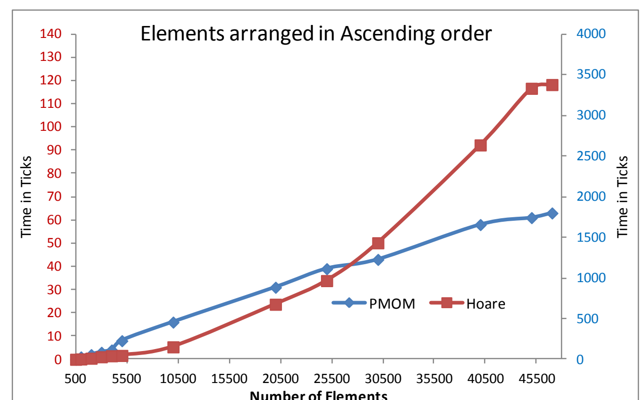

Figure 14.

Case 3: **List of type L3**: Elements of set S are in random order. The data table for the readings for Case 3 is depicted in Fig. 15 and the blue line in Fig. 16 shows results of Quicksort using PMOM while the red line graph shows Quicksort using Hoare's partition. Both the red and the blue line graph shows an O(n log n) curves.

Number of Elements	PMOM	Hoare
500	0	0
1000	0	1
2000	2	3
3000	3	4
4000	3	5
5000	8	8
10000	16	19
20000	30	43
25000	38	46
30000	42	55
40000	57	62
45000	62	65
47000	63	71
50000	77	76
100000	122	123

Figure 15.

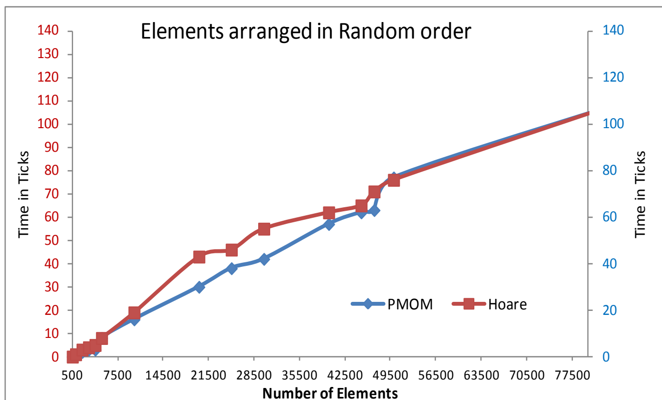

Figure 16.

IV. CONCLUSION

The results of the experiment done in this paper prove when the PMOM algorithm is used to find the pivot; that is, when the "median of medians" is chosen as the pivotal element by the partition in quick sort it helps to limit the worst-case of the problem. It is evident from the graphs in the previous sections that, Quicksort using PMOM gives a time complexity of O(n log n) even in the worst case where as Quicksort using Hoare's partition runs in O(n^2) time in for the worst-case inputs. When the elements are in random order the Quicksort using Hoare's partition and Quicksort using PMOM both show a similar behavior giving a time complexity of O(n log n).

REFERENCES

[1] Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. Time bounds for selection. Journal of Computer and System Sciences, 7:448{461, 1973.

[2] Introduction to Algorithms, Second Edition, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, The MIT Press, ISBN 0-07-013151-1 (McGraw-Hill)

[3] http://web.stanford.edu/class/archive/cs/cs161/cs161.1138 /lectures/08/Small08.pdf

[4] A. Schnhage, M. Paterson, and N. Pippenger, Finding the median, Journal of Computer and System Sciences, 13:184{199, 1976.

[5] DoritDor and Uri Zwick, Selecting the median, In Proceedings of 6th SODA, pages 88{97, 1995.Journal version in SIAM Journal on Computing, 28:1722{1758, 1999.

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

[7] Musser, David R. (1997).Introspective Sorting and Selection Algorithms, Software: Practice and Experience, Wiley. 27 (8): 983–993.

[8] Chazelle, B. 2000,The soft heap: an approximate priority queue with optimal error rate, J. ACM 47, 6 (Nov. 2000), 1012-1027.

Aviral Khattar was born in 1992 in Delhi, India. He has done B.Tech. in Computer Science and Engineering from NIIT University, Neemrana, Rajasthan, India from 2010-2014.

His internships include designing and developing a Flexible Database Management System which can be hosted in Cloud at Sasken Communication Technologies, IIT Madras Research Park for which he was awarded Certificate of Appreciation for outstanding R&D Work. He successfully completed Summer Internship Program (SIP) at Computer Science & Automation Department (CSA) at IISc, Bangalore under the guidance of Prof. N. Viswanadham.

Currently he is working at HCL Technologies as a Senior Software Developer for testing tools (Functional Testing and Web GUI testing) – Rational Functional Tester (RFT) and Rational Test Workbench WebUI (RTWW).