ALGORITHM DESIGN AND ANALYSIS REPORT

Team Members:

Nikitha Reddy Pullalarevu-A04862493

Apoorva Bapat-A04828475

Group1

Median Finding, Order Statistics and Quick Sort

Introduction:

Order Statistics: The *i*th order statistic of a set of n elements is the *i*th smallest element.

Median Finding: A median, is the "halfway point" of the set(array).

QuickSort: It uses divide and conquer algorithm to sort the given set of elements from A[1……..n].

Project Overview

- Implementation of Median of Median with groups of 3,5, and 7.
- Implementation of Randomized Median finding algorithm.
- Comparing Median of Medians and Randomized Median Finding algorithm.
- Implementation of Quick Sort, where pivot is chosen from previous algorithms.
- Implementation Randomized Quicksort.
- Implementation of simplified Quicksort.
- Compare the performance of all implementations.

Pseudocode

Randomized Median:

```
RANDOMIZED-SELECT(A, p, r, i)
1  if p == r
2       return A[p]
3  q = RANDOMIZED-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k            // the pivot value is the answer
6       return A[q]
7  elseif i < k
8       return RANDOMIZED-SELECT(A, p, q − 1, i)
9  else return RANDOMIZED-SELECT(A, q + 1, r, i − k)
```

Median Of Medians:

1. Divide the n elements of the input array into $\lfloor n/5 \rfloor$ groups of 5 elements each and at most one group made up of the remaining n mod 5 elements.
2. Find the median of each of the $\lceil n/5 \rceil$ groups by first insertion-sorting the elements of each group (of which there are at most 5) and then picking the median from the sorted list of group elements.
3. Use SELECT recursively to find the median x of the $\lceil n/5 \rceil$ medians found in step 2. (If there are an even number of medians, then by our convention, x is the lower median.)
4. Partition the input array around the median-of-medians x using the modified version of PARTITION. Let k be one more than the number of elements on the low side of the partition, so that x is the kth smallest element and there are $n-k$ elements on the high side of the partition.
5. If $i = k$, then return x. Otherwise, use SELECT recursively to find the ith smallest element on the low side if $i < k$, or the $(i-k)$th smallest element on the high side if $i > k$.

QuickSort:

```
QUICKSORT(A, p, r)
1  if p < r
2       q = PARTITION(A, p, r)
3       QUICKSORT(A, p, q − 1)
4       QUICKSORT(A, q + 1, r)
```

Partition:

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4       if A[j] ≤ x
5            i = i + 1
6            exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

Randomised Partition:

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  exchange A[r] with A[i]
3  return PARTITION(A, p, r)
```

Working Of Code:

- Compile main.cpp file and run the program to get the below ouput screen.

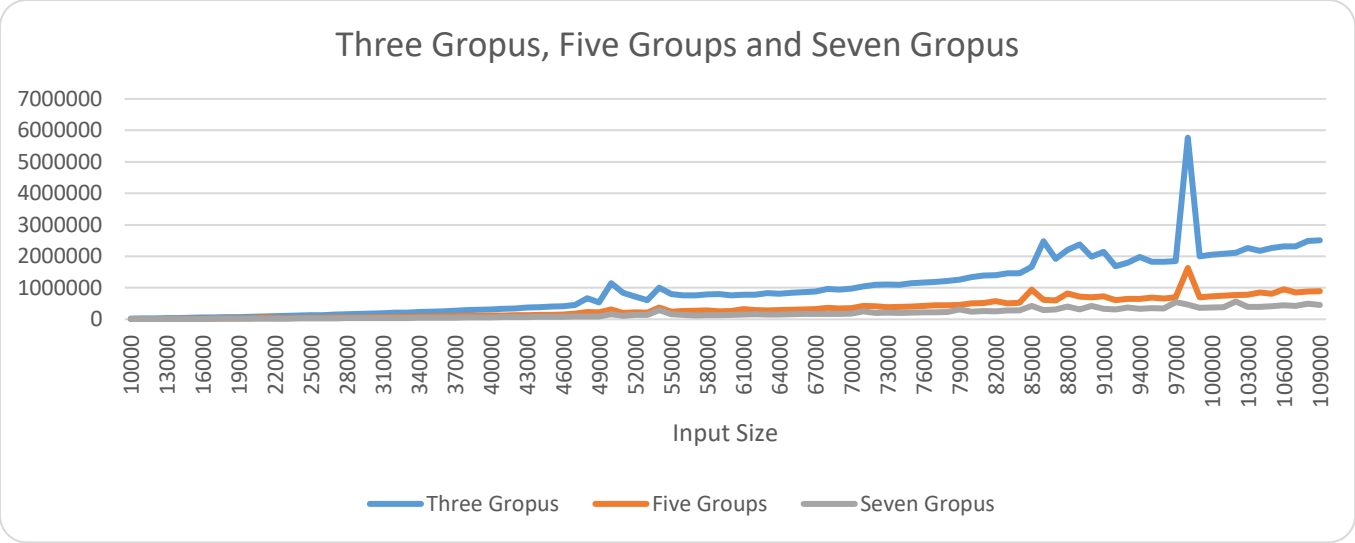
```
Select any options below to perform test
Enter 1--> to compare all quicksort
Enter 2-->to compare Median of Median And Random Median
Enter 3--> to compare Three types of median of medians
Enter 0--> to quit
Enter Your Choice:
```

- Select any of the three options or zero to quit.
- Ouput Files will be produced for each test input the values in excel to get graph analysis.

Results:

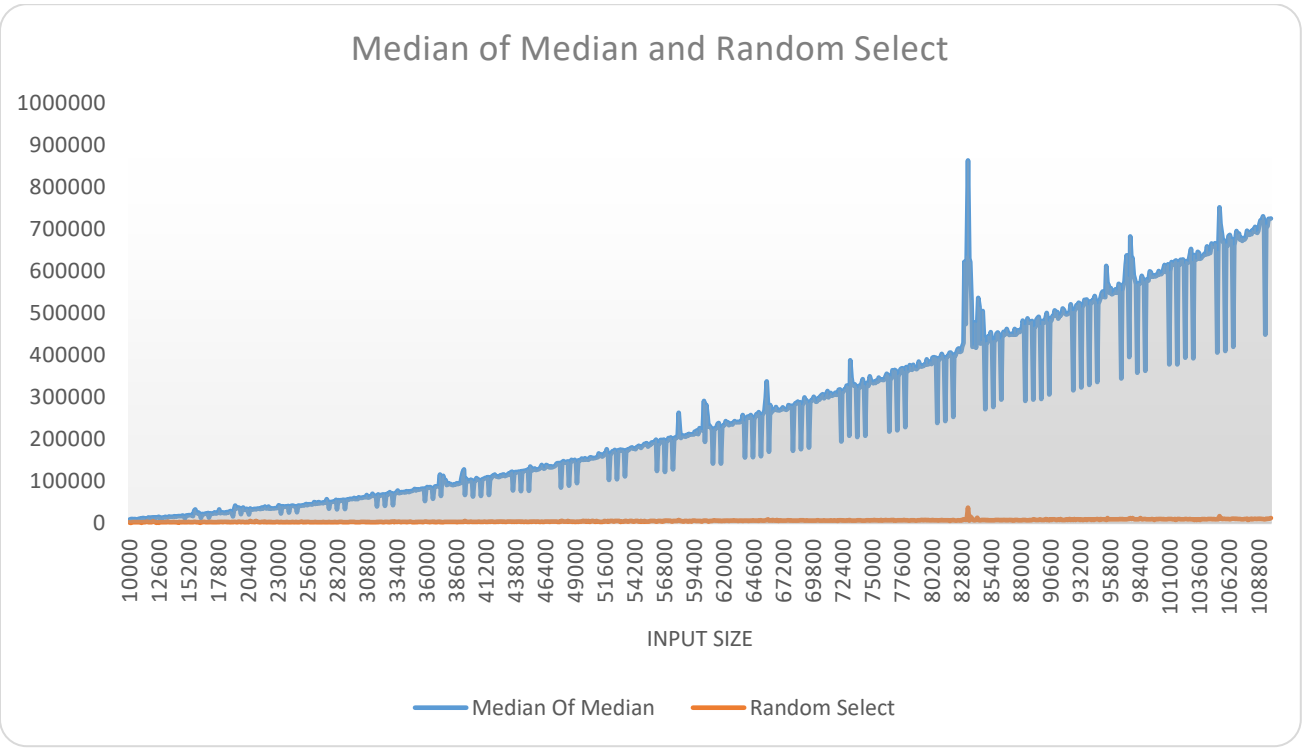
Graph: y – axis : time in micro-seconds x-axis: input size.

Comparison of Median Finding Using different Groups:

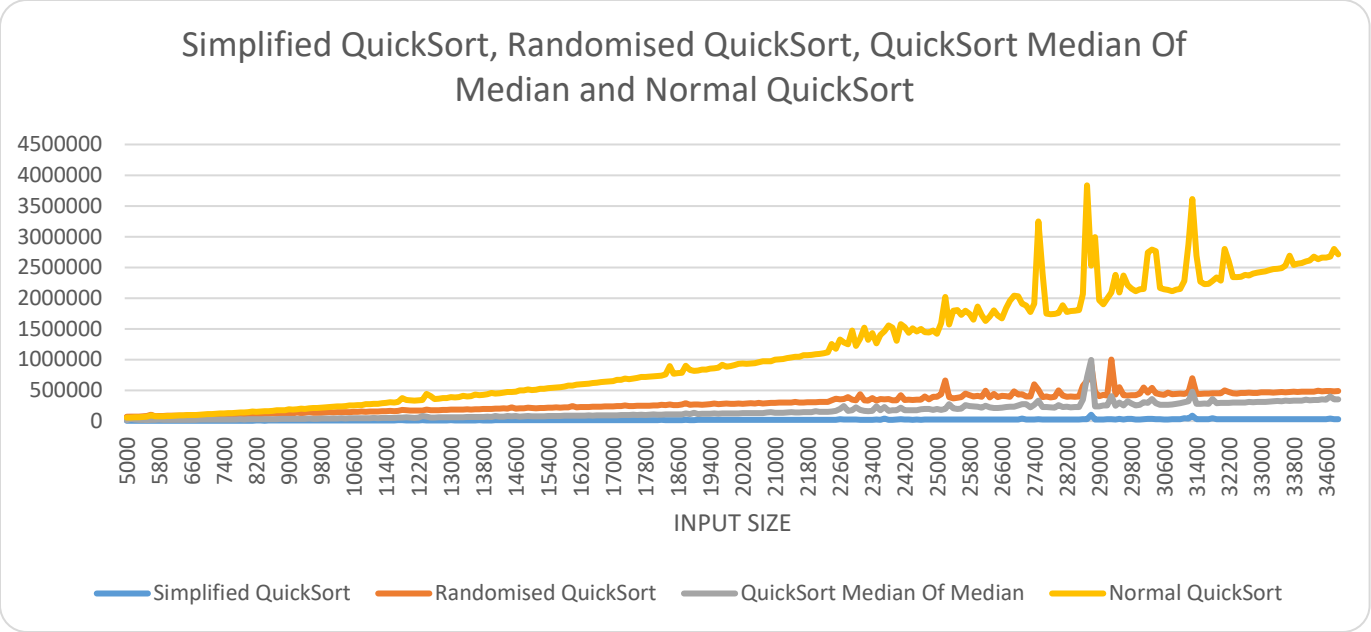


Seven Groups performed well.

Comparison of Median Finding and Random Median Finding:



Comparison of All Quicksort Functions:



Analysis:

- Simplified Quick Sort Performance was highest.
- Quick Sort using median of medians Performance was below Simplified Quick Sort.
- Randomized Quicksort Performance was below Quick Sort using median.
- Normal Quicksort Performance was below all other implementations.

References:

1. Pseudocode: Introduction to algorithms by cormen.
2. Random Numbers generation: https://en.cppreference.com/w/cpp/numeric/random/uniform_int_distribution
3. Execution Time of a function: https://www.geeksforgeeks.org/measure-execution-time-function-cpp/