CSE 2046 ANALYSIS OF ALGORITHM
HOMEWORK 2

COMPARING DIFFERENT ALGORITHMS TO FIND MEDIAN

Student Name	**Student Number**
Ahmet Önkol	150117018
Ahmet Lekesiz	150118509

Contents

Abstract

In this project, our main goal is designing an experiment to compare various algorithms for finding median depending on their performance. Also we are supposed to compare theoretical and experimental findings.

Theoretical Complexities

Insertion Sort

Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time.

The best case input is an array that is already sorted. In this case insertion sort has a linear running time **(i.e., O(*n*)).**

The worst case for insertion sort is when corresponding data is sorted in reversed order. For this case running time is **quadratic (i.e., O(n^2)).**

The average case is also **quadratic (i.e., O(n^2))**, which makes insertion sort impractical for sorting large arrays. For sorting small size of arrays insertion sort is one of the fastest algorithms but on the other hand for large size of arrays, it is not as fast as smallest ones.[1]

Merge Sort

Merge sort is a Divide and Conquer algorithm. It divides the input array in two halves, calles itself for the two halves and then merges the two sorted halves.

Time complexity of Merge sort is **O(nLogn) in all 3 cases(worst, average and best)** as merge sort always divides the array into two halves and takes linear time to merge two halves.[2]

Max-heap

Max-heap is a complete binary tree in which the value in each internal node is greater than or equal to the values in the children of that node.

Time complexity for all 3 cases(worst , average and best) is O(nlogn) since time complexity of heapify is O(Logn) and insert heap is O(n). So overall time complexity of Max-heap is O(nLogn).

Quick Select Pivot First

Quickselect is a selection algorithm to find the *k*th smallest element in an unordered list.It uses the same overall approach as quicksort, choosing one element as a pivot and partitioning the data in two based on the pivot, accordingly as less than or greater than the pivot.

For best and average cases when **the first element is picked as pivot**, time complexity will be **O(nlogn)** since it is not sorted either in a normal way or reversed.

The worst case occurs when the partition process always picks the greatest or smallest element as pivot. If we consider the strategy where **the first element is always picked as pivot**, the worst case would occur when the array is already sorted in increasing or decreasing order. Thus time complexity will be **$O(n^2)$).**

Quick Select Median of Three

Best case time complexity for the **Median of Three quick select** is given in the case when there is equal partitioning of the array about the pivot. It is given by the relation $T(n) = 2T(n/2) + n$ which gives the result **O(n log n).**

The average case time complexity of the **Median of Three** quick select is the same as that of a standard quick select as randomized quick select only helps in preventing the worst case. It is equal to **O(n log n).**

For the worst case , on a very contrived data set the **Median of Three** would result in a pivot that is the second minimum item of the remainder to be sorted. Time complexity would be **$O(n^2)$).**

Quick Select Median of Medians

Quick Select Median of medians finds an approximate median in linear time only, which is limited but an additional overhead for quickselect. When this approximate median is used as an improved pivot, the worst-case complexity of quickselect reduces significantly from quadratic to *linear*, which is also the asymptotically optimal worst-case complexity of any selection algorithm. Thus, **Time complexity for all 3 cases(worst , average and best) is O(n).**

Experimental Setup

We take inputs with size 10,100,1000 and 10000 with different characteristics which are sorted, sorted in reverse order and average(random generated inputs). For finding average characteristics we generated 50 random input sets for every given size and added them to input files. We named input files as a combination of their characteristics and size such as sorted10, reversed1000, average100 etc. With all those steps, we can clearly observe the difference between algorithms for various input sizes and characteristics, and be able to see the growth of every algorithm depending on the corresponding size and it's complexity.

Deciding on Metrics for Complexity Measurements

We are intended to use a physical unit of time for metrics since we have few functions that are not part of the basic operation and still used by the program. But when we are calculating system time that is spent by corresponding algorithms, the values vary abnormally since we used loops, needed declarations and function callers that are part of our main class and not part of our algorithms.Those things would create complexity for algorithms unnecessarily. Therefore we decided to select the basic operations of each algorithm and count them to make comparisons. So we used a counter for complexity measurement.

Results

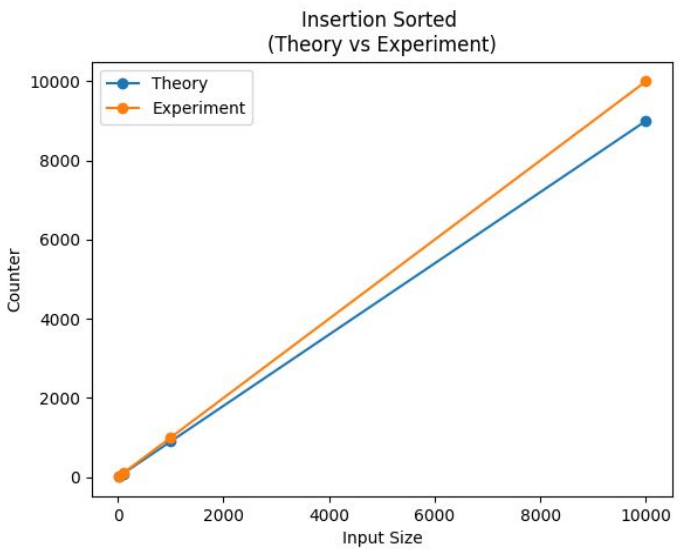
Comparing Findings with Theoretical Values for every Algorithm

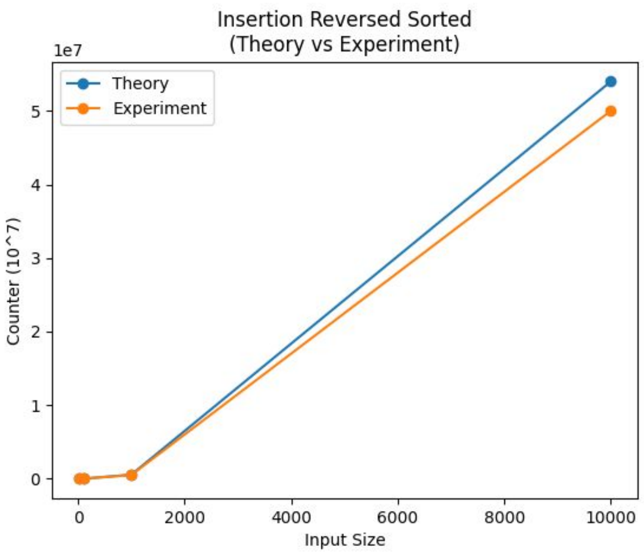
Insertion Sort

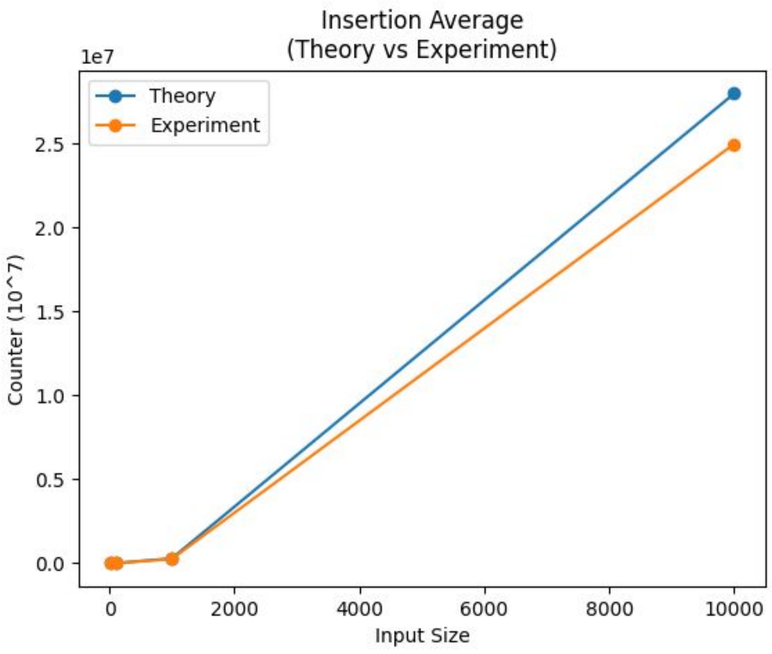
As we have mentioned above, **in the best case** ,which data is sorted, time complexity will be **O(n).** In our findings when input is in sorted order and size is increased by n times, counter values are also increased n times as it is expected since time complexity is **O(n)**. (9 // 99 // 999 // 9999) See in table.

In theory, **the worst case** will occur when data is sorted in reversed order and time complexity would be **quadratic (i.e., O(n^2))**. In our findings when input is in reversed order and size is increased by n times, counter values will increase n^2 times, so findings meet with theoretical values. (54 // 5049 // 500499 // 50004999) See in table.

For **the average case**, time complexity is the same with the worst case which is **O(n^2)** in theory.In our findings when size is increased by n times, counter values are about to increase n^2 times, so findings meet with theoretical values. (28 //2555 //250385// 24990755)
See in table.

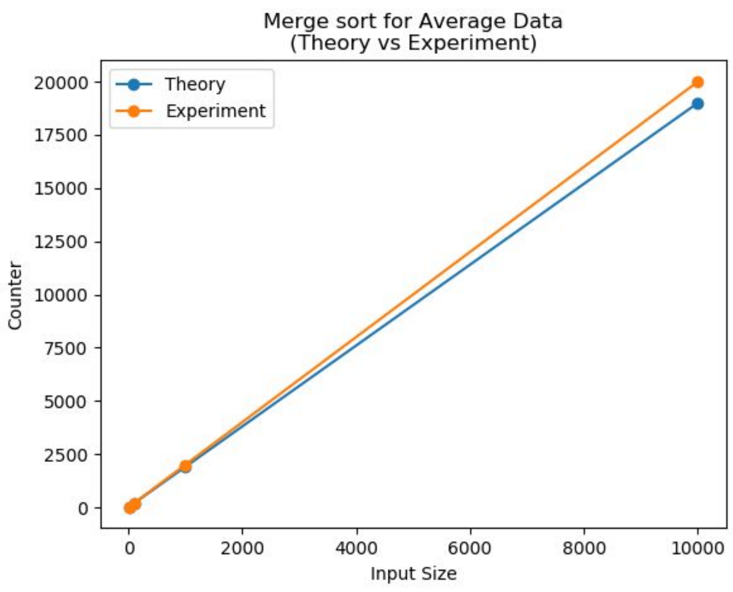

Insertion Sorted
(Theory vs Experiment)
Theory
Experiment
Counter
Input Size
0
2000
4000
6000
8000
10000

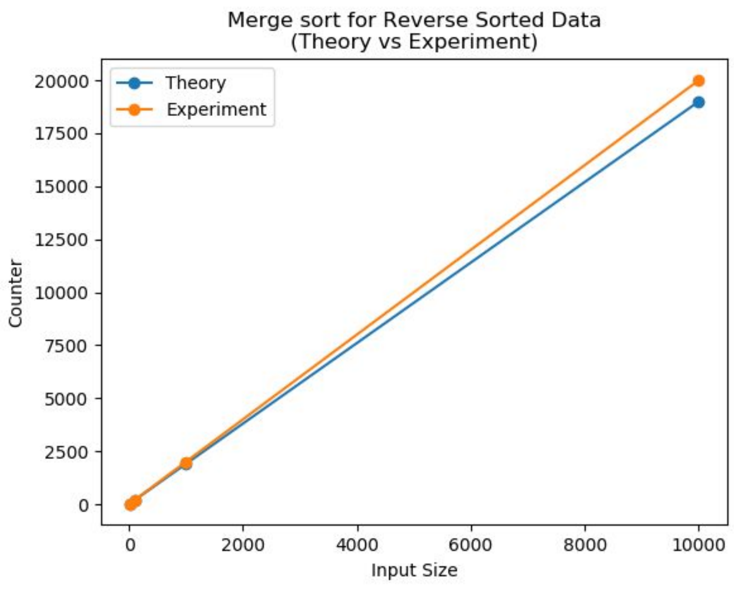

Insertion Reversed Sorted
(Theory vs Experiment)
1e7
Theory
Experiment
Counter (10^7)
Input Size
0
1
2
3
4
5
2000
4000
6000
8000
10000

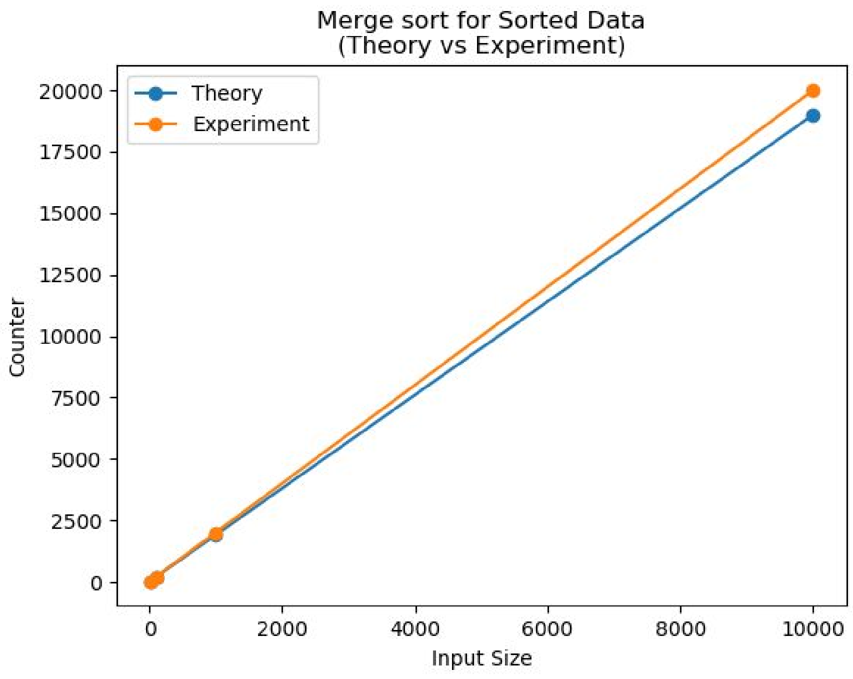

Insertion Average
(Theory vs Experiment)
1e7
Theory
Experiment
Counter (10^7)
Input Size
0.0
0.5
1.0
1.5
2.0
2.5
0
2000
4000
6000
8000
10000

Merge Sort

Time complexity of Merge sort in theory is **O(nLogn) in all 3 cases(worst, average and best)** since it divides the array into two halves and takes linear time to merge two halves as we mentioned above. In experimental findings since we are using 10 and it's multiples for any kind of input, counter values will be the same because of the reason that 10log10 equals 10.For instance when input size is 1000 and 100,counter values in findings are 1999 and 199 and their proportion to each other is **10.04** .In theory values are 1000 and 100 when size is 1000 and 100.And their proportion to each other would be **10**. So we can say that findings meet with theoretical values. See in table.

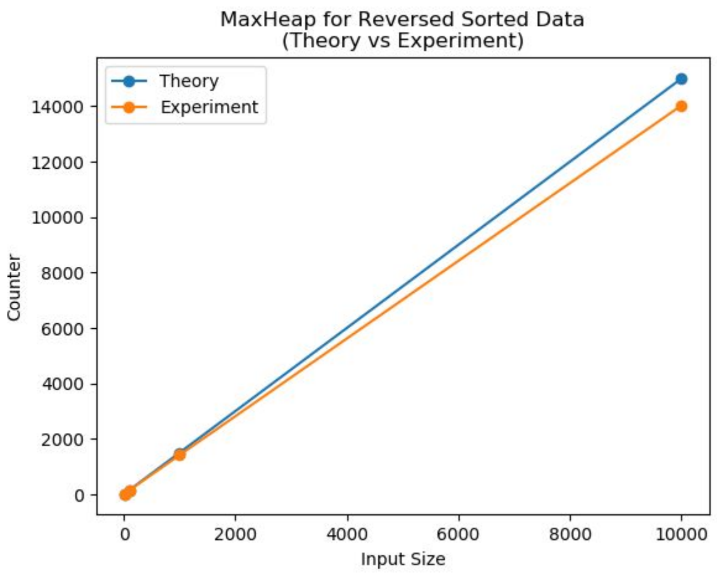


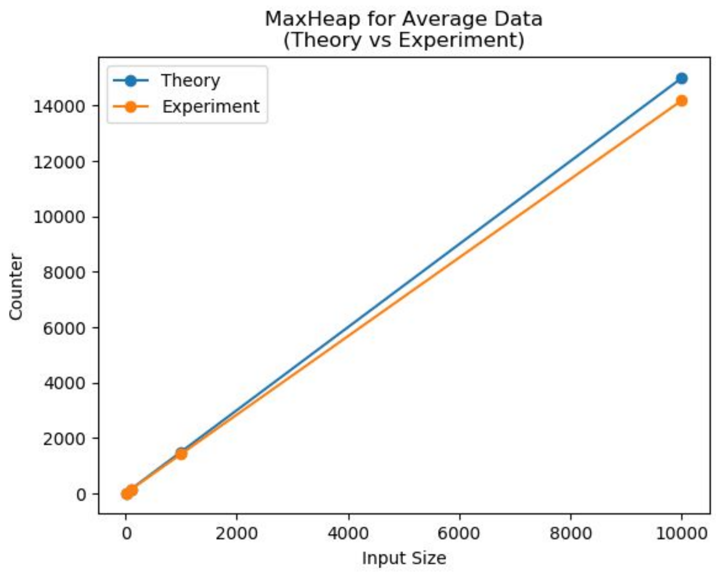


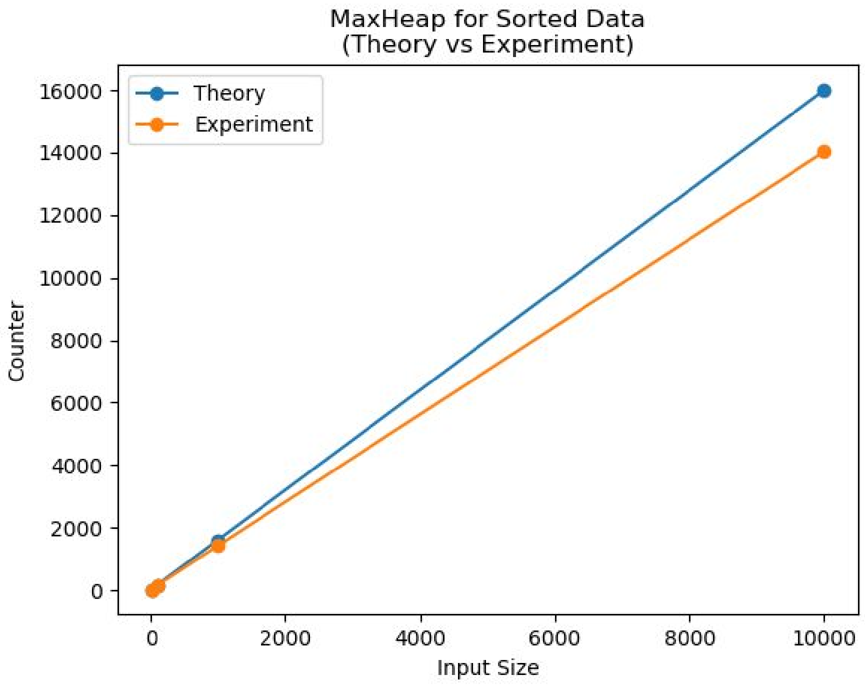


Max Heap

In theory **Time complexity for all 3 cases(worst , average and best) is O(nlogn)** since time complexity of heapify is O(Logn) and insert heap is O(n). So overall time complexity of Max-heap is **O(nLogn)**.In experimental findings if you look at max heap counter values **for sorted input,** when the size is 1000 and 10000, counter values are 1501 and 15001 and their proportion to each other is **9,99**. **For reversed order** input values are 1414 and 14030 and their proportion to each other is **9,92. For average case** values are 1420 and 14188 and their proportion to each other is **9,99**. In theory values are 1000 and 100 for given sizes since 10log10 is 10 and data is increasing 10 times for each input set and their proportion to each other would be **10**. So we can say that findings meet with theoretical values with small amounts of differences. See in table.



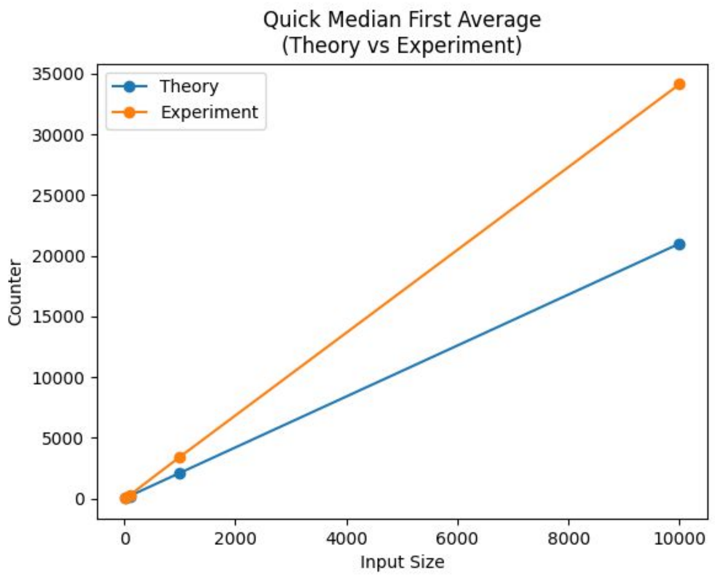


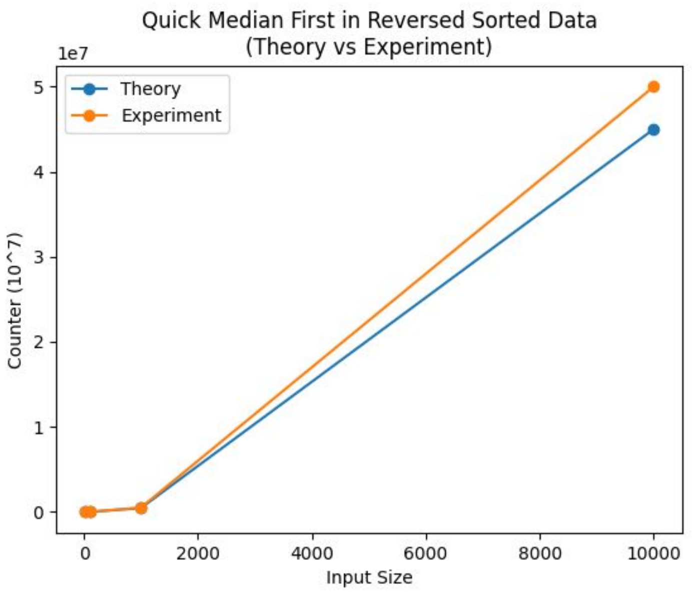


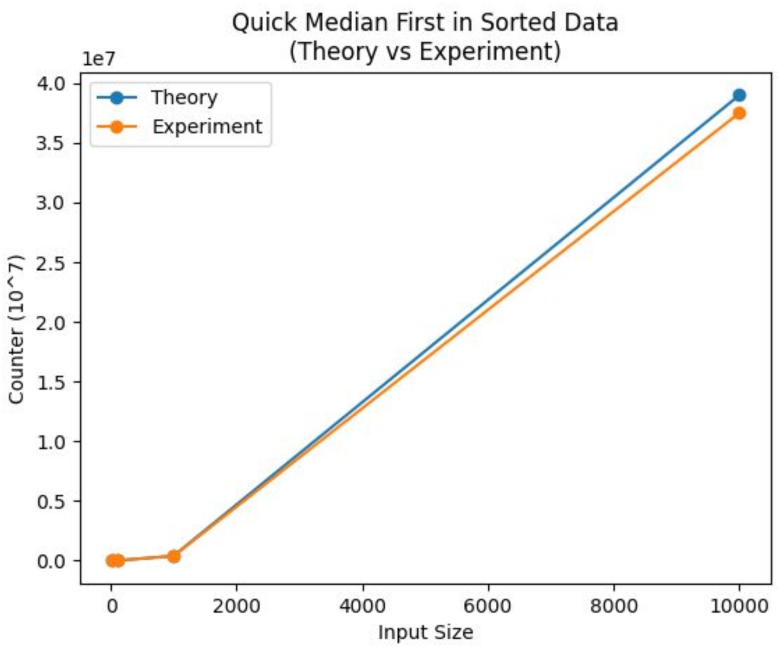
Quick Select Pivot First

In theory **For best and average cases** when **the first element is picked as pivot**, time complexity will be **O(nlogn)** since it is not sorted either in a normal way or reversed. In experimental findings **For the average case** when the size is 100 and 1000, values are 277 and 3415 and their proportion to each other is **12,33**.In theory values are 1000 and 100 for given sizes and their proportion to each other would be **10.** So we can say that it is close to theoretical values. See in table.

The worst case would occur when the array is already sorted in increasing or decreasing order. Thus time complexity will be **O(n^2)).** If we look at cases that input is whether sorted or sorted in reversed order , counter values' growth is quadratic.For instance when input is sorted and size goes through 10 to 100 to 1000 to 10000 ; counter values will be (39 // 3774 // 375249 and 37502499 see in table).The situation is same for reversed order since values are (45 // 4950 // 499500 // 49995000 see in table) growing in quadratic form. We can clearly observe that when size is increased n times, counter values are increased n^2 times.





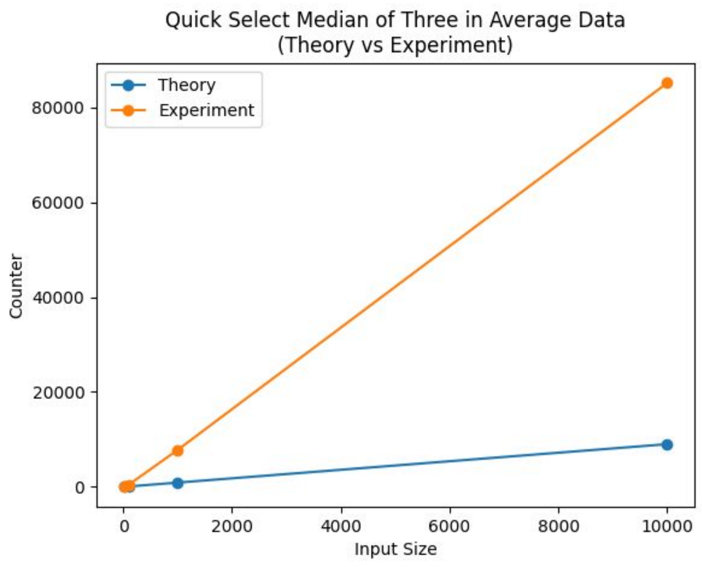


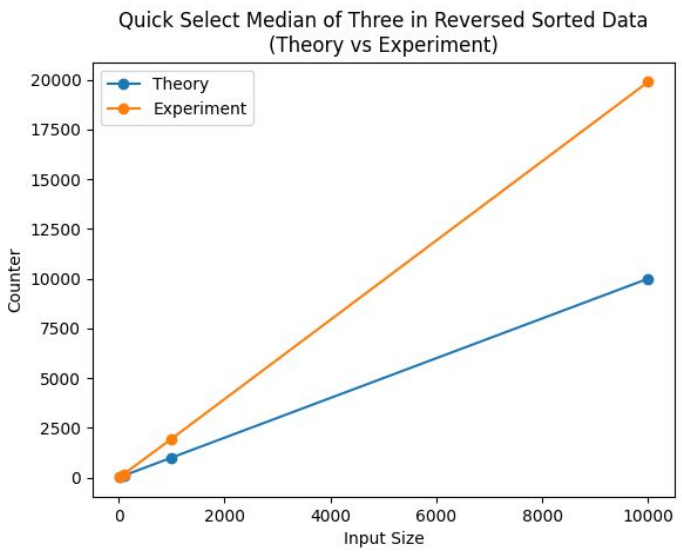
Quick Select Median of Three

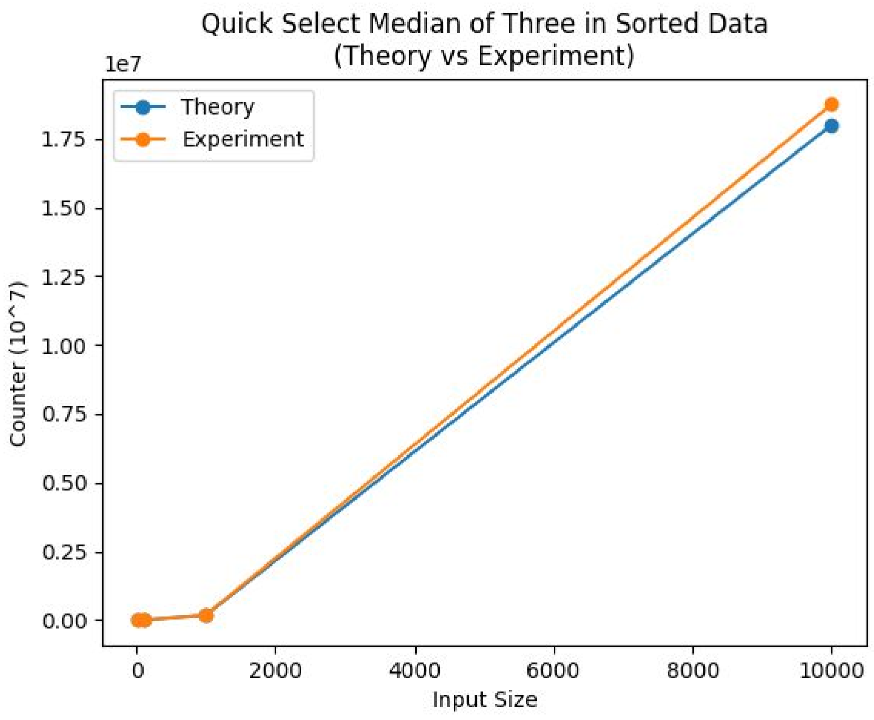
In theory **Best case time complexity** for the **Median of Three quick select** is **O(n log n).** If we look at findings, when data is in reverse or average order, counter values are growing linearly.

The average case time complexity is better than the worst case. In other words, it will act like linear time growth but at the same time it is not like O(n). So we can say that it is better than O(n^2) and close to O(nlogn). (9 // 430 // 7764 // 85126) See in table.

For the worst case , on a very contrived data set the **Median of Three** would result in a pivot that is the second minimum item of the remainder to be sorted. Time complexity would be **O(n^2)).** When we look at experimental findings for the sorted input list, while size goes through 10 to 100 to 1000 to 10000 ; counter values are 18 // 1850 // 187250 and 18747500. See in table.







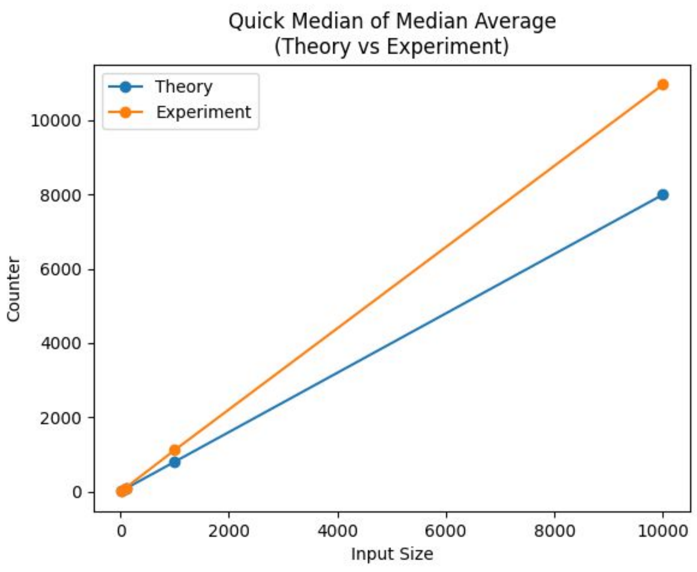
Quick Select Median of Medians

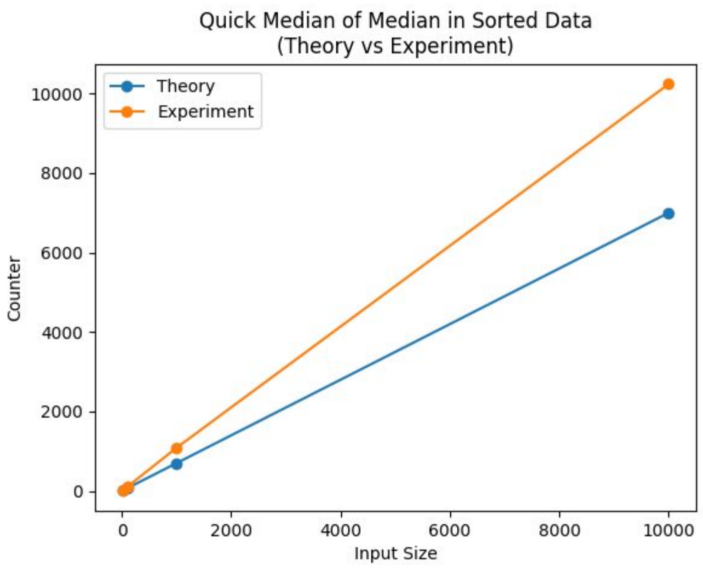
As we mentioned above the median of medians algorithm will find an approximate median in linear time only, which is limited but an additional overhead for quickselect.Thus, **time complexity for all 3 cases(worst , average and best) is O(n).** If we look at experimental findings for any input characteristic that size goes through 10 to 100 to 1000 to 10000, counter values grow like in linear time.

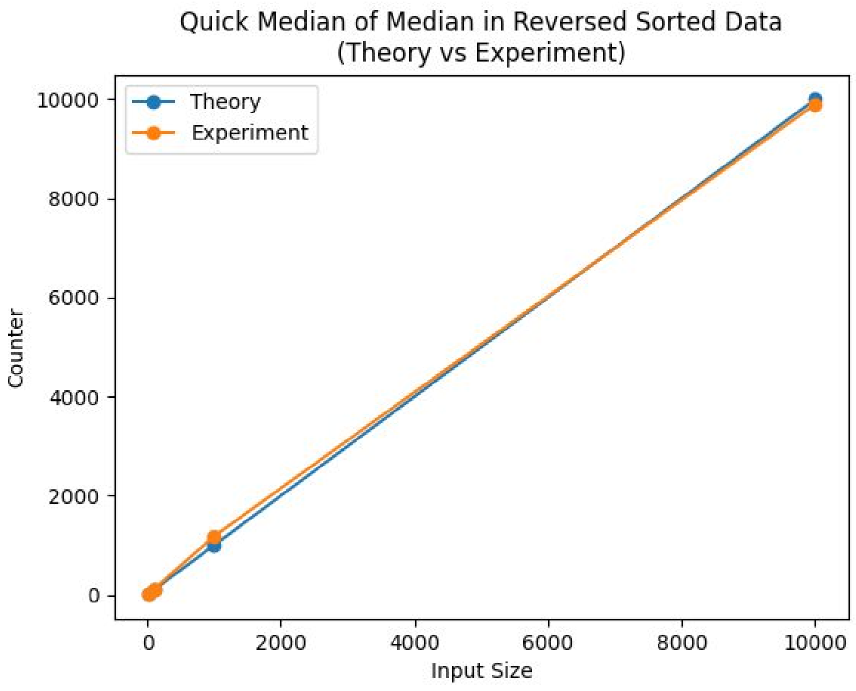
For sorted input (7 // 103 // 1085 // 10240 see in table)

For reversed order input (10 // 116 // 1180 // 9902 see in table)

For average case input (8 // 102 // 1114 // 10965 see in table)



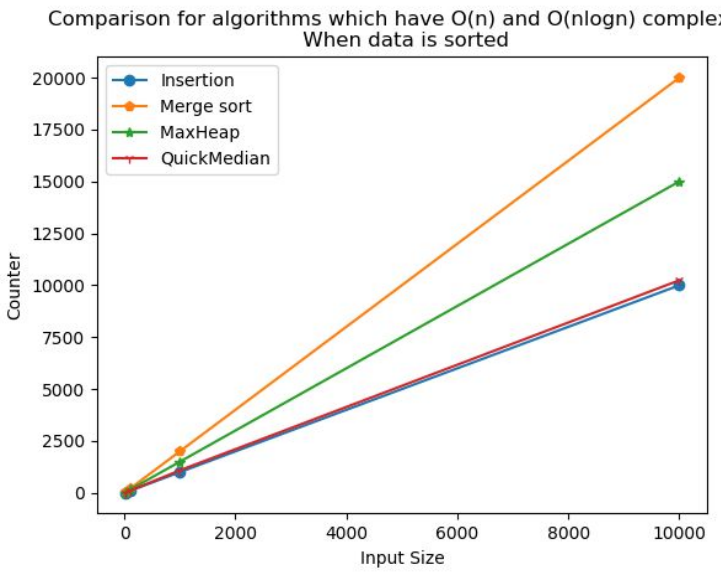


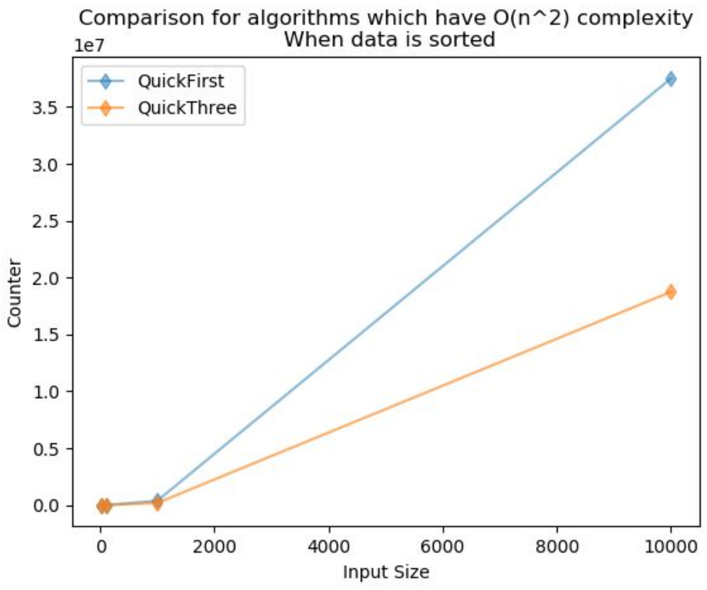


Comparing Algorithms to Find Median

In the below section we compare all algorithms for finding median using their experimental time complexity findings with given data characteristics. While doing that, we have divided the comparison into two section where time complexities are quadratic(O(n^2)) and non quadratic (O(n) || O(nlogn)).So that we can observe the difference better since when the corresponding time complexity for any algorithm is O(n^2), counter values are increasing enormously compared to any other non quadratic algorithm.

Sorted Data

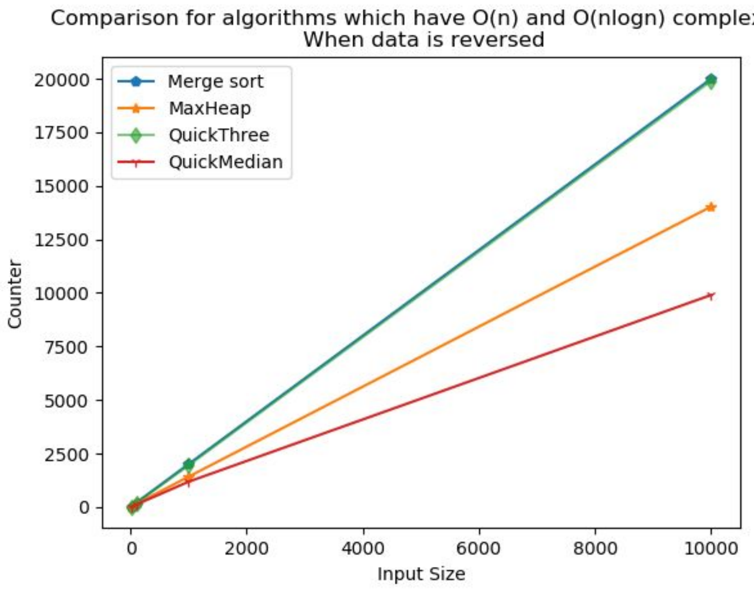


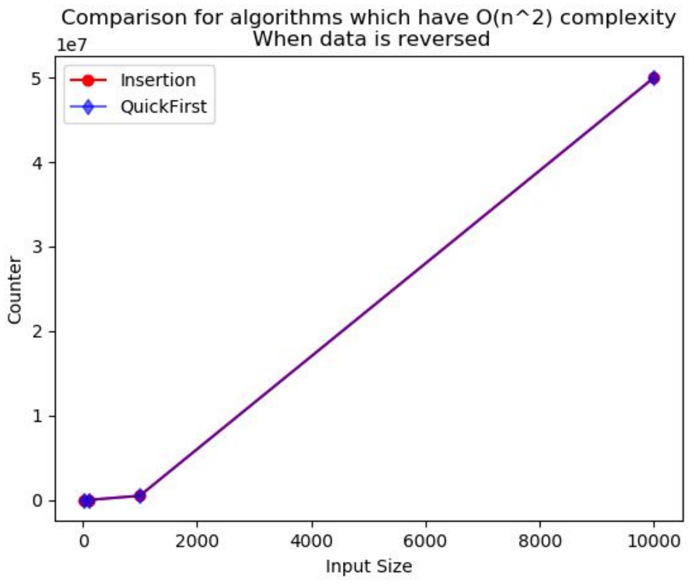


As it can be seen from the tables, for sorted data best algorithms are median of medians and insertion sort. If we look at findings for input sizes, counter values for insertion sort are 9 // 99 // 999 // 9999. And for the median of medians algorithm values are 7 // 103 // 1085 // 10240. So we can say that for the input size 10, the best algorithm is Median of Medians. On the other hand for the rest of the inputs, the best algorithm is Insertion Sort. See in table.

Furtherly, whatever the input size, Quick Select Pivot First Algorithm is the worst algorithm to find median when the data is sorted for all sizes, compared to other 5 algorithms.

Reversed Data

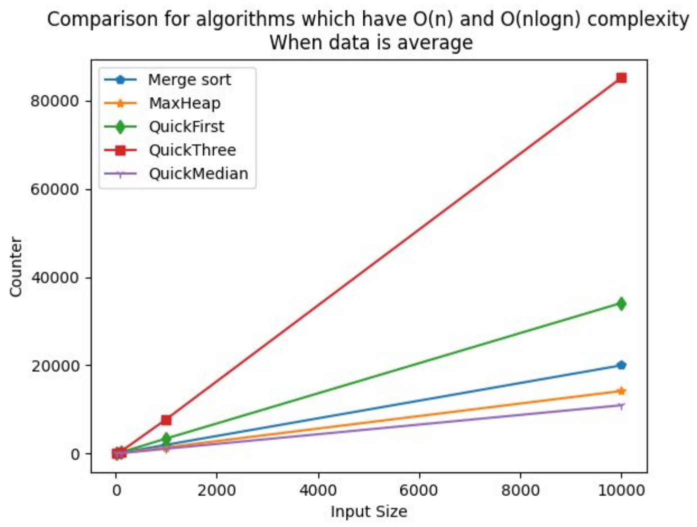


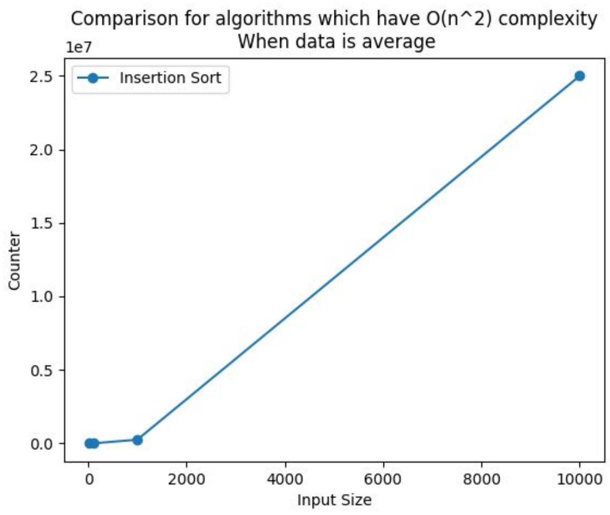


It is obvious that, Median of Medians algorithm is the best algorithm for finding median when the data is sorted in reverse order for all sizes.

On the other hand the worst algorithm for finding median when the data is sorted in reverse order is Insertion Sort. But counter values are really close with Quick Select Pivot First Algorithm. Both of them have quadratic time complexity. See in table.

Average Data





We have compared 6 algorithms in average, random generated, data. We have observed that whatever the input size, Quick Select Median of Median algorithms is best one compared to other algorithms. In addition to that, whatever the input size, Insertion Algorithm is the worst algorithm to find median, compared to other 5 algorithms.

Table for Counters

Sorted Data

	ALGORITHMS					
Input characteristics	Insertion sort	Merge sort	Max Heap	Quick Select Pivot First	Quick Select Median-of-3	Quick Select Median of Medians
sorted10	9	19	16	39	18	7
sorted100	99	199	151	3774	1850	103
sorted1000	999	1999	1501	375249	187250	1085
sorted10000	9999	19999	15001	37502499	18747500	10240

Reversed Data

	ALGORITHMS					
Input characteristics	Insertion sort	Merge sort	Max Heap	Quick Select Pivot First	Quick Select Median-of-3	Quick Select Median of Medians
reversed10	54	19	15	45	10	10
reversed100	5049	199	146	4950	176	116
reversed1000	500499	1999	1414	499500	1944	1180
reversed10000	50004999	19999	14030	49995000	19897	9902

Average Data

	ALGORITHMS					
Input characteristics	Insertion sort	Merge sort	Max Heap	Quick Select Pivot First	Quick Select Median-of-3	Quick Select Median of Medians
average10	28	19	15	21	9	8
average100	2555	199	142	277	430	102
average1000	250385	1999	1420	3415	7764	1114
average10000	24990755	19999	14188	34133	85126	10965

Teamwork

We worked on this project as a group of two people.In the programming part, we have worked together while doing pair programming. Ahmet Lekesiz created test cases and Ahmet Önkol got the results from algorithms and used them to create graphs. Finally, together we have written the report while comparing our findings with theoretical ones.

References

1. Heineman, G. T., Pollice, G., & Selkow, S. (2016). *Algorithms in a nutshell*. O'Reilly.
2. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (1989). *Introduction to algorithms*. MIT Press.