MARMARA UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

CSE2046 – ANALYSIS OF ALGORITHMS

HOMEWORK I

Submitted to:

Asst.Prof.Ömer Korçak

Students:

Erdi Türkay 150119853

İsmail Öksüz 150119516

Emre Sağıroğlu 150119766

Contents

Purpose of the Project

The aim of the project is to compare different algorithms to find the k'th smallest element in an unsorted list.

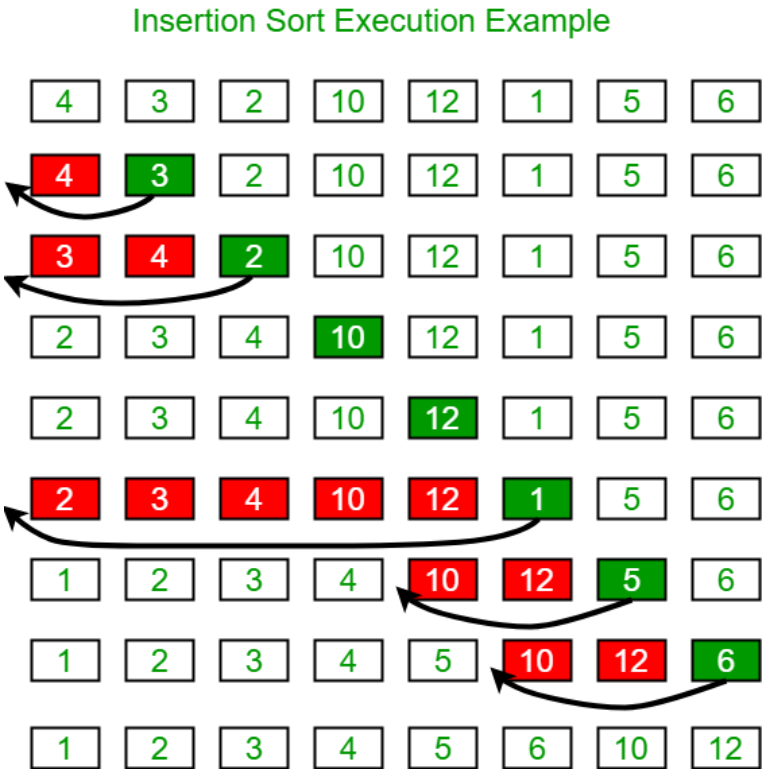
Firstly, the program generates random arrays which index of 10, 100, 1000, 10000, 100000. Then, all algorithms are processed in arrays of the specified index and it is obtained how many milliseconds the algorithm does the operation. Finally, the algorithms are compared, and it is decided which one is more efficient in terms of time. We will analyze the results of the experiment with using some tables and plots.

Insertion Sort

Insertion sort is a simple algorithm that uses in sorting arrays. This algorithm based on divide an array in two parts as sorted and unsorted.

Best-Case: If all elements sorted order in array already.

Worst-Case: If all elements sorted reverse order.



Time complexity of insertion sort is O(n) for the best case

$O(n^2)$ for the average case

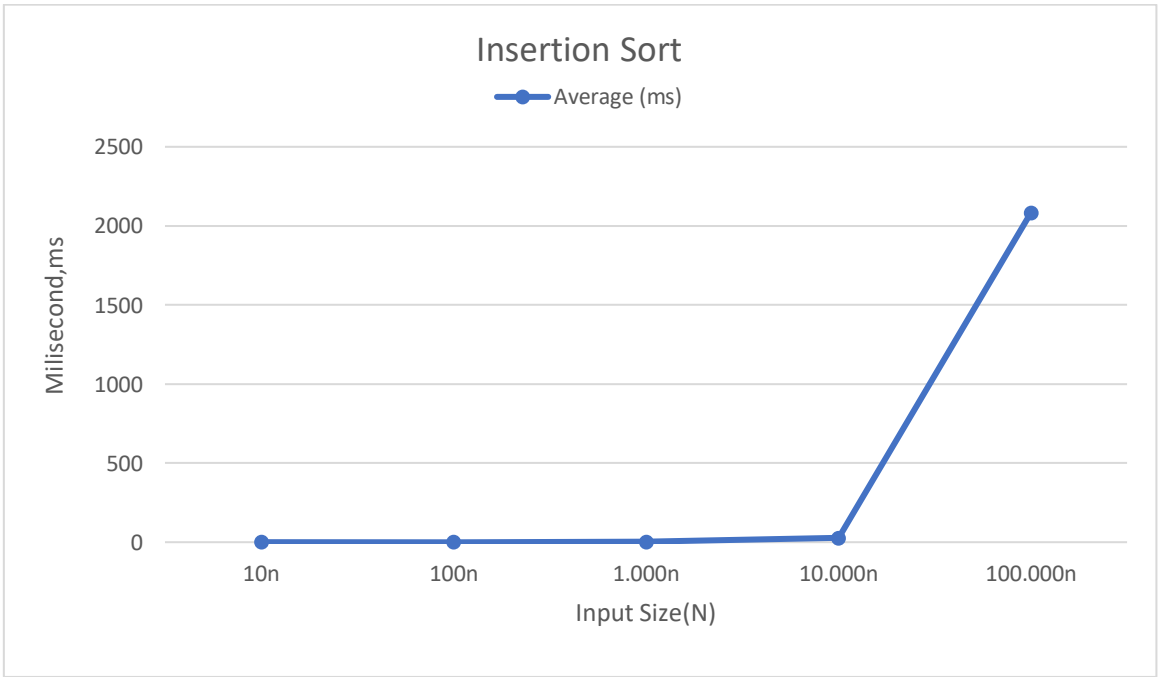
$O(n^2)$ for the worst case.

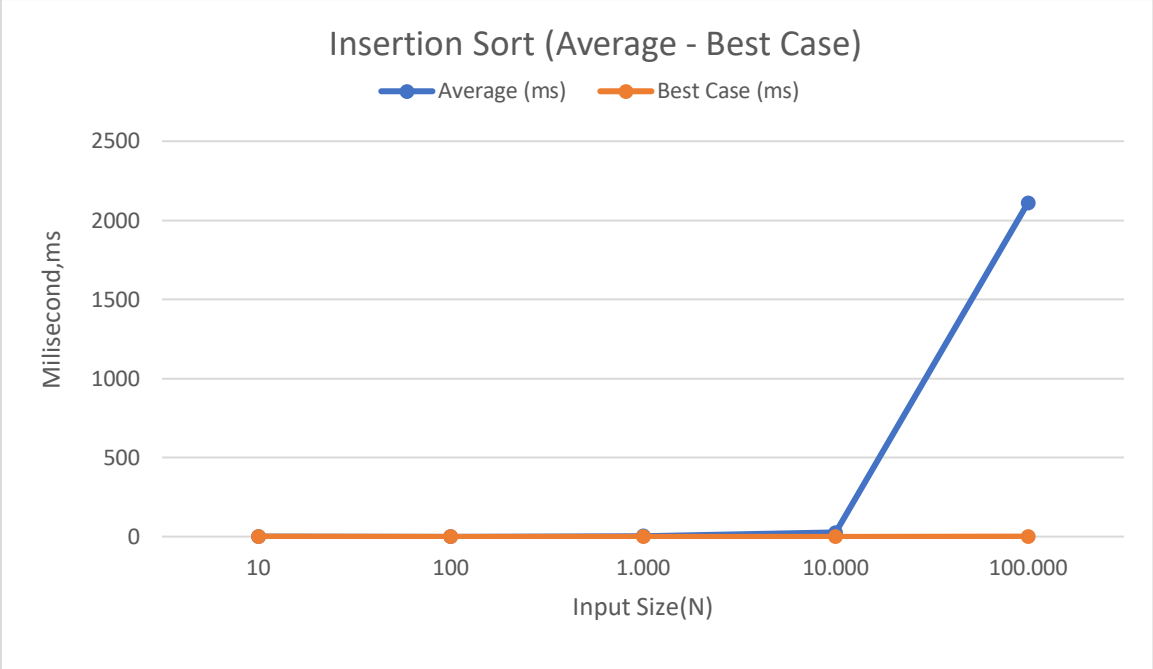
According to the results of experiment for Insertion Sort

n	Experiment 1 (ms)	Experiment 2 (ms)	Experiment 3 (ms)	Average (ms)
10	0,0111	0,0079	0,0073	0,0087
100	0,0611	0,0631	0,0582	0,0608
1000	2,7669	2,5178	2,4005	2,5617
10000	24,6685	28,0861	27,1635	26,6393
100000	2109,3822	2050,3310	2087,2219	2082,3117

Best Case of Insertion Sort

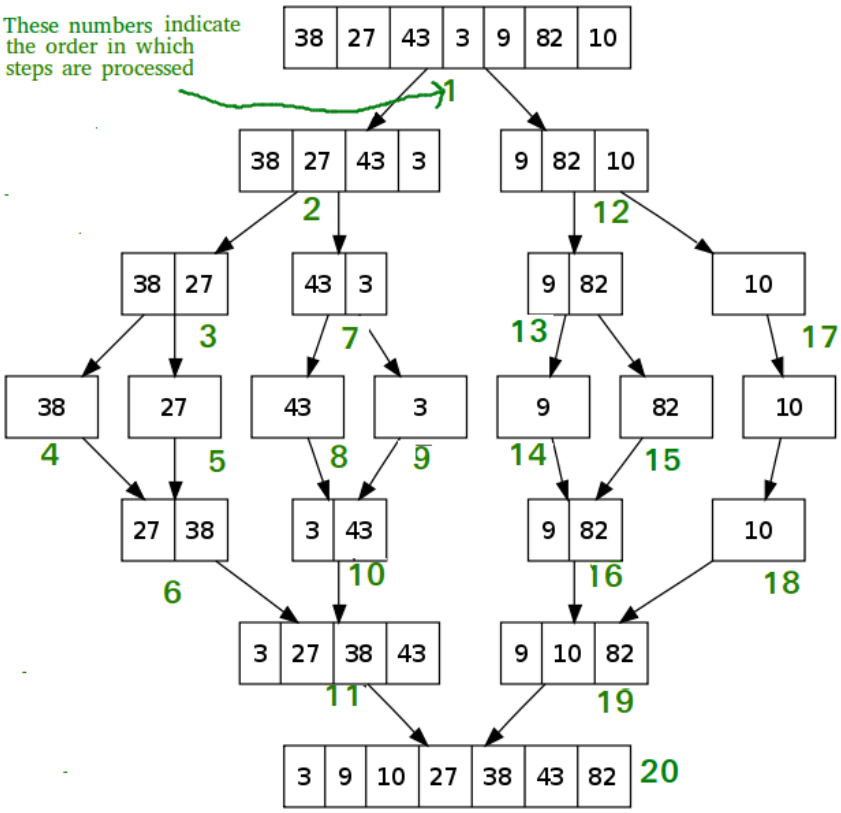
n	Experiment 1 (ms)	Experiment 2 (ms)	Experiment 3 (ms)	Average (ms)
10	0,0023	0,0012	0,0008	0,0014
100	0,0094	0,0193	0,0033	0,0106
1000	0,0049	0,0051	0,0073	0,0057
10000	0,0105	0,0107	0,0109	0,0107
100000	0,0999	0,0984	0,0983	0,0988





Merge Sort

Merge sort is a divide and conquer algorithm. It means divide the problem to smaller problems to solve problems. In merge sort we split the array in half. Dividing step continues until we are left with the individual items.



Time complexity of merge sort is O(n log(n)) for the best case

O(n log(n)) for the average case

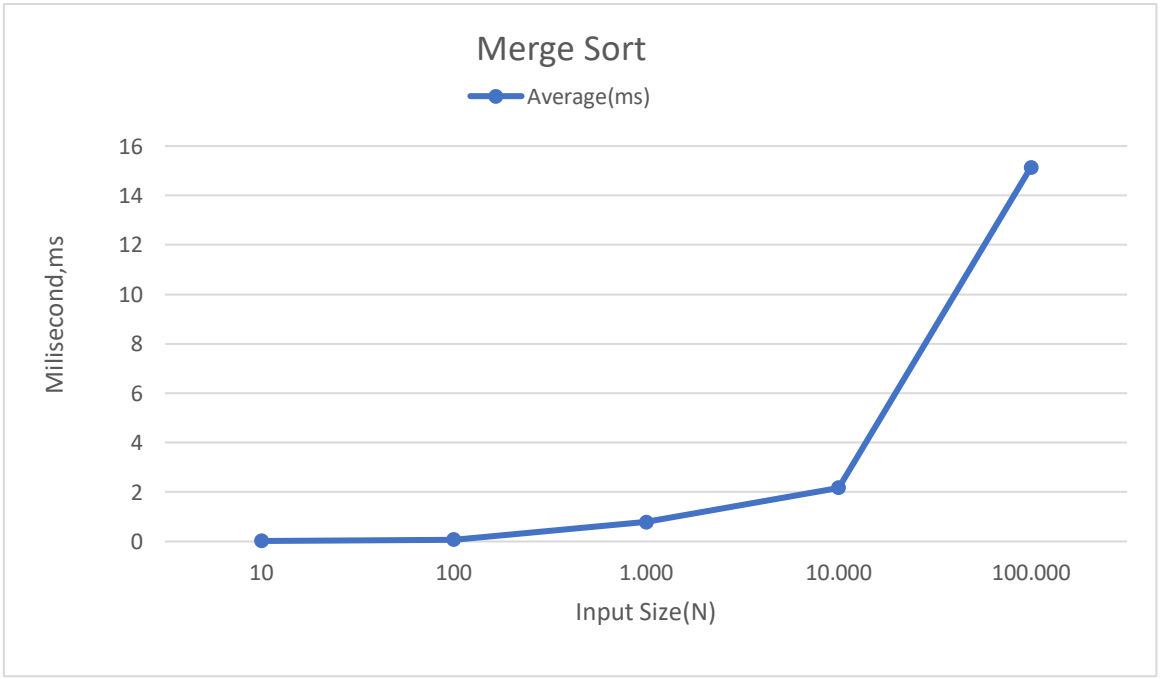
O(n log(n)) for the worst case.

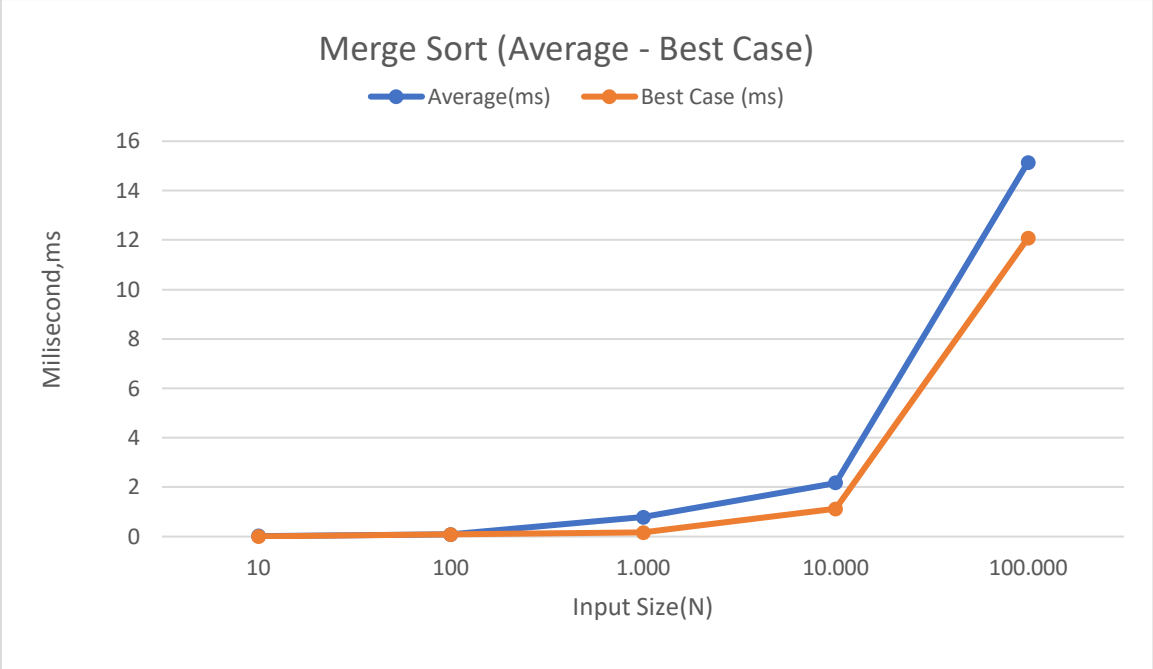
According to the results of experiment for Merge Sort

n	Experiment 1 (ms)	Experiment 2 (ms)	Experiment 3 (ms)	Average (ms)
10	0,0186	0,0281	0,0191	0,0219
100	0,0791	0,0954	0,1538	0,1094
1000	0,7838	0,6662	0,9488	0,7996
10000	2,1743	1,4691	1,4407	1,6947
100000	15,1415	14,6494	14,7052	14,8320

Best Case of Merge Sort

n	Experiment 1 (ms)	Experiment 2 (ms)	Experiment 3 (ms)	Average (ms)
10	0,0074	0,0077	0,0081	0,0077
100	0,0876	0,0863	0,0818	0,0852
1000	0,1616	0,1116	0,2213	0,1648
10000	1,3828	1,1647	0,8281	1,1252
100000	11,2133	11,3807	13,6326	12,0755





Quick Sort

Quick sort is a recursive algorithm. It aims to sort by simply selecting a number which is first element of the array in this experiment and classifying all other numbers as greater or less than the pivot.

Best-Case: Quick Sort works optimally if we always divide the arrays and subarrays into two equal size array.

Worst-Case:

Time complexity of quick sort is O(n log(n)) for the best case

O(n log(n)) for the average case

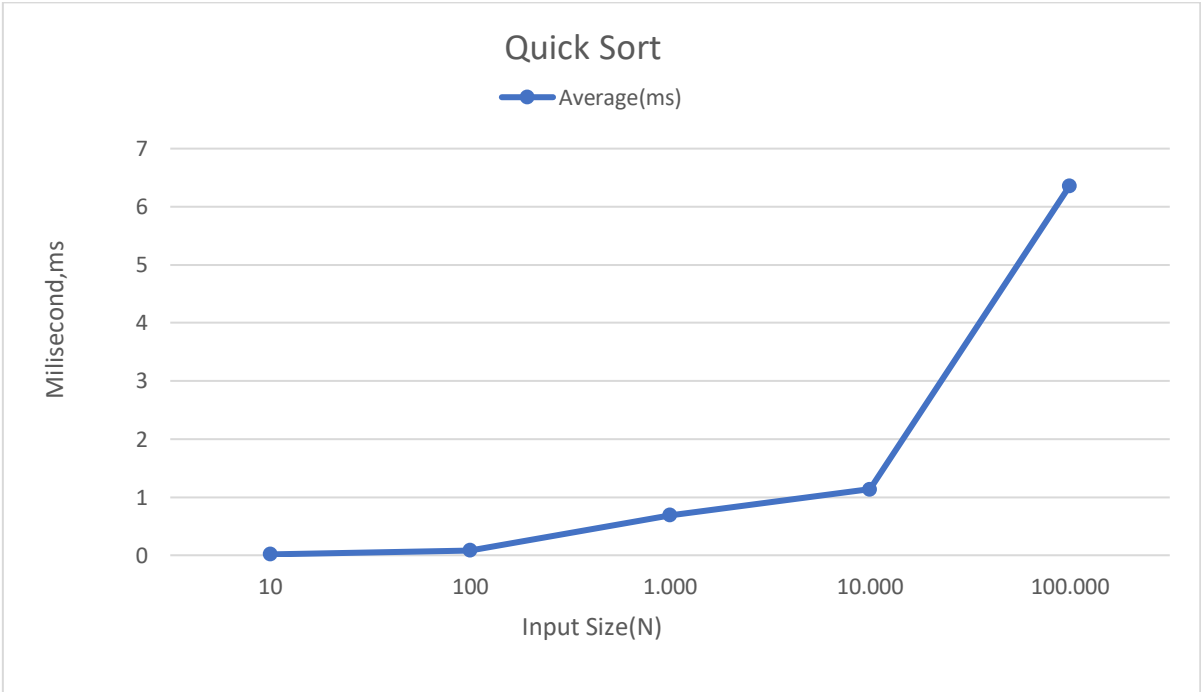
$O(n^2)$ for the worst case.

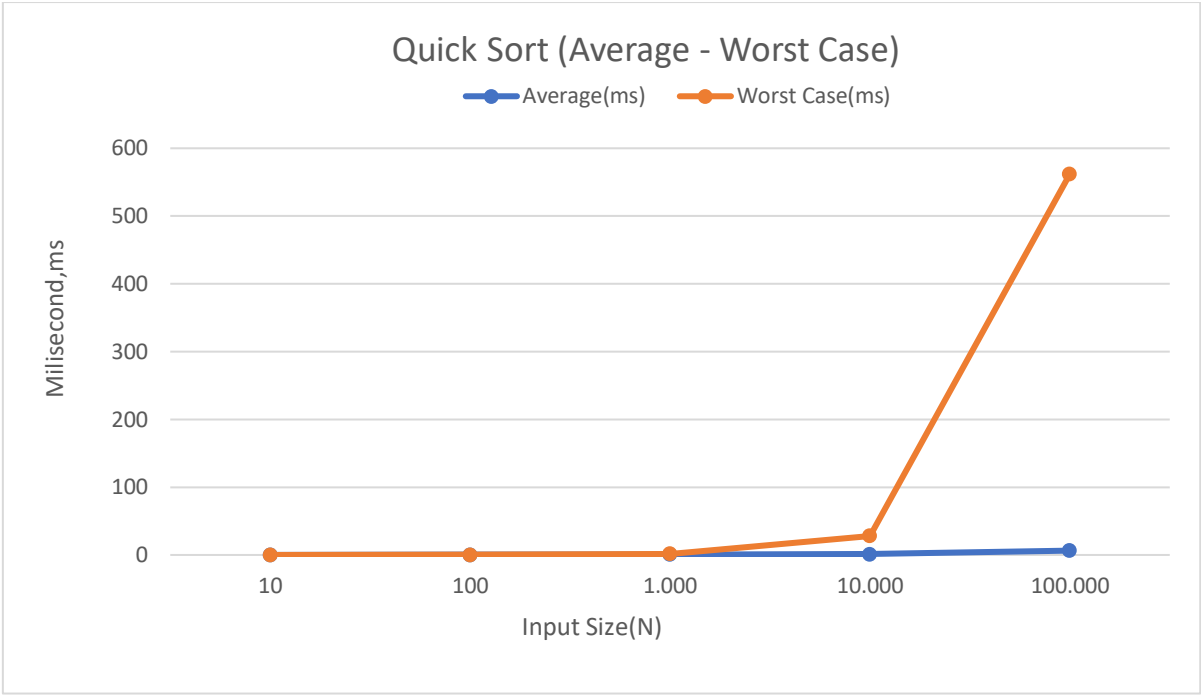
According to the results of experiment for Quick Sort

n	Experiment 1 (ms)	Experiment 2 (ms)	Experiment 3 (ms)	Average (ms)
10	0,0151	0,0073	0,0076	0,01
100	0,0825	0,0524	0,0596	0,0648
1.000	0,6871	0,3596	0,3427	0,4631
10.000	1,1376	1,1746	1,1679	1,16
100.000	6,3623	6,5984	6,5162	6,4923

Worst Case for Quick Sort

n	Experiment 1 (ms)	Experiment 2 (ms)	Experiment 3 (ms)	Average (ms)
10	0,0045	0,0042	0,0031	0,0039
100	0,1820	0,2664	0,0974	0,1819
1000	1,6284	1,4705	1,7147	1,6045
10000	25,7061	26,6569	31,9547	28,1059
100000	553,8959	553,8328	575,6686	561,7991

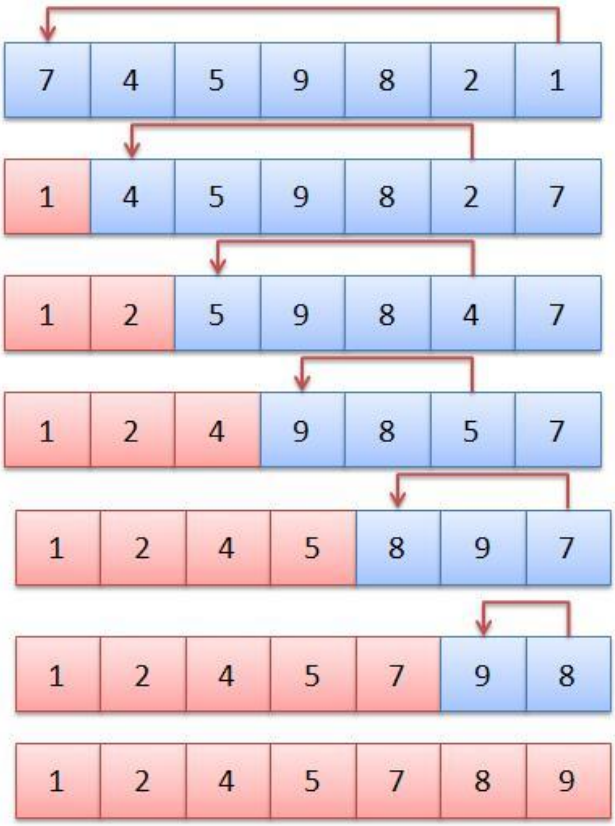




Selection Sort

Selection sort is simply find where the smallest number in the array is at each step. By replacing this number with the number at the beginning of the array, the smallest numbers are selected and it is thrown to the beginning.

Best case, worst case and average case have same time complexity which is $O(n^2)$.



Time complexity of selection sort is $O(n^2)$ for the best case

$O(n^2)$ for the average case

$O(n^2)$ for the worst case.

According to the results of experiment for Selection Sort

n	Experiment 1 (ms)	Experiment 2 (ms)	Experiment 3 (ms)	Average (ms)
10	0,0113	0,0087	0,0102	0,01
100	0,1534	0,1404	0,0674	0,1204
1.000	5,1813	3,5334	3,6010	4,1052
10.000	25,2934	23,4830	27,0247	25,2670
100.000	2228,9853	2142,7588	2177,4163	2183,0534

Heap Sort

Heap sort creates a heap tree in the background and sorts by taking the number at the top of this tree. It is similar to selection sort in finding a minimum element and placing at the first position way.

Best case, worst case and average case have same time complexity which is O(n log(n)).

Time complexity of heap sort is O(n log(n)) for the best case

O(n log(n)) for the average case

O(n log(n)) for the worst case.

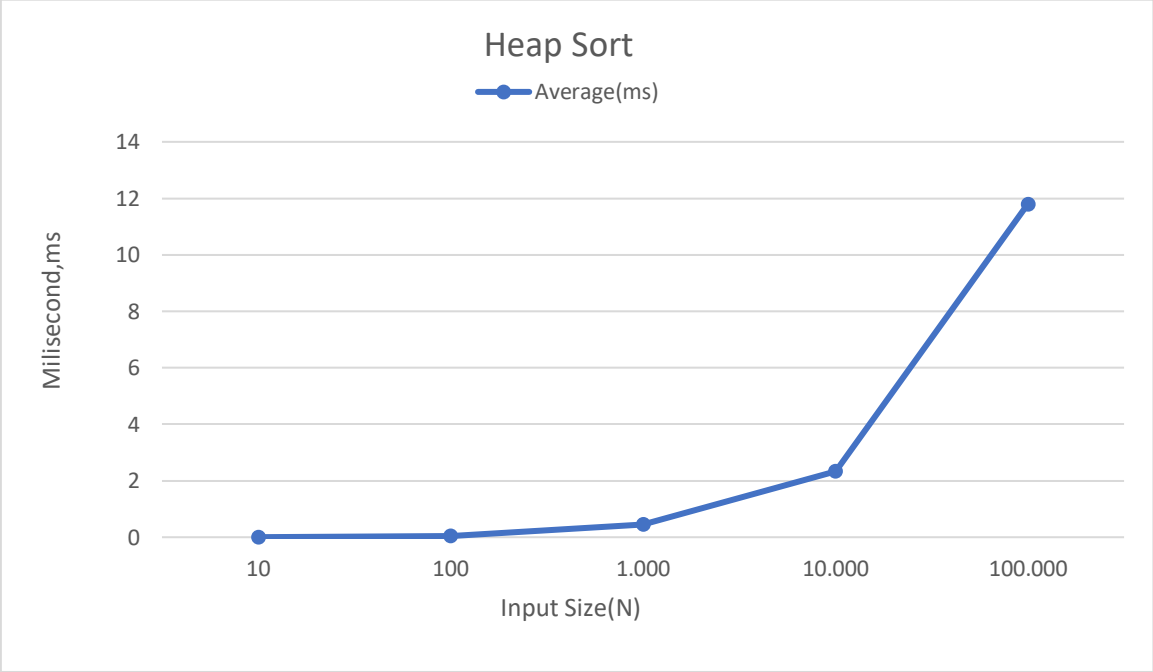
According to the results of experiment for Heap Sort

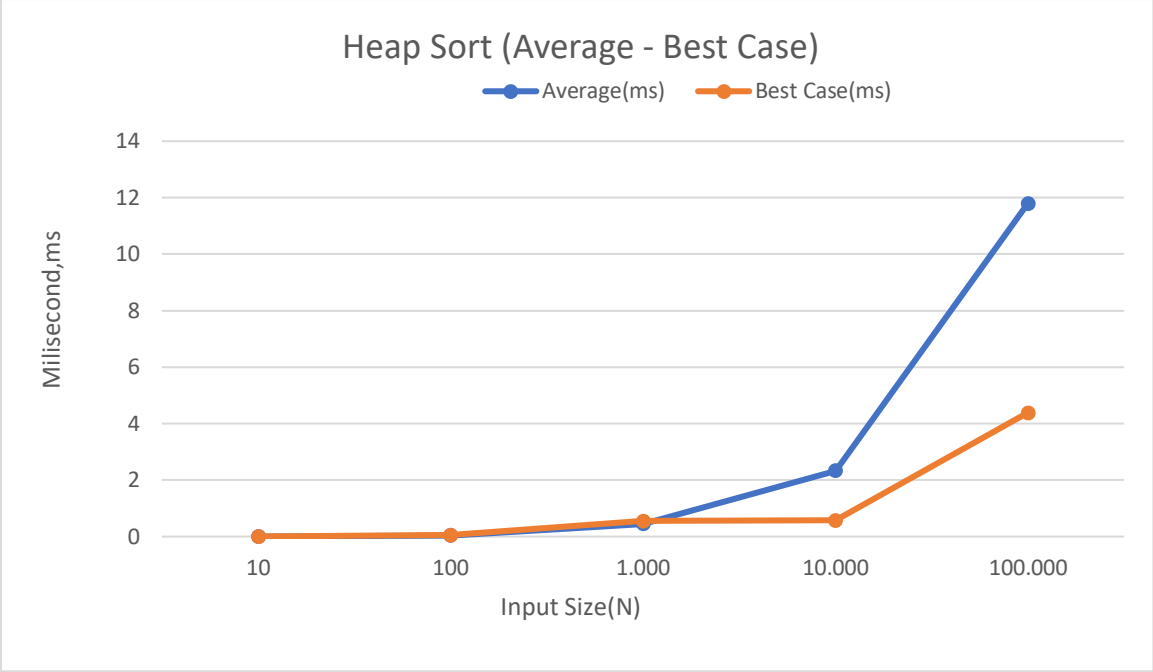
n	Experiment 1 (ms)	Experiment 2 (ms)	Experiment 3 (ms)	Average (ms)
10	0,0094	0,0346	0,0216	0,0021
100	0,0471	0,0593	0,0289	0,0451
1.000	0,5809	0,3090	0,4589	0,4496

10.000	3,0145	2,0978	1,8951	2,3358
100.000	11,6626	11,5405	12,1933	11,7988

Best Case of Heap Sort

n	Experiment 1 (ms)	Experiment 2 (ms)	Experiment 3 (ms)	Average (ms)
10	0,0038	0,0039	0,0031	0,0036
100	0,0496	0,0517	0,0567	0,0526
1.000	0,2034	0,1514	0,2011	0,5559
10.000	0,5825	0,5781	0,5700	0,5768
100.000	4,2700	4,7896	4,1042	4,3879



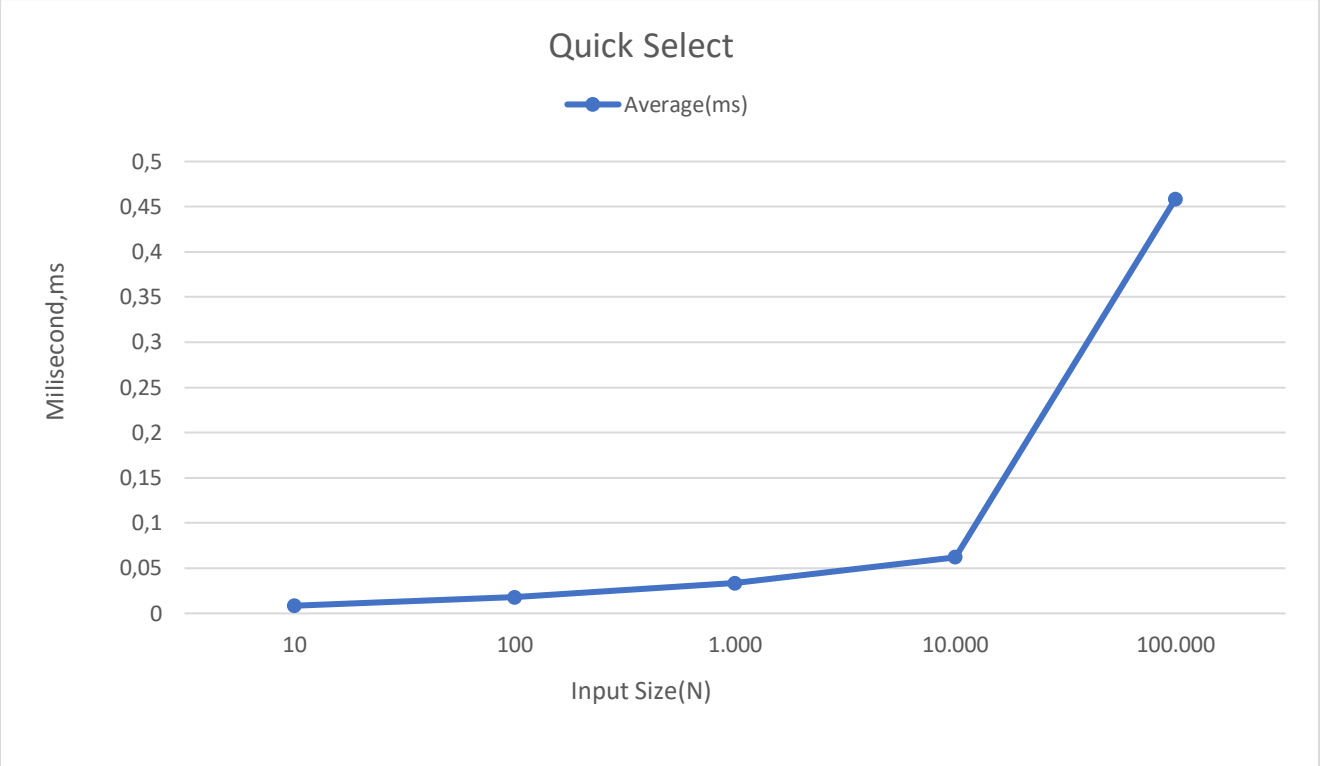


Quick Select

The distinction is because it just recurs for the segment that contains the k-th smallest element, rather than both sides (after discovering pivot). The rationale is simple: if the index of a partitioned piece is greater than k, the left part is repeated. We have discovered the k-th smallest element and returned if index equals k. We recur for the right part if index is less than k.

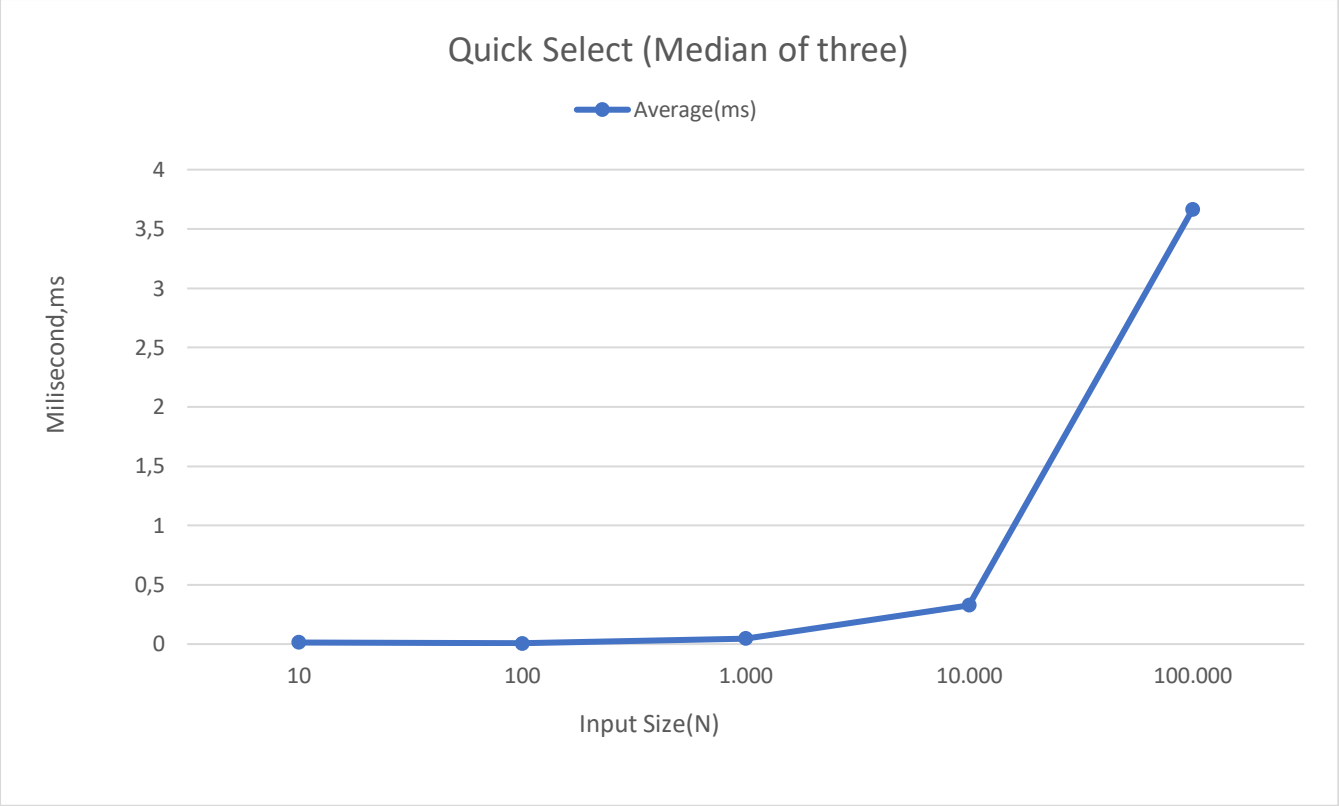
According to the results of experiment for Quick Select

n	Experiment 1 (ms)	Experiment 2 (ms)	Experiment 3 (ms)	Average (ms)
10	0,0087	0,0091	0,0078	0,0085
100	0,0193	0,0279	0,0065	0,0179
1.000	0,0170	0,0575	0,0258	0,0334
10.000	0,0564	0,0666	0,0632	0,0620
100.000	0,4488	0,7754	0,1516	0,4586



Quick Select (Median of three)

To find the kth element in the list, quickselect chooses a pivot element that divides the list into two parts. On the left side, the element which is smaller than the pivot will exist and on the right side, the element which is bigger than the pivot will exist. The pivot will be chosen by using the first, the middle and the last indexes in the array in the unordered list. The median of those three numbers will be the pivot. This process is applied repeatedly on the part where the searched element lies. While the kth element will be found, those processes recursively will execute.

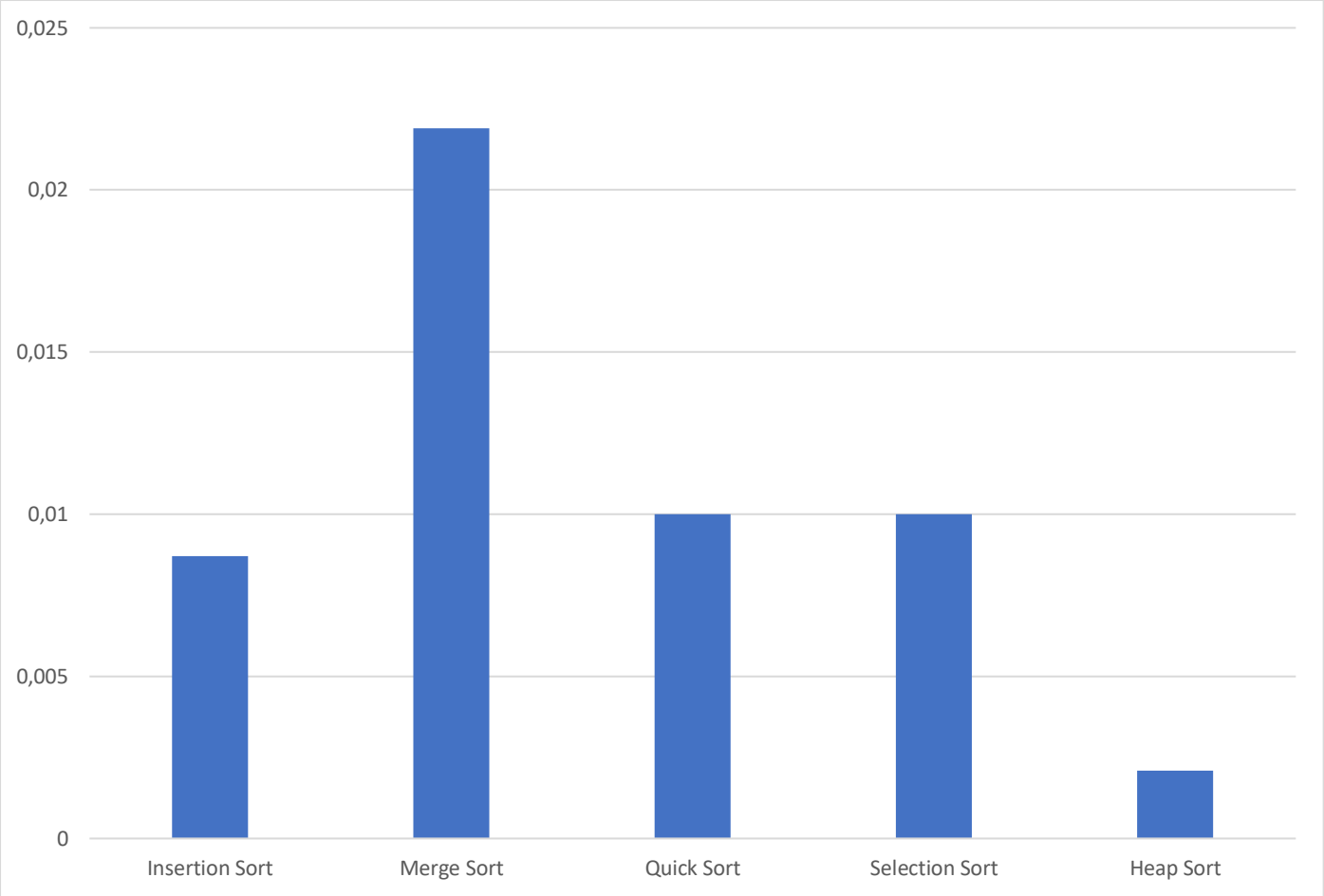


According to the results of experiment for Quick Select (Median of three)

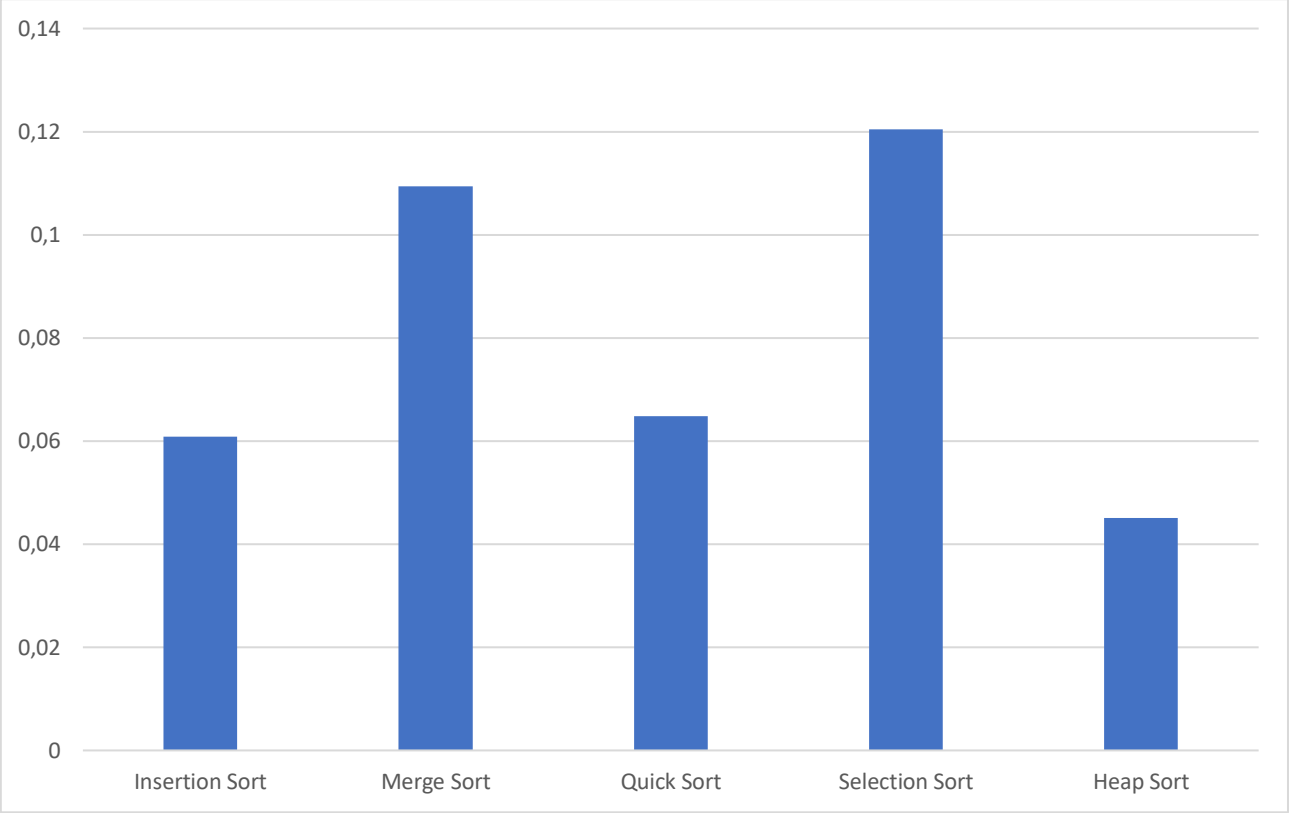
n	Experiment 1 (ms)	Experiment 2 (ms)	Experiment 3 (ms)	Average (ms)
10	0,0113	0,0217	0,0146	0,0158
100	0,0109	0,0049	0,0072	0,0076
1.000	0,0559	0,0597	0,0227	0,0461
10.000	0,3790	0,2458	0,3569	0,3272
100.000	3,0307	3,2270	4,7334	3,6637

ANALYSIS OF ALGORITHMS

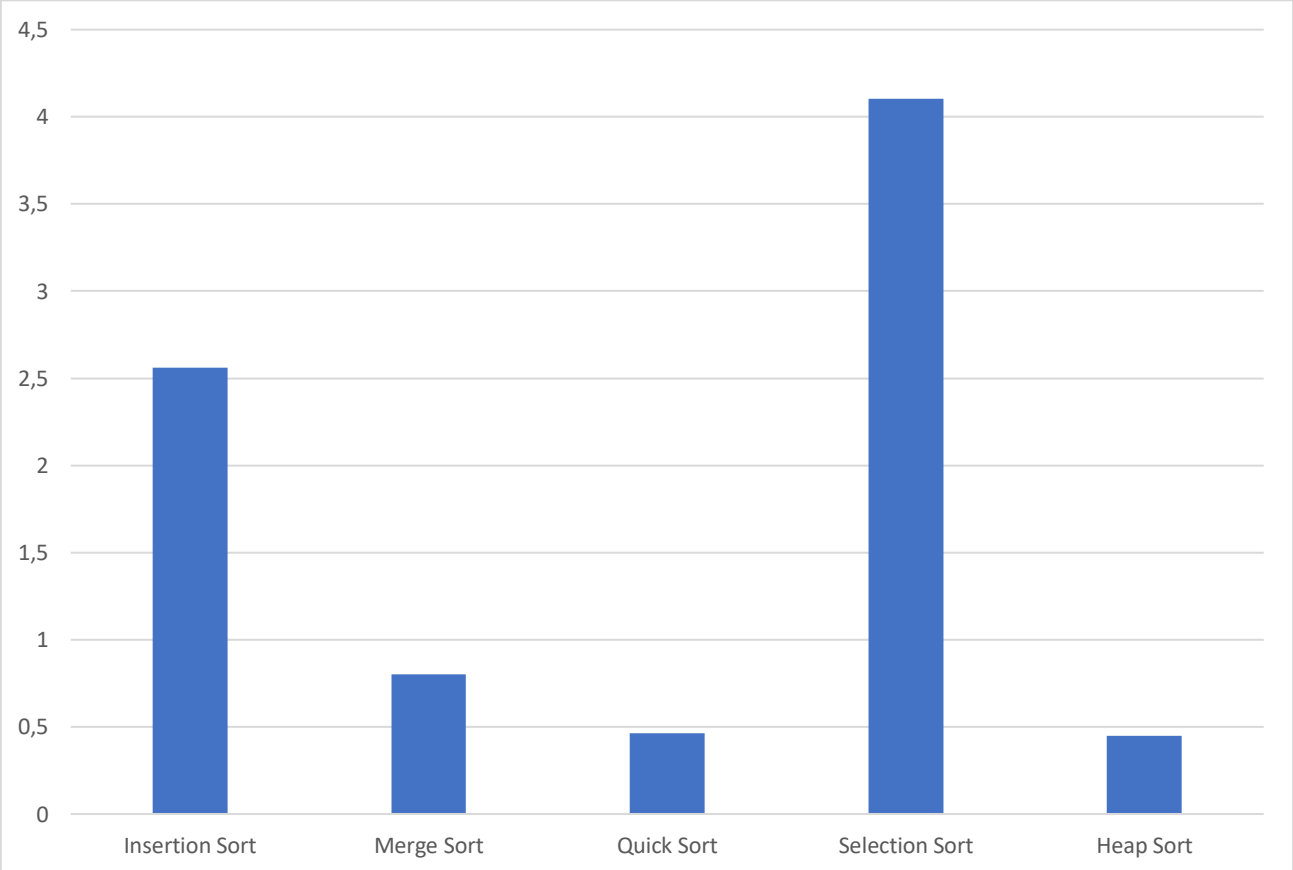
Execution Time For All Sorting Algorithms (N=10)



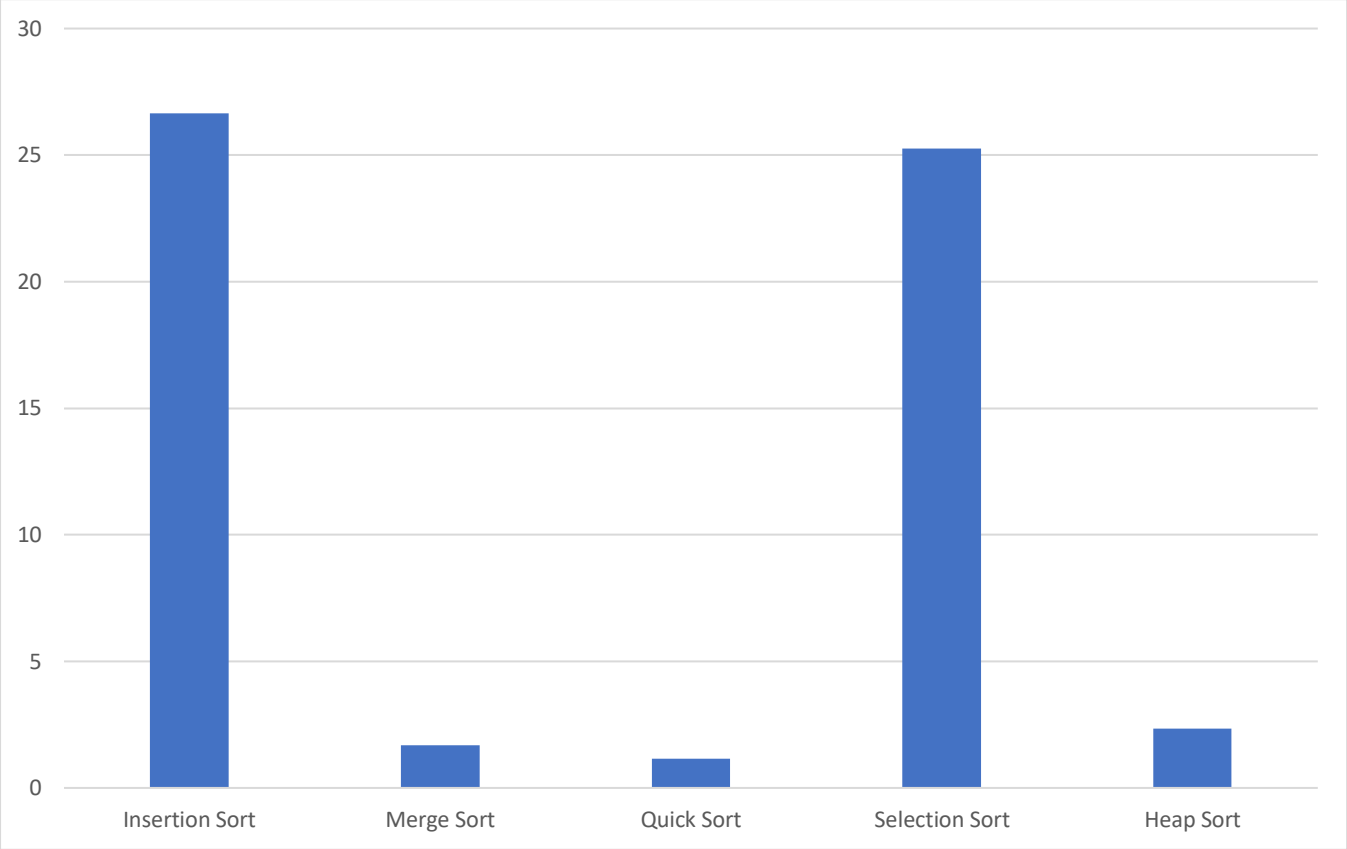
Execution Time For All Sorting Algorithms (N=100)



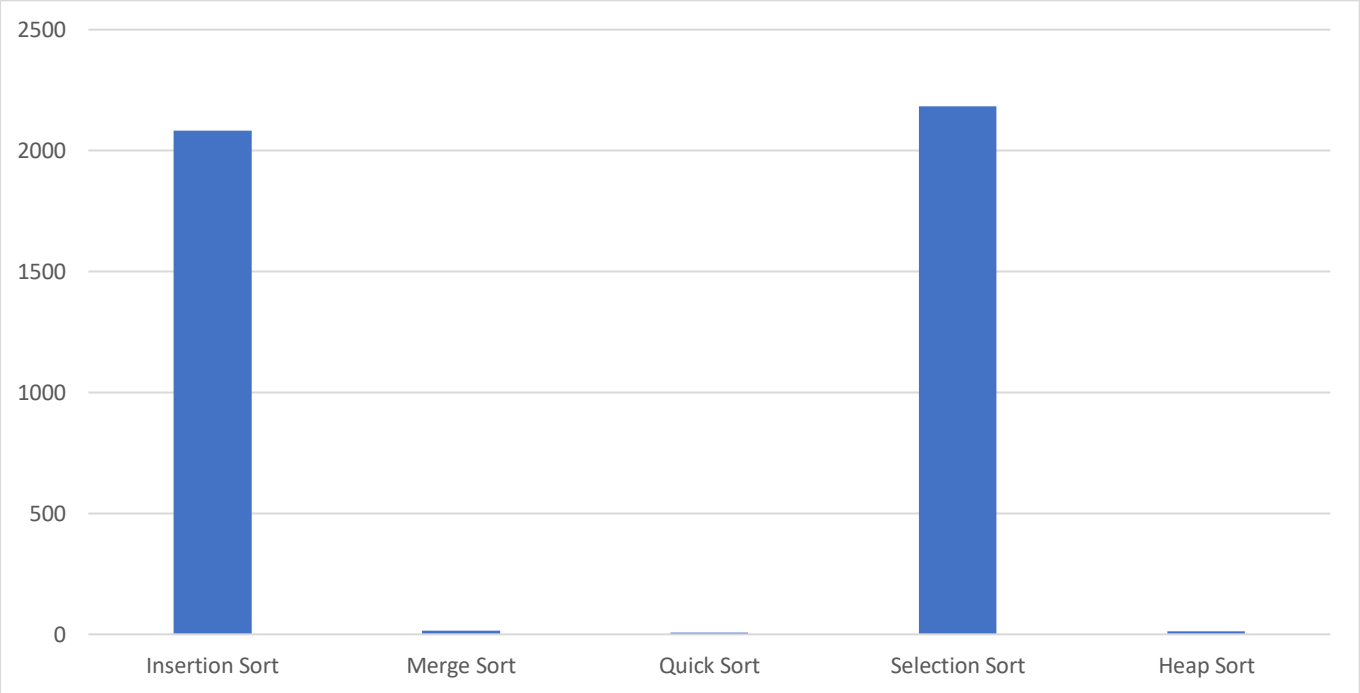
Execution Time For All Sorting Algorithms (N=1.000)



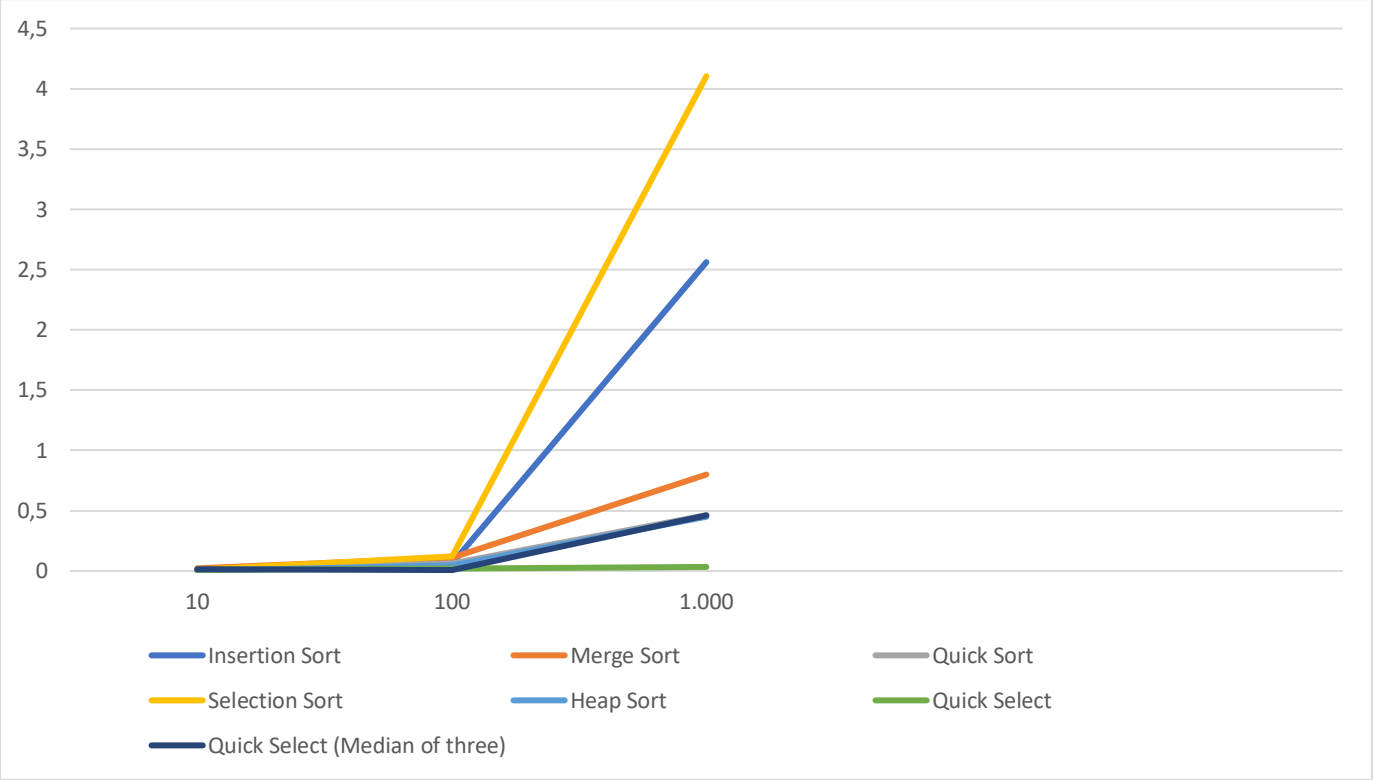
Execution Time For All Sorting Algorithms (N=10.000)



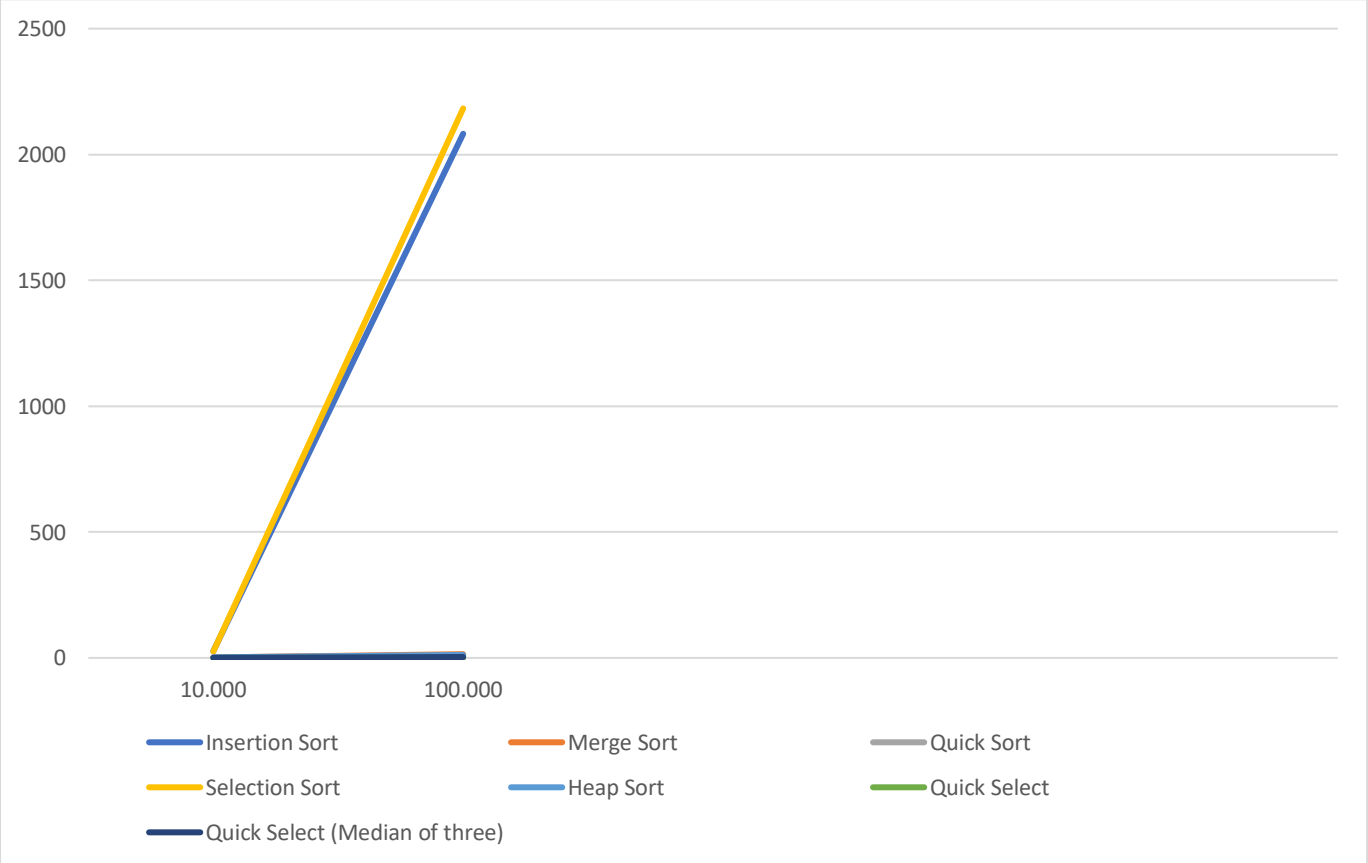
Execution Time For All Sorting Algorithms (N=100.000)



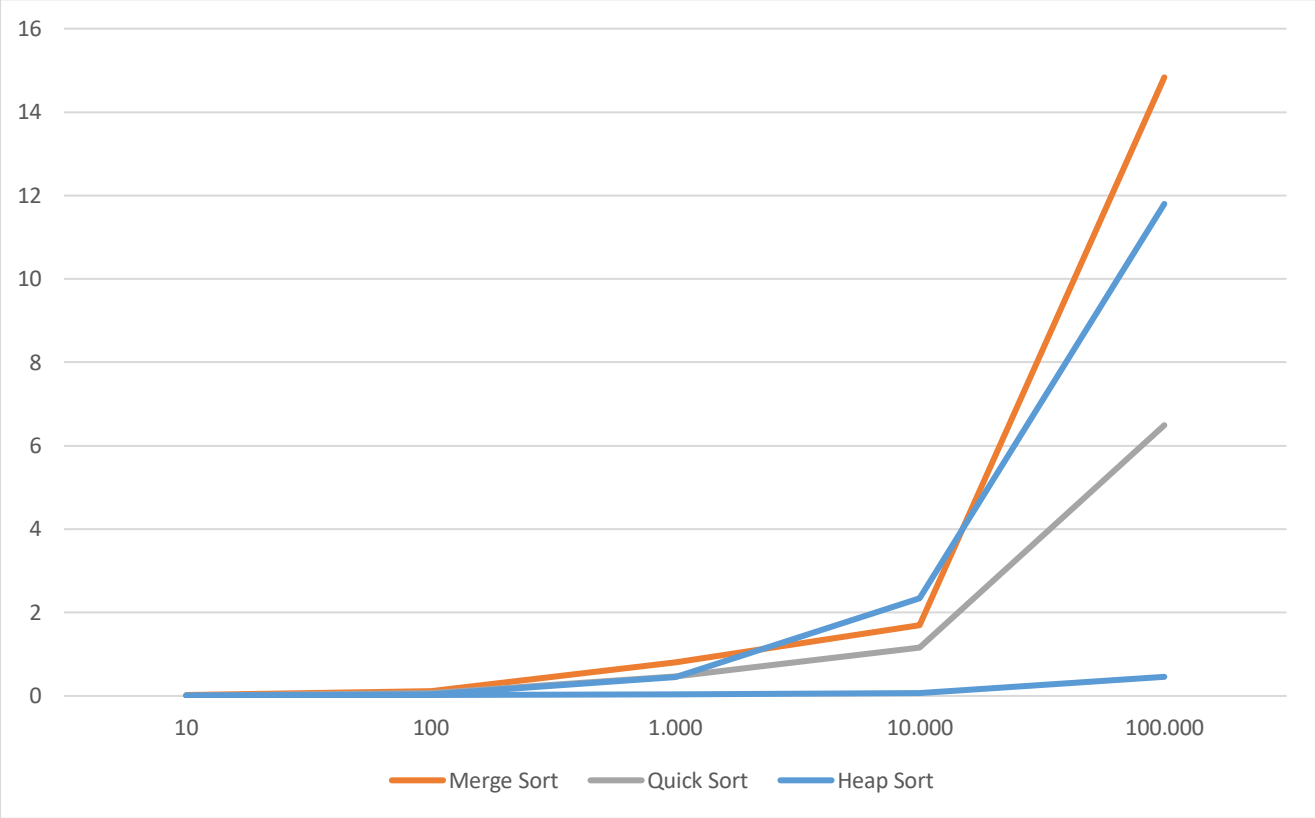
Execution Time Comparison (N between 10-1.000)



Execution Time Comparison (N between 10.000-100.000)



Execution Time Comparison Only For Merge Sort-Quick Sort-Heap Sort



General Evaluation & Summary

For this project, the input sizes were taken as 10, 100, 1.000, 10.000 and 100.000. The reason why we used such as those different sizes of inputs is to make more accurate inferences and represent the effects of the memory usage on the time complexity algorithms.

According to the results in the data above, while the input size is smaller, the execution time of the Selection Sort is more than the others. Whereas, while the input size becomes larger, the difference between the execution time of the Selection Sort and the execution time of the Heap Sort will decrease.

In conclusion; while the input size increases, the execution time difference between the Selection Sort and the Heap Sort comes closer to each other. Therefore; it can be estimated that the execution time of the Heap Sort will be more than the execution time of the Selection Sort for higher number of input sizes.

References:

- P. F. Robert Sedgewick, An Introduction to the Analysis of Algorithms.
- https://mimoza.marmara.edu.tr/~omer.korcak/courses/CSE246/CSE246_Materials.html
- https://www.geeksforgeeks.org/insertion-sort/
- https://www.geeksforgeeks.org/merge-sort/
- https://en.wikipedia.org/wiki/Quicksort
- https://en.wikipedia.org/wiki/Heapsort

The Division of Labor:

	Code	Report
Erdi Türkay	**X**	
İsmail Öksüz		**X**
Emre Sağıroğlu		**X**