****MARMARA UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING**

CSE2046 PROJECT 1

Name Student Number

Mustafa Tunahan BAŞ 150119055

Sadık AKGEDİK 150119052

**Introduction:** In this project, we were asked to compare various selection and sorting algorithms in an unsorted list of n numbers. We were asked to choose the inputs as necessary.

**Task Division:** We shared tasks equally. Sadık mostly worked on the coding part and Tunahan mostly worked on the graphing part. We wrote the report together.

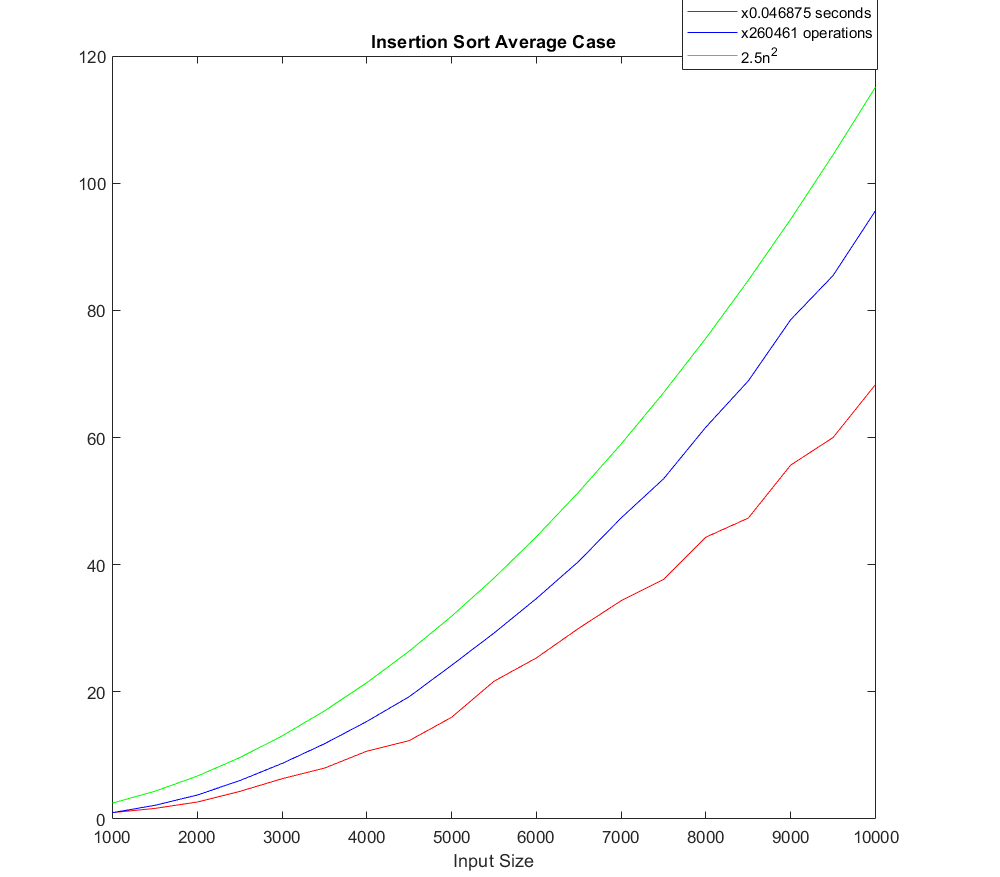
**The language used and algorithms:** The algorithms ( as stated in the homework pdf) are sorting and selecting algorithms such as Insertion-sort, Merge-sort, Quick-sort, partial Selection-sort, partial Heap-sort, quick select, and quickly select with a median of three pivot selection. We used Python programming language to implement these algorithms. The reason we chose Python was that it was easier to implement most of the algorithms and run them with various sizes and characteristics. Then we used MatLab to convert the data into graphs.

**How did we graph the data in general and how to interprete it?**

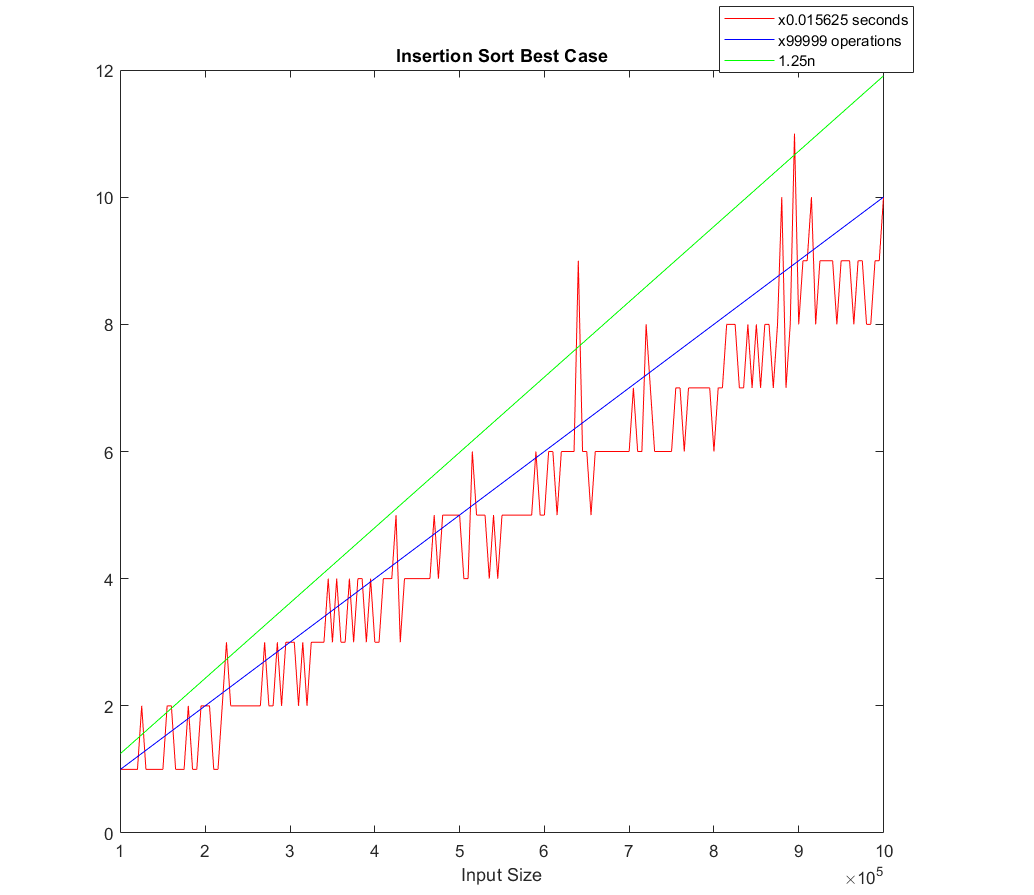
Since the data (execution time,basic operation count,asymtotic function lines) are all of different magnitudes, execution time is at most around 7-8 seconds whereas asymptotic functions like n^2 can easily get to up to 10^8, we decided to multiply the execution time with a power of 10 to get it to match the asymptotic functions.But this led to rather strange results and we felt that showing the data this way would be manipulative since there were no certain rules to by which power of 10 the execution time should be multiplied so we chose another technique (Although the graphs achieved with the first manipulative technique are not included in this report they can be found near the other graphs in the files [the ones whose names end with 2 are the graphs of the second method]) : We used the ratios obtained by dividing the data of each data type (execution time,asymptotic function,basic operation count) by the first instance of that information type to graph the data. In other words, we represented all the data as a product of the first instance of that data. One can retrieve the actual values and units of the data by multiplying that data with the corresponding values included in the legend.

**GRAPHS AND EXPLANATIONS FOR ALGORITHMS:**

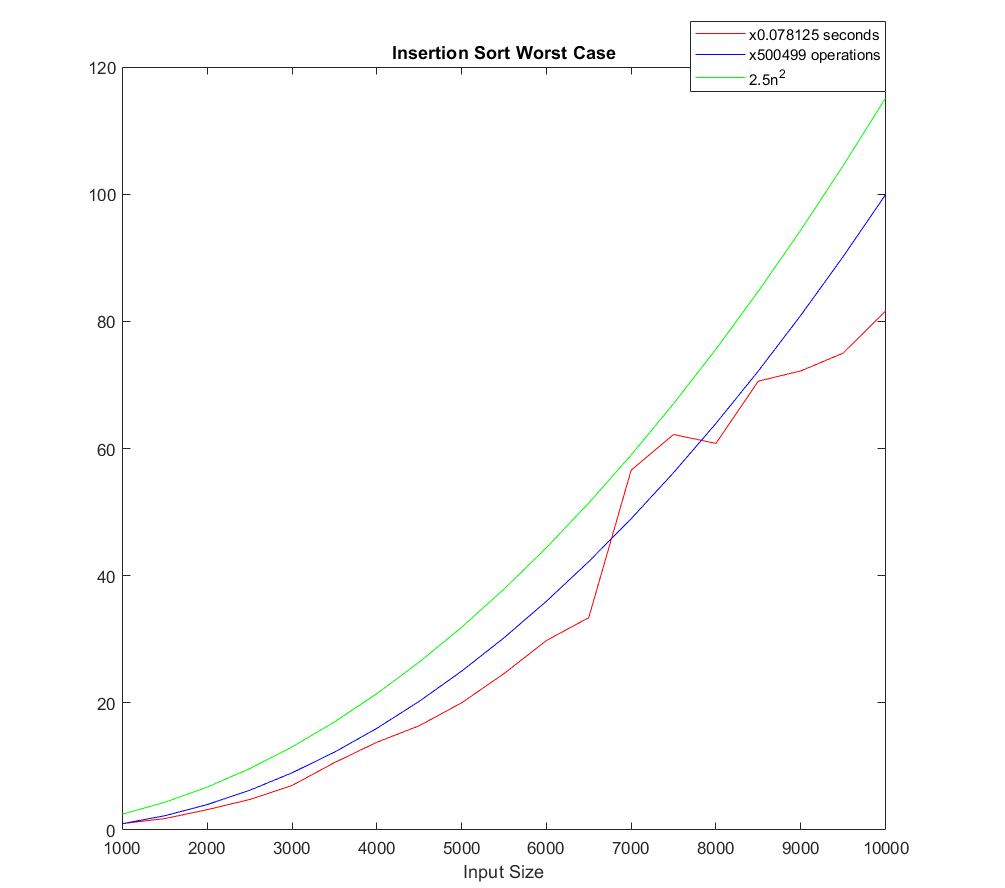
INSERTION SORT



The Average case of Insertion-Sort: The average case is when the list is random and unsorted. We gave started the input size from 1,000 and ended at 10,000 to see the rise in time clearly since Insertion sort has O() time complexity. We couldn’t directly compare it with algorithms that have long time complexity because when we tried to sort a list with 100,000 elements, Insertion sort couldn’t handle it but for example, Quick-sort could sort that list is merely a fraction of a second. What we understand from this is if the list is smaller insertion sort can be efficient but if the list gets larger it is not going to perform well.

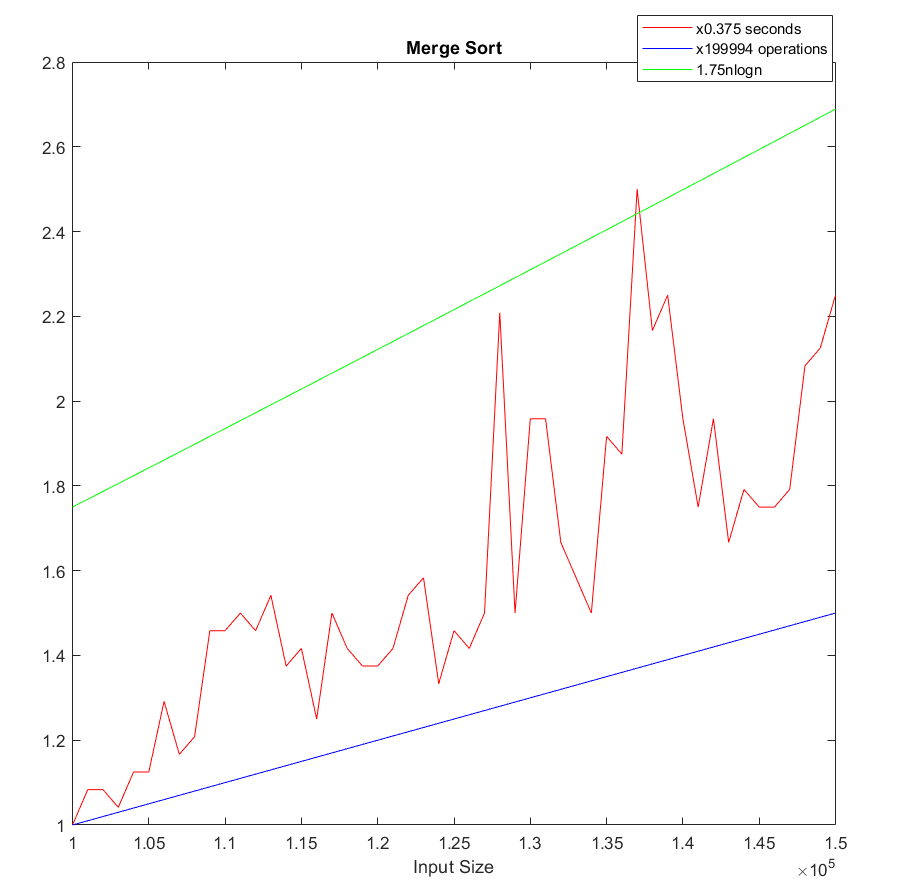


The Best case of Insertion sort: In this case, the given array was already sorted and we needed to increase the list size by 10 to get an execution time different than 0. It is very hard to comment on this since the graph has a very unusual pattern. The best case should take O(n) time and it is fast if you want to sort an almost sorted list.



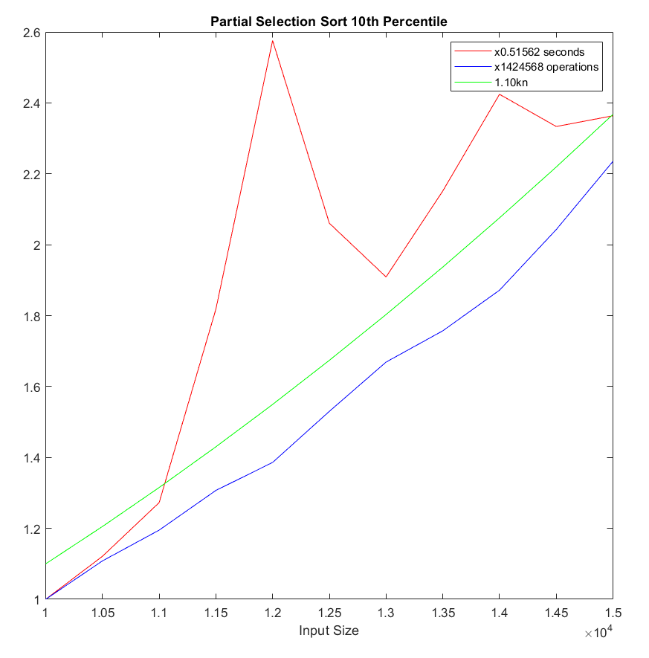
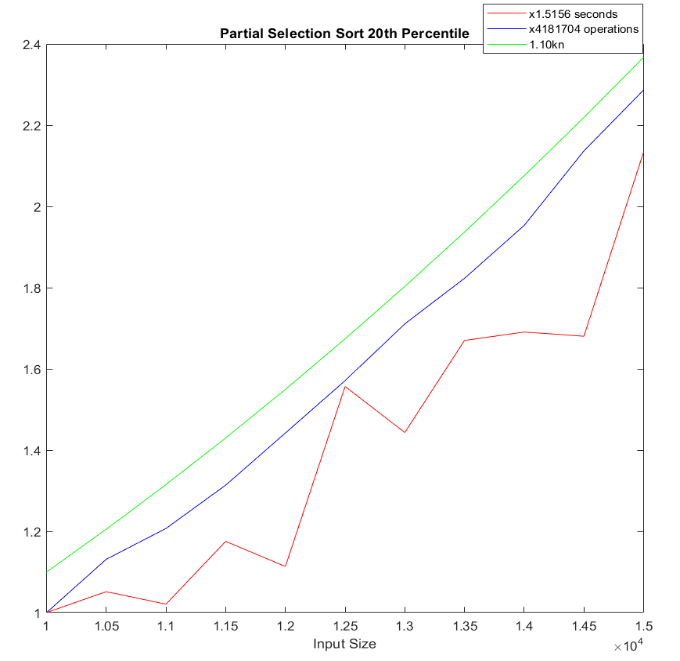
The worst case of Insertion sort: In this case the array is completely in decreasing order. It is similiar to the average case but of course slower than it.

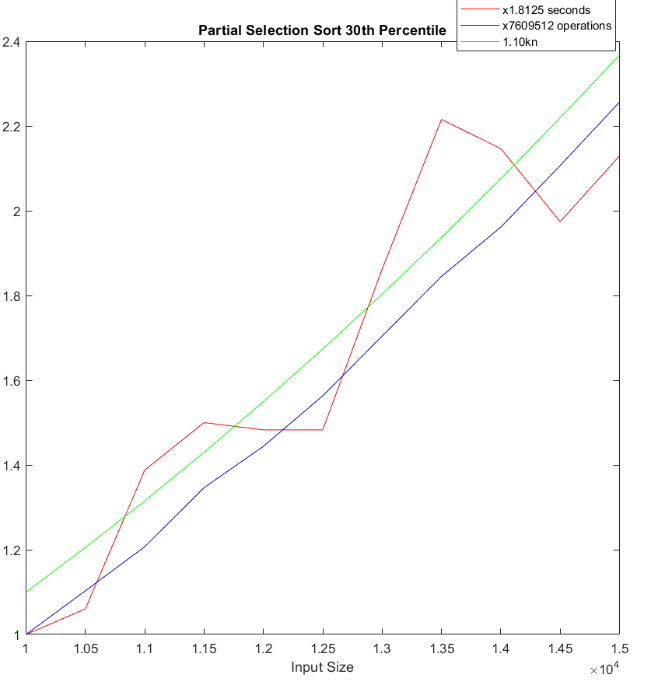
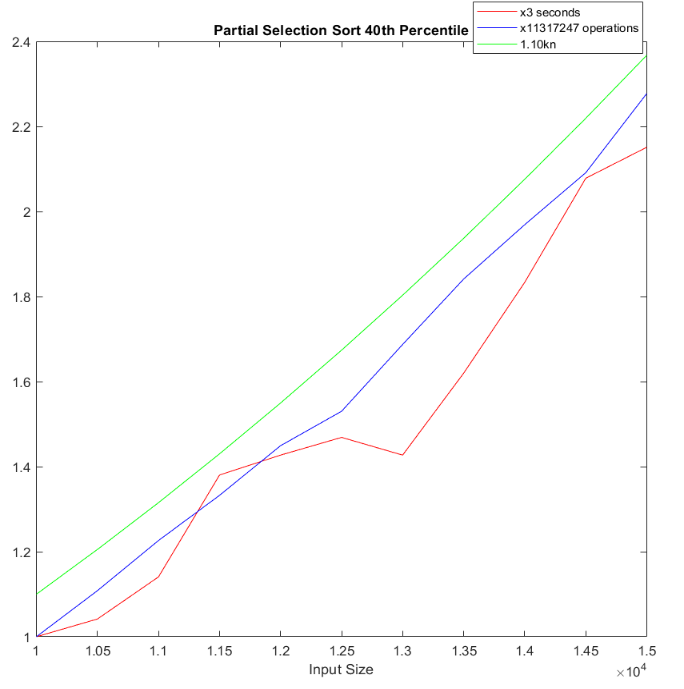
MERGE-SORT

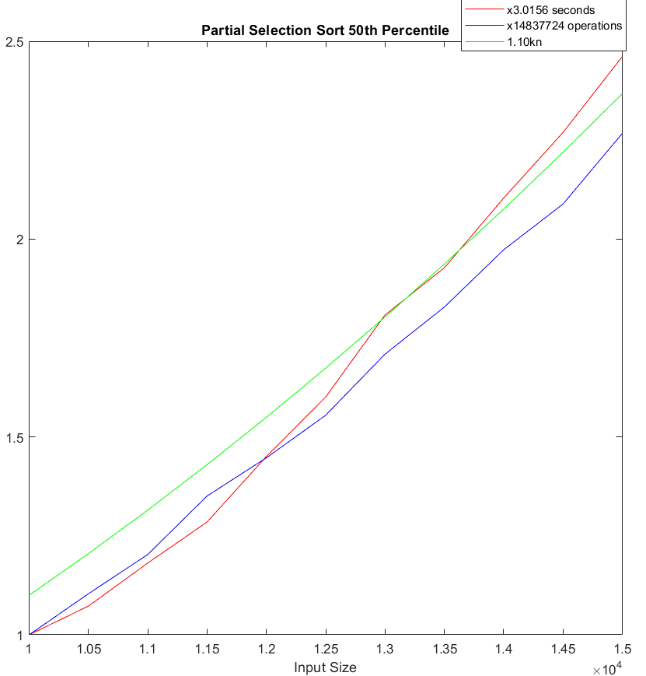
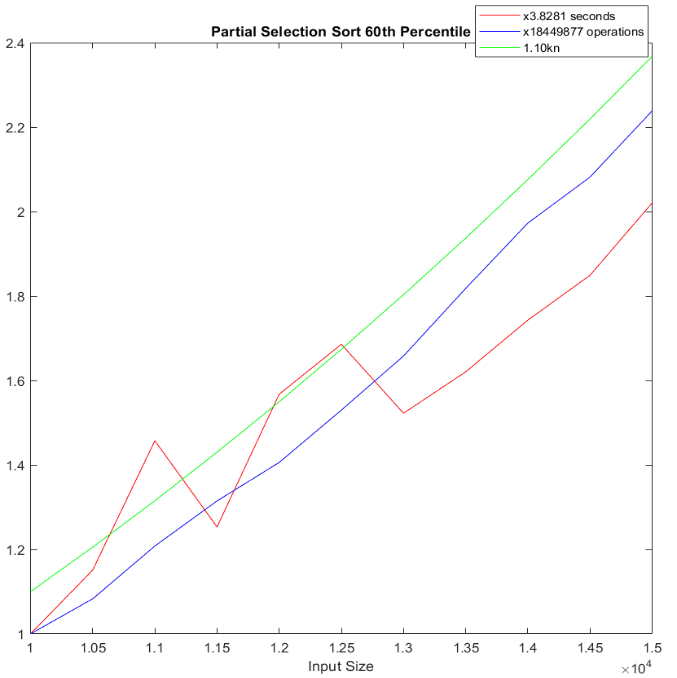


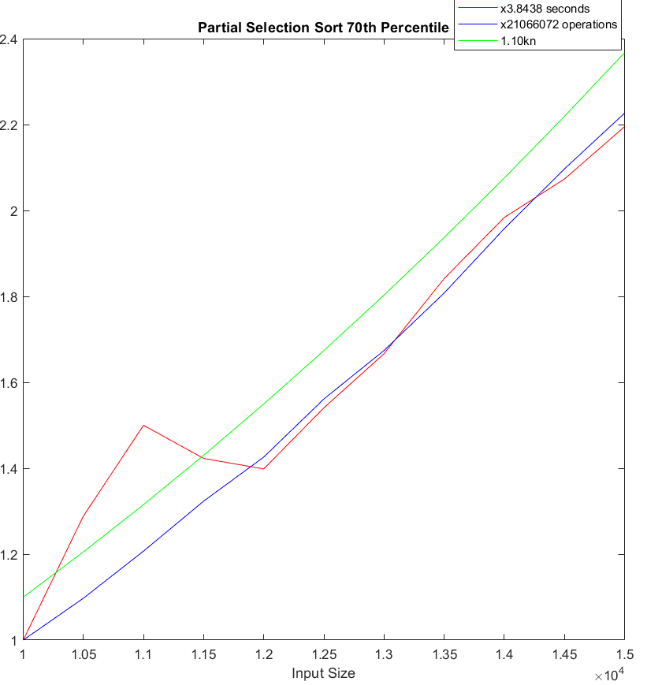
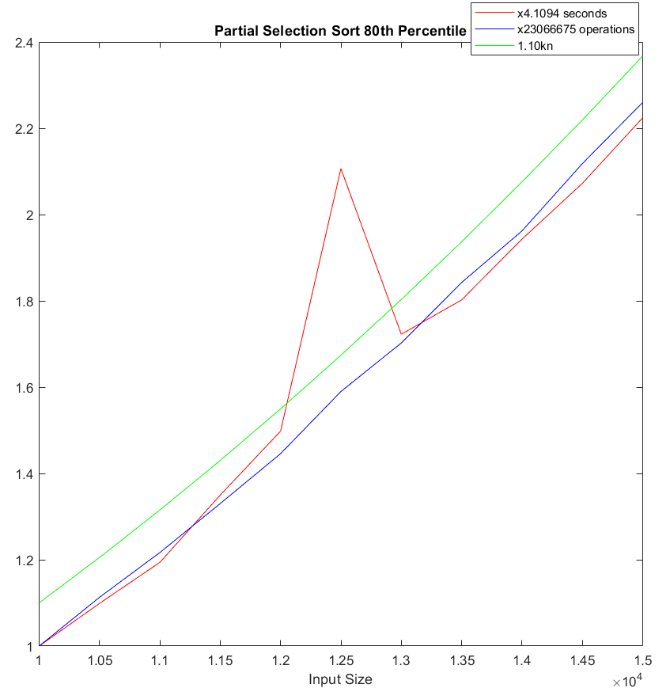
Merge sort is a very reliable algorithm with no worst or best cases. It has O(Nlog N) complexity. The algorithm is pretty consistent. We chose to input the same as Quick-sort. Merge sort is a little bit slower than Quick-sort but it is not a big difference and since Merge-sort doesn’t have a worst-case like Quick-sort it may be more useful in some situations.

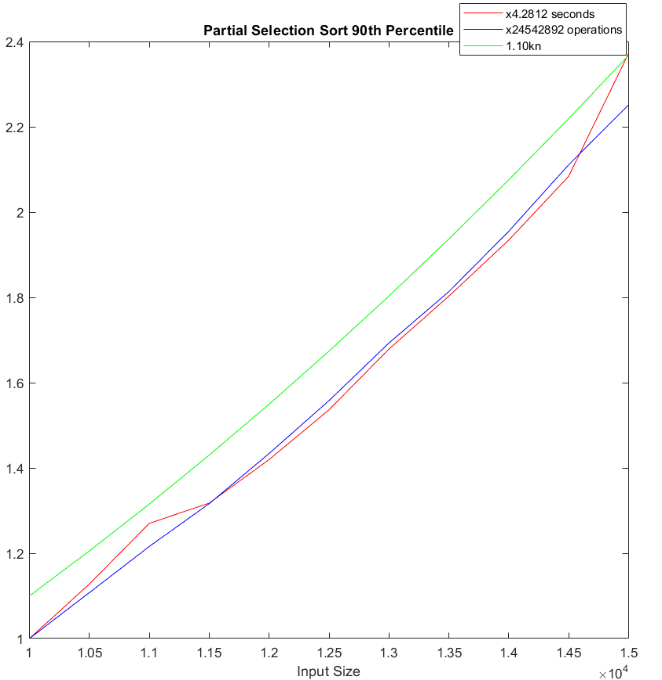
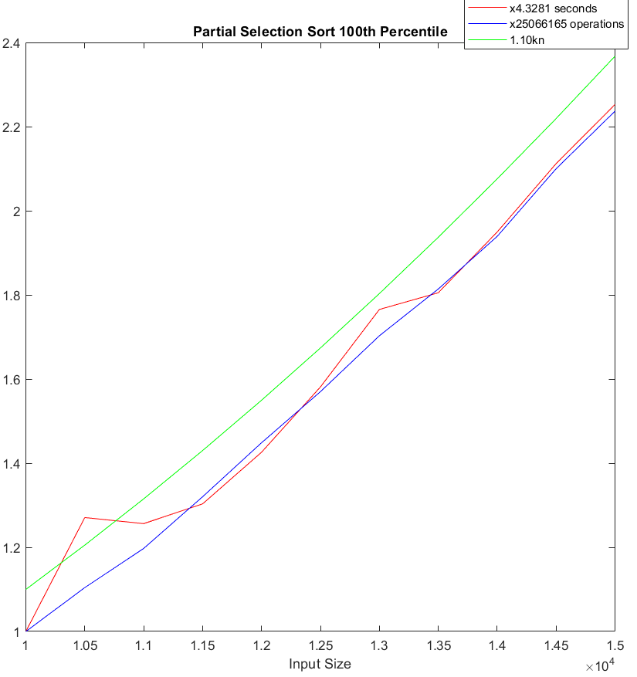
PARTIAL SELECTION SORT WITH PERCENTILES

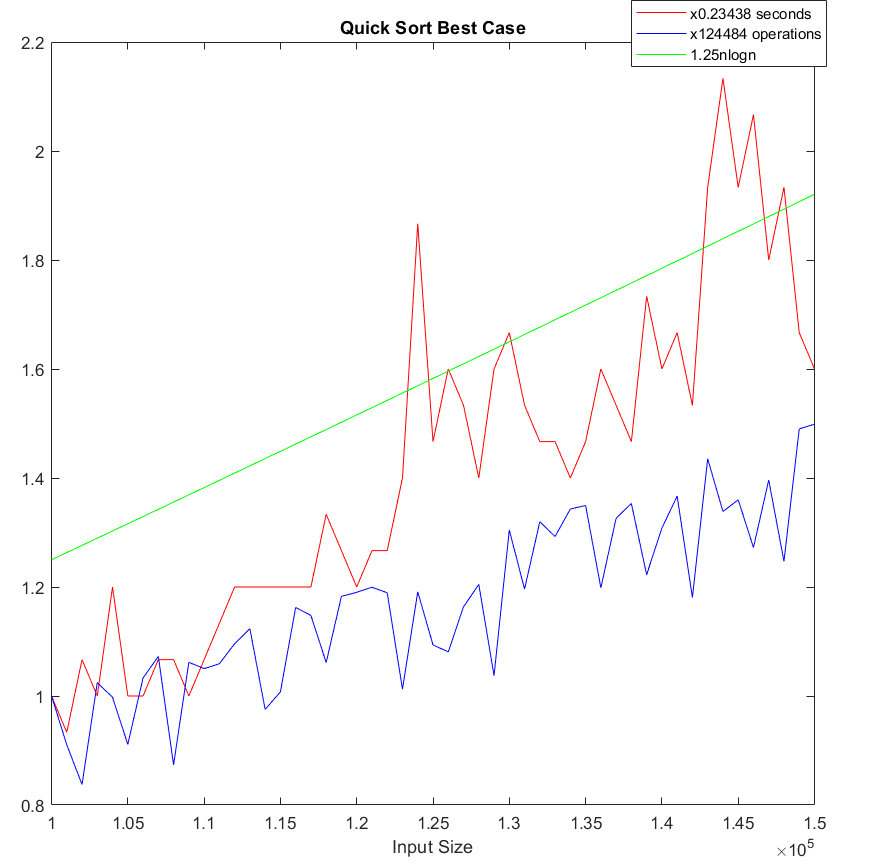
 

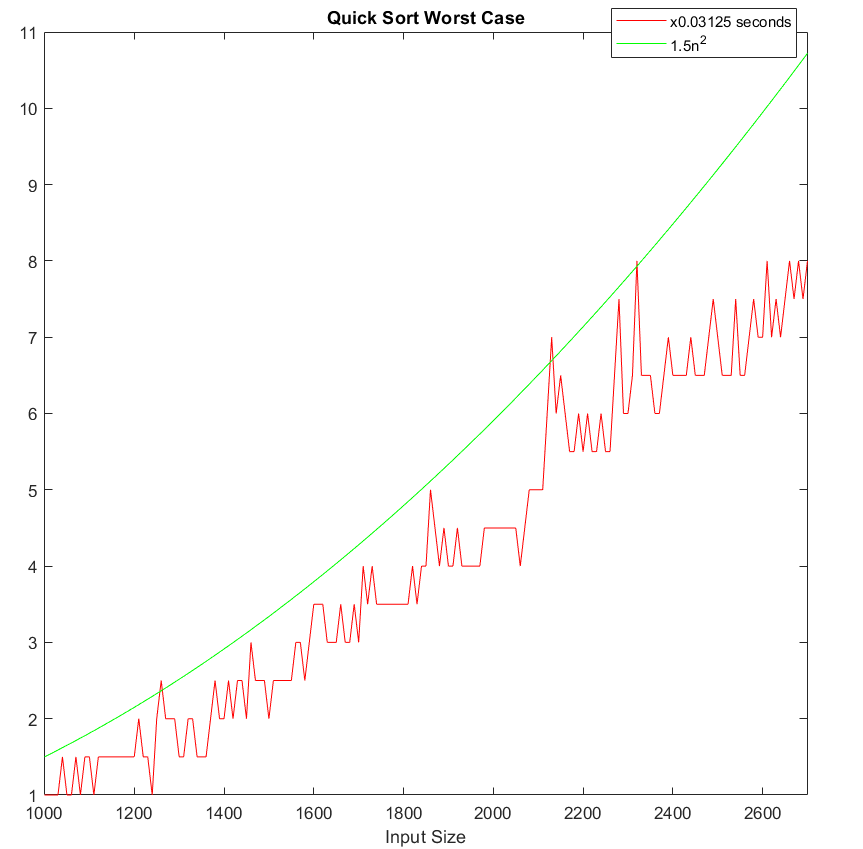
 

The partial selection sort has O(k\*n) time complexity. We divided the partial selection sort into 10 percentiles. As shown in the graphs it is pretty accurate to the expected values most of the time. There are some spikes in some percentiles but mostly it fits the expected values.

QUICK-SORT

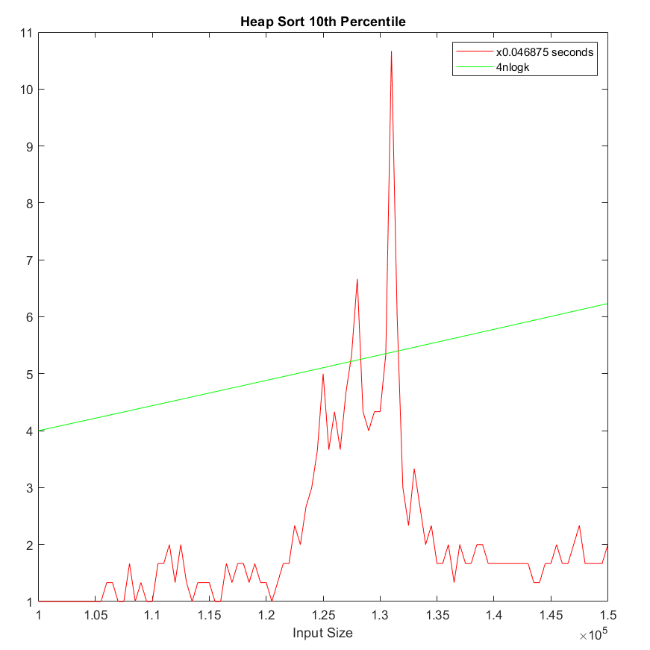
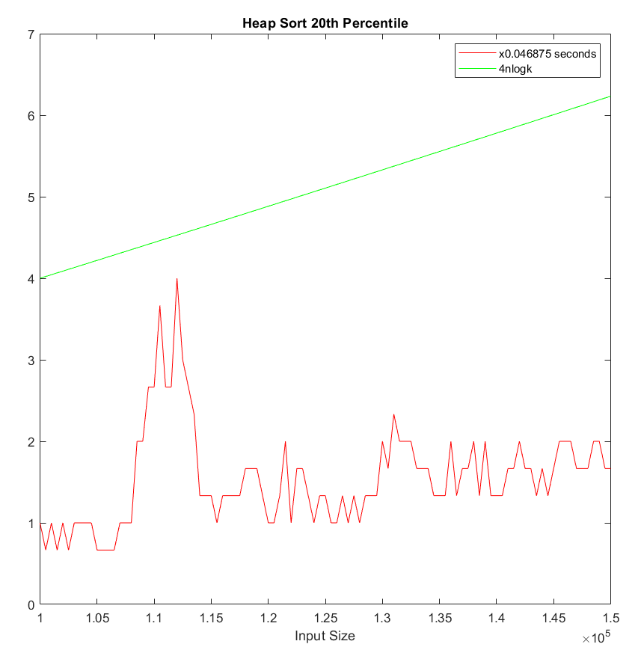


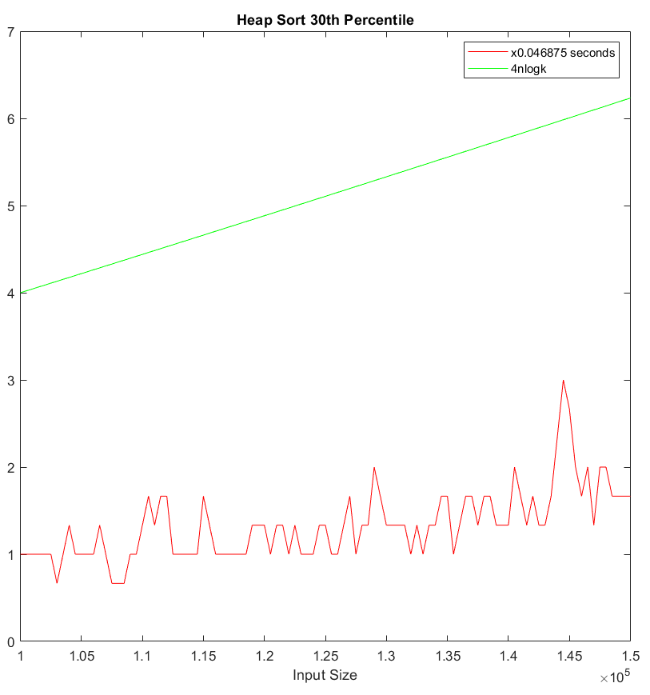
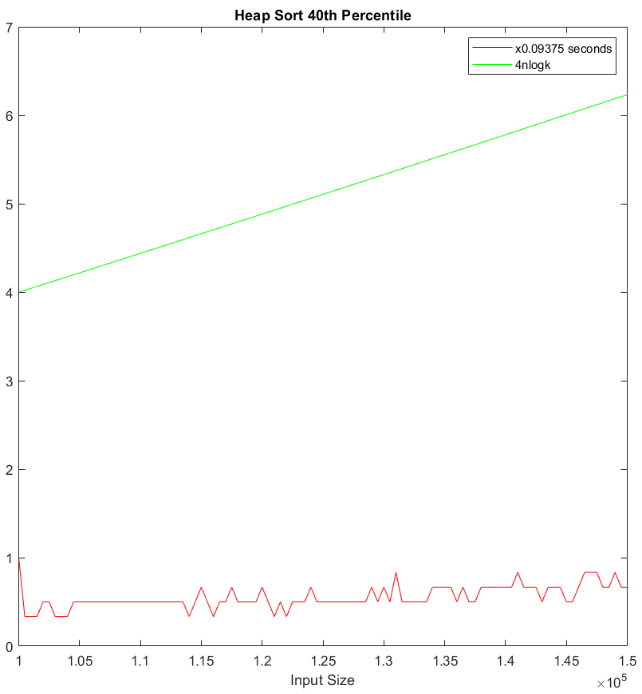
**BEST CASE:** Quicksort's best and average case is the same and it is a completely random unsorted list. It is an extremely fast sorting algorithm, way faster than other algorithms we used to completely sort a list. It has time complexity O(Nlog N). We chose the first element in the list as the pivot element and did the partitioning according to that. The spikes and weird shapes in the graph are most likely due to the performance of the computer used to sort the lists and the randomness of the inputs. We started the list size from 100.000 and ended at 150.000. While we were testing we sorted lists with size 1 million and quicksort could do it at about 3 seconds minimum which is very fast. However, we decided that we don’t need that large. The reason why execution time is above the basic operation is because in quicksort there are many operations other than basic operations which also increases the execution time.

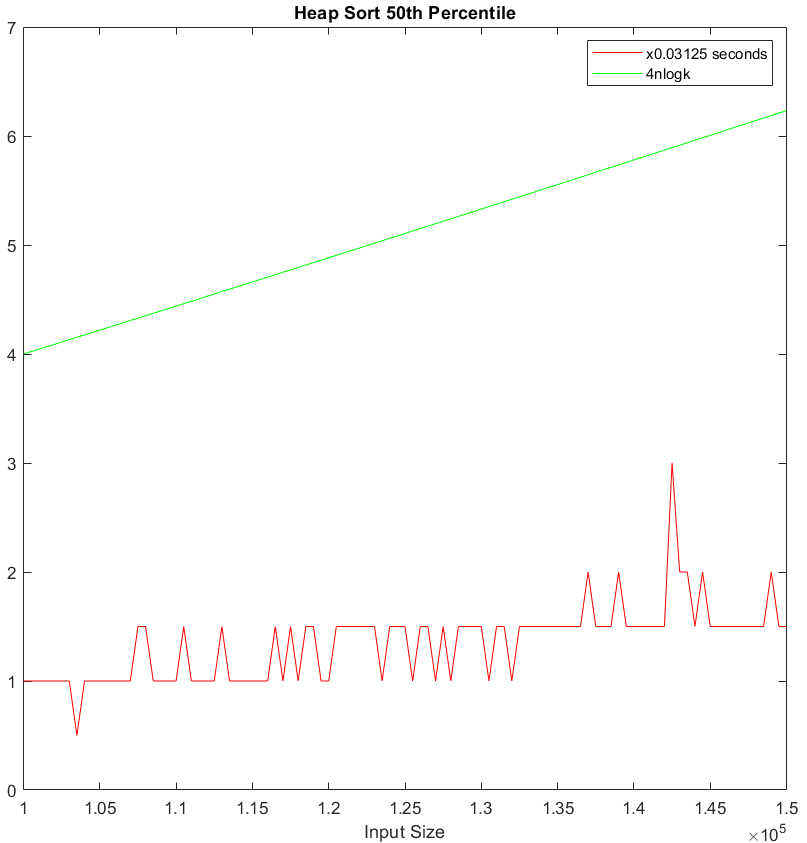
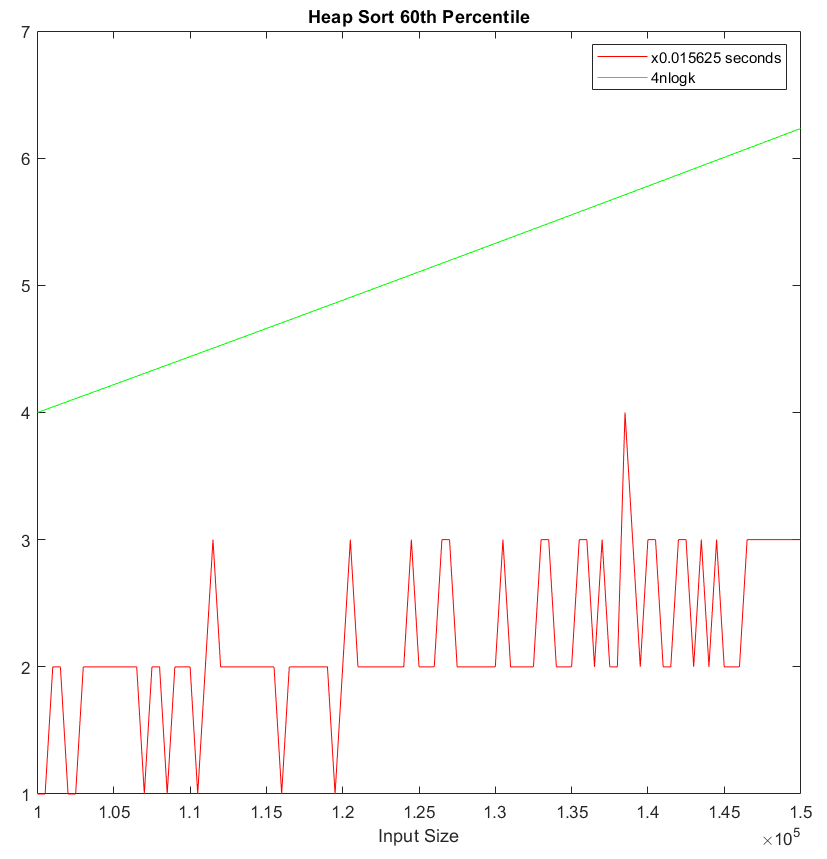


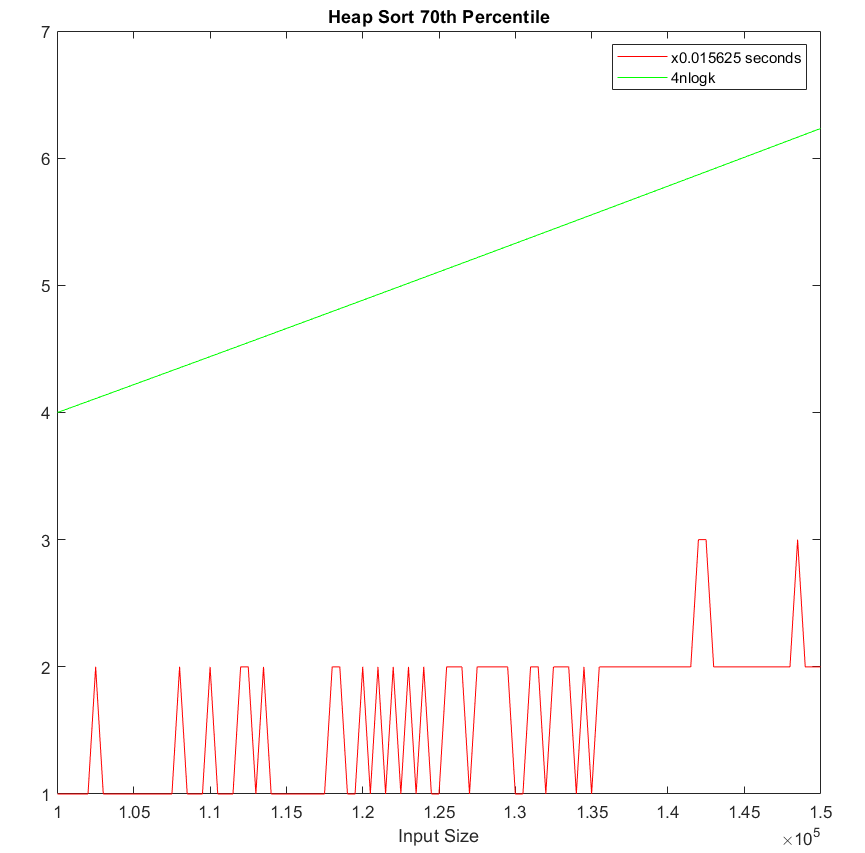
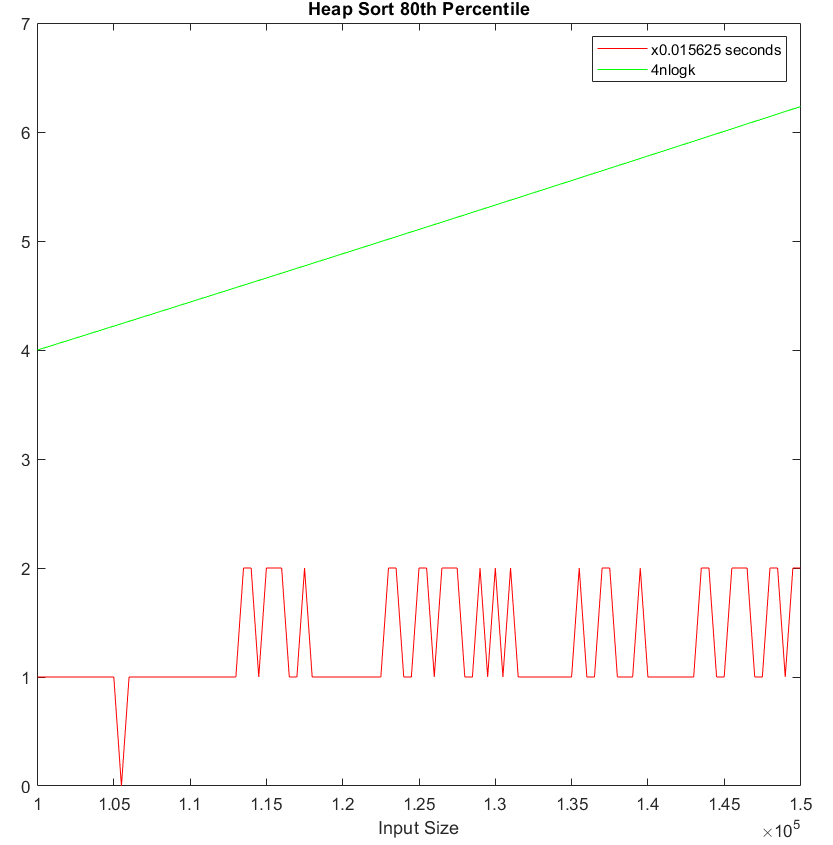
**WORST CASE:** Quick-sort’s worst-case has O() time complexity and it happens when the list is already sorted. This has a lot of problems on its own. Firstly, the algorithm performs very poorly in the worst case. We also encountered another problem which is recursion limit and stack overflow. Python by default has a 1000 recursion limit to prevent stack overflow. Quick sort’sworst case was easily passing this limit and it was giving an error. We increased that limit to 2800 but we still encountered stack overflow and couldn’t run the algorithm on a list with a size more than 2700.

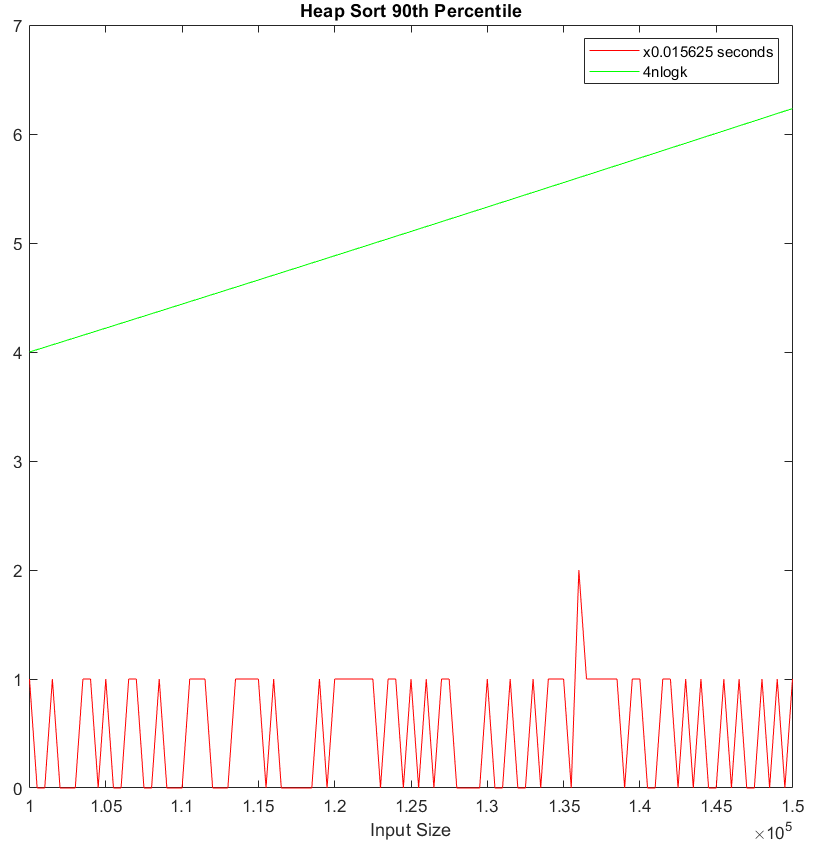
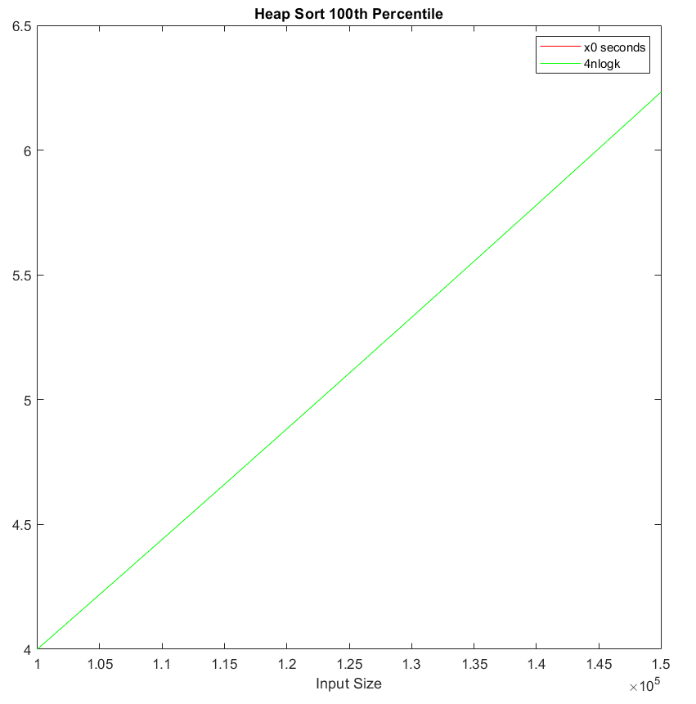
PARTIAL HEAP SORT

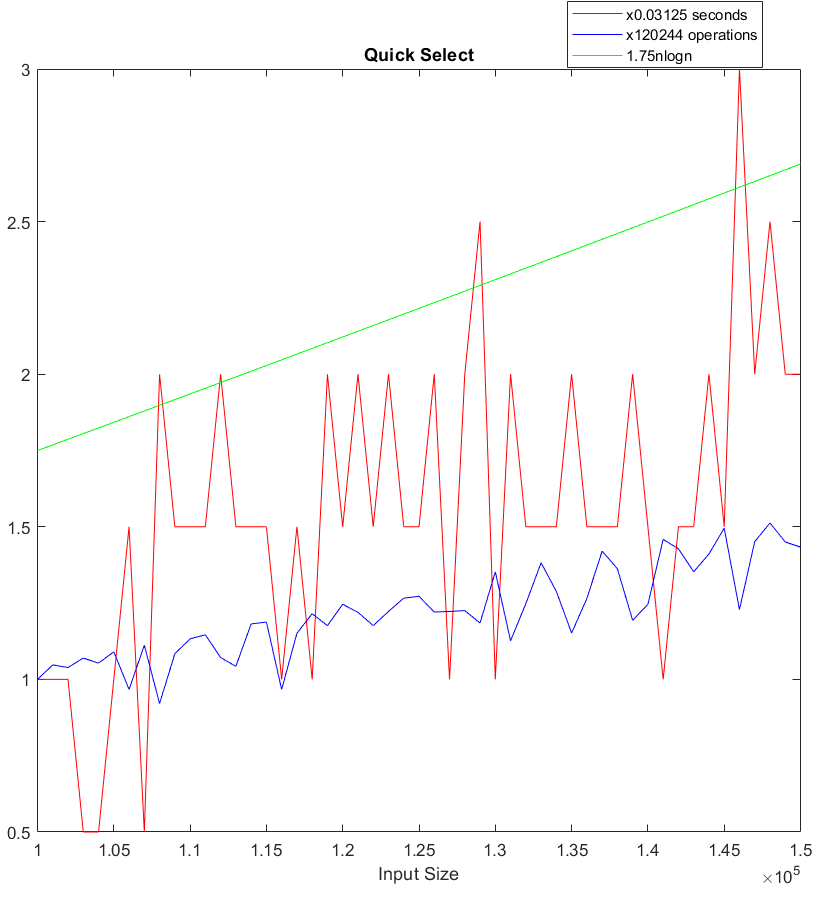
 

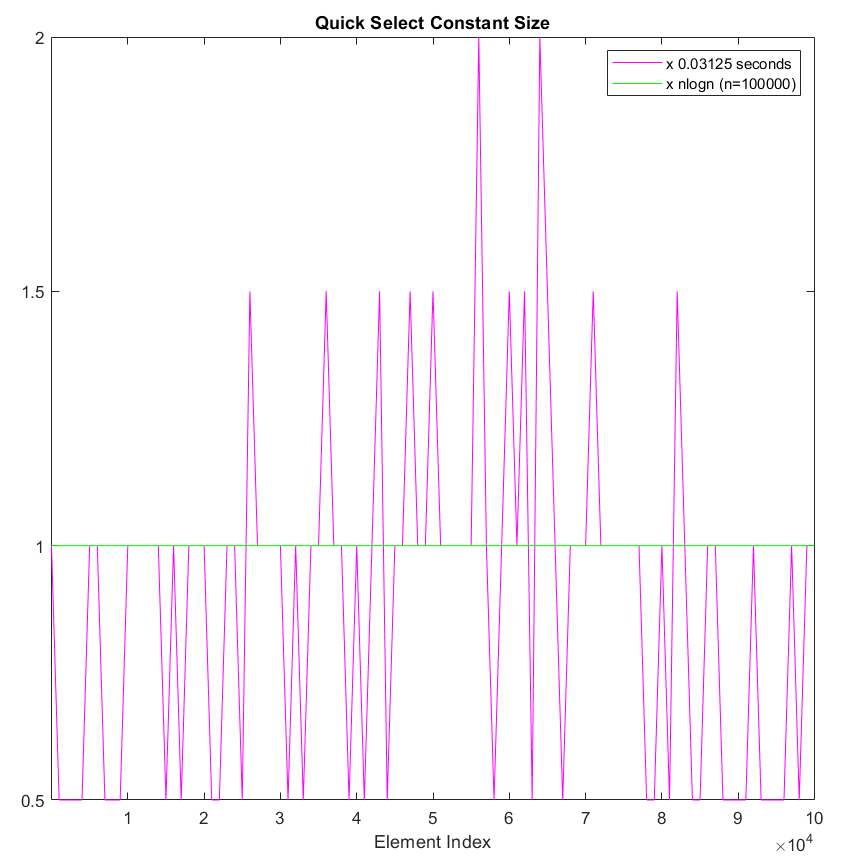
 

Partial Heap sort has O(N log K) time complexity. Partial heap sort is removing the root of a max heap n-k times and returning the root element. Because of this, as we go to the higher percentiles, it gets faster and faster to the point it takes almost 0 seconds to perform. The spikes at the first percentiles are because of the inconsistency of the computer's performance. The spikes at the higher percentiles are because the times are extremely close to 0 seconds and it's hard to measure accurate.

QUICK SELECT

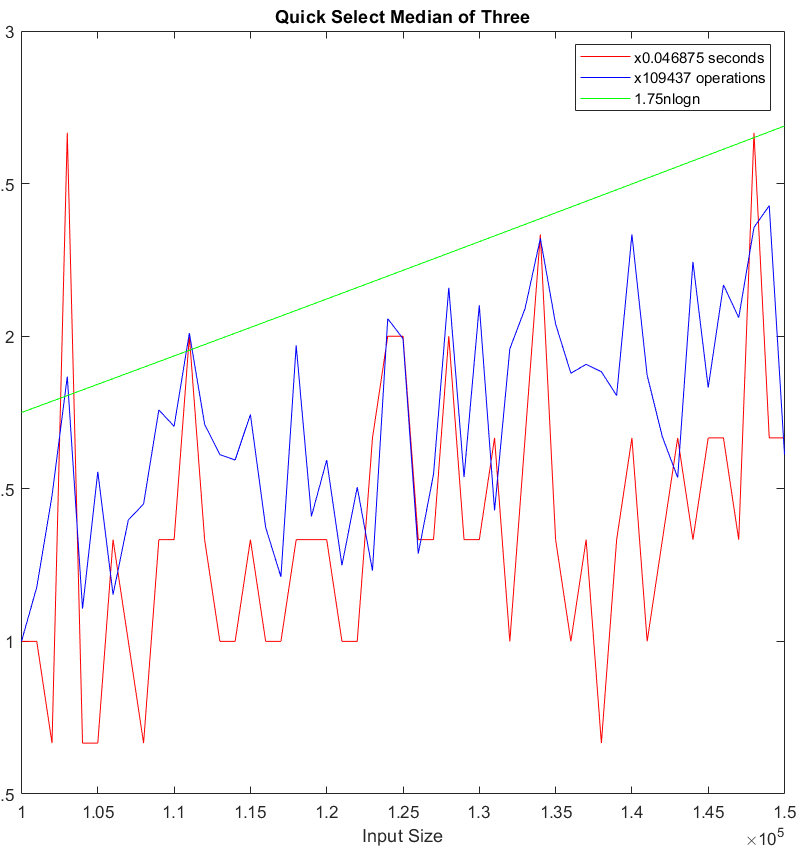


Quick select is using the same partitioning as quick-sort and is closer to O(n) but it has a worst case of O() . In a completely random unsorted list, if the partitioned element appears to be the smallest or largest element in the list, it is the worst case. It is definitely fast to pick a single element in a large array.



As we can see here quick select is not related to the index of the element . The gaps seem a lot in this graph but it is because it all takes very small time and they are very close to each other. No matter which index we choose, quick select shows similiar performance each time.

QUICK SELECT WITH MEDIAN OF THREE PIVOT SELECTION



We think that the main benefit of median of three pivot selection is getting rid of the worst case. With pivot being the median of first , middle and last elements median it can never be the smallest or largest element. In this way it is more consistent than normal Quick select with first element or last element as the pivot.

.

**Conclusion:** Depending on your goal, you may want to choose different algorithms for different purposes because each one has an advantage over the other at some point. There are also lots of factors affecting the speed of an algorithm. Although many of them meet their theoretical expectations, we have seen that some take more time than their basic operations and computer performance is not always consistent.

**References:**

<https://www.geeksforgeeks.org/insertion-sort/>

<https://www.geeksforgeeks.org/selection-sort/>

<https://www.geeksforgeeks.org/quick-sort/>

<https://www.geeksforgeeks.org/merge-sort/>

<https://www.geeksforgeeks.org/quickselect-algorithm/>

<https://www.geeksforgeeks.org/heap-sort/>

<https://stackoverflow.com/questions/7559608/median-of-three-values-strategy>