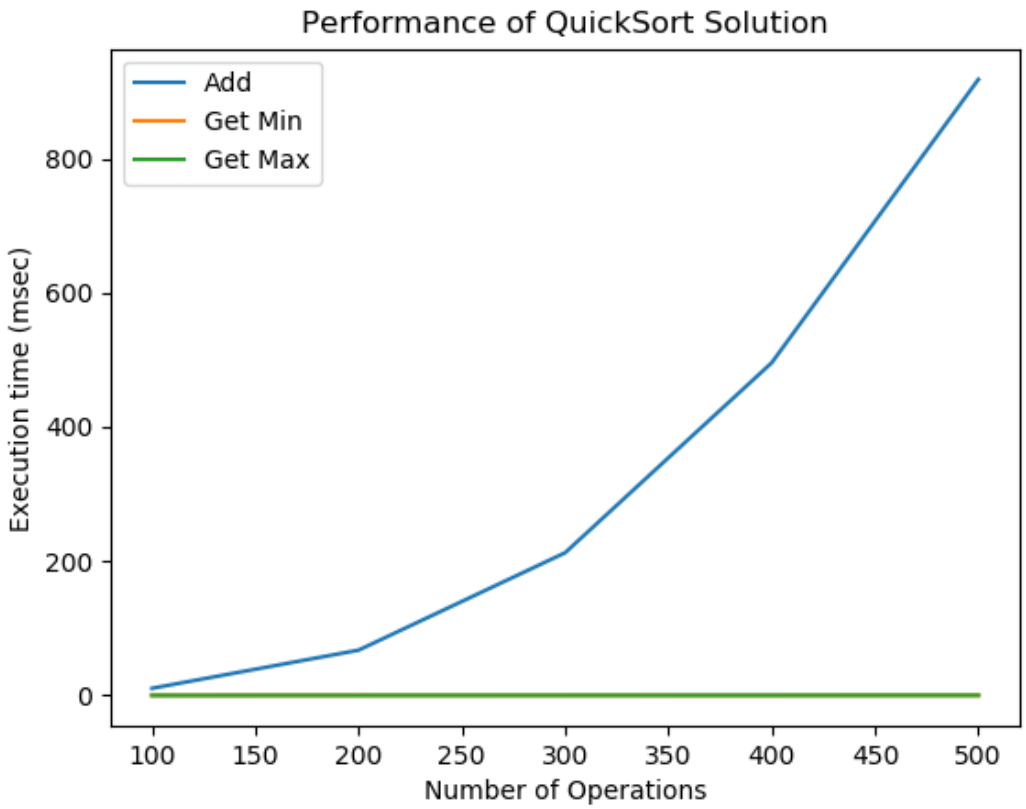
PCS2-Hw3

The aim of the project was to run a benchmark analysis of four different sorting algorithms: QuickSort, Binary Search Tree, HeapSort and BubbleSort.

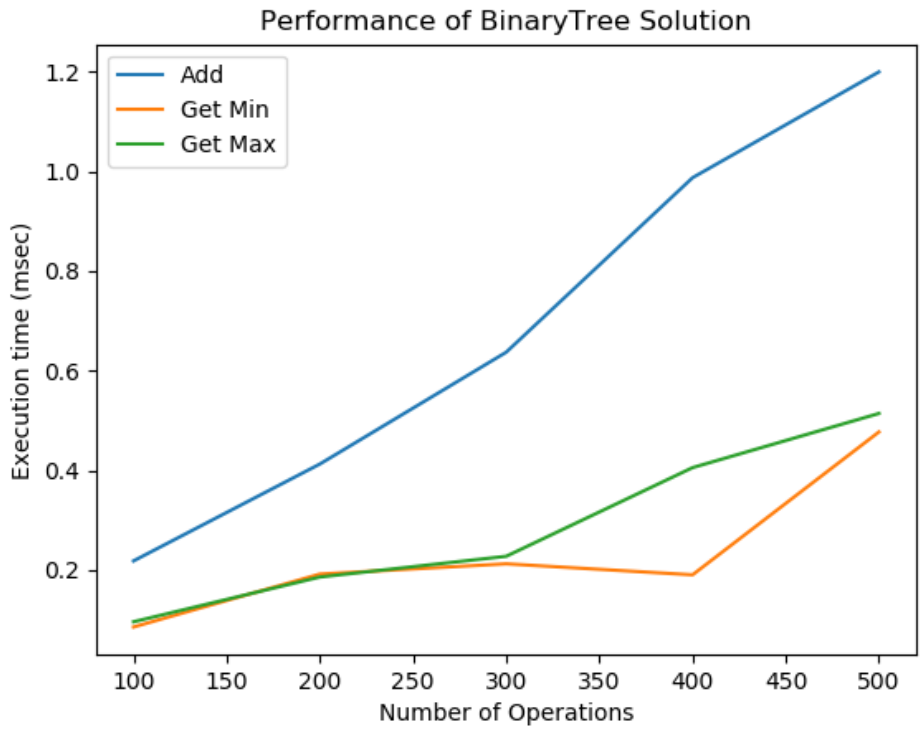
These alghorithms have been tested on random generated list of *n* elements. The performance of these algorithms are strictly related to the hardware of the system on which it has been runned. In this particular case :

- *Intel Core i7-7500U up to 3.5 Ghz*
- *8192 MB DDR4 Dual-Channel*

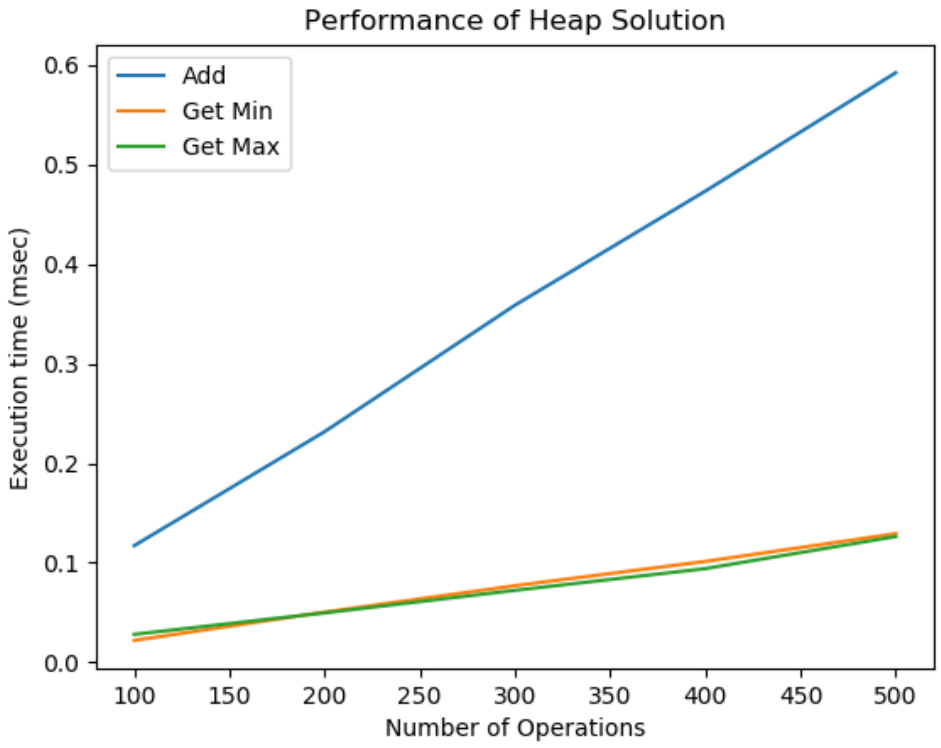
The following plots will describe the performance of each algorithm, in particular, will be showed the results in terms of execution time and number of operations.



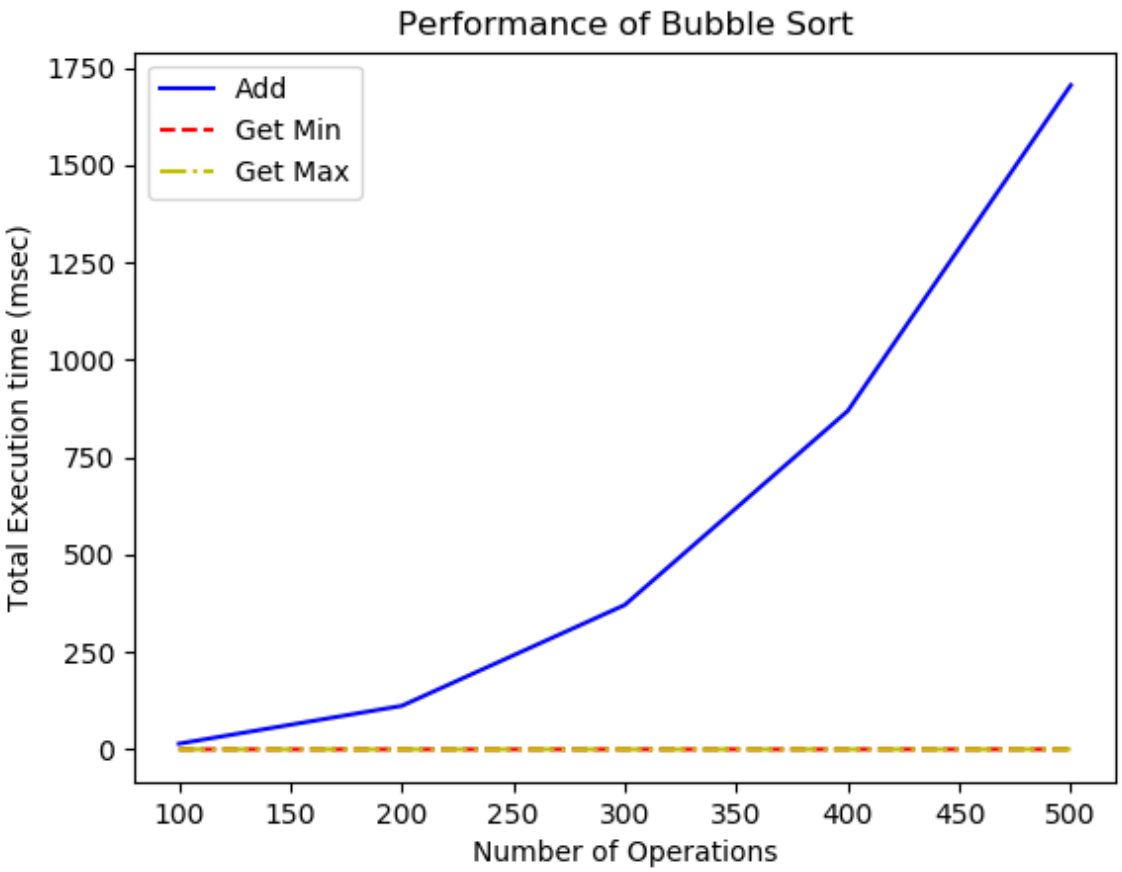
As we can see in the plot the execution time of getting both the minimum and maximum value remains almost inalterated whith the increase of the number of operations, the action of adding new items instead becomes slower.



The execution time of all the operation showed in the graph increase in an almost linear way with more operations, the add function, also in this case, increase faster.

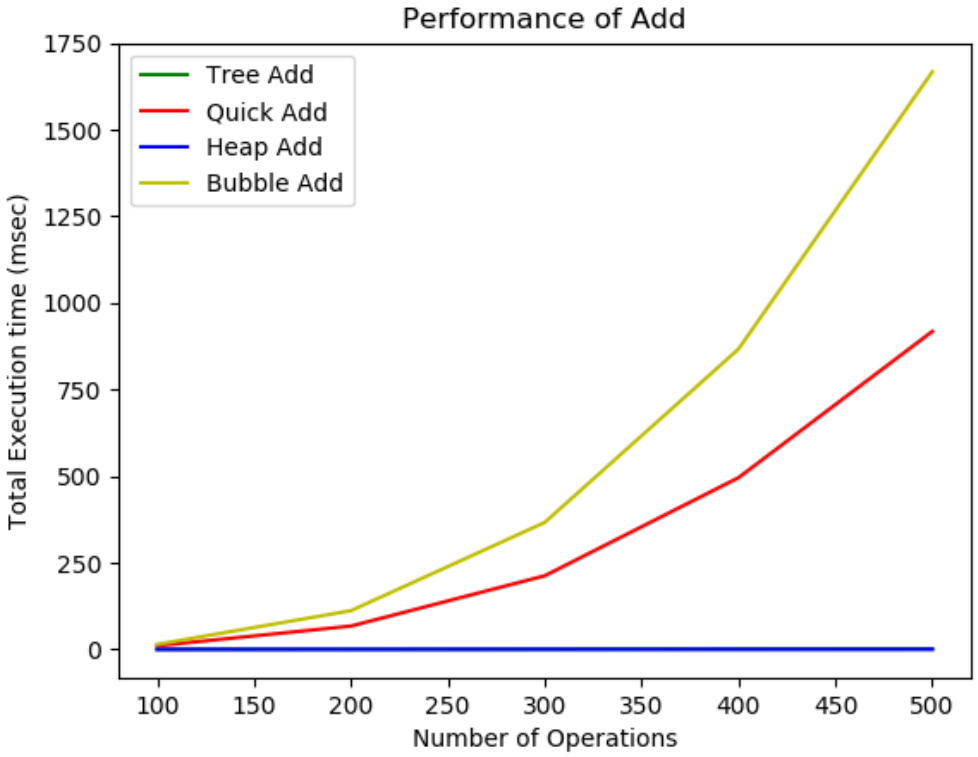


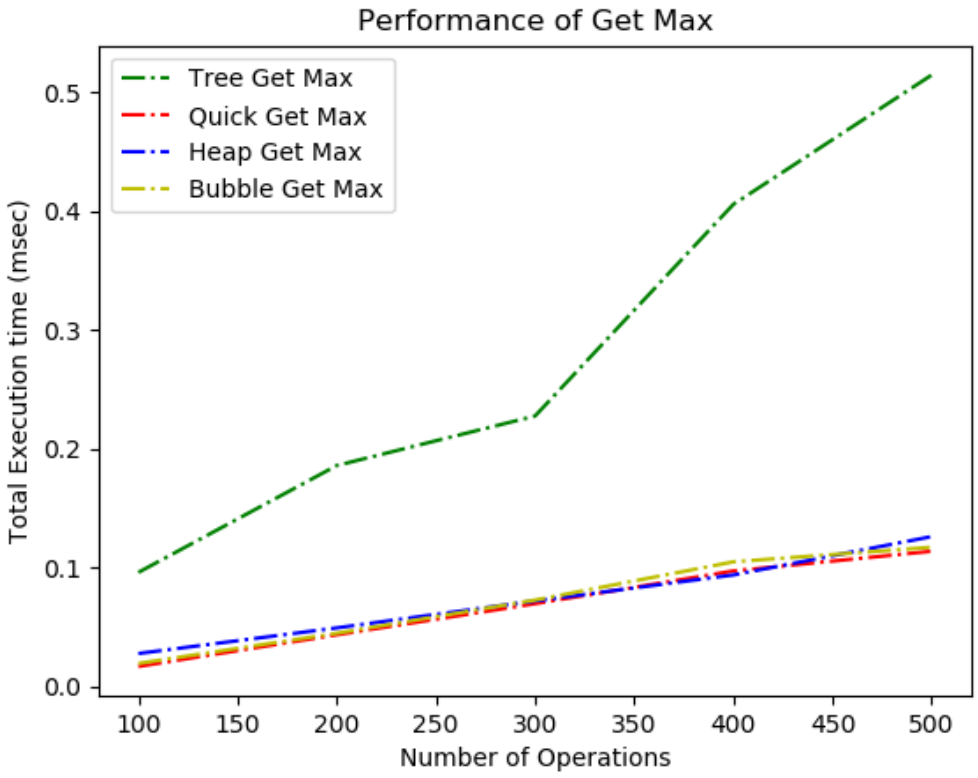
The result of the Heap algorithm is almost perfecty linear, both for the add operation and the operation of returning minimum and maximum values.

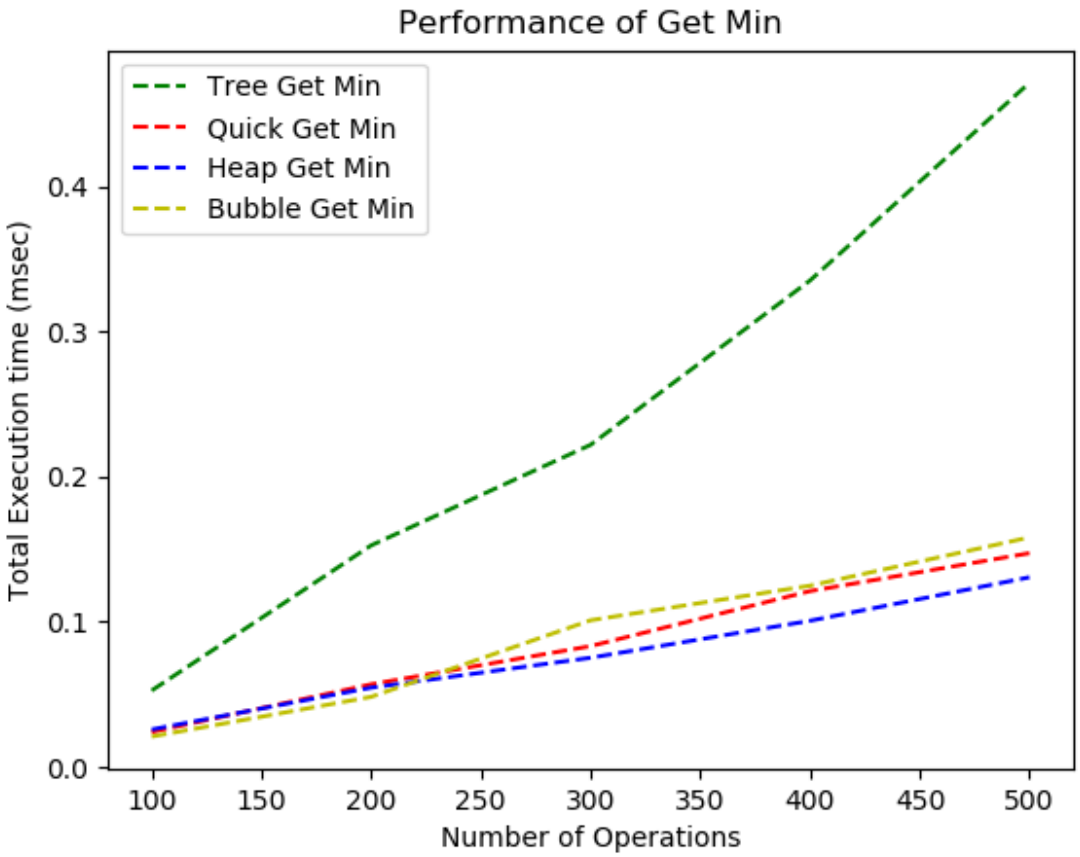


Like the QuickSort algorithm, the BubbleSort one has an increase in time of the add operation in a esponential-like way, the other two operations are nearly unaffected by the increasing of the number of operations.

Overall Performances







The above graphs represent the results of the benchmark of all the four different algorithms.

As we can see in the first one the execution time of the add operation of both Heap and Binary Tree functions are nearly the same, instead the QuickSort and the BubbleSort algorithm both takes more time to add new elements.

With the last two graphs we can clearly see differences in the execution speed time to get the minimum and the maximum value in all the four functions. As showed the speed of the Binary Search Tree function is strictly related to the number of operations runned, so this function is much more slower with high number of operations.