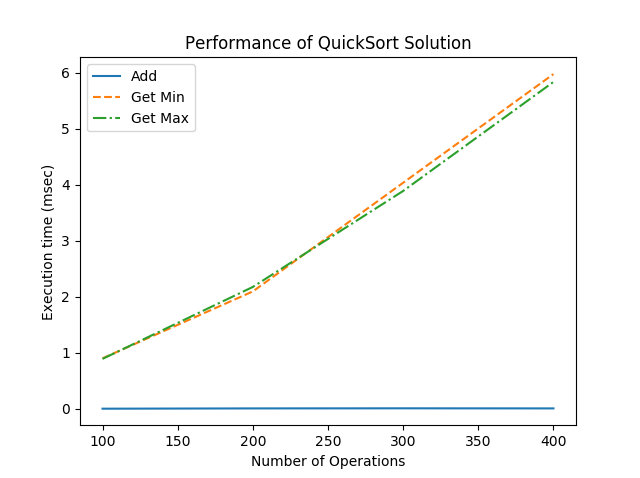
In this project we are taking in consoderation four sorting algorithms, to see which one can give us the maximum and the minimum value of a sorted random list, within the shortest possible time. The four functions are:

* QuickSort
* HeapSort
* BubbleSort
* Binary Search Tree
* **QuickSort Analysis**

As we discussed in the previous project, QuickSort is Divide and Conquer algorithm; it takes last element as pivot (which is an element that divides the list of numbers in two smaller list), places the pivot element at its correct position in sorted array, and places all smaller (than the pivot) to its left and all greater elements to its right, through the partition() process.



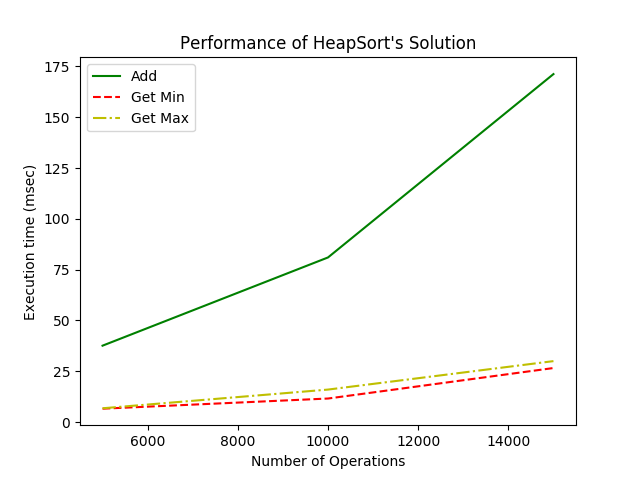
It’s time complexity is ***nlogn*** for all the cases (best, average and worst) where n is the number of items being sorted.

As we can see from the graph, *Addi s* the fastest function, while *Get Min* and *Get Max* are the slowest ones.

* **HeapSort Analysis**

HeapSort is an in-place sorting algorithm, more specifically is a comparison based sorting technique based on Binary Heap data structure (a special case of balanced binary tree data structure where items are stored in a special order such that value in a parent node is greater(or smaller) than the values in its two children nodes).

Given an array to sort, Heapsort first transforms the keys of the array into a heap. This one is then sorted by repeatedly swapping the root of the heap with the last key in the bottom row, and then sifting this new root down to an appropriate position to restore heap order.



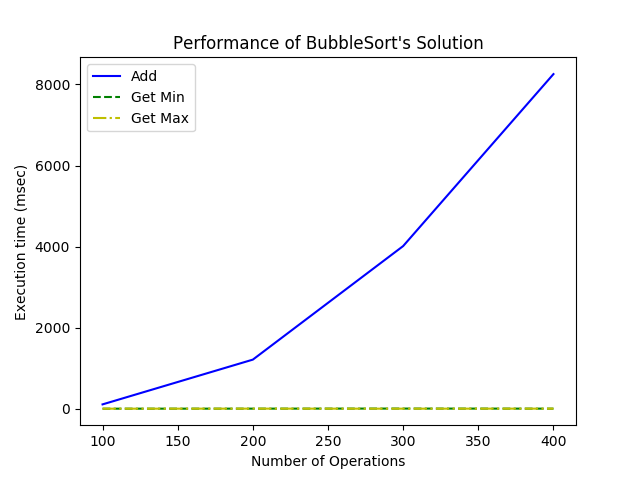
Its time complexity is ***nlogn*** for all the cases (best, average and worst).

As we can see from the graph, Get Min and Get Max are faster than Add.

Heap sort is an in-place sorting algorithm but is not a stable sort because it doesn’t retrieve the same order of equal elements in the sorted array.

* **BubbleSort Analysis**

Bubble Sort is the simplest sorting algorithm used to sort a given set of n elements provided in form of an array with n number of elements by comparing adjacent pairs and [swaps](https://en.wikipedia.org/wiki/Swap_(computer_science)" \o "Swap (computer science)) them if they are in the wrong order until the list is sorted (if n elements are given, the process is repeated n-1 times). It doesn’t require extra memory.



Its time complexity is

* ***n*** for the best case;
* ***n2*** for both average and worst case;

where n is the number of items being sorted.

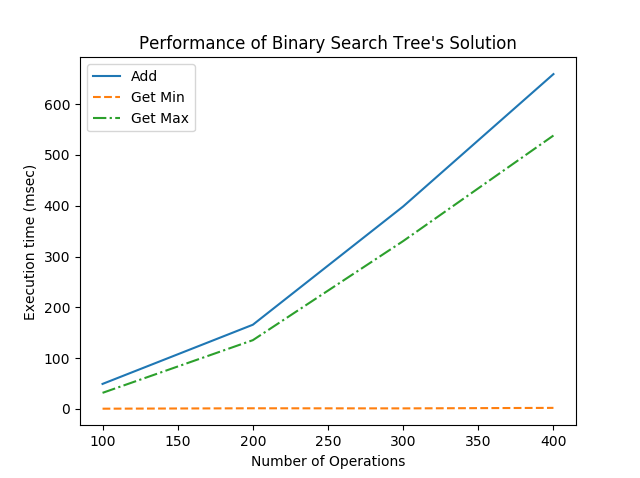
As we can see from the graph, both *Get Min* and *Get Max* are faster than Add.

This algorithm is generally too slow and not useful for the majority of the problems; it can be used only with lists that are mostly sorted (without ordered elements in near positions).

* **Binary Search Tree Analysis**

Binary search tree is a node-based binary tree data structure; each of its nodes store a key. For each node, we have to different sub-trees (left and right). The tree satisfies the following properties hich has the following properties:

* The left subtree of a node contains only nodes with keys lesser than the node’s key;
* The right subtree of a node contains only nodes with keys greater than the node’s key;
* The left and right subtree each must also be a binary search tree.



Binary Search Tree time complexity:

* ***n*** when we have the best case;
* ***nlogn*** when we have the average case;
* ***n2*** when we have the worst case;

as we can see from the graph, *Get min* represent is the fastest one, *Get Max* is tending to *Add* which isthe slowest one.

At the end of this analysis, we can say that between the array sorting algorithms (QuickSort, HeapSort and BubbleSort) the fastest is QuickSort; while the Binary Search Tree is faster compared to the array sorting algorithms.