This report explains how **Bubble Sort, Binary Tree Node, QuickSort, Add, Get Max, Get Min, Python Heap** functions, **time execution** and the various **Plots** of the algorithms were carried out.

* **Bubble Sort** is the easiest sorting algorithm that works by repeatedly swapping the adjacent elements if they are not in order.
* **Heap** is a special tree structure in which each parent node is less than or equal to its child node. It uses an array to represent the heap, the array then becomes smaller and the sort is finished.
* **Tree Node** is made up of nodes, each node contains a left reference, a right element and a data element. A binary tree can only have two nodes but can also have no nodes at all.
* **Quick Sort** is a divide and conquer algorithm, it picks an element as pivot and partitions the array around the selected pivot. It always selects the first element as the pivot.

The **import** function is a library of functions that imports random, heapq, matplotlib and time if these aren’t imported would automatically give an error.

**import** heapq= gives access to smaller values  
**import** time = measures the time taken  
**import** random = generates a random value from a certain number given  
**import** matplotlib.pyplot **as** plt = creates plots or graphs of functions

A **class** for the four algorithm is defined with the **\_\_init\_\_, add, get\_min, get\_max** variables and a parameter **self** is assigned to them except for the case of add where another parameter called **‘value’** is added which adds value to the list.

The class of Binary Heap always has a parent node with two parameters (self, i). The **self.heaplist** explains if one is bigger than the other or less, a new variable j is formed which is equal to the previous variable i.

**Time EXECUTION**

For the time measurement execution, time is being defined with the total add, min and max being called to zero, the for loop is implemented and iterates through the list then the three parameters (add, min, max) are then returned. The for loop iterates through the numbers in the list, “start” is equal to the list of lists where the add, min, max, is added to the first called variable a, the total time of executions gives.

The Execution Time is: **0x00000260AEF476A8**

Using different numbers (3,5,7,9) to test if the code works when ran show different ascending orders of result below:

[5] 5 5

[5, 7] 5 7

[3, 5, 7] 3 7

[3, 5, 7, 9] 3 9

[5] [-5] 5 5

[5, 7] [-7, -5] 5 7

[3, 7, 5] [-7, -5, -3] 3 7

[3, 7, 5, 9] [-9, -7, -3, -5] 3 9

[5] 5 5

[5, 7] 5 7

[3, 5, 7] 3 7

[3, 5, 7, 9] 3 9

The maximum number of operations is 1000, step = 200, repetitions = 5, this explains the number of operations in this case 1000 times and five repetitions.

The values of Bubble, Add, GetMin, Getmax is assigned to an empty list, the for loop is used for the range of the operations and steps: the list appends (which adds to a list) with the random function (which is imported) and a random integer from 0 to 1000. The total time for add, min, max is equal to zero for each of the functions, the values of the given variable goes from 0 – 1000 from the execution time which starts from the origin (0) to a given number required. The execution time is repeated for all variables to differentiate how much time they all take and is represented in the graphs below.

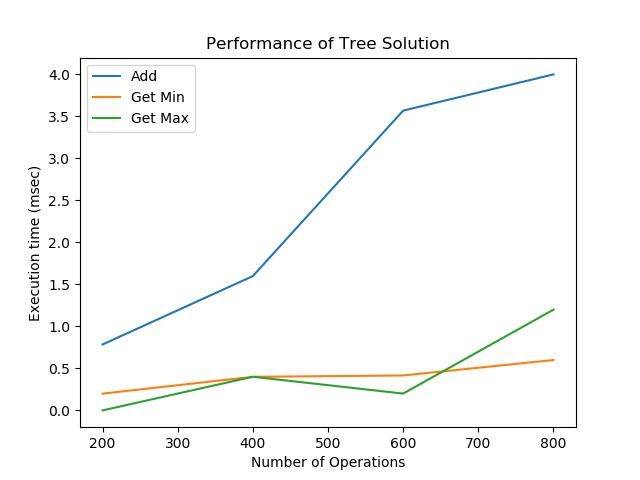
* **GRAPH**

**On the X axis :Number of Operations.**

**Y Axis : the Execution time (msec)**

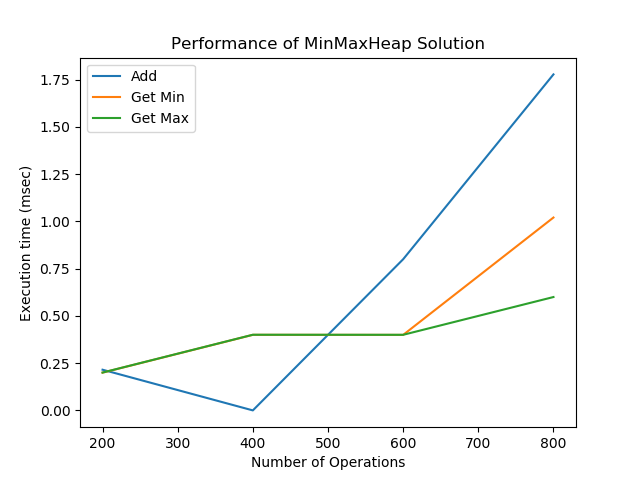
* **Tree Solution**

this graph includes **add, get min and get max** indicated with specific colours (blue, red and green). The **add** increases from the 1.0 to 4.0, **get max** increases a little above 0.5 decreases to 0 and increases again while **get min** increases then becomes constant.



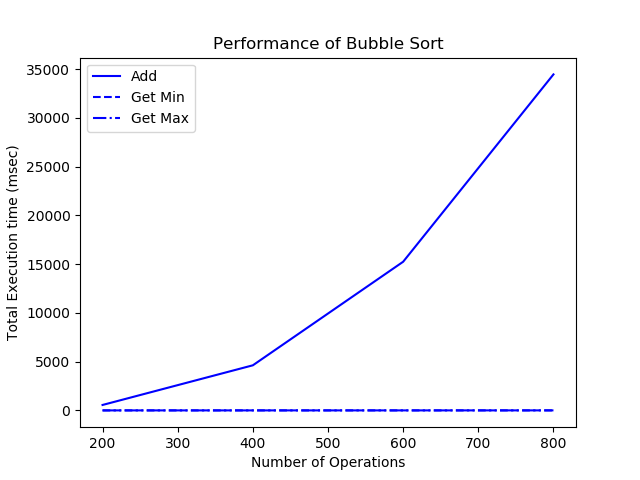
* **MinMaxHeap**

In this graph, add increases from 0.25 ,decreases to 0 then increases rapidly, get min increases, get max , increases, decreases a little and increases again.



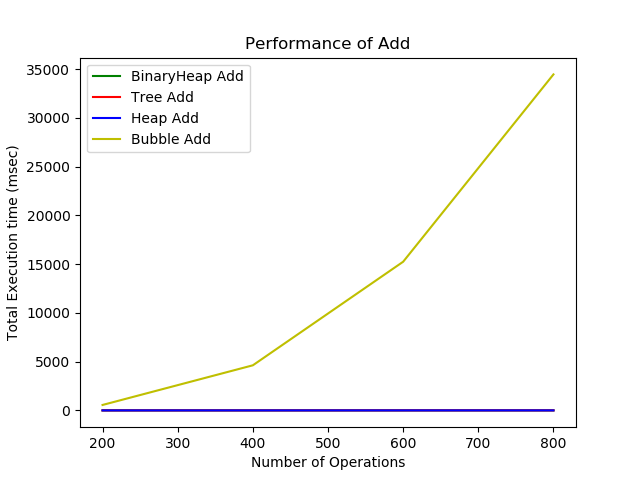
* **Bubble Sort**

The get min (- -) remains constant even as the number of operations increases while add increases fast. The total execution time increases by 5000 while add increases continuously.



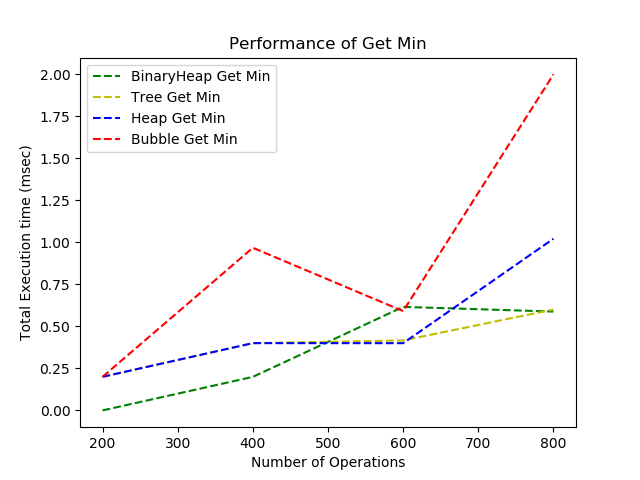
* **Add**

The Bubble add(g) increases as the number of operations increases while Heap Add(b) remains constant even as the number of operations increases



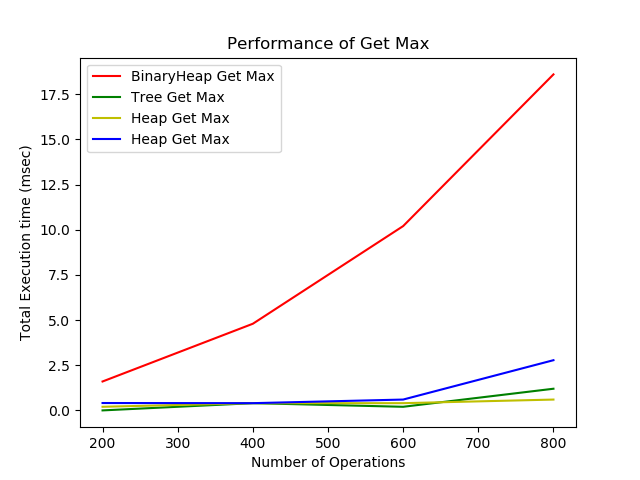
* **Get Min**

The Execution time changes with 0.25, Bubble Get Min(r) increases, decreases and increases again, Heap Get Min(b) increases , decreases a little , becomes constant then increases , Binary Heap Min(g) increases and decreases.



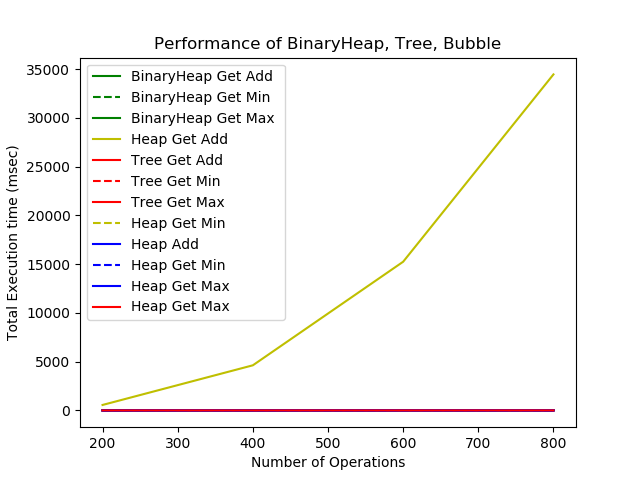
* **Get Max**

Heap Get Max (y) does not increase or decrease, Tree Get Max(g) and Heap Get Max(b) stays constant then after the 600th number of operations increases while BinaryHeap(r) increases as the number of execution time increases.



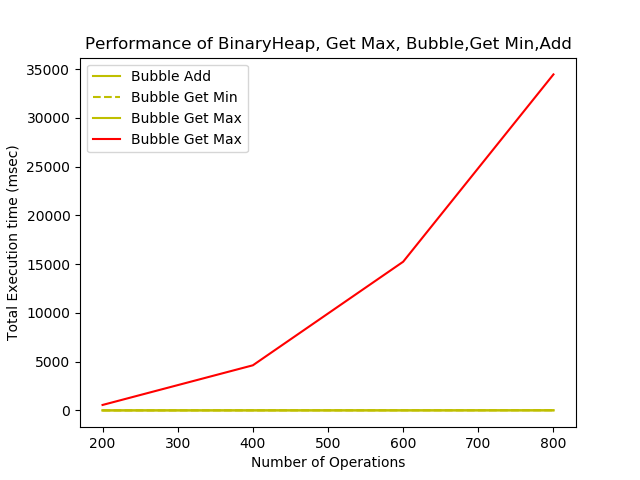
* **BinaryHeap , Tree, Bubble**

Heap Get Max (r), Tree Get Max and Add(r) are all constant while Heap Get Add(y) increases rapidly.



* **BinaryHeap, Tree, Bubble, Add**

Bubble Get Max (r) increases rapidly, while Bubble Get Max (y) and Bubble Add(remains constant



In conclusion, all the graphs vary from one other because the various execution time changes , some increase after 5000 while some after 0.25. The Get Min graph and Tree graph are the best because they show all the four algorithms.

SophiaCO