**REPORT ON QUICKSORT, BUBBLESORT, BINARY TREE AND HEAP CLASSES.**

**INTRODUCTION:**

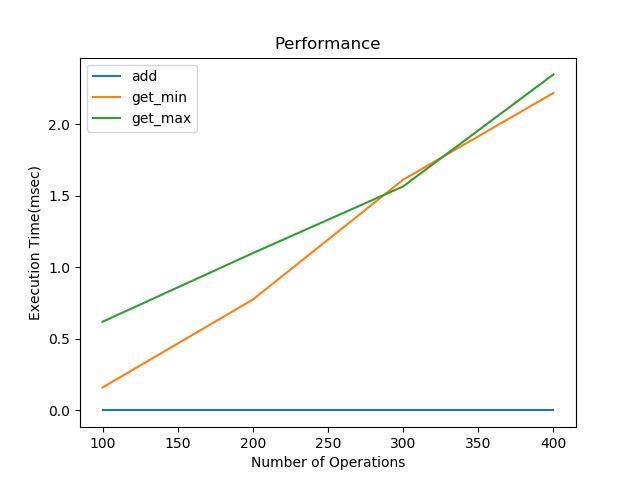
Here I am writing a report on quicksort, bubblesort, heap and binary search tree functions ; their performance according to various random list.

The goal of my work is to create a benchmark analysis for each of the classes, so the comparision of their performance, their graphs and implementations could be more understandable.

The main target of my implementation is to create a class for each function and to keep track of the minimum and the maximum value in a list of number which is random. The reason for which I am using a random list is to produce a result which is not bias. Inside every class you can find three common operations: add, get min and get max. Then I am measuring the execusion time. According to that, a graph is produced to have a schematic representation of their performance for better understanding.

**Quicksort:** quicksort is a comparison sort, meaning that it can sort items of many type for which a “less than” relation is defined. It is an efficient sorting algorithm. When implemented well it can be two or three times faster than its main competitors mergesort and heapsort. Quicksort is a divide and conquer algorithm. Below there is a table and a graph representing the performance of the function.

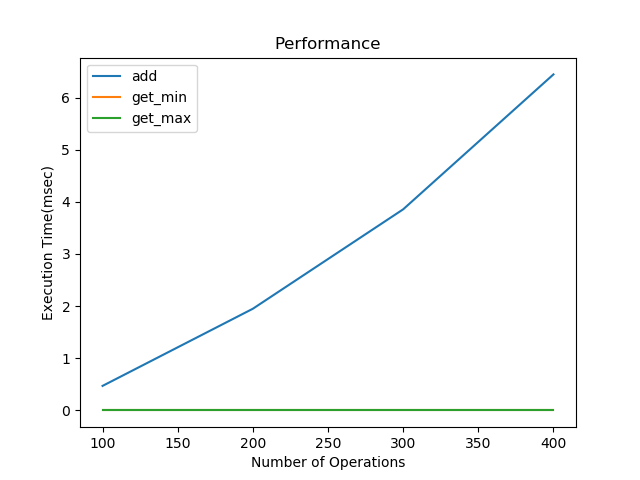
|  |  |  |
| --- | --- | --- |
| Time | | |
| Number of Operations | add | get min | get max | |
| 100 | 0.0 | 0.0017199993 | 0.0015600013 | |
| 200 | 0.0 | 0.0034499979 | 0.0042850017 | |
| 300 | 0.0 | 0.0089433312 | 0.0092866690 | |
| 400 | 0.0 | 0.0176199972 | 0.0173825025 | |



From the table and graph you can see that the time for add is constant which is the best case for this function. For get min and get max the case is different. Increasing the number of operation also increases the execution time. It is a worst case for the function.

**Bubblesort:** sometimes referred as sinking sort, is a simple [sorting algorithm](https://en.wikipedia.org/wiki/Sorting_algorithm) that repeatedly steps through the list, compares adjacent pairs and [swaps](https://en.wikipedia.org/wiki/Swap_(computer_science)) them if they are in the wrong order. This is also a compatison sort. Although the algorithm is simple, it is too slow and impractical for most problems even when compared to insertion sort. According to the time measured for performing the three operations, a table and graph are being added below.

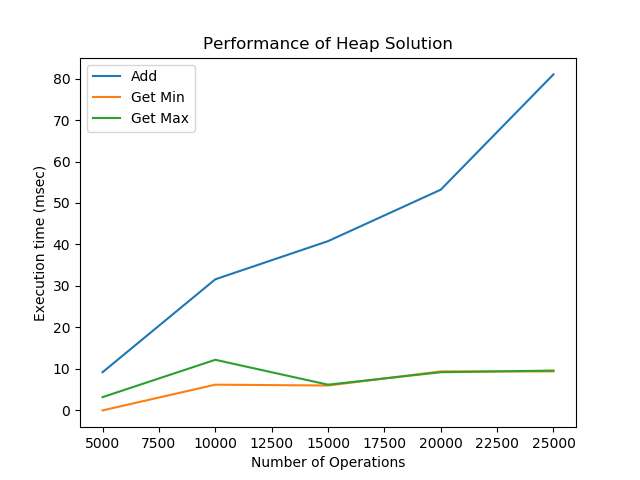
|  |  |  |
| --- | --- | --- |
| Time | | |
| Number of Operations | add | get min | get max | |
| 100 | 0.0070300006 | 0.0 | 0.0 | |
| 200 | 0.0142999994 | 0.0 | 0.0 | |
| 300 | 0.0461500000 | 0.0 | 0.0 | |
| 400 | 0.0921149998 | 0.0 | 0.0 | |



From both the table and the graph, it is visible that the time needed for adding values in the list increases as the number of operations increase. This is an exponential increment and is also the worst case for this function. While time needed for get min and get max is the same and constant for each number of operations and this is the best case for this function

**HeapSort:** heapsort is also a comparison based sorting algorithm. The improvement consists in the use of a heap data structure rather than a linear-time search to find the maximum. It is somewhat slower in practice on most machines than a well-implemented quicksort. Heapsort is an in-place algorithm but it is not a stable sort. According to the execution time, a table and graph are being added below.

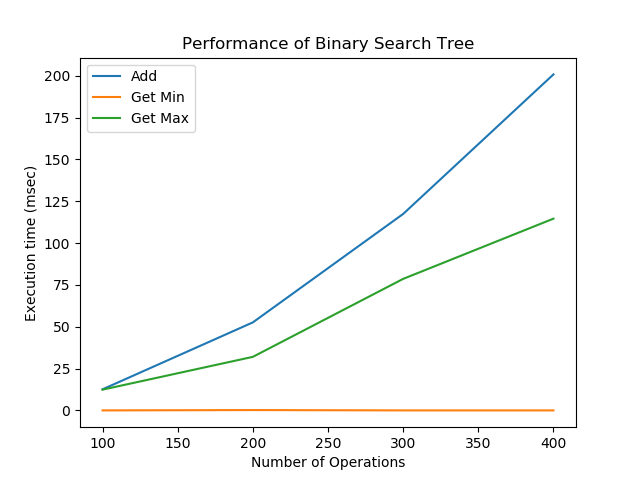
|  |  |  |
| --- | --- | --- |
| Time(msec) | | |
| Number of Operations | add | get min | get max | |
| 5000 | 10 | 0 | 2 | |
| 7500 | 18 | 1 | 6 | |
| 10000 | 31 | 3 | 11 | |
| 12500 | 35 | 3 | 8 | |
| 15000 | 39 | 3 | 3 | |
| 17500 | 45 | 1 | 1 | |
| 20000 | 50 | 5 | 5 | |



From the table and the graph, it is visible that both the operations of get min and get max take less time respect to the operation of adding a value. And with the increasing number of operations the execution time becomes same for both get min and get max, which is the best case. On the other hand, for the operation of add, increasing number of operations increases also the execution time. And this is the worst case for this function.

**Binary Search Tree :** it is a non-linear data structure, where one node is marked as a root node called key. The keys that are less than the parent are found in the left subtree and the keys that are greater than the parent are found in the right subtree. Binary tree is implemented using two pointers.

Here is a graph showing the performance of binary search tree:



Here it is visible that, time for get min is constant and this is the best case for this function. On the other hand, get max and add functions are the worst cases because increasing the number of operation the execution time increases.

**CONCLUSION:**

Quicksort, heapsort and bubblesort are the array sorting algorithms and binary search tree is a common data structure operation. Binary search tree is better than the linear time required to find items by key in an (unsorted) array. On the other hand, quicksort is faster than heapsort and bubblesort when it is well implemented. Thus we can understand in case of sorting which function is worth using.