**Report**

**Introduction**

The aim of this report is to do a benchmark analysis for each one of the following:

* Python Lists with Quicksort and MergeSort
* Binary Tree insertion, get random element, get max, delete random element
* Heap insertion, get max, delete max

using timeit to measure execution time and producing plots for each of the above in order to visualize the performance of the different algorithms.

**Structure of the files and how the data are collected**

Five files (“Quicksort and MergeSort.py”, “Binary Tree.py”, “MaxHeap.py”, “MinHeap.py” and “Python Heap.py”) were created. Each of the files is structured in this way:

* The first part is the implementation of the algorithm
* The second part is the one in which the times are taken
* The third part is dedicated to the creation of the plot

When one of the files is executed, in the same place where the file is located on the computer a directory called “Data” is created (if this is not already present); .txt files containing the times and plots are saved each time inside this directory.

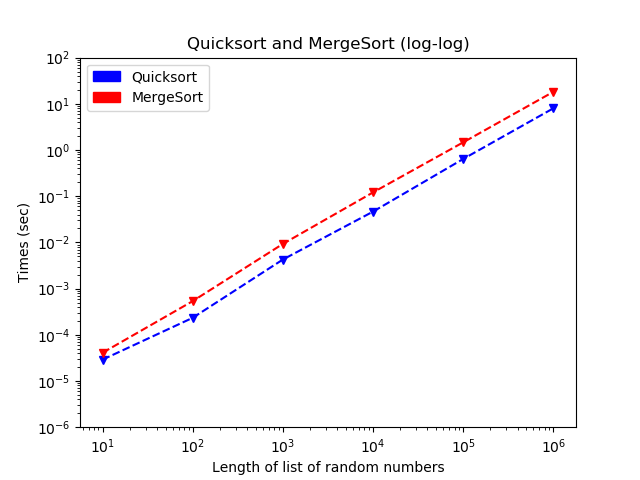
There are three variables that are important to define the conditions in which the execution times of the programs are taken. The first variable is “maxpoweroften”, which is the maximum power of ten corresponding to the maximum length of the list of random numbers. The second variable (“numberoftests”) is the number of tests that are executed. In each of the .txt files each test is represented by a line, whereas each column contains the times related to lists of equal length. The third variable is “maxvalue”, which is the upper limit of the range of numbers from which the random numbers are generated. Certain values for these variables were chosen and they are the same in all the files. In this way it was possible to compare the plots and look for similarities and differences between them. These conditions were 6 as maxpoweroften, 5 as numberoftests and 10 million as maxvalue.

**Comments**

The aim of this work is to understand how the time required for each action to be executed changes as the length of the list increases. To have a clear picture of the situation a set of random lists of increasing length was generated, times were taken for each performed action and log-log plots from the collected data were made.

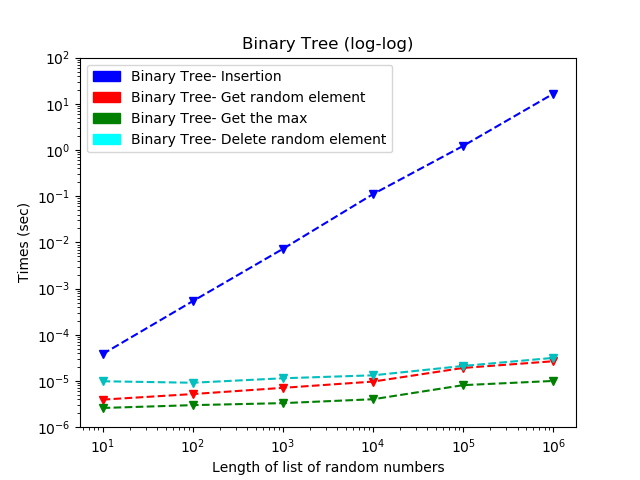
**Quicksort and MergeSort**

In the log-log plot the two algorithms are represented as two parallel lines, so the slope is the same. Quicksort is faster than MergeSort.



**Binary Tree**

Looking at the log-log plot it can be noticed that insertion times increase as the length of the random lists increases, but times needed to get a random element, to find the max and to delete a random element are less affected by the increase in length of the list and the lines representing them are almost parallel to the x-axis. Deleting a random element takes generally more time than getting a random element.

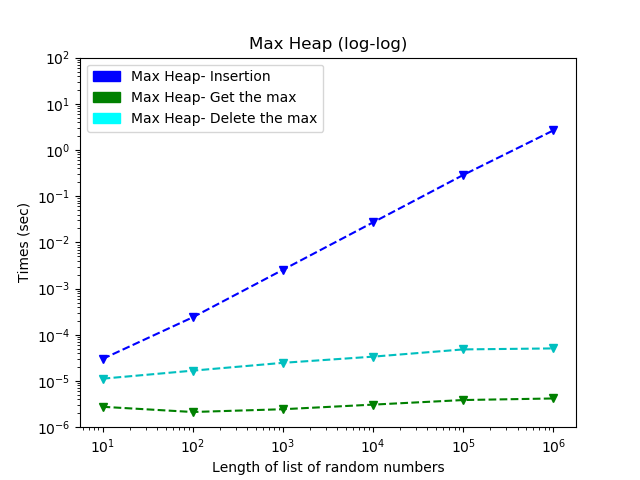


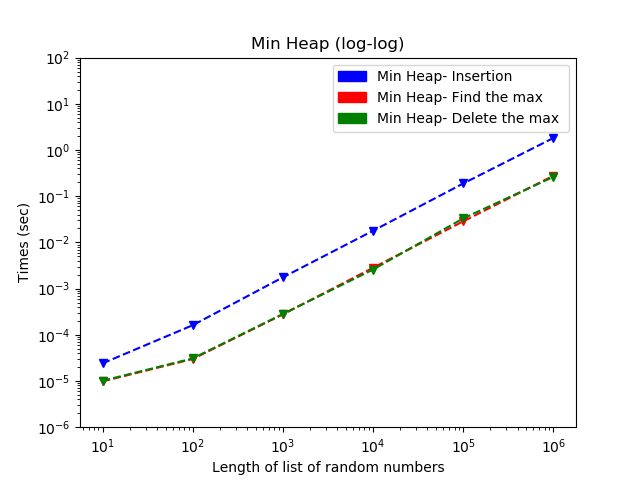
**Max Heap and Min Heap**

Log-log plots of two kinds of Heap were compared. Insertion times in Max Heap and Min Heap are similar, whereas there are significant differences in the times needed to get the max and delete it.

In Max Heap the max is always in the root so it is easy to find it, but it is necessary to reorganize the Heap after the deletion of the max. The time needed to delete the max is more than the time needed to get the max as the program needs more time when it has to delete the max and rearrange the list than when it has to just find the max.

In Min Heap for finding the max it is necessary to look for it in all the leaves, so the time increases significantly as the length of the list increases. In Min Heap the times needed to find and to delete the max are almost the same because once the max is found the deletion does not take a great amount of time. The time needed to get the max was significantly more in the Min Heap.





**Python Heap**

In this file the heapq module is used. The resulting plot was examined. It can be found that, as expected, the plot is similar to the plot of MinHeap, but the times are smaller.

