Data Structures II
Final Project Report

Syed Nayl Moid - $sm3916$
Shehreyar Abdi - $sa03576$
Ali Shujjat - $as03856$

May 13, 2019

1 Introduction

We've all been in a hurry sometimes. And an elevator skipping floors certainly doesn't help. Therefore, in our project we are attempting to solve the elevator problem that presents itself in many taller buildings that use elevators.

1.1 Defining our problem

The issue arises when multiple people on different floors call the same number of limited elevators, all which are going in different directions. The concept is to improve on time even if extra energy has to be used.

1.2 Expectations

We solve this problem using a simple unsorted array, a priority queue implemented using a singly linked-list and lastly using a heap data structure.

For the unsorted array we expect the worst running time as the worst case would be $O(n)$, whereas the sorted singly linked-list is expected to run in an average time of $O(n)$ and $O(lgn)$, therefore it should be better than the unsorted array. Finally the heap implementation should give the best time of $O(lgn)$.

2 Our Results

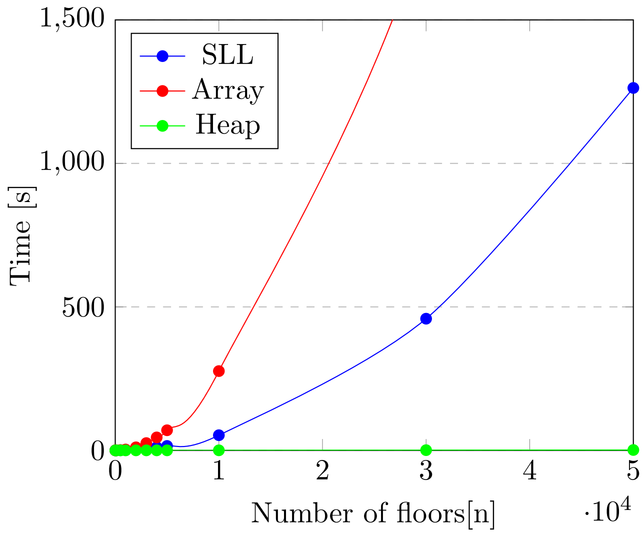
To ensure a fair comparisons between the 3 data structures, the seed, used to generate the random numbers, for all three structures was kept constant. With varying the number of floors the elevator is being called too we tabulated the results in the table shown below and then used them to plot the trend of the graphs shown below.

Here we note the time taken in seconds(s) for the program to run till completion.

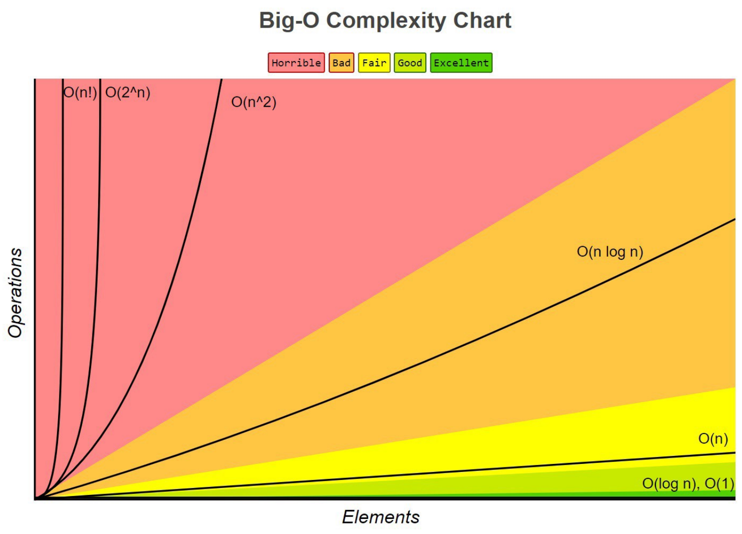
No. of Floors	Array	Linked-List	Heap
10	0.078125	0.0705	0.0468
50	0.093750	0.1235	0.0625
100	0.015625	0.1365	0.0638
500	1.062500	0.4020	0.0678
1000	2.989300	0.8283	0.0781
2000	10.70317	2.9622	0.1250
3000	25.43752	6.4468	0.1260
4000	45.21843	11.1186	0.1408
5000	70.14109	15.4431	0.1718
10000	276.3628	52.8850	0.2812
30000	x	458.8865	0.8281
50000	x	1262.927585	1.4062

After discussions with our amazing professor, it was decided to add an extra condition to simulate real world data. Therefore when the program has run for more than 10 seconds new floors are appended into the list with new passengers eager to leave. However, the number of elevators remains 3.

Runtime complexity for array, sorted linked-list and heap data structures



The above graphs were plotted in accordance to the results we obtained for different number of floors. For larger values of 30,000 and 50,000 took too long to be noted properly therefore the line for it has been extrapolated.



3 Data Analysis

Shown above is a Big-O complexity chart. In accordance we compare the results we have plotted in our graph. Our results match the running times of our selected data structures. The time for heap does not rise much higher from the x-axis. The time for the array starts off the same but then begins to increase with greater number of floors. The time for the sorted linked-list is somewhat in-between the curves of the array and heap.

4 Conclusion

We have successfully implemented all three data structures to solve for our elevator problem. We arrived at results that match our expectations mentioned at the start of this paper. In the future we plan to improve on this to work for n number of elevators, taking input from the user.

5 References

1. https://www.geeksforgeeks.org/

2. http://interactivepython.org/courselib/static/pythonds/Trees/BinaryHeapImplementation.html

3. https://www.tutorialspoint.com/

4. https://www.kingsiii.com/help-phone-blog/all-about-elevators/common-misconceptions-about-elevators-and-elevator-entrapments.asp

5. https://towardsdatascience.com/understanding-time-complexity-with-python

6. http://www.xm1math.net/texmaker/