OOP DESIGN

I have made classes and have put them in the src folder. The classes are LSBSTApp and the LSAVLTApp.

The LSBSTApp.class

This class takes the load shedding data schedule of the City of Cape Town from the ./Load_Shedding_All_Areas_Schedule_and_Map.clean.final.txt text file and puts it into a Binary Search Tree. It takes the data and splits it into array objects and arranges them accordingly into a Binary Search Tree. It invokes the Binary Search Tree code when a user enters a stage. The class has a printAreas function which prints out the stage, date and time for a specific area entered by the user. It is worth noting that all these areas in the data text file all experience 2 hours of load shedding. The printAllAreas function prints out all this data for all areas in any order. OpCounts just counts the number of comparisons done by the program. The WriteText and CreateText functions are for writing the OpCounts values and creating a text file, respectively, whenever the program is run.

The LSAVLTApp.class

This class does the exact same thing the LSBSTApp does, but it just puts the data into an AVL tree. The OpCount function counts the number of comparisons done by the program before it gets to the data it is looking for in the AVL tree. The CreateText and WriteText functions are for creating and writing the OpCounts values in a text file for every execution done by the program.

Experiment and Goals

Aim

The purpose of this experiment is to compare efficiency levels between a Binary Search Tree and an AVL tree. The proof of the two's efficiencies will be seen through the numbers their respective counters run up when a user enters a stage.

Method

1. The Binary Search Tree was tested first by inserting and searching 3 separate keys from the data. The result was recorded in the text file. Then the printAll method is invoked. "java PowerAVLT "date/time"" was typed into command prompt to search for a key.

   "javaPowerAVLTApp" command was used to print all dates.

   Input/output redirection in the shell was used to write this data into a file.
2. Three insert and search counts were recorded. An unknown input was used to find out the number of comparisons the program does when trying to find a key that isn't part of the data set.
3. The above steps were repeated for the LSBSTApp
4. The average, worst- and best-case subset comparisons were recorded into a text file. For the
5. Finally step 4 was repeated except with a csv file. The graph was generated but the data was recorded.

Test Results
AVL and BST

SAMPLE SIZE	BEST CASE	AVERAGE CASE	WORST CASE

250	1934	5204	3475
500	10358	17837	23013
750	33957	43206	86483
1000	69457	86842	229408
1250	132205	154848	501095
1500	224768	252328	962625
1750	352332	384495	1682099
2000	520661	558471	2723455
2250	734572	777552	4231377
2500	993948	1047438	6254445

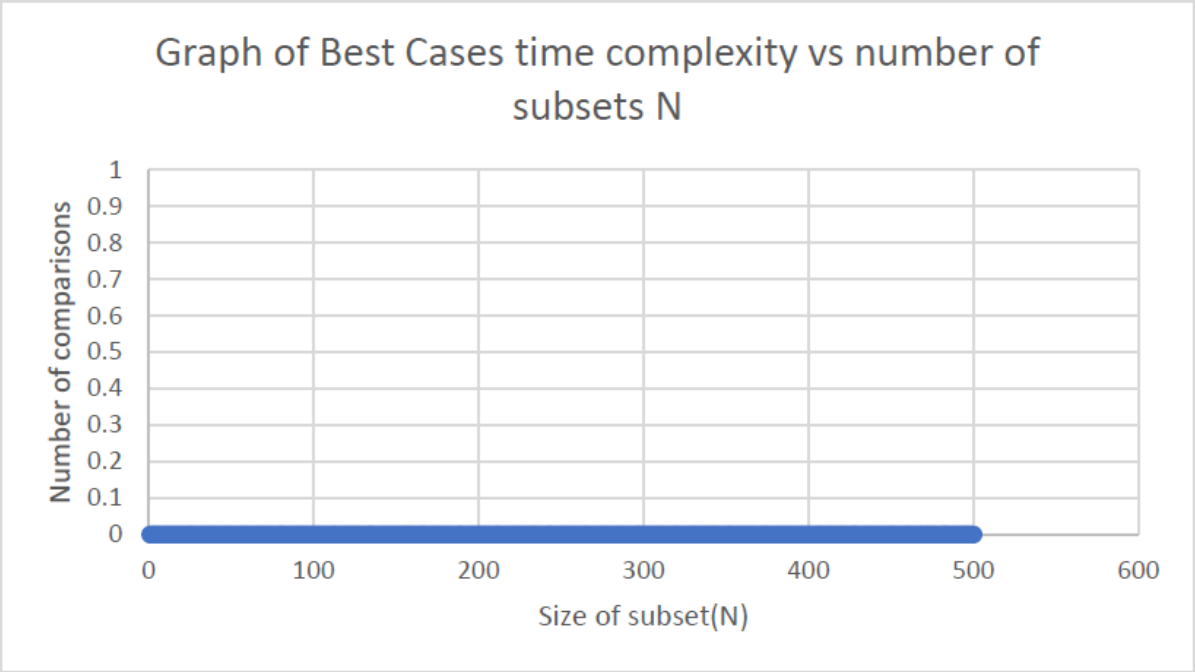
TABLE 3: LSAVLTApp

LSBSTApp

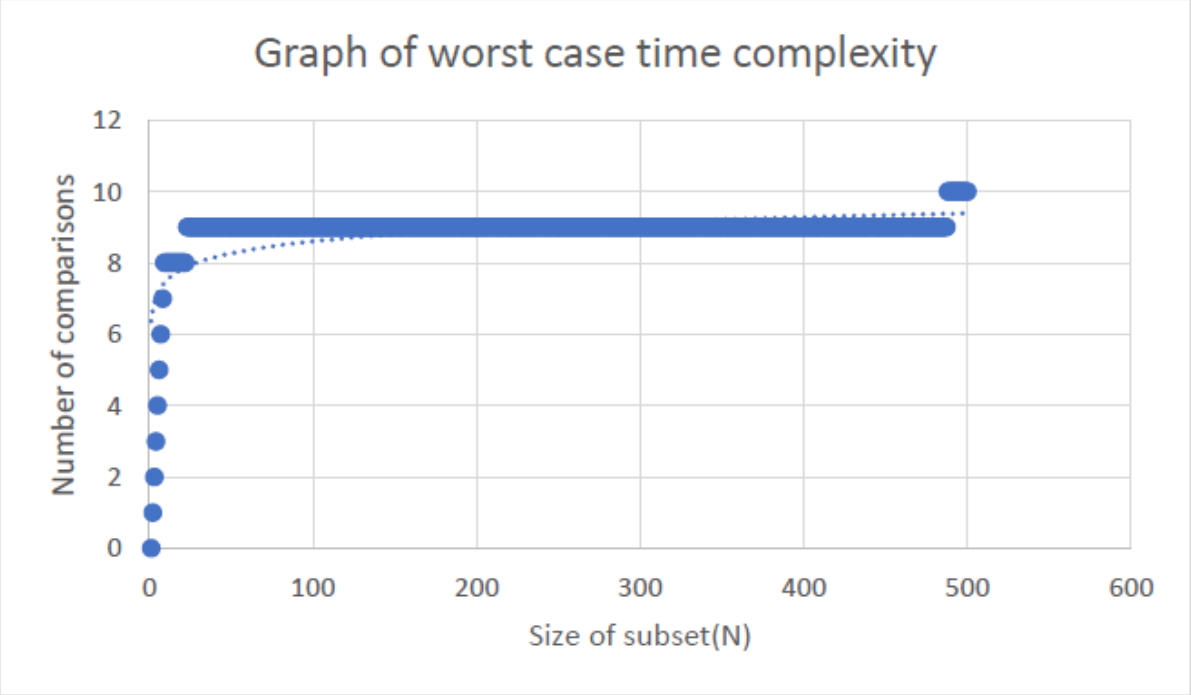
SAMPLE SIZE	BEST CASE	AVERAGE CASE	WORST CASE
250	1996	6192	3893
500	11826	21494	28445
750	36945	52573	105946
1000	85253	107051	284656
1250	164715	192657	629362
1500	283706	318018	1222110
1750	450249	491223	2159562
2000	671961	719397	3552995
2250	956683	1010689	5529317
2500	1311617	1372413	8228615

TABLE 4 :LSBSTApp

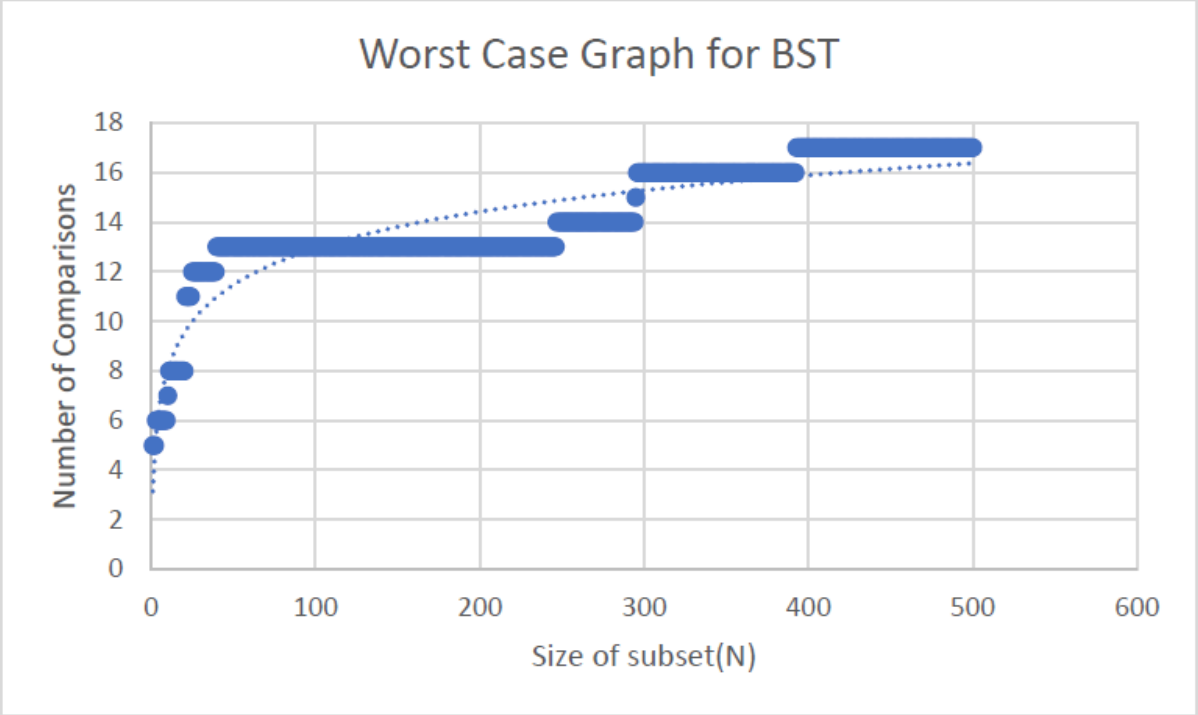
AVL Best case Insertion and Search



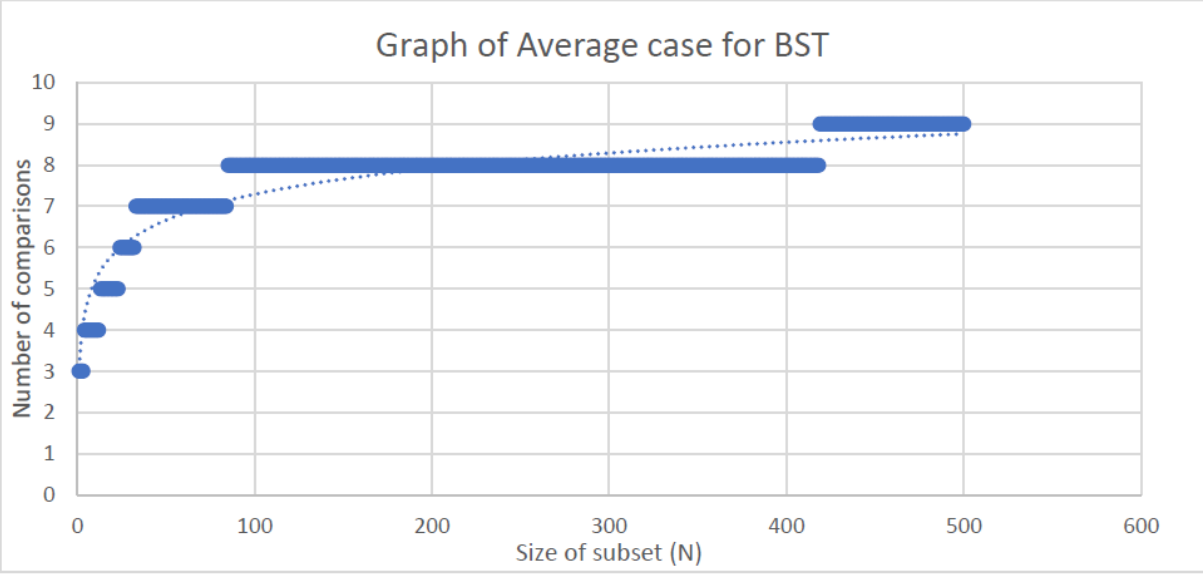
AVL Worst Case Search



BST Worst Case



BST Average Case



BST Insert Cases

The worst, average and best case for the insert comparisons were found to be the same as the respective search comparison. This is because the BST doesn't adjust or rotate the tree during insertion. Therefore, the search and insert method use the same comparisons.

AVL Insert

The average case for the entire case =3462/500=6.9. This is expected as it falls in the Olog(N) time complexity range. This is better than the average case scenario which is 4954/500=9.9 because during insertion, rotations and balancing methods cause some nodes to travel

slightly more. The worst case is therefore lower than the average case although it is still Olog(N). The best case for the insert is O (1). This is because the best case is when one node is inserted into the AVL tree.

Results

BST

Average

The BST plummeted in performance with the average insertion and search times having a complexity of o(n/2) which translates to O(N).

Worst

The worst case has an O(N). This was expected as the BST becomes a linked list when inserting sorted data so the search and insert count will increase.

Conclusion

The BST has a higher search and insertion count than the BST. The time complexity for the insertion of an AVL is average case – O(log(N)), best case O (1) and worst case – O(log(N)). This also applies for the search.

For the BST, for both search and insertion, its best case is O (1), average case – O(log(N)) and worst-case O(N). I therefore conclude that AVL tree has faster search and insert times than the BST.

Creativity Points

I used Microsoft excel to plot the graphs

I used the bash terminal-scripts to generate data for the graphs