Report for Assignment 2

Object-Oriented Design

Part 1:

I created an AVL class to store methods unique to an AVL which include the insert, balance and height methods. It also contains the printDateTime and printAllDateTimes methods. All methods a normal BST uses have remained in the BST and BSTNode classes (described under part 2). For the PowerAVLApp, the cleaned_data.csv file was loaded into a File object. The main method uses the File object to iteratively insert (and balance) nodes into a BST. If there are parameters used when the main method is executed, printDateTime executes with the input as its parameter. If there are no parameters in the execution, the printAllDateTimes method is called and all values in the AVL tree are outputted.

Part 3:

For the PowerBSTApp, the BSTNode and BST classes were reused from the previous assignment. Originally, The methods in BST and BSTNode came from https://www.moreofless.co.uk/binary-search-tree-bst-java/ , https://www.geeksforgeeks.org/print-binary-tree-vertical-order/ and https://www.geeksforgeeks.org/print-binary-tree-vertical-order/ .The main method uses the File object to iteratively insert nodes into a BST. If there are no parameters in the execution, the printAllDateTimes method is called and all values in the AVL tree are outputted.

BSTNode contains the constructor for the node as well the get(returns a Keys Value) and put(append) methods. It also, contains the verticalOrder method and its helper methods (printVerticalLine and findMinMax) which are responsible for the printAllDateTimes method running correctly. BST contains the put (for appending BST) and get (returning the value in a node) as well the containsNode methods which checks to see if the BST contains a specified node.

The PowerBSTApp class contains the printDateTime method to print info for a specified date and the printAllDateTimes method to print all dates' info.

PowerBSTApp also contains the main method which constructs a BST using the BSTNode and BST classes to store the data and then executes either the printAllDateTimes or PrintDateTime method depending on arguments.

Part 5 and 6:

I created the test class to output data to a text file from a subset of input values from the dataset for a PowerAVLApp and the BSTtest class to output data to a text file from a subset of input values from the dataset for PowerBSTApp.

Both these methods are essentially modified main methods from their respective apps to allow N (subset size) as a parameter instead of a search input.

The Experiment

The goal of the experiment was to demonstrate the speed difference when performing insert and search between the AVL tree and BST.

The first step I used for my experiment was to create the test and BSTtest classes. Both of these classes take N, which is an integer used to describe subset size, as a parameter.

The main methods create a database as in PowerAVLApp and PowerBSTApp by inserting the entire dataset from the csv file. test creates an AVL and BSTtest creates a BST.

Another file object using the dataset (csv) is also created. A string array (subset) of size N is created and then iteratively appended until it contains the dateTime keys from the dataset up to a subset of size N. This contains all the inputs values

A for loop is then executed to implement the printDateTime method for every index in the subset array. This prints out the output values corresponding to the dateTime keys up the Nth position on the dataset.

The last three lines outputted for every iteration print out the opCount, insertCount and searchCount.

The second step was to create a simple bash shellscript (tester2.sh) . The script contains a for loop which iteratively executes the test (or BSTtest) program from N=1 to N=500.

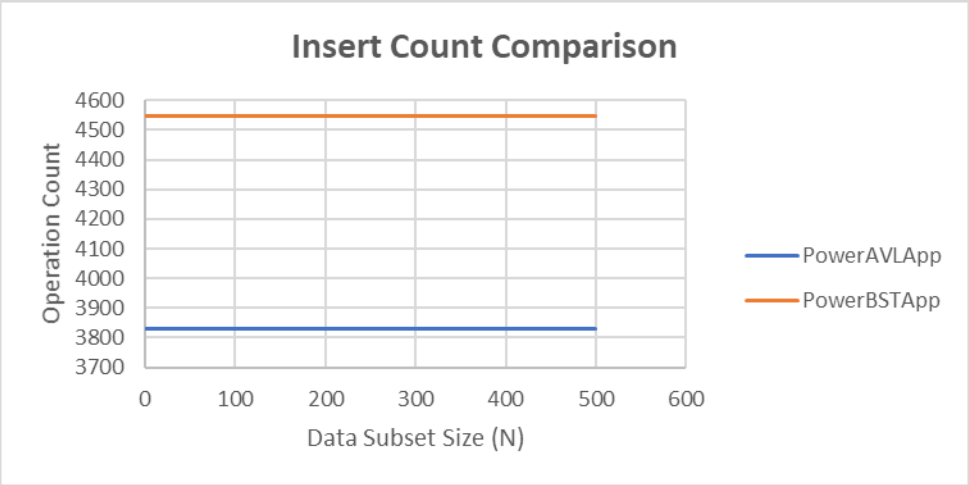
The third step was done in the terminal. I executed the script and redirected the output to a separate text file for each app.

The fourth step was to create app/class called DataExtractor. This app creates a file object using the output for a given app's test. The main method executes a while loop with stopping condition "hasNextLine()" for the file. Inside this loop, the string for the nextLine is split into an array then an if-statement checks if the first index on the line is a specified word. If it is, the entire line is printed out.

The fifth step was to use the DataExtractor to get the data from the AVL test output by redirecting the output to text files using the terminal. opCount is first specified in DataExtractor. Then searchCount is specified and then insertCount is specified. We end up with 3 text files containing experiment data for the AVL and 3 text files containing experiment data for the BST.

Lastly, the data was copied into 2 excel files, 1 for the AVL and 1 for the BST. This data is the final form ready to be analysed and/or graphed.

Results

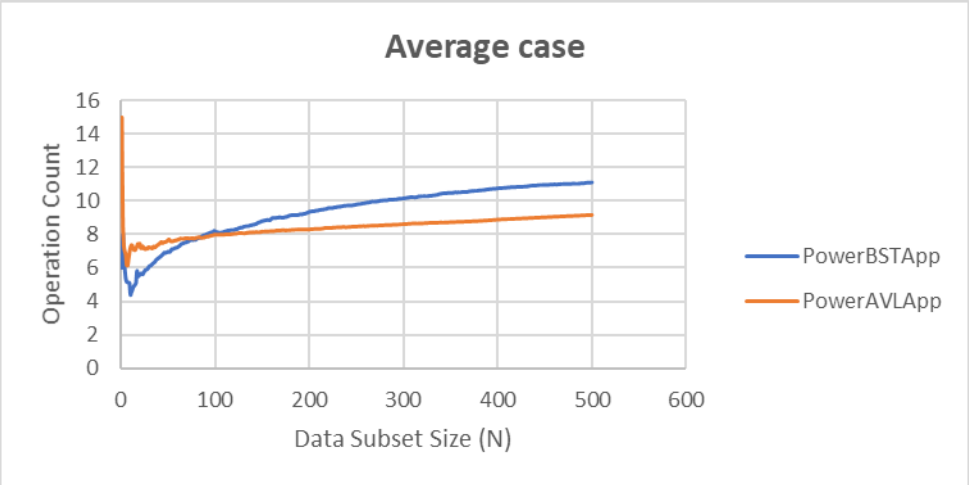


As the graph above shows, the insertion count for PowerAVLApp was constant at 3831. Best case = worst case = average case = O(3831)

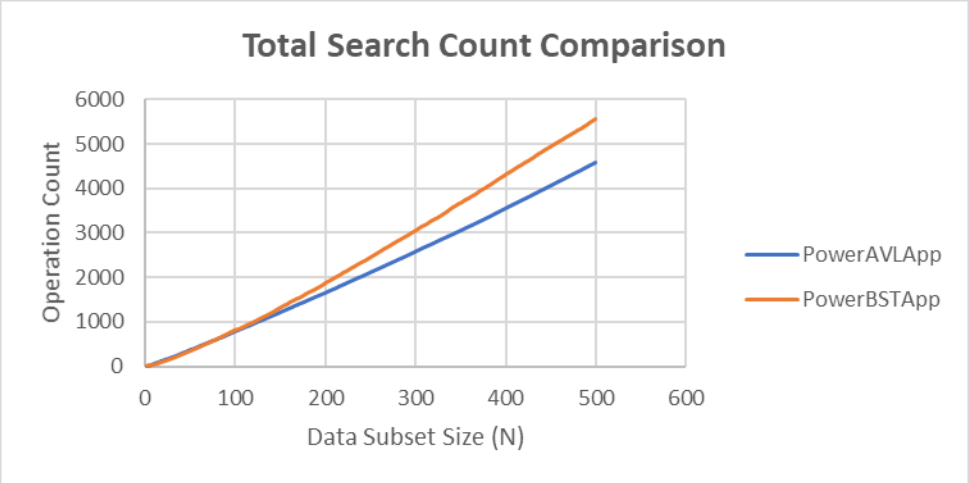
The insertion count for PowerBSTApp was also constant but at 4546. Best case = worst case = average case = O(4546). The difference in insertion count is most likely due to the slight difference is coding method I used for their respective insertion methods.

The best case search count for both PowerAVLApp and PowerBSTApp was 1. This means that the best case is O(1).

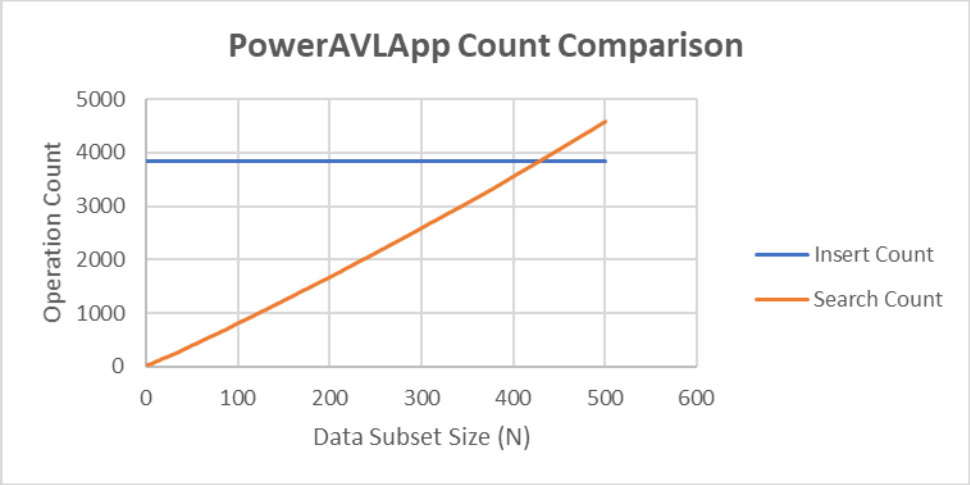
The worst case search count for PowerAVLApp was 11 . The worst case for PowerBSTApp was 23.



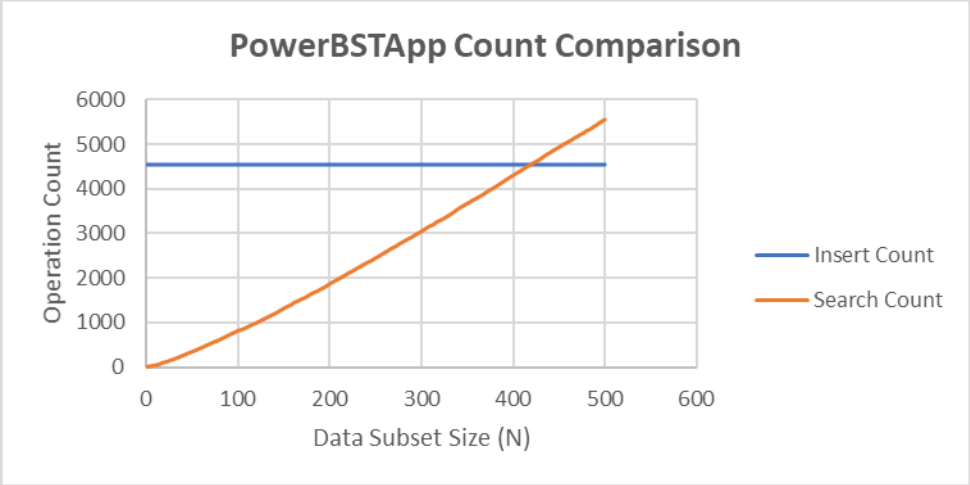
The graph above shows the search count for the average case. PowerBSTApp starts off slightly quicker before it rises above PowerAVLApp to the point that is increasingly slower. PowerBSTApp graph resembles a Log graph whereas PowerAVLApp looks more constant (flatter).



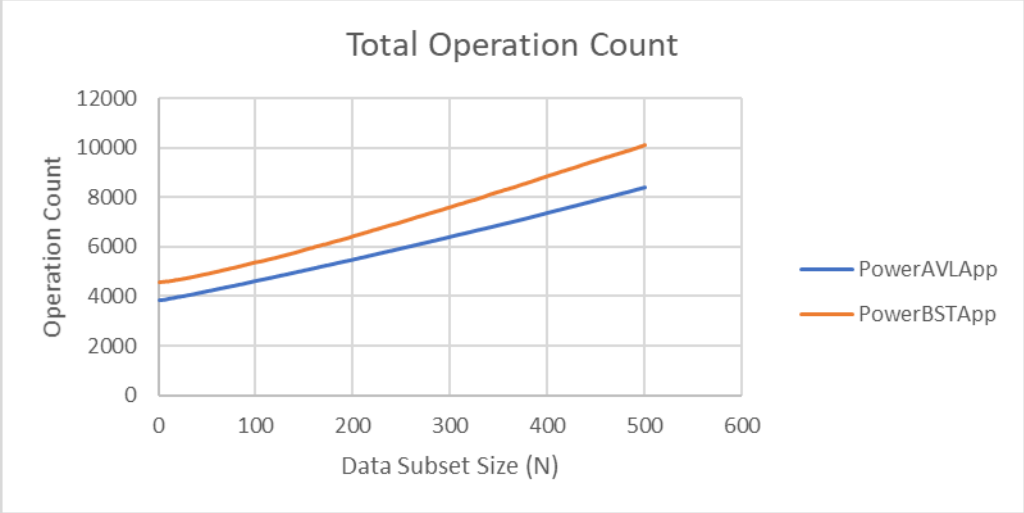
The total search count comparison shown above also illustrates that PowerAVLApp is faster search wise overall. It has a flatter curve (lower gradient).



The graph above compares insert count and search count for PowerAVLApp. The insert count is constant and the search count increases with N increasing. Search count only exceeds insert count once N>430 . For the average case, search count is much smaller than insert count.



The graph above compares insert count and search count for PowerBSTApp. The insert count is constant and the search count increases with N increasing. Overall search count only exceeds insert count once N>410 . For the average case, search count is much smaller than insert count.



The graph above illustrates the total operation for both PowerAVLApp and PowerBSTApp. PowerAVLApp is clearly the faster app overall at every subset size N. It's lower insertion count gives it a starting advantage but its graph also has a lower gradient which means it handles searching better as well.

Part 6 Results

For PowerAVLApp:

Total insertCount = 3989

Total searchCount = 4508

Total opCount = 8497

For insert count: Best case = worst case = average case = 3938

For search count: Best case= 1, Worst case= 9, Average case = 7.7

The app performed significantly more efficiently than with the unsorted data.

For PowerBSTApp:

Total insertCount = 124750

Total searchCount = 126251

Total opCount = 251001

For insert count: Best case = worst case = average case = 124750

For search count: Best case= 1, Worst case= 157485, Average case = 252.2

The app performed as inefficiently as possible with the unsorted data. This is because, with linear input data, the binary tree structure is linear. This therefore made the worst possible case for insertion and search.

Trial Runs

Test number and input	**PowerAVL (Part 2)**	**PowerBST (Part 2)**
Test 1: "16/12/2006/19:51:00"	16/12/2006/19:51:00,3.388,233.220 insertCount = 3831 searchCount = 5 opCount = 3836	16/12/2006/19:51:00,3.388,233.220 insertCount = 4546 searchCount = 2 opCount = 4548
Test 2: "16/12/2006/23:20:00"	16/12/2006/23:20:00,1.222,241.580 insertCount = 3831 searchCount = 3 opCount = 3834	16/12/2006/23:20:00,1.222,241.580 insertCount = 4546 searchCount = 3 opCount = 4549
Test 3: "17/12/2006/00:29:00"	17/12/2006/00:29:00,0.612,243.680 insertCount = 3831 searchCount = 4 opCount = 3835	17/12/2006/00:29:00,0.612,243.680 insertCount = 4546 searchCount = 4 opCount = 4550
Test 4: "17/17/2006/00:29:00	Date/time not found insertCount = 3831 searchCount = 10 opCount = 3841	Date/time not found insertCount = 4546 searchCount = 11 opCount = 4557

Test 5: no arguments	16/12/2006/17:24:00,4.216,234.840 16/12/2006/17:25:00,5.360,233.630 16/12/2006/17:38:00,4.054,235.240 16/12/2006/17:27:00,5.388,233.740 16/12/2006/17:26:00,5.374,233.290 16/12/2006/17:28:00,3.666,235.680 16/12/2006/17:32:00,3.668,233.990 16/12/2006/17:39:00,3.384,237.140 16/12/2006/19:24:00,3.262,232.640 16/12/2006/21:14:00,3.376,235.730 … 17/12/2006/01:40:00,3.214,241.920 17/12/2006/01:38:00,3.680,239.550 17/12/2006/01:41:00,4.500,240.420 17/12/2006/01:32:00,4.460,240.080 17/12/2006/01:39:00,1.670,242.210 17/12/2006/01:42:00,3.800,241.780 17/12/2006/01:43:00,2.664,243.310 insertCount = 3831 searchCount = 0 opCount = 3831	16/12/2006/17:38:00,4.054,235.240 16/12/2006/17:39:00,3.384,237.140 16/12/2006/17:40:00,3.270,236.730 16/12/2006/17:58:00,4.058,234.680 16/12/2006/18:26:00,4.868,233.840 16/12/2006/17:44:00,5.894,232.690 16/12/2006/17:42:00,3.266,237.130 16/12/2006/17:41:00,3.430,237.060 16/12/2006/17:59:00,2.472,236.940 16/12/2006/18:27:00,4.866,233.790 … 17/12/2006/01:21:00,4.652,237.920 17/12/2006/01:29:00,2.080,241.610 17/12/2006/01:42:00,3.800,241.780 16/12/2006/22:25:00,2.428,239.680 17/12/2006/00:46:00,2.450,240.820 17/12/2006/01:17:00,2.822,239.550 17/12/2006/01:43:00,2.664,243.310 insertCount = 4546 searchCount = 0 opCount = 4546

Creativity employed

1. I used many stages for data extraction to allow full control over what data I acquired and how I could implement it. These include my test classes and my data extraction class as well as my bash script.

2. I coded PowerAVLApp so that it didn't need many new methods and instead just shared most its methods with those required by PowerBSTAppp.

Git usage log

1: commit c40ba00fc1f529b68eedb5eb282526027284c3eb

2: Author: Mahmood-Ali Parker <ali@HP.localdomain>

3: Date: Mon Mar 18 01:37:28 2019 +0200

4:

5: final commit with all completed source code

6:

7: commit 76073d60987e1a9403d9cd16ad5b3817ffb2d04b

8: Author: Mahmood-Ali Parker <ali@HP.localdomain>

9: Date: Sun Mar 17 05:57:09 2019 +0200

10:

...

56: Author: Mahmood-Ali Parker <ali@HP.localdomain>

57: Date: Sat Mar 16 00:25:06 2019 +0200

58:

59: AVL class added to store methods required that are different or altered from those used by BST. Copied from notes and not been fixed yet.

60:

61: commit f9d06af1dc7346318ceed21f8c48dca6aa5bafd1

62: Author: Mahmood-Ali Parker <ali@HP.localdomain>

63: Date: Thu Mar 14 20:28:13 2019 +0200

64:

65: Initial version with recycled classes BST, BSTNode, PowerBSTApp from Assignment 1. Other files are untouched