UNIVERSITY OF CAPE TOWN
Department of Electrical Engineering

CSC2001 Assignment 1 Project 2019 Report

by Nicholas Antoniades ANTNIC008

March 6, 2019

1 | Questions

1.1 Aim of the experiment

The goal of this assignment is to compare the number of computations required to search through a traditional unsorted array data structure to that of a balanced binary search tee. Both are to be implemented using Java and real power consumption data.

1.2 OO Design

PowerArrayApp

The PowerArrayApp class needed to read in a CSV file and create an array of objects for each line in the file. This array can then be searched through checking for matching instances. Three other classes were required when creating the PowerArrayApp class: timeStamp, textWrite and opCount.

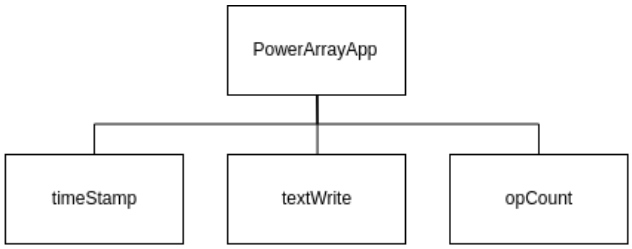

Figure 1.1: PowerArrayApp OO Design

PowerBSTApp

The PowerBSTApp class needed to read in a CSV file and create an array of objects for each line in the file. Then take that array, sort it, and then create a binary search tree with it. This array can then be searched through checking for matching instances. Five other classes were required when creating the PowerArrayApp class: timeStamp, textWrite, opCount, BinaryTree and Node.

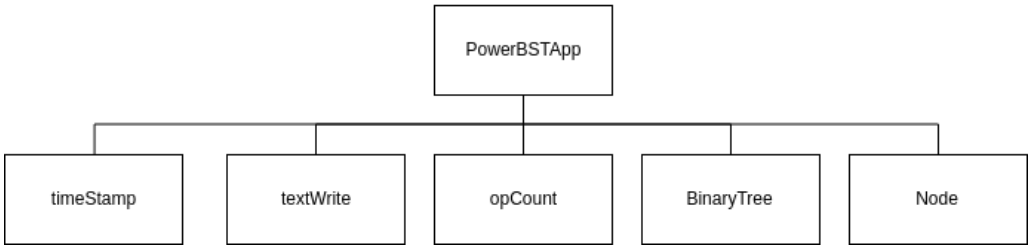

Figure 1.2: PowerBSTApp OO Design

Class descriptions:

timeStamp : Used to read in a CSV and create an array of timeStamp objects which have the date, global active power and voltage.

textWrite : The textWrite class takes in a timeStamp object and writes outputs it to a specified file.

opCount : The opCount class is used to keep track of the total number of operations

BinaryTree : The BinaryTree class takes in an un-ordered array and creates a balanced binary tree. This class allows for adding, searching and iterating through the binary tree.

1.3 Testing

By adding additional code to the BST and array methods, the number of comparison could be discretely counted. All operations (<, >, =) that are being performing in the code are counted.

PowerArrayApp

Printing the first 10 and last 10 lines printAllDateTimes ().
16/12/2006/17:24:00 234.840 4.216
16/12/2006/17:25:00 233.630 5.360
16/12/2006/17:26:00 233.290 5.374
16/12/2006/17:27:00 233.740 5.388
16/12/2006/17:28:00 235.680 3.666
16/12/2006/17:29:00 235.020 3.520
16/12/2006/17:30:00 235.090 3.702
16/12/2006/17:31:00 235.220 3.700
16/12/2006/17:32:00 233.990 3.668
16/12/2006/17:33:00 233.860 3.662
—
—
17/12/2006/01:34:00 241.540 2.358
17/12/2006/01:35:00 239.840 3.954
17/12/2006/01:36:00 240.360 3.746
17/12/2006/01:37:00 239.790 3.944
17/12/2006/01:38:00 239.550 3.680
17/12/2006/01:39:00 242.210 1.670
17/12/2006/01:40:00 241.920 3.214
17/12/2006/01:41:00 240.420 4.500
17/12/2006/01:42:00 241.780 3.800
17/12/2006/01:43:00 243.310 2.664

PowerBSTApp

Printing the first 10 and last 10 lines printAllDateTimes ().
16/12/2006/19:51:00 233.220 3.388
16/12/2006/23:20:00 241.580 1.222
17/12/2006/00:29:00 243.680 0.612
16/12/2006/20:35:00 233.370 3.226
16/12/2006/17:37:00 232.910 5.268
16/12/2006/19:23:00 234.360 3.334
16/12/2006/18:25:00 233.740 4.870
16/12/2006/20:30:00 234.540 3.262
17/12/2006/00:32:00 241.860 2.376
17/12/2006/00:22:00 240.560 0.276
—
—
16/12/2006/23:15:00 242.390 0.386
17/12/2006/00:42:00 243.650 0.382
16/12/2006/23:52:00 238.890 3.458
16/12/2006/18:38:00 234.020 2.912
16/12/2006/17:41:00 237.060 3.430
16/12/2006/19:21:00 234.020 3.332
16/12/2006/23:47:00 241.230 2.540
17/12/2006/00:09:00 242.090 0.838
16/12/2006/22:01:00 237.680 1.786
16/12/2006/17:43:00 235.840 3.728

Running the tests using a bash script

A bash script was created in which both applications were tested using multiple test cases for a range of subsets of the given power data. This data set sizes range from 1 to 500 elements of data increasing in increments of 20. The number of comparison operations required for the search in each subset is calculated and is then appended to a txt file.

The PowerArrayApp and PowerBSTApp implementations needed to be modified to allow command line parameters to call different functionality, adjust data set size, specify an output textfile and search through the data. Using these command line parameters to control the searches that occur when running the bash script.

The test values that were used in the trial runs for Part 2 and Part 4 can be seen in the table below.

Test cases	**Search**	**Status**
Test 1	16/12/2006/19:27:0	Present in dataset
Test 2	17/12/2006/01:36:00	Present in dataset
Test 3	16/12/2006/20:47:00	Present in dataset
Test 4	13/2013/20:32:00	Not in dataset
Test 5	7789997	Not in dataset

Table 1.1: Test values

1.4 Instrumentation and results

PowerArrayApp

It can be seen that as the size of the data set increases linearly, so does the number of comparisons required when doing a search.

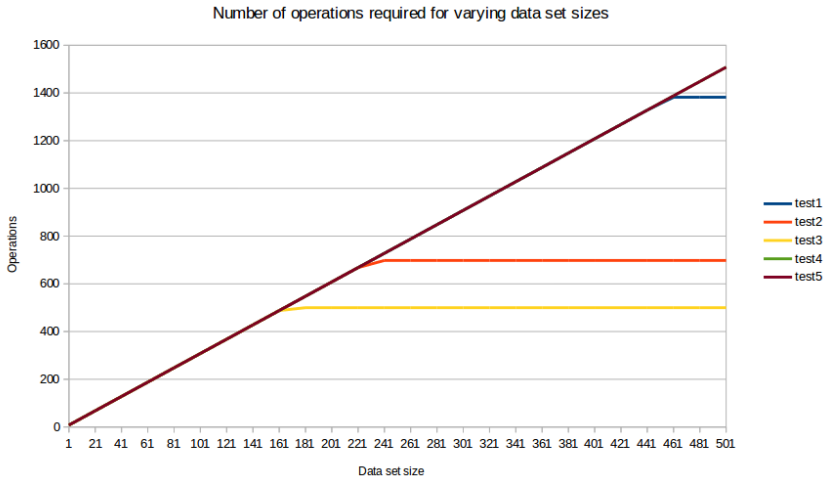

Figure 1.3: PowerArrayApp test results

PowerBSTApp

It can be see that as the size of the data set increases linearly, the number of comparisons initially increases for small data sets, but quickly levels out and becomes more constant as the size of the data set increases.

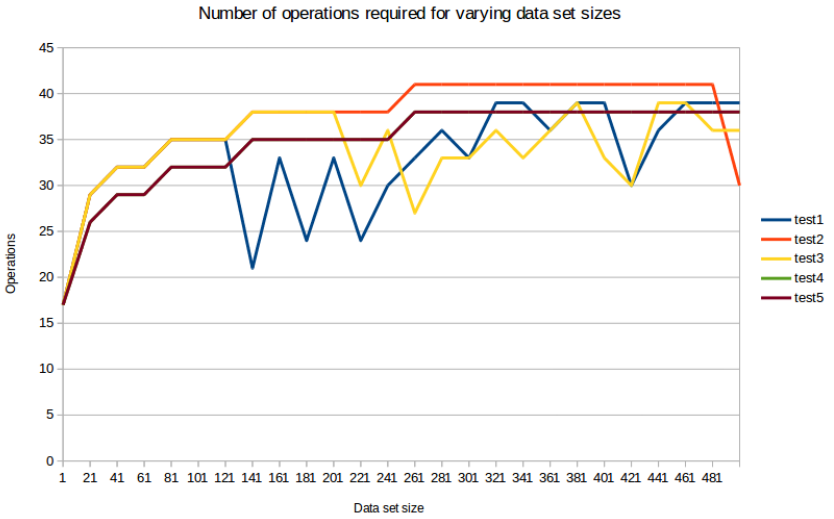

Figure 1.4: PowerBSTApp test results

1.5 Review and Summary

My application allows the user to test the number of operations when making various searches. As well as allowing for input arguments that allow for the user to adjusting the data set size, and whether or not o display the operational count value and write it to a file.

By running the bash script file the test results can be reproduced. By adjusting the bash script, alternative searches can be tested.

From the PowerArrayApp output graph, it can be seen that linear search runs on average O(n) n being the number of elements in the array.

From the PowerBSTApp output graph, it can be seen that Binary search runs in worst case on logarithmic time in the with O(log n) comparisons n being the number of elements in the array.

It can be seen that the applications that were create performed there task to completion with accuracy.

The code used for creating the BST was an adpated version of Derek Bana's from the tutorial at http://www.newthinktank.com/2013/03/binary-tree-in-java/

git log

```
1: commit 645ca0bed330628bfbc248b2d83fd5d065ecfeed
2: Author: Nicholas <antnic008@myuct.ac.za>
3: Date: Tue Mar 5 20:54:08 2019 +0200
4:
5: Finished with the testing, now for the report
6:
7: commit 31c4c1f443691df0e9e1240d6625cbbb49ac6165
8: Author: Nicholas <antnic008@myuct.ac.za>
9: Date: Tue Mar 5 15:24:44 2019 +0200
10:
...
122: Author: Nicholas <antnic008@myuct.ac.za>
123: Date: Sat Feb 23 12:16:04 2019 +0200
124:
125: Initial files
126:
127: commit f25048e125ed9b497346ba407e5f8c0127903155
128: Author: Nicholas <antnic008@myuct.ac.za>
129: Date: Fri Feb 22 19:02:45 2019 +0200
130:
131: This is my first attempt
```