CSC2001F

Practical Assignment 1

Data Structure Comparison

Niceta Nduku

NDKNIC001

6 March 2019

Introduction

This aim of this report is to compare two types of data structures, traditional array and Binary search tree.

OOP Design Process

The design process looks at what classes were created and how they interact in order of dependency. Figure 1 is a UML that shows how the classes interact with each other. It was essential to create an object that contains the values in order to make storing and searching easier.

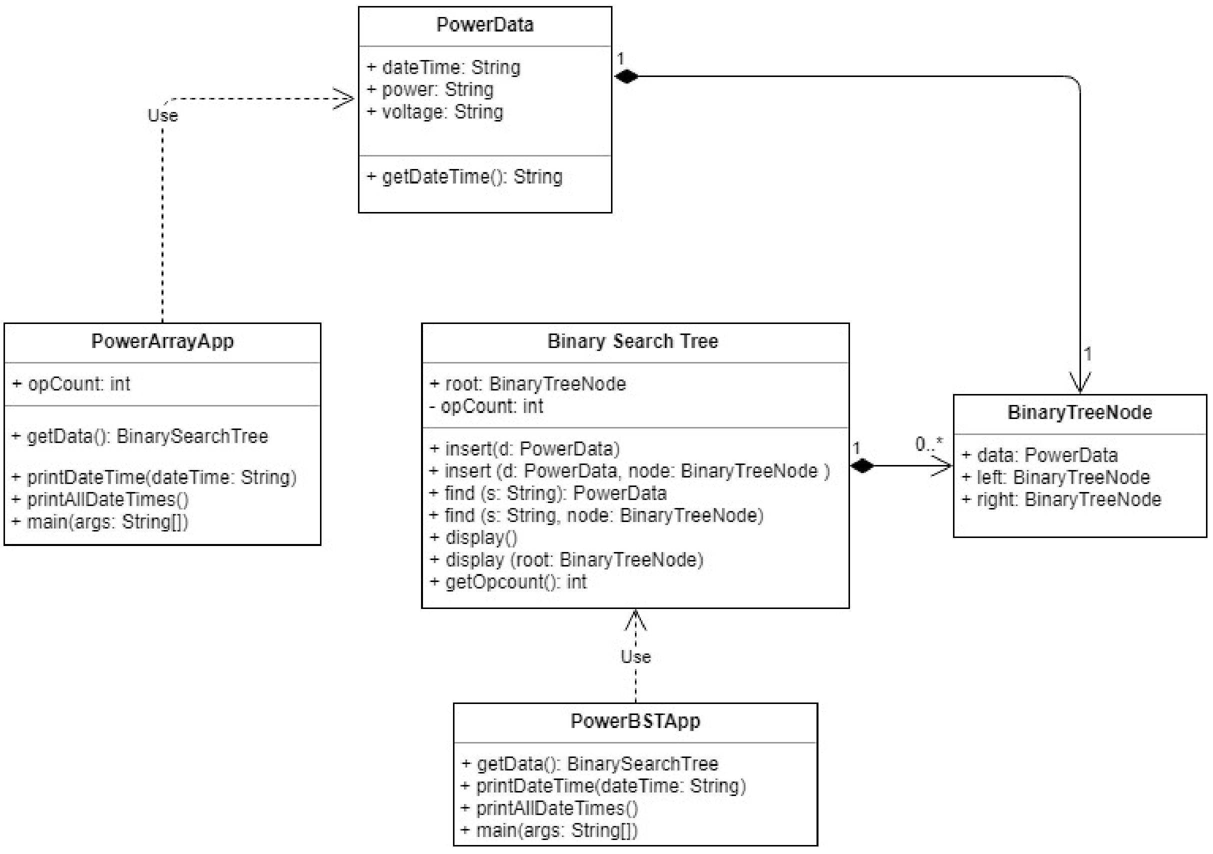

Figure 1: UML diagram showing interaction of classes

PowerData

This is an immutable class that creates the object that is going to be stored in the data structures. The instant variables of PowerData are Date/Time, Power and Voltage which are extracted and set when the csv file is read. The class has methods, getDateTime and overridden toString from the object class. getDateTime returns the Date/Time of the object.

PowerArrayApp

PowerArrayApp, is the program that creates an array that stores the PowerData and outputs the data depending on what the user requests. There are three methods in this class and the main method that runs the program. getData() is the method that first creates an empty array of 500 items and reads the CSV file. It reads the first line to remove the strings that are not being stored. While its reading the file lines, it creates and new PowerDate item, taking the first second and fourth values in the line as the Date/Time, power and voltage respectfully. The new PowerData is then stored into the array. This is done until there are no more lines to read in the file. The complete array is then returned.

The next method is printDataTime that takes in a string parameter which is assumed to be the Date/Time. It creates an array of type PowerData from getData() that will be used for iteration. The array will be looped over searching for a PowerData item that has a Date/Time that matches the one that was inserted into the method call and breaks the loop once it has found the matching Date/Time and prints it. An integer that was set to zero was incriminated for each time a comparison of dates was done and is printed to the user. If there was no matching date a message is printed.

The printDateTime method takes in no parameters. When invoked, it creates an array of type PowerData from getData and prints each line iteratively.

The main method runs the application by checking what the user inputs. If the user inputs a string it will call printDateTime with whatever the user input. If there was no string argument, it will call printAllDateTimes.

BinarySearchTree
The second data structure that stores PowerData items. The original code was from Patrick Marais [1] and was modified to add nodes of PoweData types. It constains the general insert and find(search) of a binary tree. The DateTime values are compared for the tree to know where to insert the next node. Find takes in a string and compares the Date/Time of the current node date (in this case PowerData) to the string. It does this recursively going down the tree until it finds a matching date.

The display method that takes in the root was obtained from SJ [2]. It prints out all the data items of the tree recursively, going down the tree. A display method that takes in no parameters was added that calls display with the root.

PowerBSTApp
This class has the same methods as the PowerArrayApp. However, instead of an array, it creates a new Binary Search Tree and adds the data to a tree by calling insert for each new line in the CSV file.

PrintDateTime and printAllDateTimes work the same way as the as PowerArrayApp but call on methods in BinarySearchTree such as find and display to print out Date/Times.

Experiment
The aim of the experiment was to compare the speeds of the two data structure types, Binary search tree and the traditional unsorted array. This was done using a bash script that automated the process.

I commented out parts of PowerArrayApp and PowerBSTApp so that the only output from running the programs was the operation count. I created two separate bash scripts, Experiment1.sh and Experiment2.sh that ran PowerArrayApp and PowerBSTApp simultaneously. Each script had a loop of N {1..500}. In each loop, it created a subset of the data CSV file taking N lines from the second line of the cleaned_data. It then sends the lines to a temporary CSV file. Still under the same loop, an array of the lines in the temporary file are created and the app will be run for the first item in the array, that is the Date/Time. The results of each N output are stored in an excel sheet.

Generated graphs for both the applications' output using matplotlib. The three graphs show the number of operations for the Best Case, Worst Case and Average Case for each application.

Out of Practical Scope
I created a script that has; an array that contains known date/Times and a false date/time. The user is prompted to type the application name and the script runs the app for the 3 known date/times, the false date/time and with no parameters. The output for each case is printed to 3 separate files for each case. Only the first 10 and last 10 lines of the output are stored for the no parameter case.

Results

Part Two Trial Test

Date/Time input	Output	Operations
16/12/2006/19:51:00	16/12/2006/19:51:00 3.388 233.220	1
16/12/2006/21:39:00	16/12/2006/21:39:00 3.302 236.280	249
16/12/2006/17:43:00	16/12/2006/17:43:00 3.728 235.840	500

Table 1: Output of running PowerArray app with known Date/Times

Date/Time input	Output	Operations
16/12/2007/17:43:00	Date/Time not found	500

Table 2: Output of running PowerArray app with an unknown Date/Time

No parameters
Output:
16/12/2006/19:51:00 3.388 233.220 16/12/2006/23:20:00 1.222 241.580 17/12/2006/00:29:00 0.612 243.680 16/12/2006/20:35:00 3.226 233.370 16/12/2006/17:37:00 5.268 232.910 16/12/2006/19:23:00 3.334 234.360 16/12/2006/18:25:00 4.870 233.740 16/12/2006/20:30:00 3.262 234.540 17/12/2006/00:32:00 2.376 241.860 17/12/2006/00:22:00 0.276 240.560 16/12/2006/23:15:00 0.386 242.390 17/12/2006/00:42:00 0.382 243.650 16/12/2006/23:52:00 3.458 238.890 16/12/2006/18:38:00 2.912 234.020 16/12/2006/17:41:00 3.430 237.060 16/12/2006/19:21:00 3.332 234.020 16/12/2006/23:47:00 2.540 241.230 17/12/2006/00:09:00 0.838 242.090 16/12/2006/22:01:00 1.786 237.680 16/12/2006/17:43:00 3.728 235.840
Operations: 0

Table 3: First 10 and last 10 lines of the output from running PowerArrayApp with no parameters

Part Four Trial Test

Date/Time input	Output	Operations
16/12/2006/19:51:00	16/12/2006/19:51:00 3.388 233.220	4547
16/12/2006/21:39:00	16/12/2006/21:39:00 3.302 236.280	4559
16/12/2006/17:43:00	16/12/2006/17:43:00 3.728 235.840	4558

Table 4: *Output of running PowerBSTApp with known Date/Times*

Date/Time input	Output	Operations
16/12/2007/17:43:00	Date/Time not found	4561

Table 5: *Output of running PowerBTSApp with an unknown Date/Time*

No parameters
Output:

```
16/12/2006/17:24:00 4.216 234.840
16/12/2006/17:25:00 5.360 233.630
16/12/2006/17:26:00 5.374 233.290
16/12/2006/17:27:00 5.388 233.740
16/12/2006/17:28:00 3.666 235.680
16/12/2006/17:29:00 3.520 235.020
16/12/2006/17:30:00 3.702 235.090
16/12/2006/17:31:00 3.700 235.220
16/12/2006/17:32:00 3.668 233.990
16/12/2006/17:33:00 3.662 233.860
17/12/2006/01:34:00 2.358 241.540
17/12/2006/01:35:00 3.954 239.840
17/12/2006/01:36:00 3.746 240.360
17/12/2006/01:37:00 3.944 239.790
17/12/2006/01:38:00 3.680 239.550
17/12/2006/01:39:00 1.670 242.210
17/12/2006/01:40:00 3.214 241.920
17/12/2006/01:41:00 4.500 240.420
17/12/2006/01:42:00 3.800 241.780
17/12/2006/01:43:00 2.664 243.310
Operations: 4546
```

Table: First 10 and last 10 lines of the output from running PowerBSTApp with no parameters

Part Five Experiment Results

PowerArrayApp

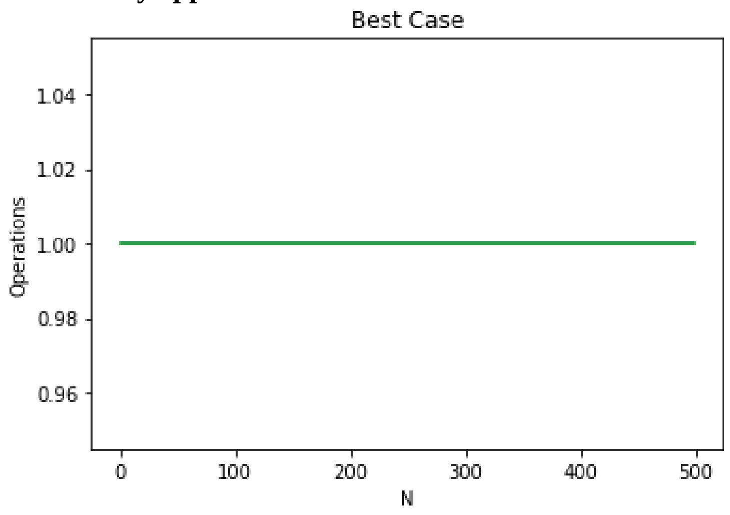

Figure 2: Best case of the number of operations as N increases for PowerArrayApp

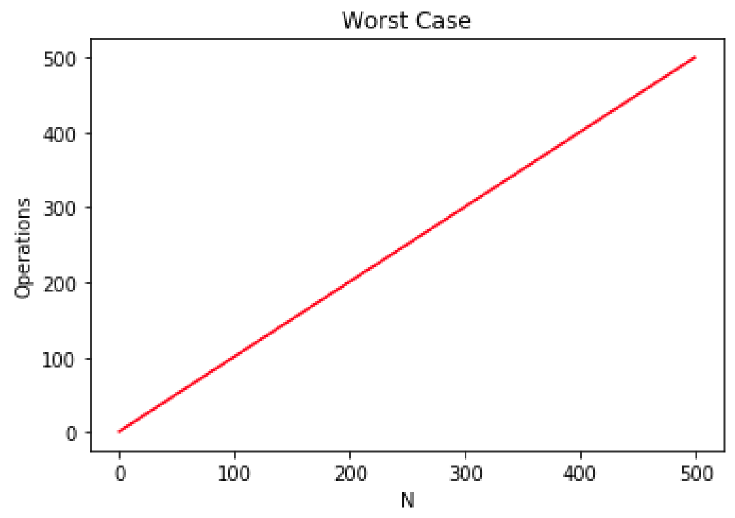

Figure 3: Worst case of the number of operations as N increases for PowerArrayApp

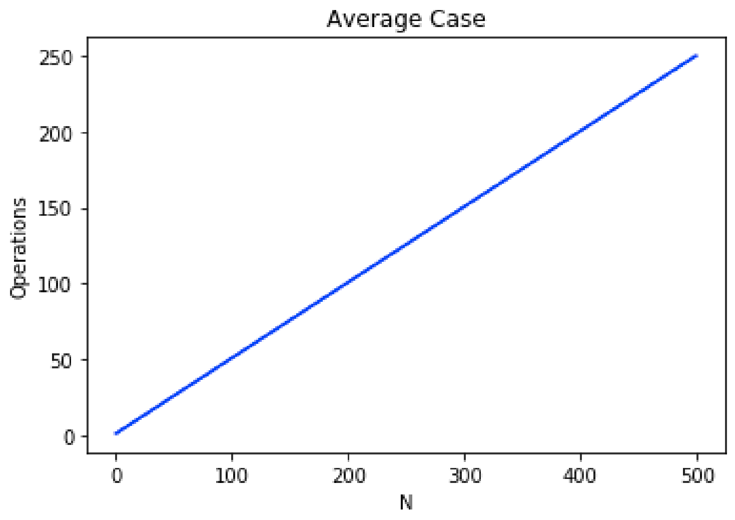

Figure 4: Average case of the number of operations as N increases for PowerArrayApp

PowerBSTApp

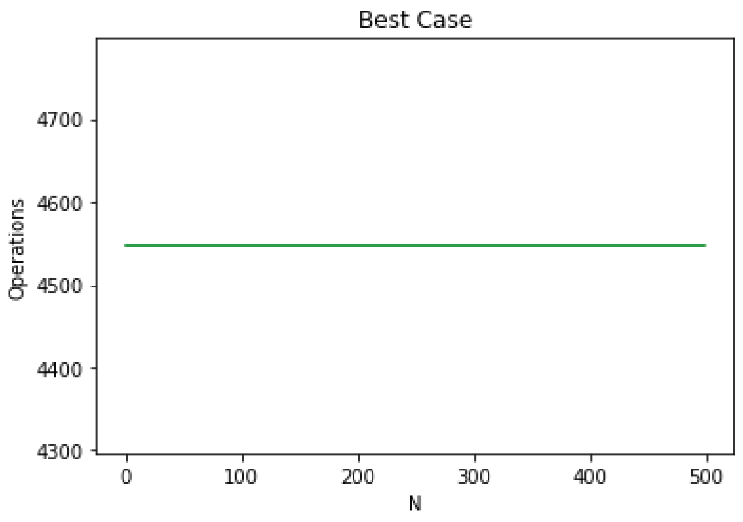

Figure 5: Best case of the number of operations as N increases for PowerBSTApp

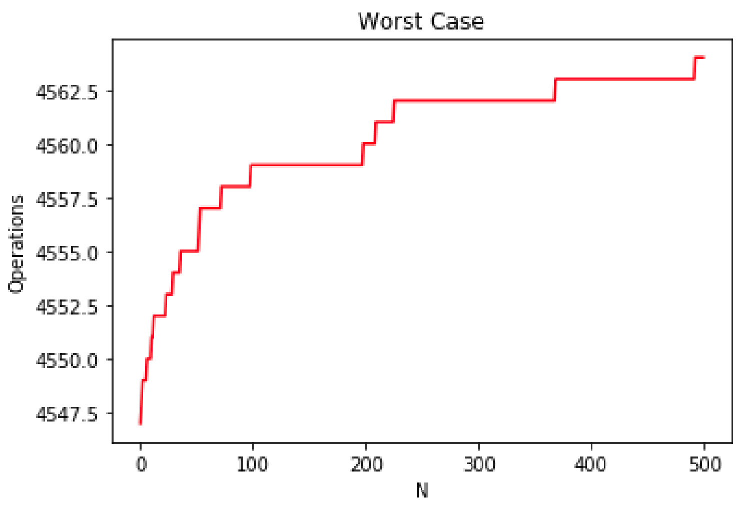

Figure 6: Worst case of the number of operations as N increases for PowerBSTApp

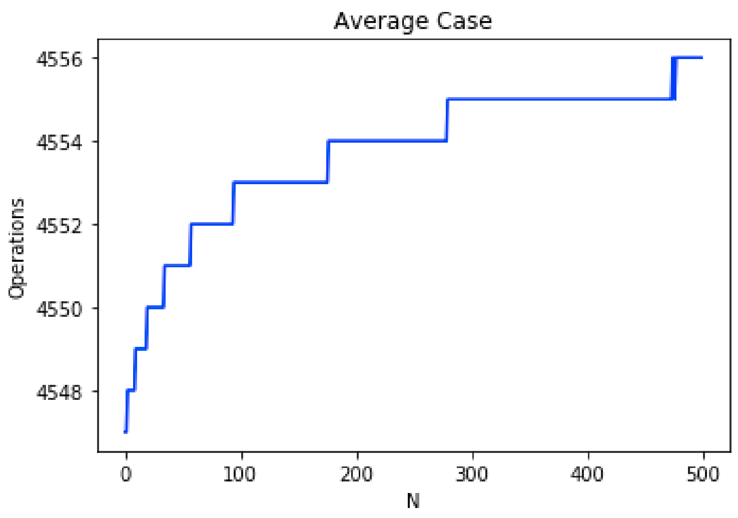

Figure 7: Average case of the number of operations as N increases for PowerBSTApp

Discussion and Conclusion

For all cases, PowerBSTApp had a greater number of operations. This is mainly because when the insert method in the binary search tree is invoked, comparisons are continuously being made and increase as the tree increases in size.

However, in figure 6 and 7, it can be seen that as the data set increases, the number of operations decrease in the binary search tree as compared to the array where it increases linearly as seen in figure 3 and 4. The worst case and average case of the binary search tree have a logarithmic pattern which means that it has a better efficiency than the linearity of the array. This is so because of the nature of binary trees. When finding a value, it does not have to iterate through the entire data structure to find a match.

For both cases, the best case was a constant value of all N values. This is because in both cases, the first item to be stored in the respective data structures is also the first value that was searched for in the subsets.

References

[1] P. Marais, *Binary Search Trees,* 2019.

[2] SJ, "Algorithms @tutorialhorizon," 16 09 2015. [Online]. Available: https://algorithms.tutorialhorizon.com/binary-search-tree-complete-implementation/. [Accessed 05 03 2019].

Appendix

Git log

```
commit 4a57a1d19e7d206826576949b2e3a77d1f4bfe3d
Author: Niceta Nduku <NDKNIC001@myuct.ac.za>
Date: Tue Mar 5 17:11:59 2019 +0200

Final project code

commit 33b666742c05c52c1383b77fd070c704ca00e583
Author: Niceta Nduku <NDKNIC001@myuct.ac.za>
Date: Tue Mar 5 14:42:38 2019 +0200

...
Author: Niceta Nduku <NDKNIC001@myuct.ac.za>
Date: Thu Feb 28 16:27:50 2019 +0200

Added my Binary Search Tree app

commit 15c5f008ccb2a2bcdb2f9c45f3d6c32e5a1cc564
Author: Niceta Nduku <NDKNIC001@myuct.ac.za>
Date: Tue Feb 26 15:48:08 2019 +0200

Practical 1: Code for the PowerArray Application and the object
power data
```