Maria Veltcheva VLTMAR001 3 March 2019

CSC2001F

ASSIGNMENT 1
STATISTICAL ANALYSIS

OVERVIEW

The aim of this report is to explore the difference in performance between an unsorted array and a binary search tree (BST). Outlined in this report is the design of the applications which implement the data structures in question, a description of the experiment undertaken as well as the results obtained during experimentation. Finally, an analysis of the results is included where the performance of each data structure is explored.

APPLICATION DESIGN

In order to analyse the performance of the array and BST data structures, various java applications were written that implement each data structure separately. The applications include methods that populate the respective data structures with data from a csv file (*cleaned-data.csv*) as well as methods that perform either a traversal or a search operation on the data structure in question. The operation invoked depends on the command line arguments passed when running the respective applications from the terminal. An in-depth description of the application design is provided below:

NOTE: The *cleaned_data.csv* file contains records that each list 8 values pertaining to individual household power consumption. The elements that populate the respective data structures are a concatenation of only the Date/Time, Power and Voltage values of the individual records in the csv.

PowerArrayApp

The first application, ***PowerArrayApp***, is the one which implements an unsorted array as its data structure. The application accepts up to four command line arguments. The first is a Date/Time string of the form “DD/MM/YYYY/hh:mm:ss” which denotes a Date/Time that will be searched for in the array, the second is a number (N) between 0 and 500 which denotes the number of records that will be loaded into the array, the third is a string that denotes the path to a textfile which will store counter values for an array of a specific size (the counter values will be explained in detail later in the report) and the fourth denotes the path to a textfile that stores a subset of data from cleaned_data.csv. When the application is run, the main() method first checks if more than one argument has been passed, if one or less arguments are passed, then the default size of the array is set to 500 and a default path to a file that stores random counter values is set, otherwise then the array size is set to the value passed and the path to the textfile is stored for use later in the program.

After that check, the main() method invokes the application’s loadData() method, which populates the array with records from the csv. When the loadData() method is invoked, the application then traverses through the csv file using a Scanner and extracts elements to be loaded into the array by splitting each record into its necessary components and concatenating them as needed. Each element is then added one by one until the array size is met.
After the array has been populated, the main() method then checks if a Date/Time string (`args[0]`) was passed as an argument. If so, then the method printDateTime(dateTime) is invoked, which traverses the array searching for the record matching the given Date/Time and prints out its Date/Time, Power and Voltage values if found and “Date/Time not found” if not found. If no

arguments are passed to the main method, then the printAllDateTimes() method is invoked, which traverses the array and prints out the Date/Time, Power and Voltage values for each array item. If a subset data file was passed as an argument (`args[0]`) then the file is traversed, and each Date/Time string is passed as a parameter to printDateTime and searched for in the array one by one.

PowerBSTApp

The second application, ***PowerBSTApp***, is the one which implements a binary search tree (BST) as its data structure. The application has the same methods that perform the same functions as those in PowerArrayApp. It also accepts the same command line arguments as those in PowerArrayApp and its main() method navigates between attribute states and method invocations in exactly the same way that the PowerArrayApp application does. The only difference is that a BST is used to store the Date/Time records as opposed to an unsorted array. The implementation of the BST data structure is done with the help of two other classes, namely the ***BinarySearchTree*** class and the ***BinaryTreeNode*** class. A brief description of each class and its interactions with PowerBSTApp is given below:

The ***BinaryTreeNode*** class will allow for the instantiation of objects with the following attributes: data in the form of a Date/Time record, as well as links to two BinaryTreeNodes (its left and right children). Along with the aforementioned attributes, the class also contains various get and set methods to allow for the retrieval and manipulation of the node's attributes. The linking of BinaryTreeNodes together constitutes a ***BinarySearchTree (BST)***. The insert method of the BST class is what links objects of type BinaryTreeNode (BTN) together. It does this by traversing the tree and comparing the Date/Time record of the node to be inserted with that of the current node at each step. If a nodes data "comes before" that of the current node then the method traverses left, and if a nodes data "comes after" that of the current node then the method traverses right (lexical ordering is used). It does this traversal until a leaf node is reached, thereafter it creates a new BTN and links it to the leaf node in question as either a left or a right child. The BinarySearchTree is rooted at some starting node, the link to which is stored in the rootNode attribute of the BST class.
When the PowerBSTApp application is loading data from the csv into a tree, it first creates an empty tree – which sets the counter attribute of the BST class (explained later) to zero and the rootNode to null. It then invokes the insert(dateTime) method as it traverses the csv, passing the Date/Time record to be stored in the node as an argument. Once the tree has been populated, the main() method navigates to either the printDateTime(dateTime) method or the printAllDatesTimes() method according to the command line arguments passed when running PowerBSTApp from the command line, as explained earlier. If the printDateTime(dateTime) method is invoked, the BST class's find method is invoked, in order to search for the given Date/Time record in the tree. The find method traverses the tree in a similar fashion to the insert method, comparing the Date/Time record given with that of each node visited during the traversal. If the Date/Time records match, the method returns the node containing the record, and if the node is not found then null is returned.
If the printAllDateTimes() method is invoked, the BST class's preOrder() method is invoked, in order to traverse the tree and print out the Date/Time, Power and Voltage values for all records in the tree. The preOrder() method works in conjuction with the BST class's visit and format methods. The format method ensures that the nodes data is formatted in a readable order (by separating its Date/Time, Power and Voltage values) while the visit method simply prints out the formatted data.

The explanations above cover two of the possible ways to run each application from the command line (excluding the ways that pass invalid arguments to the respective applications), namely:

- java PowerArrayApp
- java PowerArrayApp "DD/MM/YYYY/hh:mm:ss"

OR

- java PowerBSTApp
- java PowerBSTApp "DD/MM/YYYY/hh:mm:ss"

The other way of running the applications is by passing the size of the data structure (N), the path to a textfile that stores counter values and the path to a subset data file as arguments. This way of running the applications aids the experimentation process, which is described in the next section of the report.

EXPERIMENT DESCRIPTION

The java applications as described above do not output anything that allows the performance of the respective data structures to be analysed. They merely populate each data structure and allow for various operations to be performed on each. The aim of the experiment is to demonstrate and analyse the effect that each data structure has on the performance of the application that implements it. Each application is therefore compared according to the performance of its printDateTime method as the method follows paths of traversal in different ways due to the difference in structure between the array and the BST. The paths of traversal, if shorter or longer than one another, will result in a performance difference between the two data structures. Therefore by instrumenting the applications appropriately, the paths of traversal and differences in performance, if any, between the aforementioned data structures can be observed.

The applications are instrumented as follows:
Once the printDateTime method has been invoked, a counter is initialised to zero and is incremented every time the values of Date/Time strings are compared as these comparisons represent steps along the path of traversal. Therefore by the time the search has terminated, the counter will show how many steps were taken in order to search for the given Date/Time string. These counters are calculated throughout the experiment for datasets of different sizes and for data items at varied positions within the data set.

The method of experimentation is as follows:

1) Instrument the code as necessary (explained above).
2) Create a textfile with a subset of data from cleaned_data.csv. This is done with an application that was created to aid in experimentation. The application is called ***BuildFile*** and will be explained in detail later on in the report.
3) Populate each data structure with the same data items that are in the subset textfile.
4) Pass the subset textfile as a parameter to each application. Thereafter, the application in question will traverse through the textfile and invoke printDateTame, passing each data item as a parameter. When passing the textfile as a parameter, the path to a textfile that will store counter values must also be passed as a parameter because once the printDateTime method has searched for the given dateTime and incremented its counter accordingly, the counter value is appended to the textfile storing counter values for that specific iteration of experimentation.

A bash script is used to vary the size of the subset textfile and subsequently the size of the data set in the application that is populated with data from the textfile. The size varies from 0 to 500 and is incremented in steps of 20. Each incrementation initates a new iteration of experimentation. During each iteration, the script creates the subset textfile and then calls each application with the following parameters:

java PowerArrayApp “DD/MM/YYYY/hh:mm:ss” N “path to a textfile that stores counters” “path to subset textfile”

NOTE: The first argument here can be any arbitrary string because the application will not navigate in such a way that the argument is utilised. The second argument here denotes the number of data

items that must be loaded into the array/BST and will instruct the application to stop loading once that many items have been loaded. The same applies for PowerBSTApp.
NOTE: The script can be run from the bin directory by running the following command: ./Part5

At the end of the experiment, what remains is a set of textfiles storing counter values for different sized data sets - there is a set of textfiles for each application (they are stored in the textfiles directory under the subdirectory labelled Part5). Each file is then loaded into excel in order to extract best, worst and average case counter values. These values are then plotted and the graphs compared.

TRIAL VALUES

In order to test the functionality of PowerArrayApp and PowerBSTApp after the added instrumentation, each application was tested with three known Date/Time strings, one unkown Date/Time string and with no parameters. The operation counts were recorded in each case.
The results are as follows:

PowerArrayApp

Known Dates:

```
vltmar001@sl-dual-143:~/CSC2001F/Assignment_One/bin$ java PowerArrayApp "16/12/2006/18:18:00"
Picked up _JAVA_OPTIONS: -Xms512m -Xmx4096m

Date/time:                  16/12/2006/18:18:00
Power:                      4.472
Voltage:                    233.290

Operation Count:            12
```

```
vltmar001@sl-dual-143:~/CSC2001F/Assignment_One/bin$ java PowerArrayApp "17/12/2006/00:19:00"
Picked up _JAVA_OPTIONS: -Xms512m -Xmx4096m

Date/time:                  17/12/2006/00:19:00
Power:                      2.396
Voltage:                    239.030

Operation Count:            56
```

```
vltmar001@sl-dual-143:~/CSC2001F/Assignment_One/bin$ java PowerArrayApp "16/12/2006/19:26:00"
Picked up _JAVA_OPTIONS: -Xms512m -Xmx4096m

Date/time:                  16/12/2006/19:26:00
Power:                      3.620
Voltage:                    233.470

Operation Count:            105
```

Unknown Date:

```
vltmar001@sl-dual-143:~/CSC2001F/Assignment_One/bin$ java PowerArrayApp "01/12/2006/18:14:00"
Picked up _JAVA_OPTIONS: -Xms512m -Xmx4096m
Date/time not found

Operation Count:            500
```

No Parameters:
First ten lines

```
vltmar001@sl-dual-143:~/CSC2001F/Assignment_One/bin$ java PowerArrayApp
Picked up _JAVA_OPTIONS: -Xms512m -Xmx4096m
Date/time:                16/12/2006/19:51:00
Power:                    3.388
Voltage:                  233.220

Date/time:                16/12/2006/23:20:00
Power:                    1.222
Voltage:                  241.580

Date/time:                17/12/2006/00:29:00
Power:                    0.612
Voltage:                  243.680
```

Last ten lines

```
Date/time:                16/12/2006/22:01:00
Power:                    1.786
Voltage:                  237.680

Date/time:                16/12/2006/17:43:00
Power:                    3.728
Voltage:                  235.840

Operation Count:          0
```

PowerBSTApp

Known Dates:

```
vltmar001@sl-dual-143:~/CSC2001F/Assignment_One/bin$ java PowerBSTApp "16/12/2006/18:18:00"
Picked up _JAVA_OPTIONS: -Xms512m -Xmx4096m

Date/time:                16/12/2006/18:18:00
Power:                    4.472
Voltage:                  233.290

Operation Count:          9
```

```
vltmar001@sl-dual-143:~/CSC2001F/Assignment_One/bin$ java PowerBSTApp "17/12/2006/00:19:00"
Picked up _JAVA_OPTIONS: -Xms512m -Xmx4096m

Date/time:                17/12/2006/00:19:00
Power:                    2.396
Voltage:                  239.030

Operation Count:          13
```

```
vltmar001@sl-dual-143:~/CSC2001F/Assignment_One/bin$ java PowerBSTApp "16/12/2006/19:26:00"
Picked up _JAVA_OPTIONS: -Xms512m -Xmx4096m

Date/time:                16/12/2006/19:26:00
Power:                    3.620
Voltage:                  233.470

Operation Count:          9
```

Unknown Date:

```
vltmar001@sl-dual-143:~/CSC2001F/Assignment_One/bin$ java PowerBSTApp "01/12/2006/18:14:00"
Picked up _JAVA_OPTIONS: -Xms512m -Xmx4096m
Date/time not found

Operation Count:          6
```

No Parameters:
First 10 lines

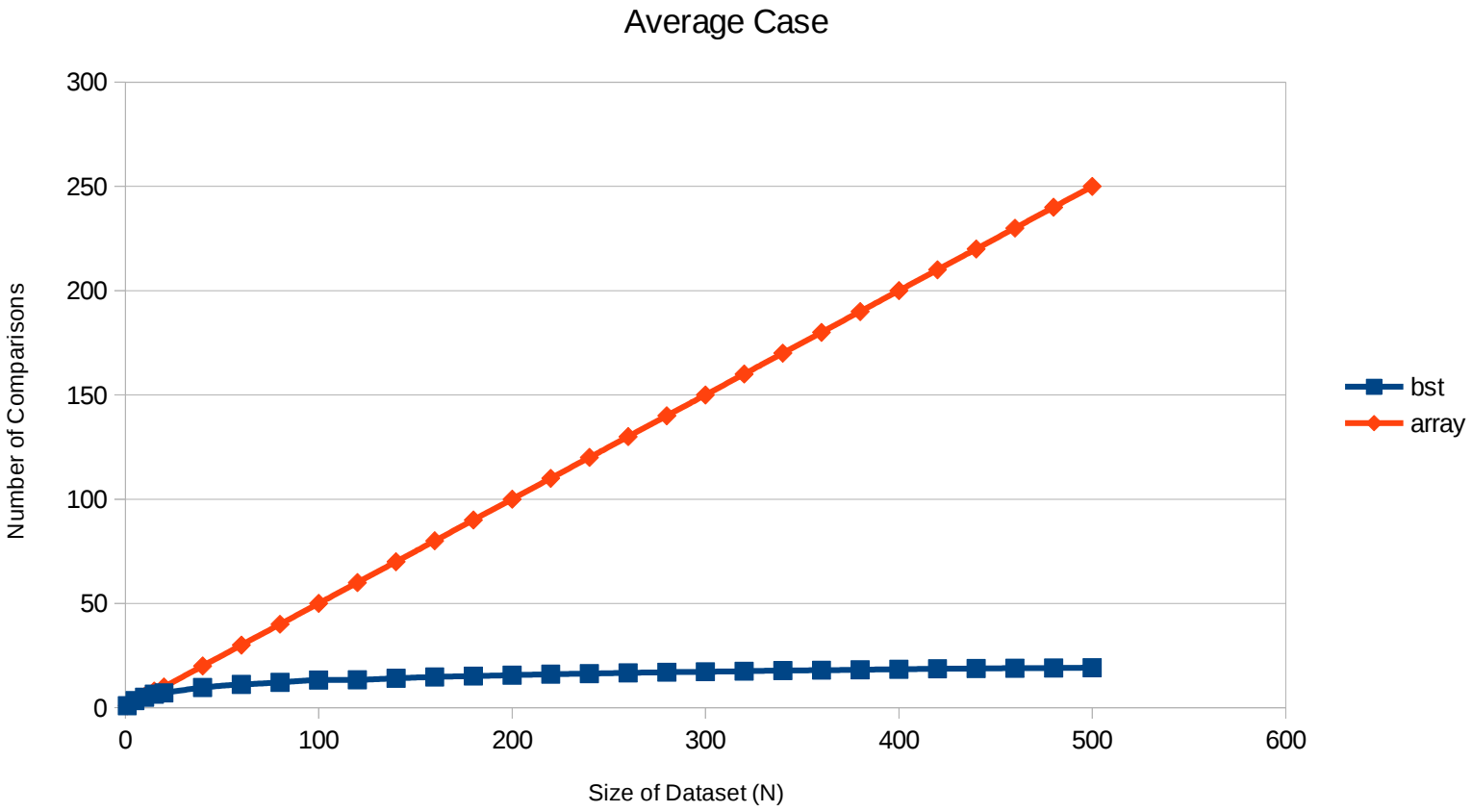
```
vltmar001@sl-dual-143:~/CSC2001F/Assignment_One/bin$ java PowerBSTApp
Picked up _JAVA_OPTIONS: -Xms512m -Xmx4096m
Date/time:                16/12/2006/19:51:00
Power:                    3.388
Voltage:                  233.220

Date/time:                16/12/2006/17:37:00
Power:                    5.268
Voltage:                  232.910
```

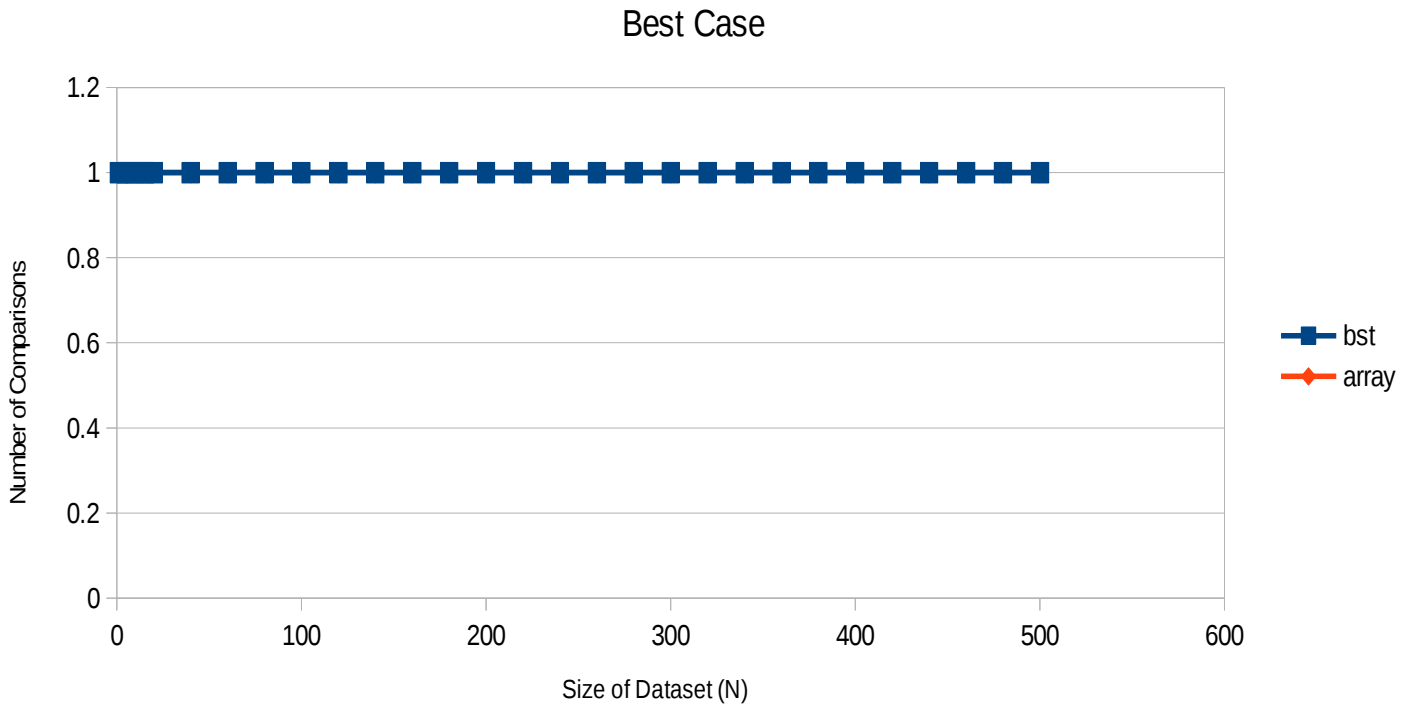
Last 10 Lines

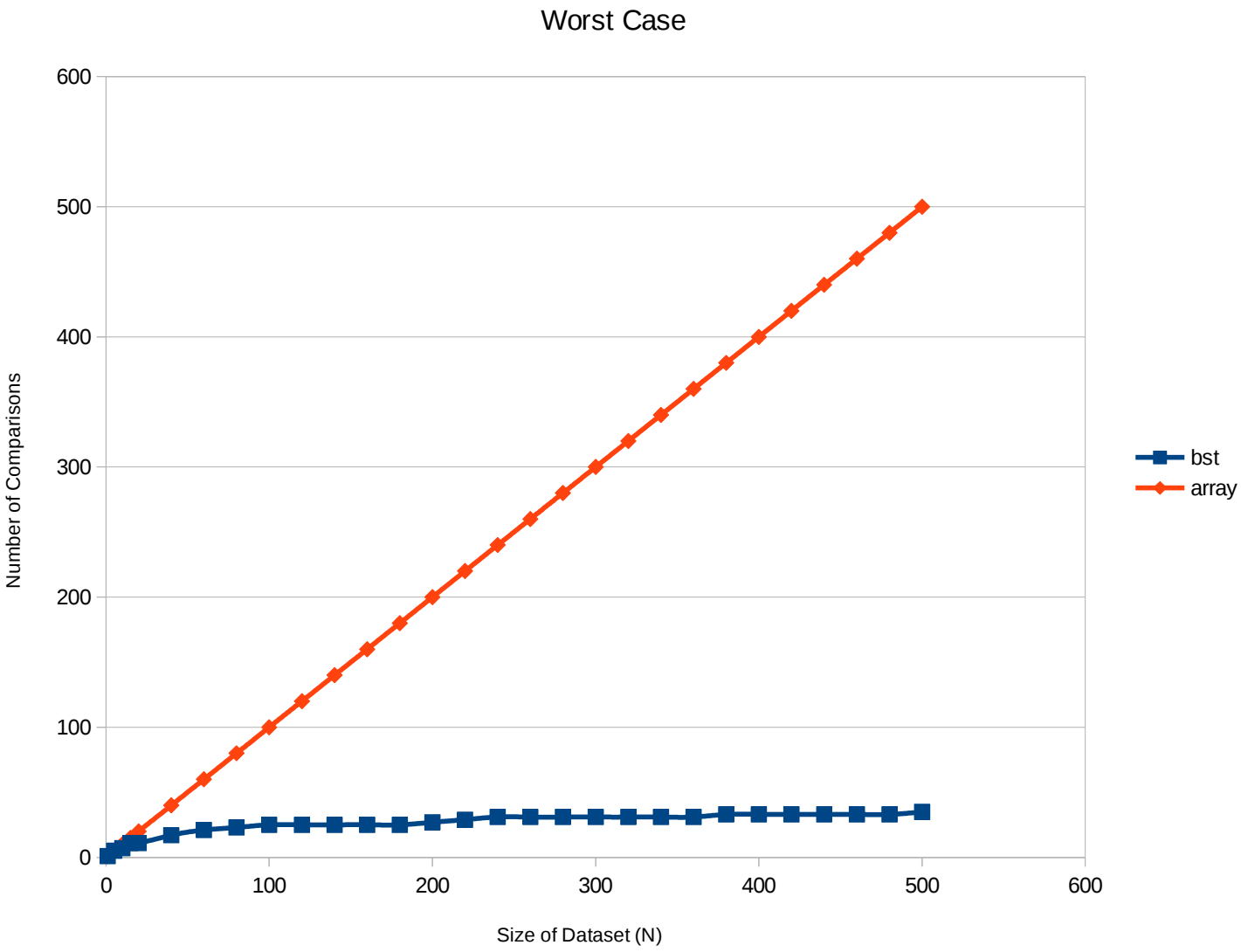
```
Date/time:                17/12/2006/01:42:00
Power:                    3.800
Voltage:                  241.780

Date/time:                17/12/2006/01:43:00
Power:                    2.664
Voltage:                  243.310

Operation Count:          0
```

RESULTS OF EXPERIMENTATION

Below are the graphs comparing the average, best and worst case comparison counts for different sized datasets poluated into the array and BST data structures.




Best Case
1.2
1
0.8
0.6
0.4
0.2
0
Number of Comparisons
0
100
200
300
400
500
600
Size of Dataset (N)
bst
array


Worst Case
600
500
400
300
200
100
0
Number of Comparisons
0
100
200
300
400
500
600
Size of Dataset (N)
bst
array

ANALYSIS OF RESULTS

In the graph depicting worst case values for different values of N, the best case value for all values of N remains the same. For both the array and the BST, the best case is always that the item to be searched for is the first item in the dataset and therefore the best case will always result in only one comparison being made.

In the graphs depicting average and worst case values, it can be seen that the search performance of the array is of order N. This is due to the fact that the array stores data linearly and therefore when an item needs to searched for that is at position n in the array, the application has to make n comparisons before it reaches n. The graphs also show that the search performance of the BST is significantly lower at an order of $\log_2(N)$. This is due to the fact that that the structure of a BST ensures that the depth of the BST is of order $\log_2(N)$ so when the application has to search for a node at depth n the path length is $\log_2(N)$. The reason for the $\log_2(N)$ depth is because if the tree is balanced and fully populated ie. every node has two children aside from the leaf nodes, then the number of nodes in the tree is 2^n, where n is the height of the tree, rearranging the equation $N = 2^n$, we get $n = \log_2(N)$. We can deduce from the performance of the BST seen in the graph that the data passed is unordered and results in balanced binary search trees being built. If the data were ordered, the binary search tree would turn into a linked list and the performance would resemblt that of the array.

The graphs show that in the worst and avergae case searches, implementing a binary search tree and populating it with unordered data results in significantly faster search times than when implementing an array.

CREATIVE COMPONENT

As described earlier in the report, a bash script was written and used for the automation of the experimentation process. For explanations on how to run the script and how it operates, please refer to the section that discusses the experiment. Along with the bash script, a java class named BuildFile was written in order to create subset text files to be used during the experiment. BuildFile's main method is invoked from the script during each iteration of experimentation, thereafter a subset textfile of a specific size is created and passed to PowerArrayApp and PowerBSTApp as a paramater. The class accepts two command line arguments, the first is the size of the subset to be created and the second is the path to the textfile that will store the subset. The class operates the same way that PowerArrayApp does when loading its data in that it traverses the the cleaned_data.csv file and loads data items one by one into the textfile until the size of the subset is reached. The only difference is that BuildFile only loads Date/Time strings as opposed to Date/Time, Power and Voltage values. More information on the BuildFile method can be found in the javadocs for this project. The application can be run from the bin folder if preffered, but it must be passed the appropriate arguments, as the application was only designed to be run from within the bash script that is

written in such a way that it passes these arguments. If the arguments are not passed then the program will produce errors.

GIT USAGE LOG

First 10 lines

```
Author: Maria Veltcheva <VLTMAR001@myuct.ac.za>
Date:   Sat Feb 23 19:25:19 2019 +0200

    A1: Code was edited to change the data item that was being inputted into the array. Data item was originally an entire row from the csv, d
ata item is now only inclusive of the necessary fields. Errors regarding adding data items to the array at the correct indices were also fixed
.

commit cb9908f7e5401cfc8f8bb47ebcfaeae9ba9516b5
Author: Maria Veltcheva <VLTMAR001@myuct.ac.za>
Date:   Sat Feb 23 18:56:29 2019 +0200

    A1: PowerArrayApp class created. Code added to class to load data items from csv file into an array.
```

Last 10 lines

```
commit d36a1e0fd476dbf6d48ce6dd99582cdf6754da21
Author: Maria Veltcheva <VLTMAR001@myuct.ac.za>
Date:   Tue Mar 5 16:39:46 2019 +0200

    A1: All classes except for the BuildFile class were edited to exclude the insertion count comparisons from the total comparisons. The Powe
rArrayApp and PowerBSTApp classes were edited to include an additional paramater that passes the path to a textfile to be travered and queried
 by printDatTime for analysis. The functionality of the bash script was changed accordingly in order to allow the classes to traverse the text
file as opposed to the script traversing the textfile. All javadocs updated to accomodate the changes in code.

commit 4c480bb148c694f2c1e8894a50e065cd35de2647
```