Report

Object-Oriented Design:

Part 1: For the PowerArrayApp, The csv file was loaded into a File object. The PowerArrayApp class contains all the methods needed to implement the data structure. The main method uses the File object to create a 2 dimensional Array for use as the database. Once the application is executed by the user, if the application is executed with a parameter, the printDateTimes method is called to search for the specified date in the parameter and return its information.
If there are no parameters in the execution, the printAllDateTimes method is called to iteratively print out all the dam details.

Part 2: For the PowerBSTApp, the BSTNode and BST classes had to be created. 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/ .
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.

Testing

Part 2	
Input	**Output**
16/12/2006/17:44:00	16/12/2006/17:44:00,5.894,0.000,232.690 opCount = 222
16/12/2006/21:35:00	16/12/2006/21:35:00,1.872,0.000,238.480 opCount = 54
16/12/2006/19:11:00	16/12/2006/19:11:00,3.414,0.000,234.190 opCount = 62
18/12/2009/11:11:11	Date/time not found opCount = 500
	16/12/2006/19:51:00,3.388,0.158,233.220 16/12/2006/23:20:00,1.222,0.046,241.580 17/12/2006/00:29:00,0.612,0.000,243.680 16/12/2006/20:35:00,3.226,0.078,233.370 16/12/2006/17:37:00,5.268,0.398,232.910 16/12/2006/19:23:00,3.334,0.000,234.360 16/12/2006/18:25:00,4.870,0.000,233.740

	16/12/2006/20:30:00,3.262,0.076,234.540 17/12/2006/00:32:00,2.376,0.056,241.860 17/12/2006/00:22:00,0.276,0.000,240.560 … 16/12/2006/23:15:00,0.386,0.000,242.390 17/12/2006/00:42:00,0.382,0.108,243.650 16/12/2006/23:52:00,3.458,0.000,238.890 16/12/2006/18:38:00,2.912,0.048,234.020 16/12/2006/17:41:00,3.430,0.156,237.060 16/12/2006/19:21:00,3.332,0.000,234.020 16/12/2006/23:47:00,2.540,0.060,241.230 17/12/2006/00:09:00,0.838,0.334,242.090 16/12/2006/22:01:00,1.786,0.096,237.680 16/12/2006/17:43:00,3.728,0.000,235.840 opCount = 0

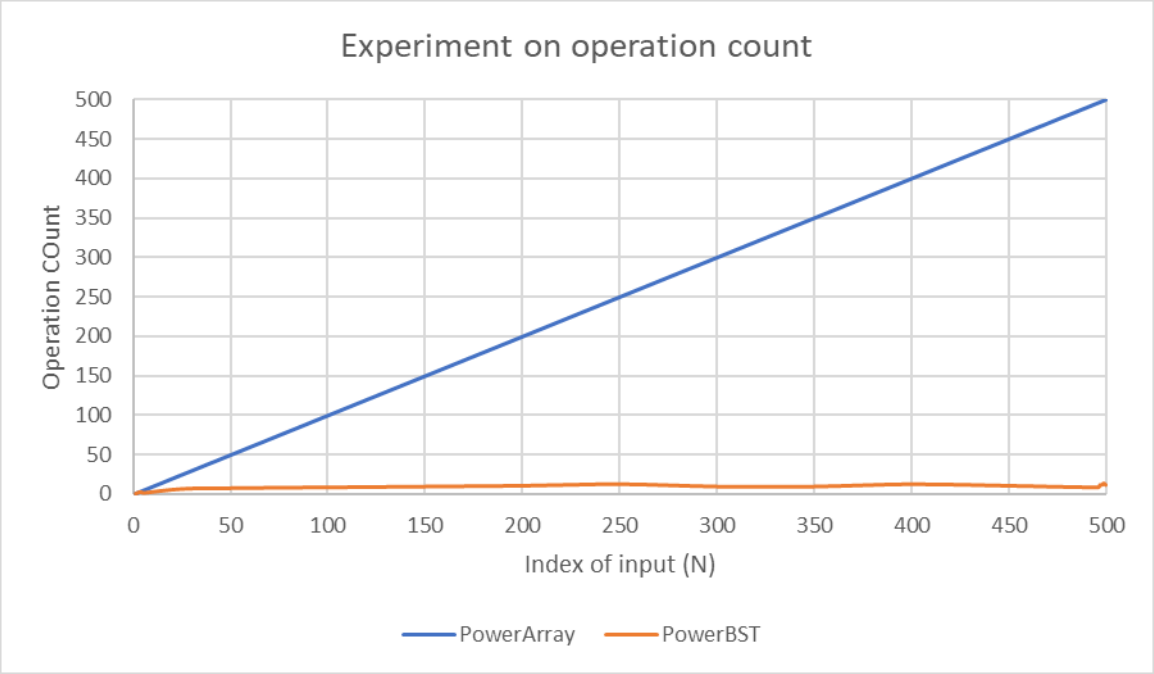
Part 4	
Input	**Output**
16/12/2006/17:44:00	16/12/2006/17:44:00,5.894,0.000,232.690 opCount = 9
16/12/2006/21:35:00	16/12/2006/21:35:00,1.872,0.000,238.480 opCount = 11
16/12/2006/19:11:00	16/12/2006/19:11:00,3.414,0.000,234.190 opCount = 8
18/12/2009/11:11:11	Date/time not found opCount = 11
	16/12/2006/17:38:00,4.054,0.422,235.240 16/12/2006/17:39:00,3.384,0.282,237.140 16/12/2006/17:40:00,3.270,0.152,236.730 16/12/2006/17:58:00,4.058,0.200,234.680 16/12/2006/18:26:00,4.868,0.000,233.840 16/12/2006/17:44:00,5.894,0.000,232.690 16/12/2006/17:42:00,3.266,0.000,237.130 16/12/2006/17:41:00,3.430,0.156,237.060 16/12/2006/17:59:00,2.472,0.058,236.940 16/12/2006/18:27:00,4.866,0.000,233.790 … 17/12/2006/00:45:00,2.456,0.076,241.180 17/12/2006/01:14:00,5.246,0.230,237.910 17/12/2006/01:16:00,3.858,0.240,238.520 17/12/2006/01:21:00,4.652,0.142,237.920 17/12/2006/01:29:00,2.080,0.000,241.610 17/12/2006/01:42:00,3.800,0.000,241.780 16/12/2006/22:25:00,2.428,0.070,239.680 17/12/2006/00:46:00,2.450,0.074,240.820 17/12/2006/01:17:00,2.822,0.188,239.550 17/12/2006/01:43:00,2.664,0.000,243.310 opCount = 0

The Experiment

The goal of the experiment was to see which data structure performed better and when which of the two performed better.

The dataset for the input values includes every dateTimes cell from the csv file.
To gather my data, I used iteration to extract the operation count values from the PowerArrayApp and the PowerBSTApp.

The graph below shows the operation count for both apps at the same value of N.



As depicted by the graph above, The best case for the PowerArrayApp occurs when N=1 and is therefore O(1), the worst case is where N = 500. The trend is linear and the worst case therefore, in general be O(n). The average case in in this instance is where the operation count = 250. In general, the average case is N/2

The best case for the PowerBSTApp is when N=1 and is therefore N(1). The worst case is where operation count = 18. The average for this set was operation count =11. There is no discernible trend for the efficiency of the app.

Given the information above, it is clear that the PowerBSTApp is much more efficient at every level when compared to the PowerArrayApp.

Creativity employed

1. I found a way to make the PowerArrayApp in as few lines as possible, while also confining the entire implementation to one java file.

2.

Git usage log

```
commit eecbf32abf45dc5ecb6712b7e19714712cebc397 (HEAD -> master)
Author: Mahmood-Ali Parker <ali@HP.localdomain>
Date:   Wed Mar 6 23:41:01 2019 +0200

    actual final

commit 3edd878cd20a4af681d4e83689639d1d29e27bb3
Author: Mahmood-Ali Parker <ali@HP.localdomain>
Date:   Wed Mar 6 23:20:33 2019 +0200

    Finished code which runs perfectly
~
```