Assignment 2

Name: Rahul Deeljore
Student Number: RHLDEE001

Intro:

In this assignment I will conduct an experiment to demonstrate the speed difference for search and insert operations between a BinarySearchTree and an AVLTree.

Aim:

The size of the BinarySearchTree and AVLTree will be varied from 1 to 211 and the count for insertion and searching will be recorded. Every element in the data structures will have its own insert and search count.

The data will then be analysed to see the difference between the BinarySearchTree and the AVLTree.

The experiment will then be repeated with the size of the dataset of 211. In this last part, the dataset will be sorted before adding them to the data structures. We will then analyse how this has affected the search and insert counts for the 2 data structures.

Method:

There are 2 csv files:
“thedams.csv” contains the unsorted dams
“dam.csv” contains the sorted dams

The classes used in this experiment are:

1. Dam
2 BinarySearchTree, BinaryTree, BinaryTreeNode, BinaryQueueNode, BinaryQueue
3. DamArrayApp
4. DamBSTApp
5. BSTrun
6. AVLrun
7. DamAVLApp

The Dam class contains the instance variables that are required to create a Dam.
There is a contructor that can be used to initalize Dam objects.
There is an appropriate toString() method to print out Dam objects.
There is a compareTo() method that compares Dam names.

The BinarySearchTree class can create a BinarySearchTree of any data type.
It contains an insert method to add values to the BinarySearchTree.
It contains a delete() method to remove values.
It has a preOrder(), inOrder() and postOrder() method to print the values from the BinarySearchTree in different sorting pattern.

The BinaryTree class contains the implementation for a Binary Tree. This class is used for the implementation of the BinarySearchTree.

The BinaryTreeNode class contains the implementation for the nodes in the BinarySearchTree and AVLTree used.

The DamArrayApp class contains a method that will read the csv file, create the Dam objects from it and put those objects in an array. This will be used to insert the objects in the BinarySearchTree and AVLTree.

The DamBSTApp will create a variable of type BinarySearchTree and store all the Dam objects inside it.
The main method will print all Dam objects from the BinarySearchTree is no argument is given.
The main method will print the Dam object whose name is given as argument. It will also print out the number of comparisons that were made before finding the required Dam. It will also print out the number of insert comparisons made when inserting the object. In the case when the Dam was not found, the user will be told so.

The DamAVLApp will create a variable of type AVLTree and store all the Dam objects inside it.
The main method will print all Dam objects from the AVLTree if no argument is given.
The main method will print the Dam object whose name is given as argument. It will also print out the number of comparisons that were made before finding the required Dam. It will also print out the number of insert comparisons made when inserting the object. In the case when the Dam was not found, the user will be told so.

BSTrun was used to obtain data to plot graphs. The size of the BinarySearchTree was varied and every element searched. The number of comparisons when searching and inserting was then recorded.

AVLrun was used to obtain data to plot graphs. The size of the AVLTree was varied and every element searched. The number of comparisons when searching and inserting was then recorded.

Trials:

1.DamAVLApp

When the DamAVLApp was run with 3 known dams the output was:

```
rahul@rahul-VirtualBox:~/files/Assignment2/AVLoriginal$ java DamAVLApp "Hartbeespoort Dam"
Name: Hartbeespoort Dam, FSC: 186.44, Percentage: 96.5
insert counts:1
Search Counts:6


rahul@rahul-VirtualBox:~/files/Assignment2/AVLoriginal$ java DamAVLApp "Ngotwane Dam"
Name: Ngotwane Dam, FSC: 19.033000000000001, Percentage: 4.6
insert counts:0
Search Counts:12

rahul@rahul-VirtualBox:~/files/Assignment2/AVLoriginal$ java DamAVLApp "Zaaihoek Dam"
Name: Zaaihoek Dam, FSC: 184.63, Percentage: 58.5
```

```
insert counts:8
Search Counts:14
```

When the DamAVLApp was run with 1 unknown dam the output was:

```
rahul@rahul-VirtualBox:~/files/Assignment2/AVLoriginal$ java DamAVLApp "Vanilla Dam"
Dam not found
```

When no arguments were given, the output was:

first 10 lines:

```
Name: Lindleyspoort Dam, FSC: 14.208, Percentage: 2.7
insert counts:3
Name: Flag Boshielo Dam, FSC: 185.13, Percentage: 40.200000000000003
insert counts:6
Name: Buffelspoort Dam, FSC: 10.183, Percentage: 71.400000000000006
insert counts:2
Name: Blyderivierpoort Dam, FSC: 54.369, Percentage: 50
insert counts:6
Name: Armenia Dam, FSC: 12.957000000000001, Percentage: 10
insert counts:7
```

Last 10 lines:

```
Name: Wriggleswade Dam, FSC: 91.471000000000004, Percentage: 98.7
insert counts:8
Name: Woodstock Dam, FSC: 373.25, Percentage: 77.099999999999994
insert counts:8
Name: Xonxa Dam, FSC: 115.86, Percentage: 100
insert counts:8
Name: Xilinxa Dam, FSC: 13.823, Percentage: 27.2
insert counts:8
Name: Zaaihoek Dam, FSC: 184.63, Percentage: 58.5
insert counts:8
```

2. DamBSTApp

When running DamBSTApp with 3 known dams the output was:

```
rahul@rahul-VirtualBox:~/files/Assignment2/AVLoriginal$ java DamBSTApp "Pella Dam"
Name: Pella Dam, FSC: 2.111, Percentage: 38
insert counts:3
search counts:6

rahul@rahul-VirtualBox:~/files/Assignment2/AVLoriginal$ java DamBSTApp "Zaaihoek Dam"
Name: Zaaihoek Dam, FSC: 184.63, Percentage: 58.5
insert counts:7
search counts:14
```

```
rahul@rahul-VirtualBox:~/files/Assignment2/AVLoriginal$ java DamBSTApp "Hartbeespoort
Dam"
Name: Hartbeespoort Dam, FSC: 186.44, Percentage: 96.5
insert counts:1
search counts:2
```

When running DamBSTApp with 1 unknown Dam the output was:

```
rahul@rahul-VirtualBox:~/files/Assignment2/AVLoriginal$ java DamBSTApp "jim morrison"
Dam not found
```

When running DamBSTApp with no arguments, the output was:

first 10 lines:

```
Name: Ngotwane Dam, FSC: 19.033000000000001, Percentage: 4.6
insert counts:0
Name: Hartbeespoort Dam, FSC: 186.44, Percentage: 96.5
insert counts:1
Name: Bon Accord Dam, FSC: 4.381, Percentage: 103
insert counts:2
Name: Albasini Dam, FSC: 28.199000000000002, Percentage: 69.2
insert counts:3
Name: Blyderivierpoort Dam, FSC: 54.369, Percentage: 50
insert counts:4
```

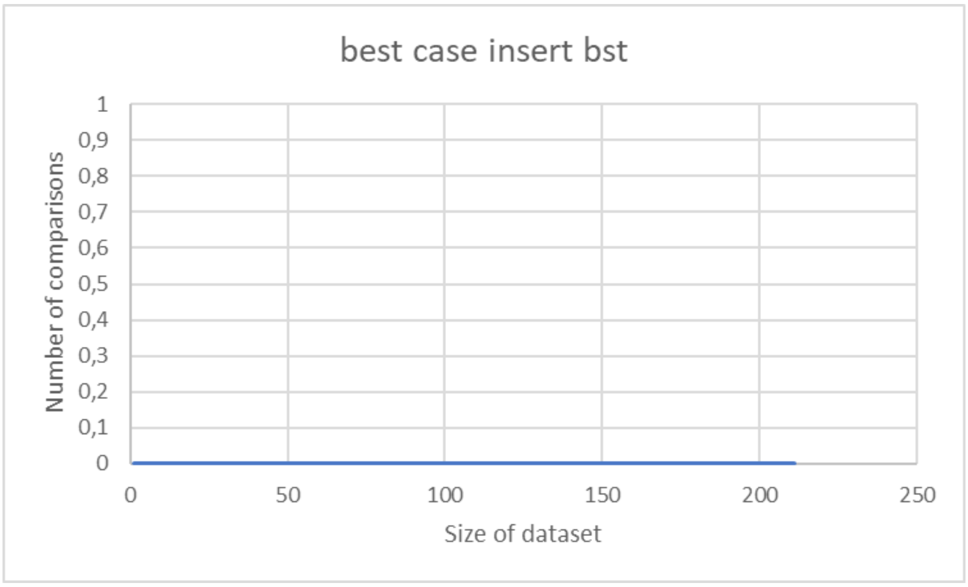
Last 10 lines:

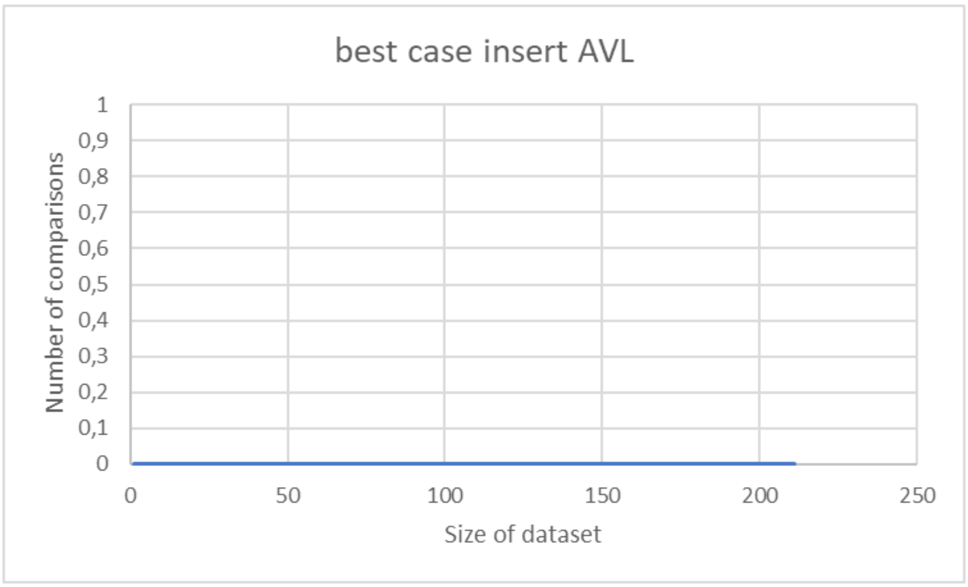
```
Name: Xonxa Dam, FSC: 115.86, Percentage: 100
insert counts:6
Name: Wriggleswade Dam, FSC: 91.471000000000004, Percentage: 98.7
insert counts:7
Name: Woodstock Dam, FSC: 373.25, Percentage: 77.099999999999994
insert counts:8
Name: Xilinxa Dam, FSC: 13.823, Percentage: 27.2
insert counts:8
Name: Zaaihoek Dam, FSC: 184.63, Percentage: 58.5
insert counts:7
```

Varying the size of the dataset:

When running the DamBSTApp and DamAVLApp for the value of dataset from 1 to 211, the following values were obtained for the insert comparisons

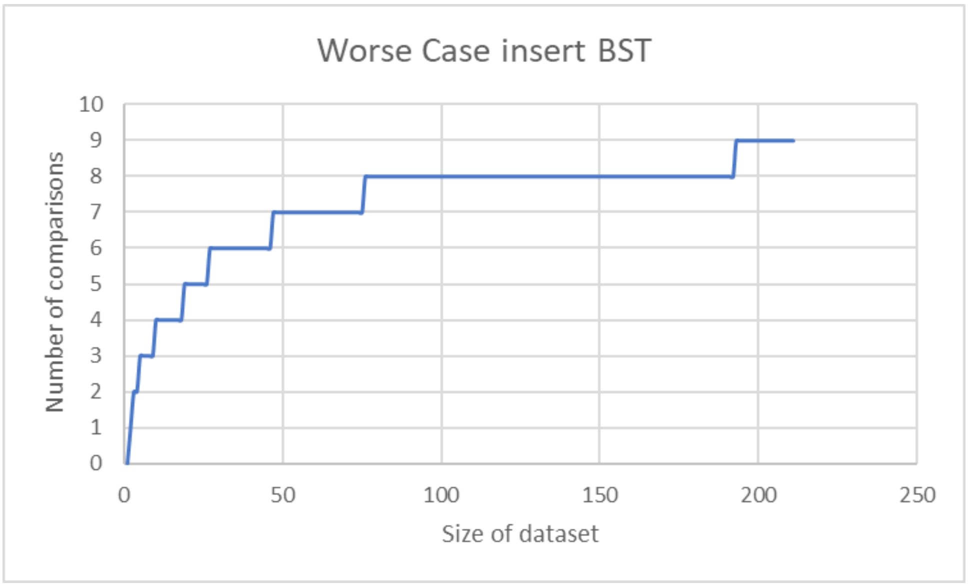
Best case:

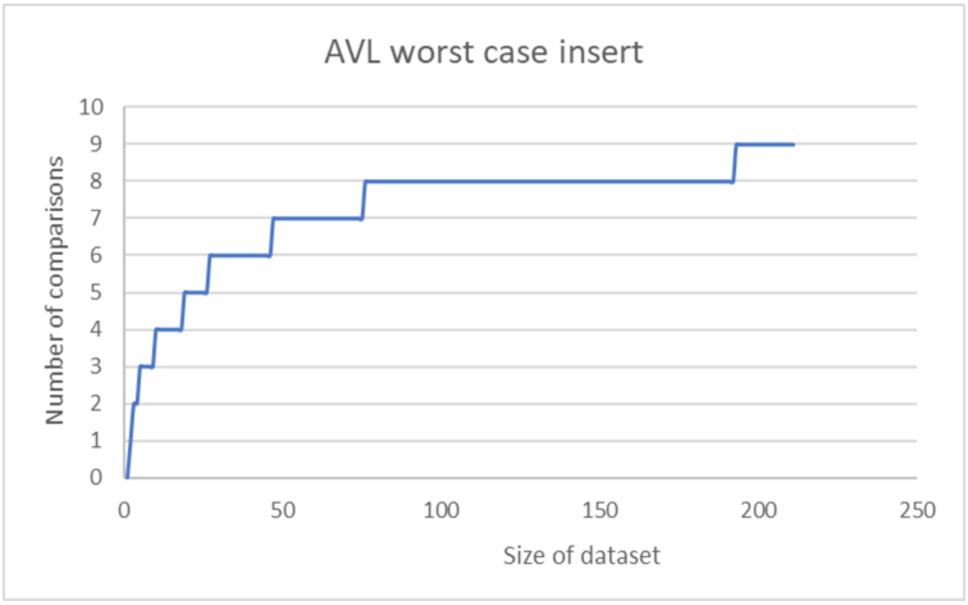




The best case occurs when the adding the first data in the data structures. During this no comparisons are made between the datatype of the node. The algorithmic complexity is O(1).

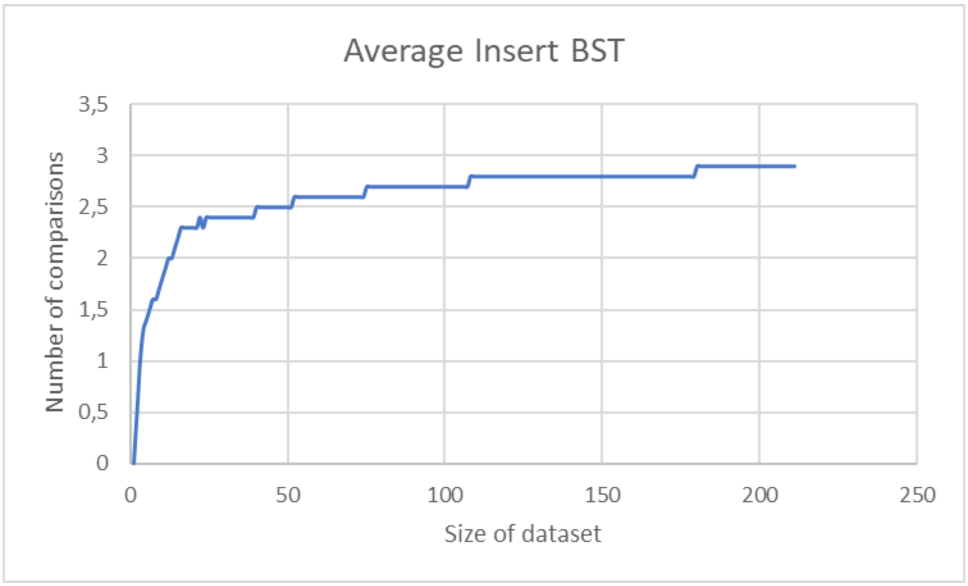
Worse Case:

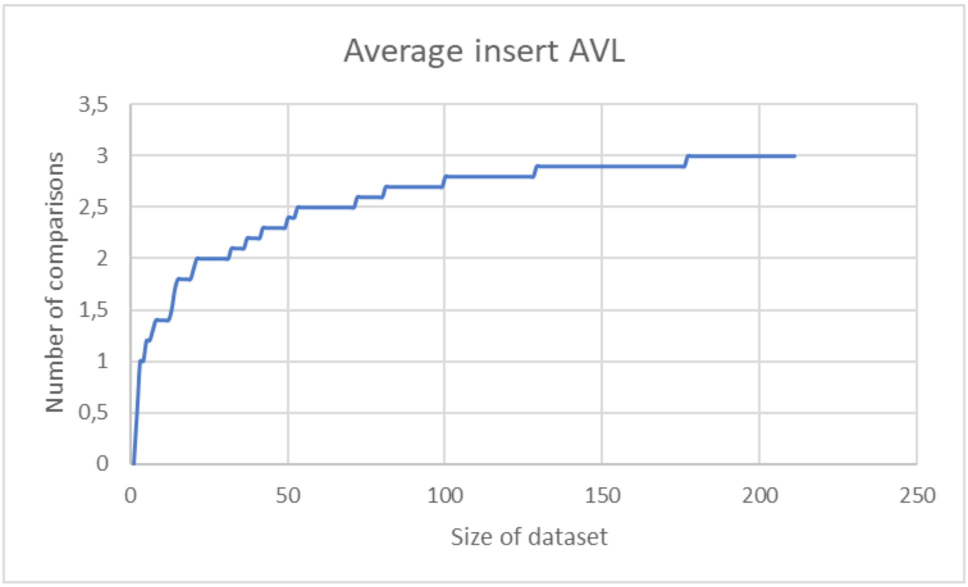




Here also we can see that for the worst case the BinarySearchTree and the AVLTree are almost similar. This is because they are fundamentally the same data structures. It can be noted that the worst case for both data structures are low as compared to an array.

Average case:

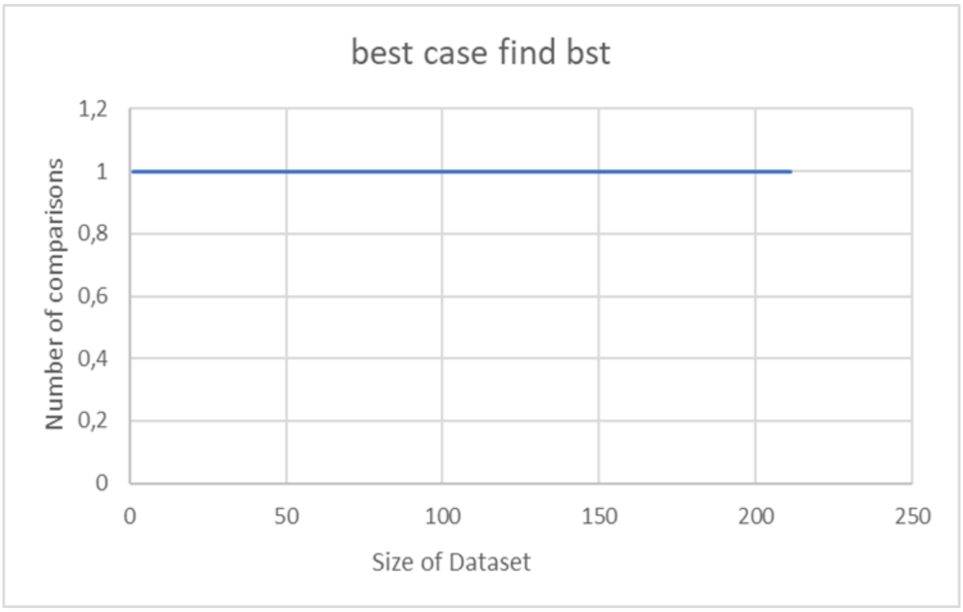


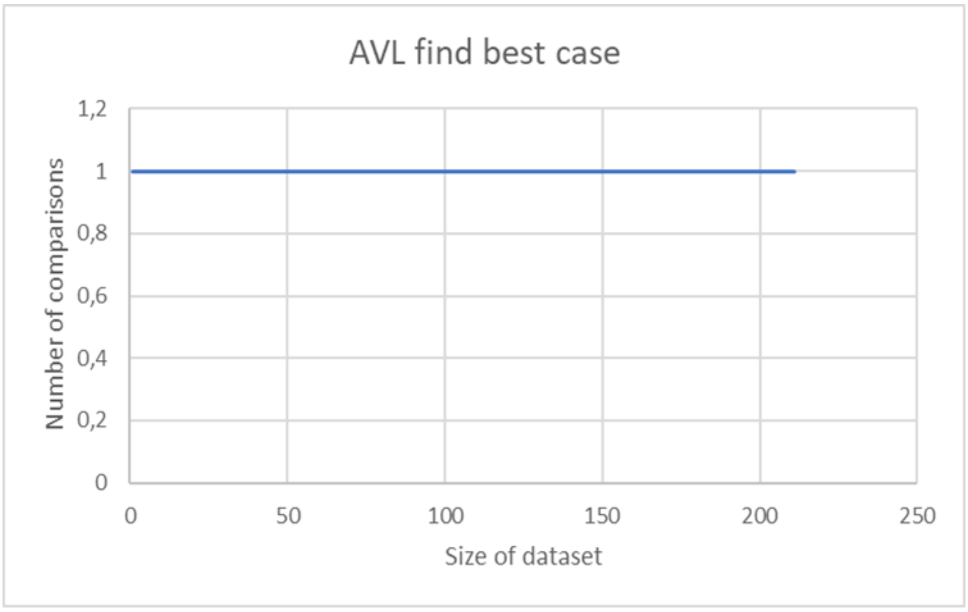


For the average case, both data structures perform nearly similarly. The algorithmic complexity for the average case is O(log n). This is why the number of comparisons are low.

When running the DamBSTApp and DamAVLApp for the value of dataset from 1 to 211, the following values were obtained for the search comparisons.

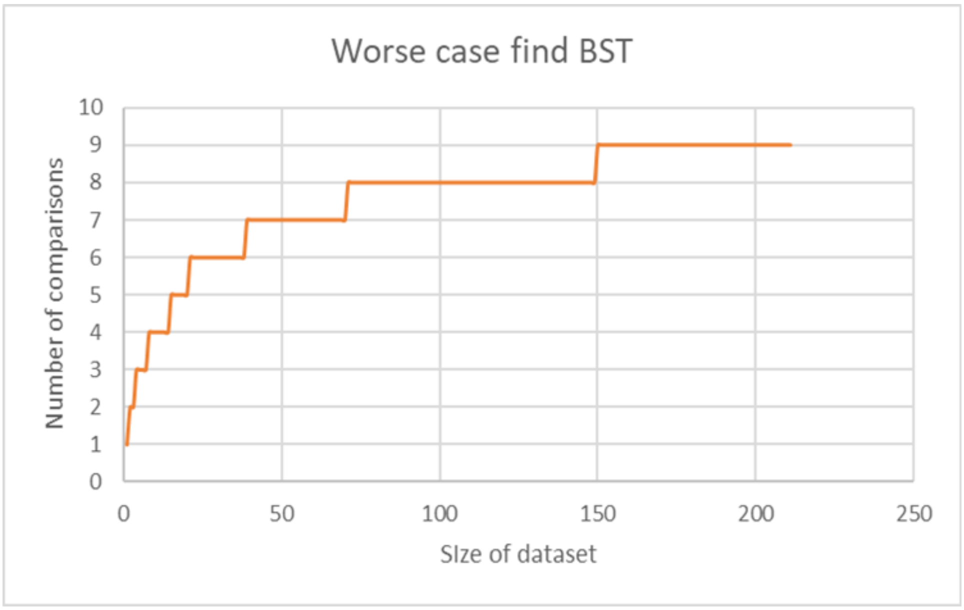
Best case:

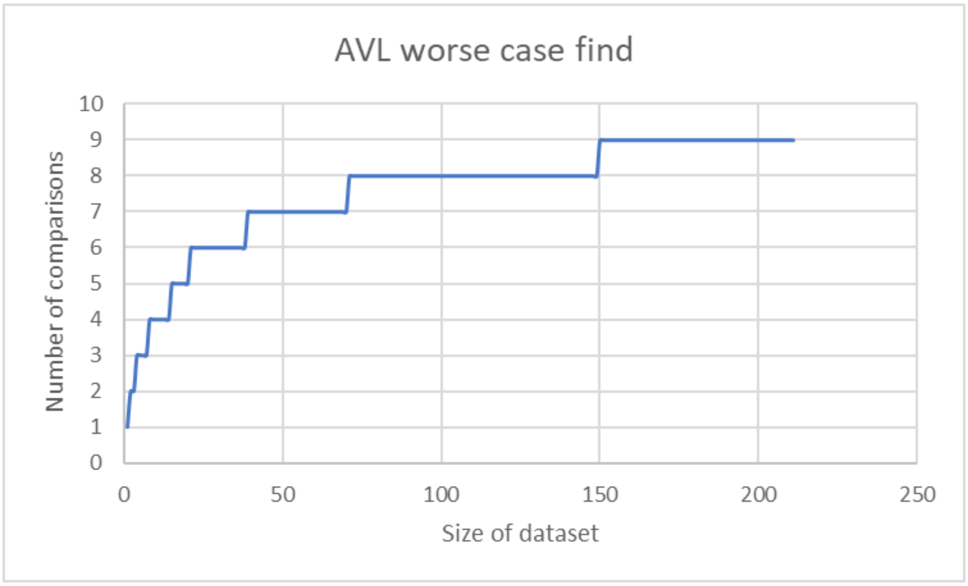




For the best case, both data structures perform similarly. This occurs when the data searched is in the root node, therefore only one comparison is made. The algorithmic efficiency is O(1).

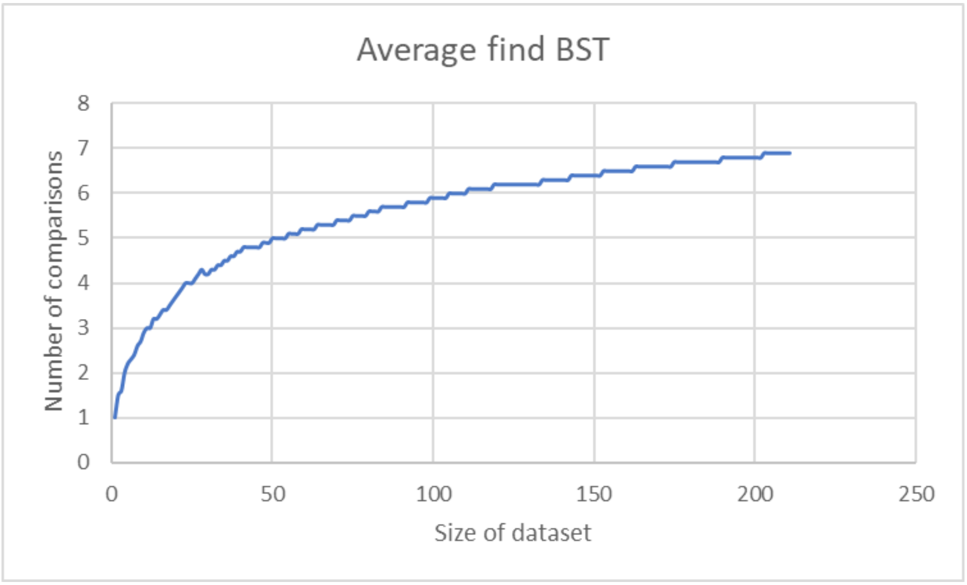
Worst case:

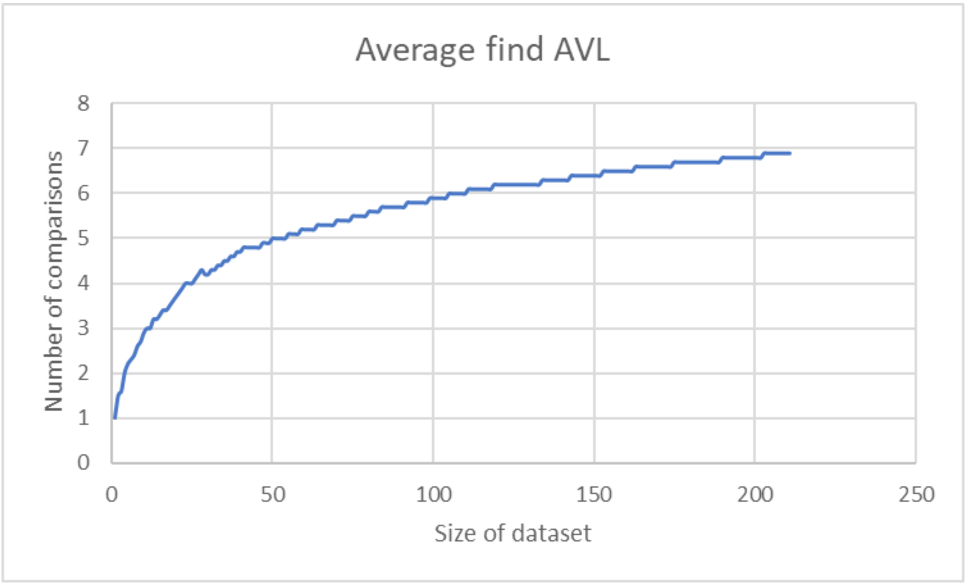




The worst case for both data structures are almost the same. This occurs when the data item searched is at the end of the tree, therefore the whole height of the tree has to be traversed.

Average case:





Here also, the two data structures perform almost the same. This is because the data structures are very similar. The average case is the total number of search comparisons/ size of dataset.

Part 6 (sorted file)

The csv file was sorted so that the Dam names would appear in alphabetical order.
The results were then analysed:

When running the sorted file with BSTrun with the full dataset, the results were as follows

Find counts:-

```
best:1worse:210avg:105.5781990521327
Insert counts:-
best:0worse:209avg:104.1611374407583
number of data:211
```

When running the sorted file with AVLrun with the full dataset, the results were as follows

```
Find counts:-
best:1worse:8avg:6.829383886255924
Insert counts:-
best:0worse:8avg:3.3412322274881516
number of data:211
```

Here we can cleary see the advantage that the AVLTree offers. While the worse cases for the BinarySearchTree is almost the size of the dataset(211) , the worse cases are still less than 10 for the AVLTree.
This is because the tree is balanced whenever there is a difference in height of more than 1 at any node. The balancing of the tree prevents a skewed structure as in the BinarySearchTree.

This linear structure means that the BinarySearchTree is essentially an array. This explains the average case of O(n/2) for the insert and search operations. The AVLTree meanwhile has an average case of O(log n).

Creativity:

Instead of using python or bash scripts, I created the BSTrun and AVLrun classes that print out the best/worse/average case for the insert and search counts for the range of dataset of 1 to 211. The data maximum, minumum and average values were calculated and printed as an array. It was then easy to plot the graphs.

Git log:

```
rahul@rahul-VirtualBox:~/files/Assignment2/AVLoriginal$ git log
commit 444c98669e959750a8eadccc34d8c272e611babe (HEAD -> master)
Author: Rahul Deeljore <Rahul_2496@hotmail.com>
Date:   Tue Mar 27 10:47:08 2018 +0200

    near final version

commit d9bc57ca2f0c1c6f53d8092fac6f84701bcb0e80
Author: Rahul Deeljore <Rahul_2496@hotmail.com>
Date:   Tue Mar 27 04:56:42 2018 +0200

    all javadoc added
```

```
commit 2e14998fc8b4c0cc1bc4a053655646c451af6189
Author: Rahul Deeljore <Rahul_2496@hotmail.com>
Date:   Tue Mar 27 04:54:59 2018 +0200

    javadoc added

commit 5f7189fb156d9fb8cb90d8c08828a1b3c5952501
Author: Rahul Deeljore <Rahul_2496@hotmail.com>
Date:   Mon Mar 19 16:36:22 2018 +0200

    part 1,2,3,4 done
```