CSC2001F – Assignment 2 Report
HMMDEN001
April 2021

I have created 3 programs for my Assignment 2, 2 of which are java and 1 is a python program.

I have created a StudentTxt.java program that will create an object of type student with traits of an ID and a Name (both of type Strings).
The program StudentTxt.java, is a class that creates student type objects (each with a name and student ID, of which both are string variables). This class has the usual relative accessor methods for both instance variables, and the constructor needed to create an instance of this class. There is also a toString method and a compareTo method in this program. The compareTo method is used in the AVL implementation in docs code, when we are searching for a specific key in the data structure.

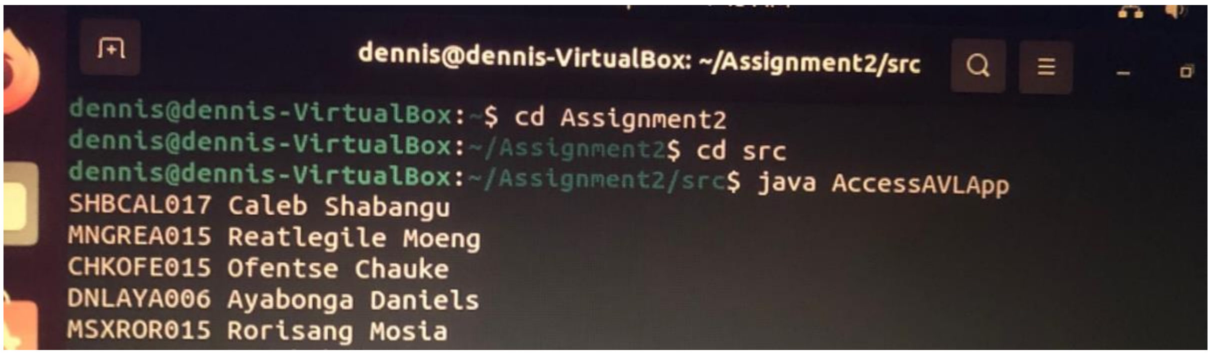
I have then created a program called AccessAVLApp that creates an array of type StudentTxt, reads in a txt file and stores the contents, line by line, as student objects within the AVL Binary Tree.
In the AccessAVLApp program I have also created two methods. First one of printStudent which takes in a string ID as a key to search for the relevant StudentTxt object within the array – and returns the name associated with the StudentTxt. Secondly, printAllStudents which makes use of the inOrder method to output the entire array to the screen.
This is the program that's used from the terminal to pass in commands such as 'java AccessAVLApp' and 'java AccessAVLApp "<someID>" ' that will use the respective printAllStudents and printStudent methods within the program.

The third program I've made is a python script that can be manipulated and used for the experimentation part of Assignment2. It helps automate the experimentation process. Automatically obtaining best, worst and average cases for the number of key comparisons within the AVL Binary Tree implementation and redirecting that output to a result.txt file in the src directory. Keeping in mind the students relative ID's are used as keys to search with. Simply have to write the pathways of the subsets into it, and redirect the results accordingly. It also includes print statements, portraying where the script is exactly within the automation process. I elaborate on the structure of the python script in my statement on page 6, hoping to score a few creativity points for this assignment2.

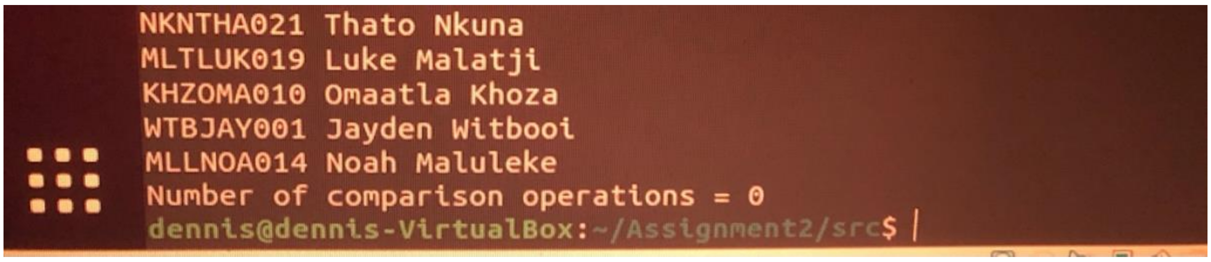
I have sourced code from Professor Suleman for the creation of an AVL Binary Tree for the purposes of this assignment, and I've referenced this in my javadocs and given him the relevant credit by showing that he was the author of the code.

Trial Tests of AccessAVLApp (Part1)

Input : java AccessAVLApp

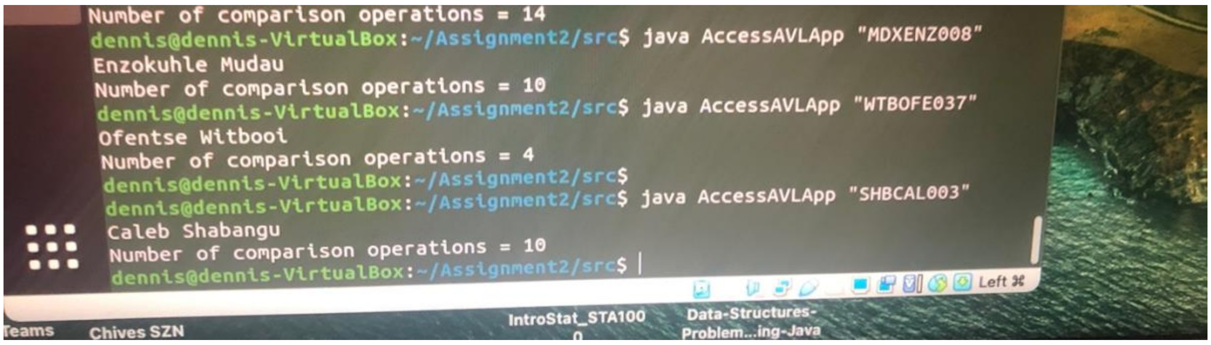


```
dennis@dennis-VirtualBox: ~/Assignment2/src
dennis@dennis-VirtualBox:~$ cd Assignment2
dennis@dennis-VirtualBox:~/Assignment2$ cd src
dennis@dennis-VirtualBox:~/Assignment2/src$ java AccessAVLApp
SHBCAL017 Caleb Shabangu
MNGREA015 Reatlegile Moeng
CHKOFE015 Ofentse Chauke
DNLAYA006 Ayabonga Daniels
MSXROR015 Rorisang Mosia
```



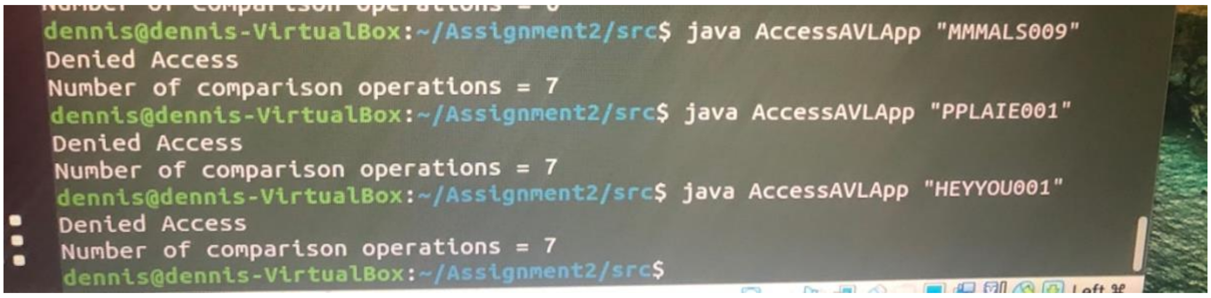
```
NKNTHA021 Thato Nkuna
MLTLUK019 Luke Malatji
KHZOMA010 Omaatla Khoza
WTBJAY001 Jayden Witbooi
MLLNOA014 Noah Maluleke
Number of comparison operations = 0
dennis@dennis-VirtualBox:~/Assignment2/src$ |
```

Testing parameters that are valid:
Input : java AccessAVLApp "<someValidKey>"



```
Number of comparison operations = 14
dennis@dennis-VirtualBox:~/Assignment2/src$ java AccessAVLApp "MDXENZ008"
Enzokuhle Mudau
Number of comparison operations = 10
dennis@dennis-VirtualBox:~/Assignment2/src$ java AccessAVLApp "WTBOFE037"
Ofentse Witbooi
Number of comparison operations = 4
dennis@dennis-VirtualBox:~/Assignment2/src$
dennis@dennis-VirtualBox:~/Assignment2/src$ java AccessAVLApp "SHBCAL003"
Caleb Shabangu
Number of comparison operations = 10
dennis@dennis-VirtualBox:~/Assignment2/src$ |
```

Testing parameters that are invalid:
Input : java AccessAVLApp "<someInvalidKey>"



```
dennis@dennis-VirtualBox:~/Assignment2/src$ java AccessAVLApp "MMMALS009"
Denied Access
Number of comparison operations = 7
dennis@dennis-VirtualBox:~/Assignment2/src$ java AccessAVLApp "PPLAIE001"
Denied Access
Number of comparison operations = 7
dennis@dennis-VirtualBox:~/Assignment2/src$ java AccessAVLApp "HEYYOU001"
Denied Access
Number of comparison operations = 7
dennis@dennis-VirtualBox:~/Assignment2/src$
```

Experimentation (Part2)

In this experiment, I intend to showcase the various measures of the number of comparisons between keys in best/worst/average case scenarios for varying data set sizes, in the AVL data structure implementation.
These comparisons of the AVL tree are faster than the BST tree from assignment 1.
The juxtaposition of the comparisons of the find and insert comparison operations will be portrayed, drawing differences and similarities between the two. This will be done via **instrumentation** of two variables that will be used to track the relevant comparisons for find and insert. These variables are redirected during the experiment process to a txt file, and we use that as the data to construct the tables and graphs seen below.

The comparison values obtained through the automation of the experiment through a python file I've created are what I use to plot my graph, of subset size on the x axis, against number of comparisons on the y axis. There's more explanation on the python file and exactly how I've automated my experiments below, in my statement stating what I've done that constitutes for creativity.

I've constructed the script in such a way that the minimum, maximum and average operation counters are retrieved and stored in a txt file (using output redirection), these values are used for the following tables and graphs respectively.

Find comparisons

The Cases for the find operation counters should be as follows

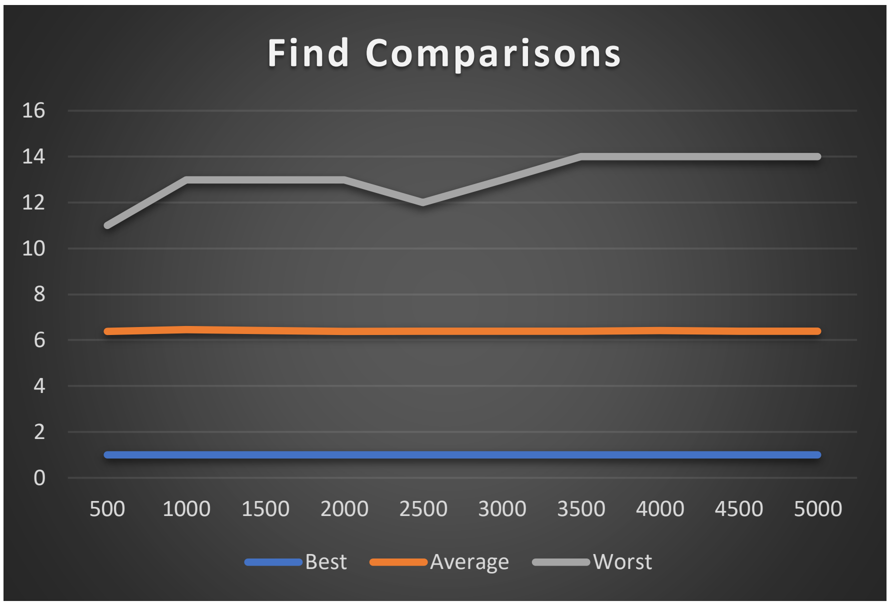
- Best case : O(1)
    - ➔ as if the first node we are checking to see if it has the key we are looking for, could be the key we are looking for.
- Average case : O(log2(N)), log to the base of 2 &where N = number of nodes of the AVL tree
- Worst case : O(log2(N)), log to the base of 2 &where N = number of nodes of the AVL tree
    - ➔ If we have to search through the entire tree to find the specific key we are looking for.

These are only expected values and may be subject to change depending on how the AVL tree ends up being structured.

Subset Sizes	Best Case	Average Case	Worst Case
500	1	6.422	11
1000	1	6.459	13
1500	1	6.406	13
2000	1	6.381	13

2500	1	6.3868	12
3000	1	6.3893	13
3500	1	6.3992	14
4000	1	6.4092	14
4500	1	6.3982	14
5000	1	6.3908	14

Number of operations counted



Different Subset Sizes

As seen above with the results, the average number of comparisons don’t change much and the number of worst comparisons range between 11 and 14. The best case for the number of comparisons made during the search for the element in the AVL Tree using the key, is always one.

This is way faster than using a BinarySearchTree (From Assignment1), for search operations.

<u>Insert comparisons</u>

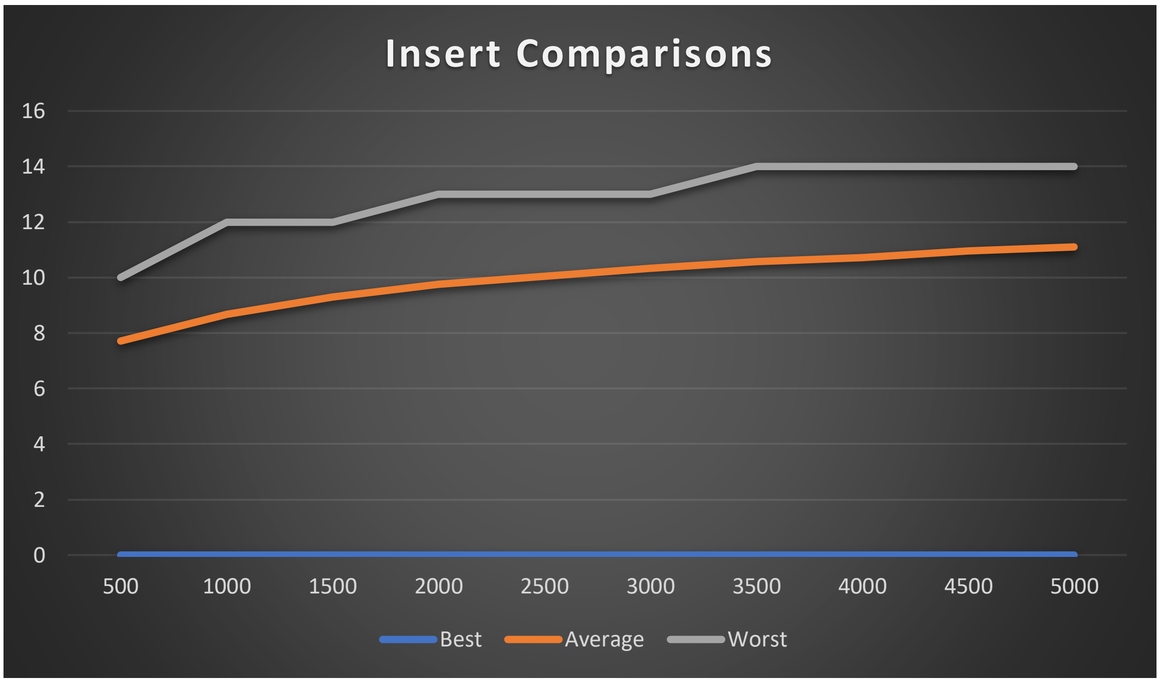
The Cases for the Insert operation counters should be as follows

- Best case: O(0)
  - ➔ as if the AVL tree is empty, would simply insert the node at root, thus no comparisons done
- Average case : O(log2(N)), log to the base of 2 &where N = number of nodes of the AVL tree
- Worst case: O(log2(N)), log to the base of 2 &where N = number of nodes of the AVL tree

➔ As we may have to insert all the way to the height of the tree, with N being the height

These are only expected values and may be subject to change depending on how the AVL tree ends up being structured.

Subset Sizes	Best Case	Average Case	Worst Case
500	0	7.71	10
1000	0	8.68	12
1500	0	9.30	12
2000	0	9.75	13
2500	0	10.05	13
3000	0	10.32	13
3500	0	10.58	14
4000	0	10.72	14
4500	0	10.96	14
5000	0	11.10	14



Number of operations counted

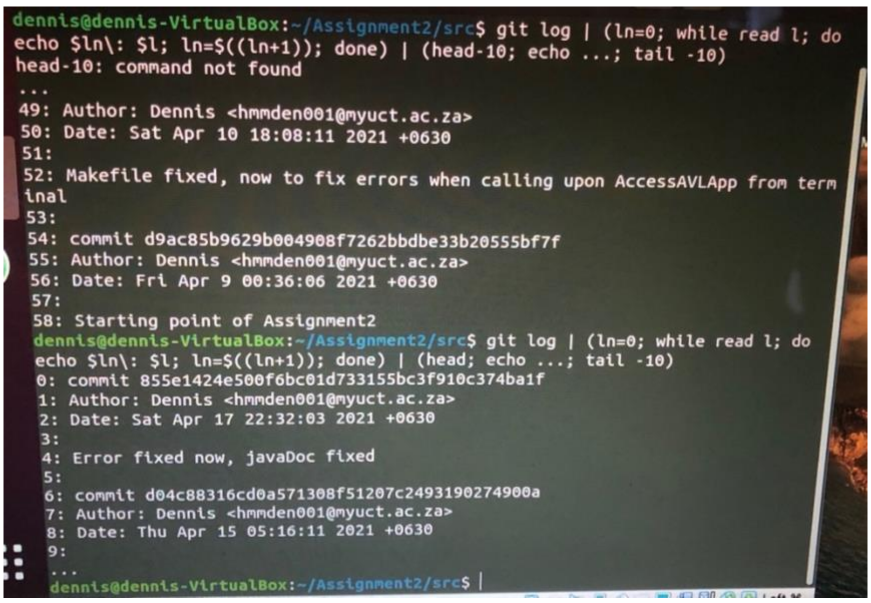
Different Subset Sizes

As seen above with the results, the average number of comparisons gradually increase as the subset size gets larger. The number of worst comparisons range between 10 and 14. The best case for the number of comparisons made during the insertion for each element in the AVL Tree using the key, is always zero (ie inserting the first element into the tree at the root).

In conclusion, as seen from the results of my experiment (tables and graphs seen above), these results for the find and insert operations relatively are much less than those from the Binary Search Tree implementation.
With find operations averaging around 6-7 operations counted, and average insert operations ranging between 7 and 11 operation counted.

What constitutes for creativity within my Assignment 2.
I have automated experiments needed for part 2 of Assignment2 and to retrieve the counter values for the best, average and worst case scenarios. This has been done with a python script. This script can be manipulated and used for the automation of the experiment part of the Assignment 2. It automates the randomization of subsets being created (different sizes as specified in the Assignment outline), automates the searching of each student object, using their unique ID as a key, and finally it gets the min, max and average values for the comparison counters and redirects the output to a txt file.
I have also created a method in the AVLTree code from the Professor where it takes in an integer value (ie opCount variable), and resets this integer value to zero. This is done so that we don't add comparisons for insert from inserting the first element to inserting the second element into the AVL Tree and etc. This method is called on, passing through the Operation Counter for Insert into it, after it's printed to the screen.

Snapshot of my git log, first ten and last ten lines :



```
dennis@dennis-VirtualBox:~/Assignment2/src$ git log | (ln=0; while read l; do
echo $ln\: $l; ln=$((ln+1)); done) | (head-10; echo ...; tail -10)
head-10: command not found
...
49: Author: Dennis <hmmden001@myuct.ac.za>
50: Date: Sat Apr 10 18:08:11 2021 +0630
51:
52: Makefile fixed, now to fix errors when calling upon AccessAVLApp from term
inal
53:
54: commit d9ac85b9629b004908f7262bbdbe33b20555bf7f
55: Author: Dennis <hmmden001@myuct.ac.za>
56: Date: Fri Apr 9 00:36:06 2021 +0630
57:
58: Starting point of Assignment2
dennis@dennis-VirtualBox:~/Assignment2/src$ git log | (ln=0; while read l; do
echo $ln\: $l; ln=$((ln+1)); done) | (head; echo ...; tail -10)
0: commit 855e1424e500f6bc01d733155bc3f910c374ba1f
1: Author: Dennis <hmmden001@myuct.ac.za>
2: Date: Sat Apr 17 22:32:03 2021 +0630
3:
4: Error fixed now, javaDoc fixed
5:
6: commit d04c88316cd0a571308f51207c2493190274900a
7: Author: Dennis <hmmden001@myuct.ac.za>
8: Date: Thu Apr 15 05:16:11 2021 +0630
9:
...
dennis@dennis-VirtualBox:~/Assignment2/src$ |
```