CSC20001F: ASSIGNMENT1

Prepared by:

Name: Lindani khuzwayo

Student Number: KHZLIN012

Due: 07 April 2021

1.Introduction

This report focuses on the uses, implementation and comparison of the efficiency of two data structures. A broad analysis of the two data structures namely, Arrays and Binary Search Trees is carried out experimentally using real-world data. The goal of this experiment is to compare the Binary Search Tree data structure with a traditional unsorted array data structure, both implemented in java with a real-world application to check if a student is on a pre-approved list for access to campus during lockdown

In addition, not only data structures are observed but the completion of the experiment was possible through the aid of object orientated programming and text files because of their simplicity, as they are commonly used for storing external information or data (inputs). A continues creation, reading and writing of text files is observed to help store critical data for the experiment.

A data structure can be defined as a labelled location in memory that can be used to store and organise any type of data in that particular space, an algorithm is a bunch of lines (code) or steps to solve a problem in a sequential manner. The combination of the two concepts allows us to efficiently write and optimize our programs, the above is observed in this experiment.

2.Desgin method: OOP and data structure

The approach of the experiment is to use classes and objects in Java which are the fundamental components of OOP's. Classes are a blueprint or set of lines of instructions to build a specific type of object, they define the behaviour of objects thus their contents.

Three set of classes were created for the purpose of the experiment namely AccessArrayApp.Class, AccessBSTApp.Class and Students.Class . Figure 1 below is a physical repersantation of the Student class.

Students
Name ID LastName

getStudentID()
getStudentName()
getSurname()
toString()

Figure1: Student class

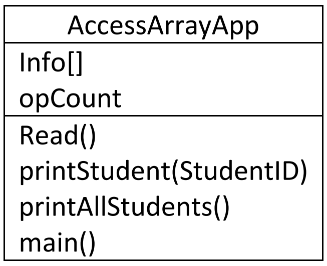


Figure2 : AccessArrayApp class

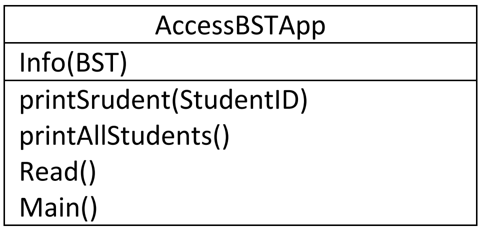


Figure3: AccessBSTApp class

Students implements the comparable interface for the purpose of overriding the comparaTo() menthod. Figure1 above is a class, a prototype for a Student it shows the Name, student ID and the student's surname or last name. From these descriptions we can construct a student and the student becomes the object. Thus, in thus students 5000 objects of students are constructed. And stored in an Array inside AccessArrayApp thus creating an array of students

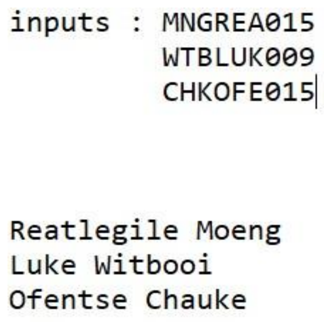
Figure 2 above is a class that makes use of Student class to create or construct objects of students i.e. giving name, id and surname for each object. The data for students is store in a text file oklist.txt thus AccessArray and reads the file and simultaneously construct object of Students while adding the object to the array Info[] of datatype Students. That is done by calling the read() method. printStudent(StudentID) is a method that receives a parameter of string from a user and goes through info[] to find the student if the student is not on info, Access is denied for that particular student . While printAllStudents() is a method used to print the content inside the array info[]. The main() method is main method of the class which also takes parameters of datatype string and stores then on a array namely args[]. Thus it is where all the other methods are called.

Figure 3 above is also a class that makes use of the Student class to create student objects of students i.e. giving name, id and surname for each object. The data for students is store in a text file oklist.txt thus AccessBSTApp reads the file and simultaneously construct object of Students while adding the objects to the a binary search tree Info(BST) of datatype Students. That is done by calling the read() method. printStudent(StudentID) is a method that receives a parameter of string from a user and goes through info(BST) to find the student if the student is not on info, Access is denied for that particular student . While printAllStudents() is a method used to print the content inside the binary search tree info(). The main() method is main method of the class which also takes parameters of datatype string and stores then on a array namely args[]. Thus it is where all the other methods are called.

As previously mentioned the goal of the experiment is to compare the binary search tree with a traditional unsorted array data structure, both implemented in Java, using a real-world application to check if a student is on a pre-approved list for access to campus during the lockdown. Oklist.txt if the preapproved list

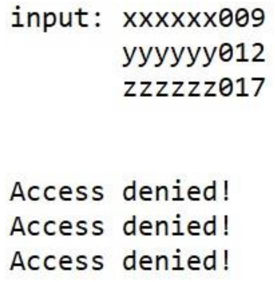
3.Testing and collection of data

Both AccesArrayApp and AccessBSTApp were tested with a set of 3 known parameters that work, another set od 3 invalid parameters lastly without any parameters the following figures were observed outputs



```
inputs : MNGREA015
         WTBLUK009
         CHKOFE015


Reatlegile Moeng
Luke Witbooi
Ofentse Chauke
```

Figure 4: set of 3 known inputs for AccessArrayApp



```
input: xxxxxx009
       yyyyyy012
       zzzzzz017


Access denied!
Access denied!
Access denied!
```

Figure 5: set of 3 known inputs for AccessArrayApp

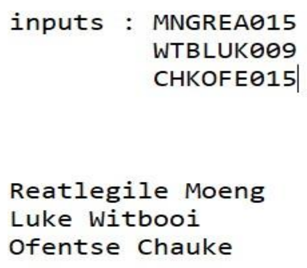

```
inputs : MNGREA015
         WTBLUK009
         CHKOFE015


Reatlegile Moeng
Luke Witbooi
Ofentse Chauke
```

Figure 6: set of 3 known inputs for AccessBSTApp

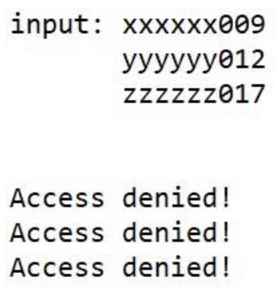

```
input: xxxxxx009
       yyyyyy012
       zzzzzz017


Access denied!
Access denied!
Access denied!
```

Figure 7: set of 3 known inputs for AccessBSTApp

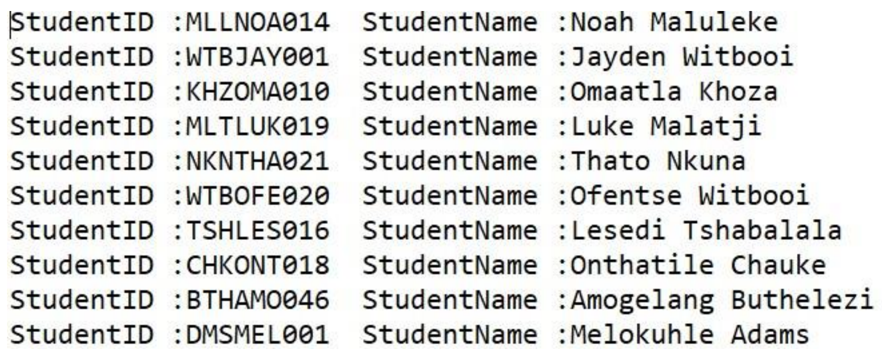
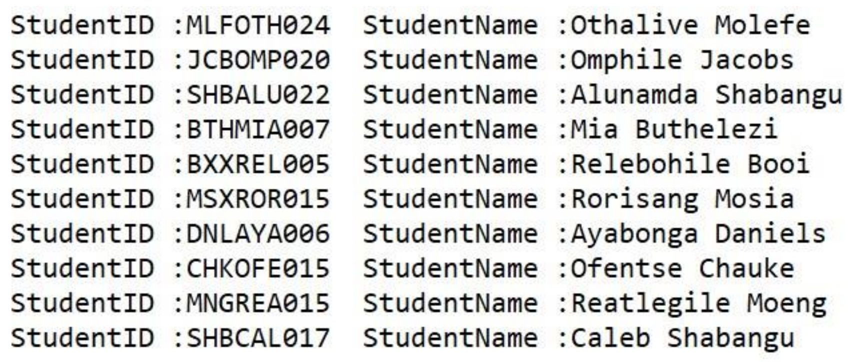
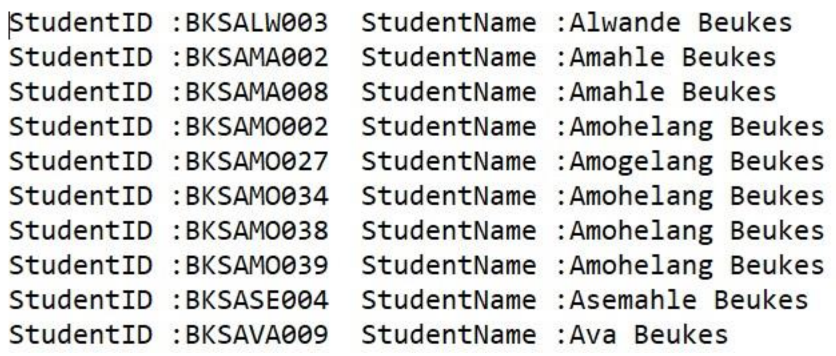

```
StudentID :MLLNOA014  StudentName :Noah Maluleke
StudentID :WTBJAY001  StudentName :Jayden Witbooi
StudentID :KHZOMA010  StudentName :Omaatla Khoza
StudentID :MLTLUK019  StudentName :Luke Malatji
StudentID :NKNTHA021  StudentName :Thato Nkuna
StudentID :WTBOFE020  StudentName :Ofentse Witbooi
StudentID :TSHLES016  StudentName :Lesedi Tshabalala
StudentID :CHKONT018  StudentName :Onthatile Chauke
StudentID :BTHAMO046  StudentName :Amogelang Buthelezi
StudentID :DMSMEL001  StudentName :Melokuhle Adams
```

Figure 8: first 10 lines without parameters AccessArrayApp



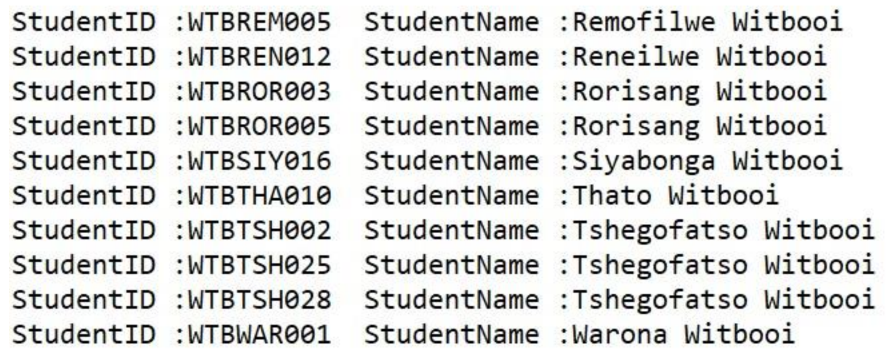
```
StudentID :MLFOTH024  StudentName :Othalive Molefe
StudentID :JCBOMP020  StudentName :Omphile Jacobs
StudentID :SHBALU022  StudentName :Alunamda Shabangu
StudentID :BTHMIA007  StudentName :Mia Buthelezi
StudentID :BXXREL005  StudentName :Relebohile Booi
StudentID :MSXROR015  StudentName :Rorisang Mosia
StudentID :DNLAYA006  StudentName :Ayabonga Daniels
StudentID :CHKOFE015  StudentName :Ofentse Chauke
StudentID :MNGREA015  StudentName :Reatlegile Moeng
StudentID :SHBCAL017  StudentName :Caleb Shabangu
```

Figure 9: last 10 lines without parameters AccessArrayApp



```
StudentID :BKSALW003  StudentName :Alwande Beukes
StudentID :BKSAMA002  StudentName :Amahle Beukes
StudentID :BKSAMA008  StudentName :Amahle Beukes
StudentID :BKSAMO002  StudentName :Amohelang Beukes
StudentID :BKSAMO027  StudentName :Amogelang Beukes
StudentID :BKSAMO034  StudentName :Amohelang Beukes
StudentID :BKSAMO038  StudentName :Amohelang Beukes
StudentID :BKSAMO039  StudentName :Amohelang Beukes
StudentID :BKSASE004  StudentName :Asemahle Beukes
StudentID :BKSAVA009  StudentName :Ava Beukes
```

Figure 10: first 10 lines without parameters AccessBSTApp



```
StudentID :WTBREM005  StudentName :Remofilwe Witbooi
StudentID :WTBREN012  StudentName :Reneilwe Witbooi
StudentID :WTBROR003  StudentName :Rorisang Witbooi
StudentID :WTBROR005  StudentName :Rorisang Witbooi
StudentID :WTBSIY016  StudentName :Siyabonga Witbooi
StudentID :WTBTHA010  StudentName :Thato Witbooi
StudentID :WTBTSH002  StudentName :Tshegofatso Witbooi
StudentID :WTBTSH025  StudentName :Tshegofatso Witbooi
StudentID :WTBTSH028  StudentName :Tshegofatso Witbooi
StudentID :WTBWAR001  StudentName :Warona Witbooi
```

Figure 11: last 10 lines without parameters AccessBSTApp

More tests were ran, now for the comparison part a counter is used to track any comparison of keys on the classes mentioned above thus the operation counter is useful when comparing the number of comparisons for the two different data structures. The following is observed

filename	AccessArrayApp(opCount)	AccessBSTApp(opCount)	n
file1.txt	1	1	500
file2.txt	1	1	1000
file3.txt	1	1	1500
file4.txt	1	1	2000
file5.txt	1	1	2500
file6.txt	1	1	3000
file7.txt	1	1	3500
file8.txt	1	1	4000
file9.txt	1	1	4500
file10.txt	1	1	5000

Table1: Best case (n-number if lines in text file)

filename	AccessArrayApp	AccessBSTApp	n
file1.txt	500	11	500
file2.txt	1000	16	1000
file3.txt	1500	28	1500
file4.txt	2000	30	2000
file5.txt	2500	14	2500
file6.txt	3000	9	3000
file7.txt	3500	23	3500
file8.txt	4000	36	4000
file9.txt	4500	19	4500
file10.txt	5000	17	5000

Table2: Worst case (n-number if lines in text file)

4.Disscusion of data and conclusion

From the above data it is clear that the AccessArrayApp and the AccessBSTApp are running with no error in the implementation of the this is due to the approach discussed above with is more of a creative and ethical ways of programming, thus the approach is also more of modular programming which is a creative programming concept. It involves separating a program's functions into independent pieces or building blocks, each containing all the parts needed to execute a single aspect of the functionality.

Thus from the two tables above it is observed that accessing data in a traditional unsorttd array is timely and expensive, the number of comparisons made is way to high compare to that of a binary search tree. This make binary search trees the best data structure for storing and retrieving data from, thus the number off operations done is to little compared to that of arrays.