Comparing Efficiency of Insert and Find Operations on a Binary Search Tree and an AVL Tree

Lindelani Mbatha - MBTLIN007
mbtlin007@myuct.ac.za

abstract

Data structures are efficient mechanisms to store and provide access to data. There are various types of data structures which differ in efficiencies of running certain operations such as insert, delete and search. This report concerns about an experiment to compare a Binary Search tree with an AVL Tree, specifically testing insert and search operations. Analysis is presented using big O analysis

1. Introduction

The goal of the experiment was to compare efficiencies of an AVL Tree and a Binary Search Tree (both implemented in Java) in conducting find operations. The experiment was then conducted through OOP, tests and analysis.

2. OOP and data structure design

Two java application to conduct the experiment were made. Application development includes the java application LSAVLApp was to test the AVL Tree data structure, and the LSBSApp was developed to test the BinarySearchTree data structure. To allow applications functionality an OOP aproach was taken which includes, an Entry class of which each line of data is an object, and a helper class to handle all file and data structure operations for both data structures. To achieve this, abstraction is used in the BinaryTree class, then both BinarySearchTree and AVLTree extends it and override some of its methods used in the app. Figure 1 below shows the OOP design of the applications.

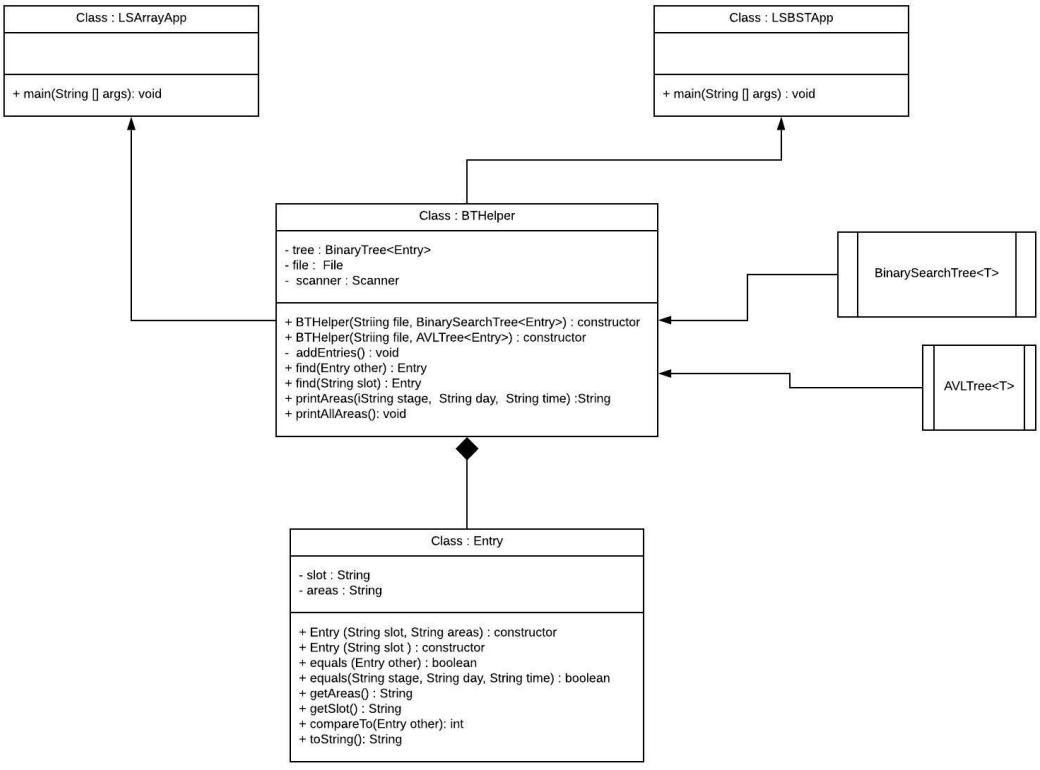

Figure 1 : UML Diagram illustrating OOP design of the applications

3. Experiment description

The experiment was conducted on the applications through instrumentation and counting comparison steps for each operation : insert, and search. For each operation in each data structure, a counter is initialized and is incremented every time there is a comparison between data objects (i.e Entry objects, which represents each data entry in the Load Shedding data). The counters are stored as instance variables and can be queried through get methods, and are outputted to stream or logged to a file through unix output redirection.

Test cases for each data structure application were developed. To ensure robustness, each application is tested on a set of 3 entries known to exist on the dataset, and then 3 invalid entries which are combinations of out of range stage or, day or, time.

The applications were then tested through randomized subsets with size ranging from 297 to 2796 with a step of 296. Each dataset was used as input data to both applications. The applications were tested to find every entry in each subset, redirecting output to a log file hence the operation count for inserting and finding to and from the data structure respectively. The operation counters for each data set were then stored continuously in a pandas data frame, where the worst, best, and average cases are determined through maximum, minimum, and average values respectively. These results from all subsets are then analysed by plotting them against subset size.

4. Experiment Results

4.1 Trial test on LSAVLApp

Test cases were developed to test key operations of the AVL Tree application. The results demonstrate querying for one entry in the application, and printing all entries.

4.1.1 Testing printAreas() method

	value	App command	Unix command	Output
Valid values	2_5_08	make runA stage=2 day=5 time=08	java -cp bin LSAVLTApp 2 5 08	The areas are: 6, 14 insert find 13 9
	5_13_22	make runA stage=5 day=13 time=22	java -cp bin LSAVLTApp 5 13 22	The areas are: 15, 7, 11, 3, 12 insert find 13 12
	8_25_18	make runA stage=8 day=25 time=18	java -cp bin LSAVLTApp 8 25 18	The areas are: 12, 4, 8, 16, 13, 5, 9, 1 insert find 13 13
Invalid Values	9_6_08	make runA stage=9 day=6 time=0	java -cp bin LSAVLTApp 9 6 08	No Areas Found
	5_32_20	make runA stage=5 day=32 time=20	java -cp bin LSAVLTApp 5 32 20	No Areas Found
	4_19_24	make runA stage=4 day=19	java -cp bin	No Areas Found

		time=24	LSAVLTApp 4 19 24	

4.1.2 Testing printAllAreas

1st 10 lines	Last 10 lines
Slot 1_10_00 Areas : 15 Slot 1_10_02 Areas : 16 Slot 1_10_04 Areas : 1 Slot 1_10_06 Areas : 2 Slot 1_10_08 Areas : 3	Slot 8_9_14 Areas : 10, 2, 6, 14, 11, 3, 7, 15 Slot 8_9_16 Areas : 11, 3, 7, 15, 12, 4, 8, 16 Slot 8_9_18 Areas : 12, 4, 8, 16, 13, 5, 9, 1 Slot 8_9_20 Areas : 13, 5, 9, 1, 14, 6, 10, 2 Slot 8_9_22 Areas : 14, 6, 10, 2, 15, 7, 11, 3

4.2 Trial Tests on LSBSTApp

Test cases were developed to test key operations of the Binary Search Tree application. The results demonstrate querying for one entry in the application, and printing all entries.

4.2.1 Testing printAreas()

	value	App command	Unix command	Output
Valid values	2_5_08	make runB stage=2 day=5 time=08	java -cp bin LSBSTApp 2 5 08	The areas are: 6, 14 insert find 192 9
	5_13_22	make runB stage=5 day=13 time=22	java -cp bin LSBSTApp 5 13 22	The areas are: 15, 7, 11, 3, 12 insert find 192 12
	8_25_18	make runB stage=8 day=25 time=18	java -cp bin LSBSTApp 8 25 18	The areas are: 12, 4, 8, 16, 13, 5, 9, 1 insert find 192 13
Invalid Values	9_6_08	make runB stage=9 day=6 time=0	java -cp bin LSBSTApp 9 6 08	No Areas Found
	5_32_20	make runB stage=5 day=32 time=20	java -cp bin LSBSTApp 5 32 20	No Areas Found
	4_19_24	make runB stage=4 day=19 time=24	java -cp bin LSBSTApp 4 19 24	No Areas Found

4.2.2 Testing printAllAreas

1st 10 lines	Last 10 lines

Slot 1_10_00 Areas : 15 Slot 1_10_02 Areas : 16 Slot 1_10_04 Areas : 1 Slot 1_10_06 Areas : 2 Slot 1_10_08 Areas : 3	Slot 8_9_14 Areas : 10, 2, 6, 14, 11, 3, 7, 15 Slot 8_9_16 Areas : 11, 3, 7, 15, 12, 4, 8, 16 Slot 8_9_18 Areas : 12, 4, 8, 16, 13, 5, 9, 1 Slot 8_9_20 Areas : 13, 5, 9, 1, 14, 6, 10, 2 Slot 8_9_22 Areas : 14, 6, 10, 2, 15, 7, 11, 3

4.3 Combined Experiment Results

After the test was automated through a python script which generated the results into a csv file for each data structure. Results Tables for both data structures can be found the logs directory.

Using the auto-generated tables, big O analysis of the results was plotted and is shown in the figures below.

Insert Operations

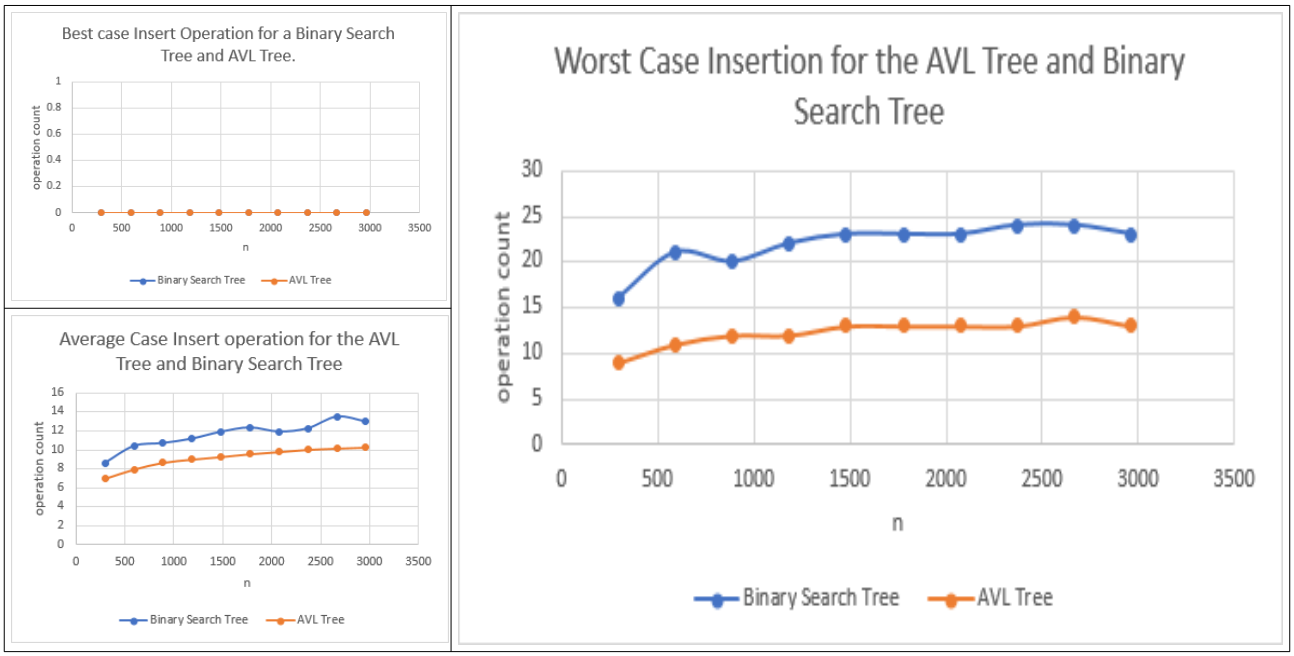

Figure 2 : Comparison of the two data structures for conducting Insert Operations.

Notice that the best case scenario appears to display results for one data structure, due to that the results for both data structures are exactly the same, and so lie on one graph.

Find Operations_

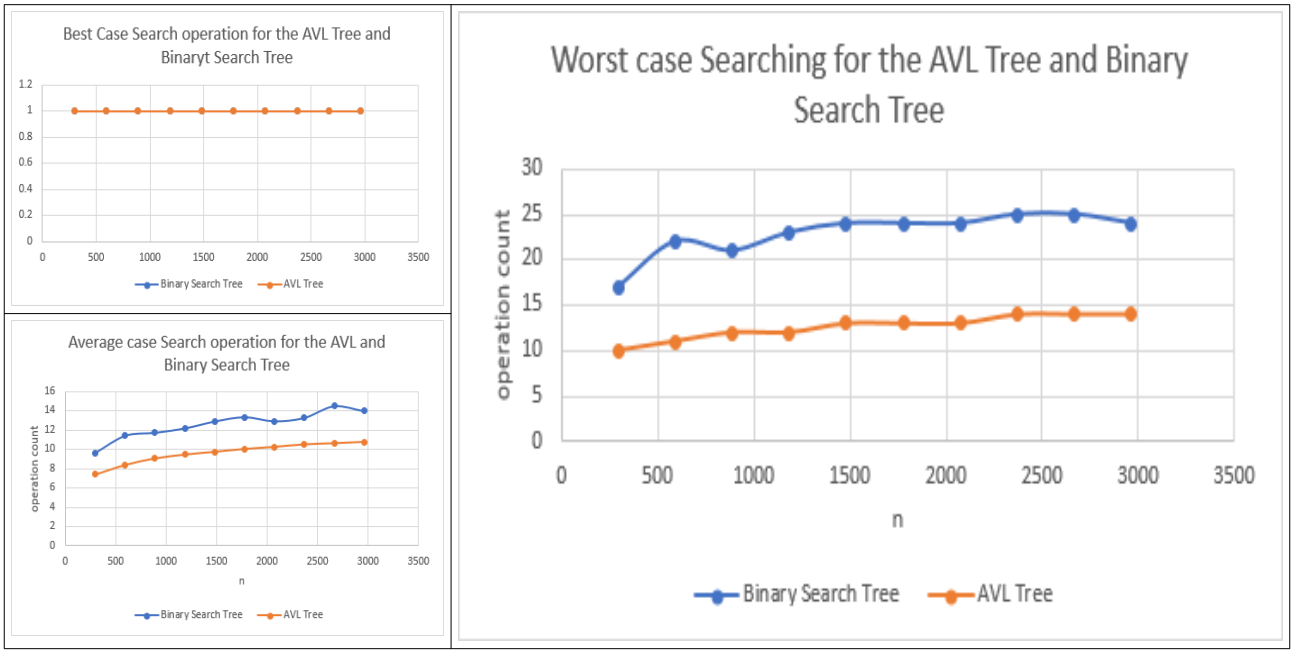

Figure 3 : Comparison of the two data structures for conducting Insert Operations.

Notice that the best case scenario appears to display results for one data structure, due to that the results for both data structures are exactly the same, and so lie on one graph.

5. Results Discussion

From figure 2 and figure 3 big O relationships are derived and shown in Table 1 below.

Table 1 : Big (O) analysis of the results

	Find			Insert		
Case	Best	Average	Worst	Best	Average	Worst
AVL Tree	O(1)	O(log(n))	O(log(n))	O(1)	O(log(n))	O(log(n))
Binary Search Tree	O(1)	O(log(n))	O(log(n))	O(1)	O(log(n))	O(log(n))

From figure 2 and figure 3 it can be realized that inserting on an AVL Tree on worst case takes logarithmic time. The same operation also takes logarithmic time in the Binary Search Tree on worst case. Although extra balancing techniques are performed in an AVL Tree, the graphs show that inserting in an AVL tree takes less comparison steps than in a Binary Search tree, hence the AVL Tree becomes more efficient than the Binary Search Tree in inserting, if comparison steps are taken into account.

Moreover, finding or searching for items in an AVL Tree in the worst takes logarithmic time. The same operation also takes logarithmic time on a Binary search tree in the worst case. Less comparison steps are required in an AVL Tree, and so, tt can be deduced that conducting a search operation is more efficient in an AVL Tree than in a Binary Search Tree.

6. Conclusions

It is more efficient to run a search or find operation in an AVL Tree then in a Binary Search Tree. However both data structures have equivalent Big O analysis', but the difference comes when individual graph values(comparison steps) are taken into account. The AVL balancing technique makes it more efficient to search in an AVL Tree than in a Binary Search Tree while keeping insert more efficient because it does not require more comparison steps once the tree is balanced.

7. Creativity

Additional functionality added to the experiment was :

- The make commands allow users to specify parameters in a user friendly manner
- Users can conduct their own tests using the application, and the results tables, and graphs will be automatically generated

8 . Development

8.1 Information

Documentation of the applications can be accessed through the doc directory Some of the useful make commands to interface with the application are as follows:

NOTE : make clean command only work when the specified operations have been run and the files to be removed exist on their respective directories.

#compile all classes
$ make

#remove class files files
$ make clean

#clean all test files
$ make clean-t

#cleans documentation
$ make clean-docs

# cleans backup files
$ make clean-backups

#removes class files, documentation and test files
$ make clean-all

#run LSAVLTApp
$ make runA stage=<stage> day=<day> time=<time>

#run LSBSTApp
$ make runB stage=<stage> day=<day> time=<time>

#Run tests through the python script
$ make test

8.2 Git Logs

Below are some of the Git logs through development of the applications.

```
0: commit f876515935f07adb1351384cff24ee347307d418
1: Author: “Lindelanimcebo” <lindelanimcebo@gmail.com>
2: Date: Wed Apr 29 03:43:56 2020 +0200
3:
4: make clean-all
5:
6: commit 89081f0ca065ca112fdcee72c729d2b84a285de0
7: Author: “Lindelanimcebo” <lindelanimcebo@gmail.com>
8: Date: Wed Apr 29 03:41:15 2020 +0200
9:
...
71: Author: “Lindelanimcebo” <lindelanimcebo@gmail.com>
72: Date: Sat Mar 14 01:48:00 2020 +0200
73:
74: created AVL tree app
75:
76: commit 40408a98d076e372654312afa04375122a865bcd
77: Author: “Lindelanimcebo” <lindelanimcebo@gmail.com>
78: Date: Sat Mar 14 00:31:32 2020 +0200
79:
80: initial commit with functioning Binary Search Tree, Makefile and Python runTests Script
```