Comparing Insert and Find Operations on a Binary Search Tree and a Traditional Array

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 a traditional array, specifically testing insert and search/find operations. Analysis is presented using big O analysis

1. Introduction

The goal of the experiment was to compare efficiencies of an Array and a Binary Search Tree 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 LSArrayApp to test the traditional array data structure, and the LSBSApp developed to test the BinarySearchTree data structure. To allow applications functionality an OOP aproach was taken which includes, model class to handle all data structure operations for each data structure, an Entry class of which each line of data is an object and a FileHelper class which handles all file operations for both applications. Figure 1 below shows the OOP design of the applications

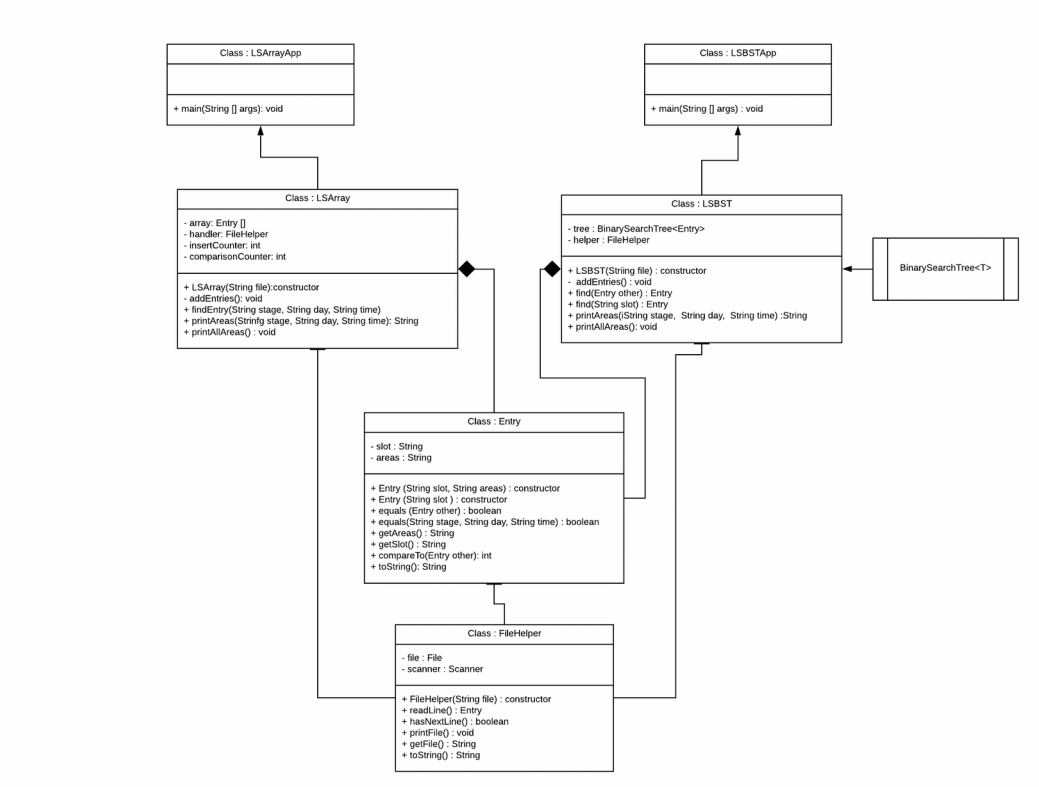


Figure 1 : UML Diagram illustrating OOP design of the applications

3. Experiment description

The experiment was conducted on the applications through instrumentation and counting comparison steps for each operation : insert, and search. Test cases for each data structure application were developed. The applications were then tested through randomized data sets with size ranging from 297 to 2796 with a step of 296 and for each case the applications were tested to find each value in each subset, keeping track of the operation count for inserting and finding to and from the data structure respectively.

4. Experiment Results

Throughout the experiment the applications were first tested with different parameter combinations to ensure functionality and reliability. The tests were as follows

4.1 Trial test on LSArrayApp

4.1.1 Testing printAreas() method

input : stage = 8 , day = 16 , time = 22
application command : make runA stage=8 day=16 time=22
unix comand : Java -cp bin LSArayApp 2 32 00
logs :

```
Mar 04, 2020 1:14:13 AM LSArrayApp main
INFO: The Areas are : 15, 7, 11, 3, 12, 4, 8, 16
insert   comparison
0        2976
```

input : stage=2, day=32 , time=00
appication comand: make runA stage=2 day=32 time=00
unix comand : Java -cp bin LSArayApp 2 32 00
logs :

```
java -cp bin LSArrayApp 2 32 00
Mar 04, 2020 1:29:40 AM LSArrayApp main
INFO: Areas not found
```

4.1.2 Testing printAllAreas

input : no arguments
application command : make runA
unix command : Java -cp bin LSArayApp
logs:

Java -cp bin LSArayApp 8 16 22	Last 10 lines
java -cp bin LSArrayApp Slot 1_1_00 Areas : 1 Slot 1_17_00 Areas : 1 Slot 1_2_00 Areas : 13	Slot 8_27_22 Areas : 14, 6, 10, 2, 15, 7, 11, 3 Slot 8_12_22 Areas : 14, 6, 10, 2, 15, 7, 11, 3 Slot 8_28_22 Areas : 14, 6, 10, 2, 15, 7, 11, 3 Slot 8_13_22

Slot 1_18_00 Areas : 13 Slot 1_3_00 Areas : 9 Slot 1_19_00 Areas : 9 Slot 1_4_00 Areas : 5 Slot 1_20_00 Areas : 5 Slot 1_5_00 Areas : 2 Slot 1_21_00 Areas : 2	Areas : 15, 7, 11, 3, 12, 4, 8, 16 Slot 8_29_22 Areas : 15, 7, 11, 3, 12, 4, 8, 16 Slot 8_14_22 Areas : 15, 7, 11, 3, 12, 4, 8, 16 Slot 8_30_22 Areas : 15, 7, 11, 3, 12, 4, 8, 16 Slot 8_15_22 Areas : 15, 7, 11, 3, 12, 4, 8, 16 Slot 8_31_22 Areas : 15, 7, 11, 3, 12, 4, 8, 16 Slot 8_16_22 Areas : 15, 7, 11, 3, 12, 4, 8, 16

4.2 Trial Tests on LSBSTApp

4.2.1 Testing printAreas

input : stage = 4 , day = 9 , time = 18
application command : make runB stage=4 day=9 time=18
unix comand : Java -cp bin LSBSTApp 4 9 18
logs :

```
java -cp bin LSBSTApp 4 9 18
Mar 04, 2020 1:38:45 AM LSBSTApp main
INFO: The areas are: 12, 4, 8, 16
insert   comparison
270100           78
```

input : stage = 9, day = 05 , time = 20
appication comand: make runB stage=9 day=05 time=20
unix comand : Java -cp bin LSBSTApp 9 05 20

```
jjava -cp bin LSBSTApp 9 05 18
Mar 04, 2020 1:41:17 AM LSBSTApp main
INFO: No Areas Found
```

4.2.2 Testing printAllAreas

input : no arguments
application command : make runB
unix command : Java -cp bin LSBSTApp
logs

1st 10 lines	Last 10 lines
java -cp bin LSBSTApp Slot 1_10_00 Areas : 15 Slot 1_10_02 Areas : 16 Slot 1_10_04 Areas : 1	Slot 8_9_04 Areas : 5, 13, 1, 9, 6, 14, 2, 10 Slot 8_9_06 Areas : 6, 14, 2, 10, 7, 15, 3, 11 Slot 8_9_08 Areas : 7, 15, 3, 11, 8, 16, 4, 12 Slot 8_9_10

Slot 1_10_06 Areas : 2 Slot 1_10_08 Areas : 3 Slot 1_10_10 Areas : 4 Slot 1_10_12 Areas : 5 Slot 1_10_14 Areas : 6 Slot 1_10_16 Areas : 7 Slot 1_10_18 Areas : 8	Areas : 8, 16, 4, 12, 9, 1, 5, 13 Slot 8_9_12 Areas : 9, 1, 5, 13, 10, 2, 6, 14 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

The test was automated through a python script which generated the results for best, average and worst cases into a csv file for each data structure. Results Tables for both data structures can be found on appendix A or the logs directory.

Using the auto-generated tables, big O analysis of the results was plotted and is shown in figure 2 and figure 3 below

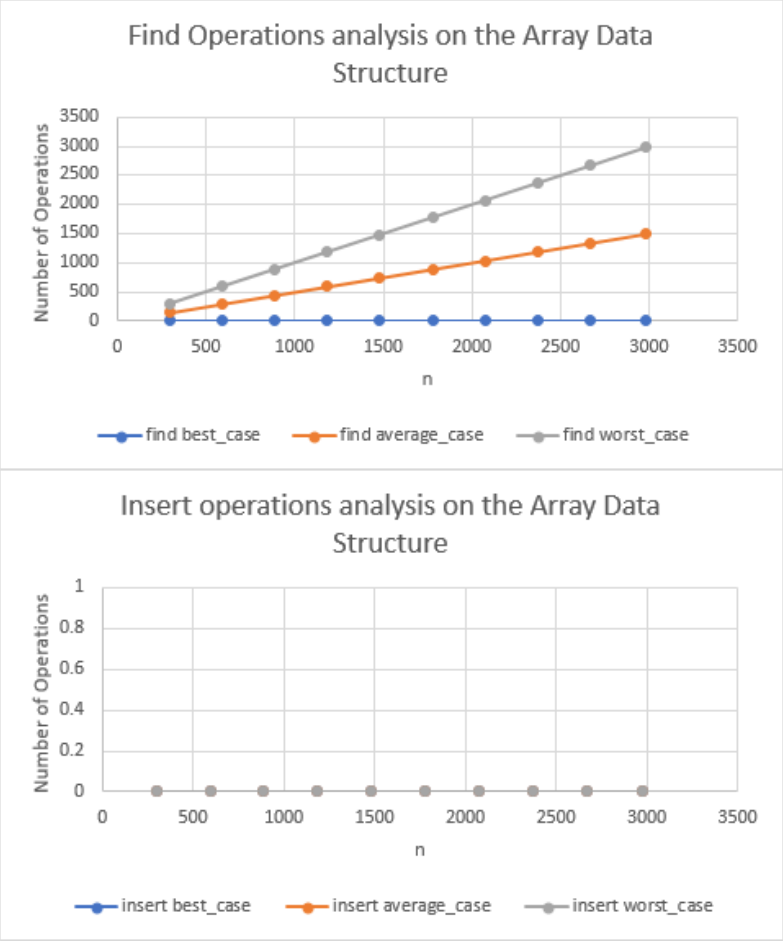
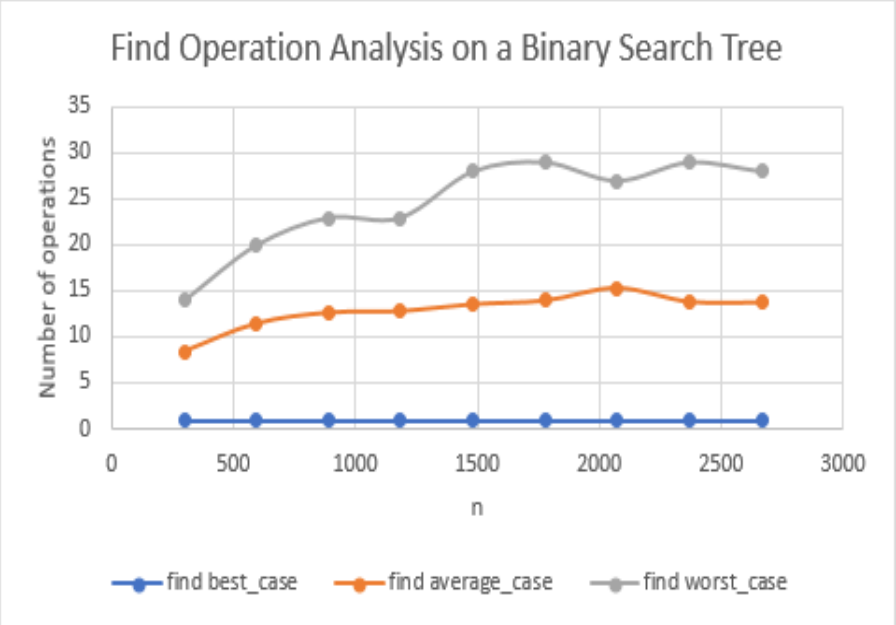


Figure 2 : Analysis for insert and find operation on the Array.



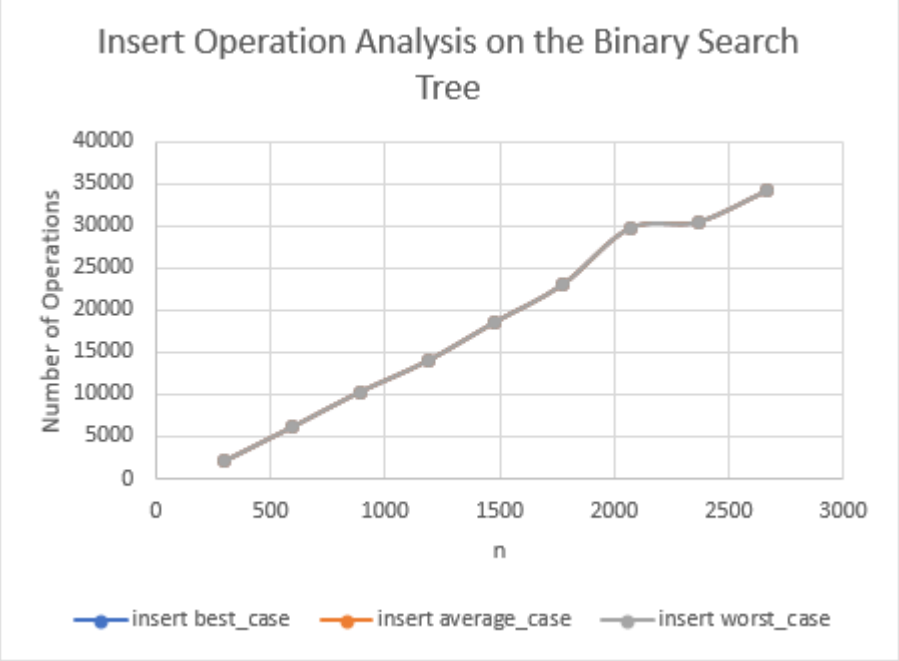


Figure 3 : Analysis for insert and find operations on the Binary Search Tree.

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
Array	O(1)	O(n)	O(n)	O(1)	O(1)	O(1)
Binary Search Tree	O(1)	O(log(n))	O(log(n))	O(n)	O(n)	O(n)

From figure 2 and figure 3 it can be realized that inserting on an Array on worst case takes constant time, this is because there are no comparison operations required since, the index of the last element is known. However the same operation takes linear time in the Binary Search Tree since multiple comparison steps are required to insert the data in a sorted manner.

Moreover, finding or searching for items in an Array on worst takes linear time, since this requires linear searching through an array and comparing the search value with the elements each time. The worst case occurs when the element in question is the last element in the array. However the same operation takes logarithmic time on a Binary search tree, since the data is already sorted, and searching divides and conquers the tree, separating it into two halves each time and searching in the relevant child tree. Worst case occurs when the element in question is in the lowest level of the tree or is a leaf.

6. Conclusions

It is more efficient to run a search or find operation using a Binary Search Tree than a traditional array. However, the trade-offs are, inserting unsorted data to a Binary Search Tree is very inefficient as compared to an Array, and the Binary Search Tree can only be implemented with comparable data. And so, in a case of persistent data, a Binary Search Tree would be a better choice than an Array, but in a case where data changes rapidly and the data changes rapidly than a binary search tree would not be a better choice. However, these conclusions do not mean that these two data structures are the best for the outlined application, more data structures can be investigated, developed and implemented.

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

Development specifications can be accessed via the ReadMe.md file. 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:

#compile all classes
$ make

#remove unused files
$ make clean

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

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

#clean all test files

make clean-t

#Run tests
make test

8.2 Git Logs

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

```
### $ git log | (ln=0; while read l; do echo $ln\: $l; ln=$((ln+1)); done) | (head -10; echo ...; tail -10)
0: commit 6c2cd9a0d8f9605efd723a963007d7116b0c1281
1: Author: “Lindelanimcebo” <lindelanimcebo@gmail.com>
2: Date: Wed Mar 4 02:17:38 2020 +0200
3:
4: test with n = 297 to 2665 with a step of 296
5:
6: commit eef3f00bd59f2a338955f09ef3069e6e6ee4e4ea
7: Author: “Lindelanimcebo” <lindelanimcebo@gmail.com>
8: Date: Tue Mar 3 05:44:05 2020 +0200
9:
...
73: Author: “Lindelanimcebo” <lindelanimcebo@gmail.com>
74: Date: Tue Feb 25 23:52:52 2020 +0200
75:
76: Added functionality to print all areas and to print details of a certain slot
77:
78: commit 431e32faf97c250e153f648d3956d5813cb1a549
79: Author: “Lindelanimcebo” <lindelanimcebo@gmail.com>
80: Date: Tue Feb 25 21:20:17 2020 +0200
81:
82: initial commit
```