CSC2001F Assignment 2 Report
Deon Fourie

I approached this assignment with the following OO design in mind:

For both the LSBSTApp and LSAVLTApp classes I created a class called LSData. This class creates key and value objects for the BST and AVLT. It also has useful methods like equals() and toString() which are used in LSBST and LSAVLT classes. LSData's key and value objects are inserted into both trees which can then be found by LSData's equals() method. LSData's toString() method is invoked whenever both applications are run to display the contents of both trees.

Another class that was created to be used in both applications is called FileIn. This class was included to keep the LSBST and LSAVLT classes simple, i.e. to handle all file operations which are the same in both classes. This class initializes scanner and file objects which are used to read in the data file. The method readLine() splits every line of the data file into key and value parameters, which are added into the LSData class.

For LSBSTApp:
The class LSBSTApp is the driver class for the LSBST class. It simply calls printAreas or printAllAreas from the LSBST class, depending on the input. The printAreas method in the LSBST class takes in 3 arguments provided by the user and concatenates them by separating the arguments with "_". This then becomes a query key. This tree is then searched to find a matching key value. If a matching key value is found, the key and area pair is printed. If the matching key does not exist, the method returns "Areas not found". Insert counters and Search counter variable are also printed, details of this will be mentioned later. The printAllAreas method does an in-order traversal of the tree. The LSBST class has a constructor which instantiates a BinarySearchTree object with a LSData datatype. The BinarySearchTree, along with BinaryTree, BinaryTreeNode, BTQueue and BTQueueNode classes were all provided by Hussein. An add() method in LSBST inserts a LSData object node into the BST. The LSBST class also has a search method() which calls the BST's find method to find a key in the BST. The FileIn class is used to read in the data file.

The addition made to Hussein's code was to include countInsert and countFind variables as well as getCountFind and getCountInsert methods into his BST class. These variables were incremented in the insert and find methods of the BST class. They increment every time a comparison operation is made. This is a very high-level overview of the LSBSTApp.

For LSAVLTApp:
The LSAVLTApp class is the driver class for the LSAVLT class. It simply calls printAreas or printAllAreas from the LSAVLT class, depending on the input. The printAreas method in the LSAVLT class takes in 3 arguments provided by the user and concatenates them by separating the arguments with "_". This then becomes a query key. This tree is then searched to find a matching key value. If a matching key value is found, the key and area pair is printed. If the matching key does not exist, the method returns "Areas not found".

Insert counters and Search counter variable are also printed, details of this will be mentioned later. The printAllAreas method does an in-order traversal of the tree. The LSAVLT class has a constructor which instantiates a AVLTree object with a LSData datatype. The AVLTree, along with BinaryTree, BinaryTreeNode, BTQueue and BTQueueNode classes were all provided by Hussein. An add() method in LSAVLT inserts a LSData object node into the AVLT. The LSAVLT class also has a search method() which calls the AVLTree's find method to find a key in the BST. The FileIn class is used to read in the data file.

The addition made to Hussein's code was to include countInsert and countFind variables as well as getCountFind and getCountInsert methods into his AVLTree class. These variables were incremented in the insert and find methods of the AVLTree class. They increment every time a comparison operation is made. This is a very high-level overview of the LSAVLTApp.

The experiment:

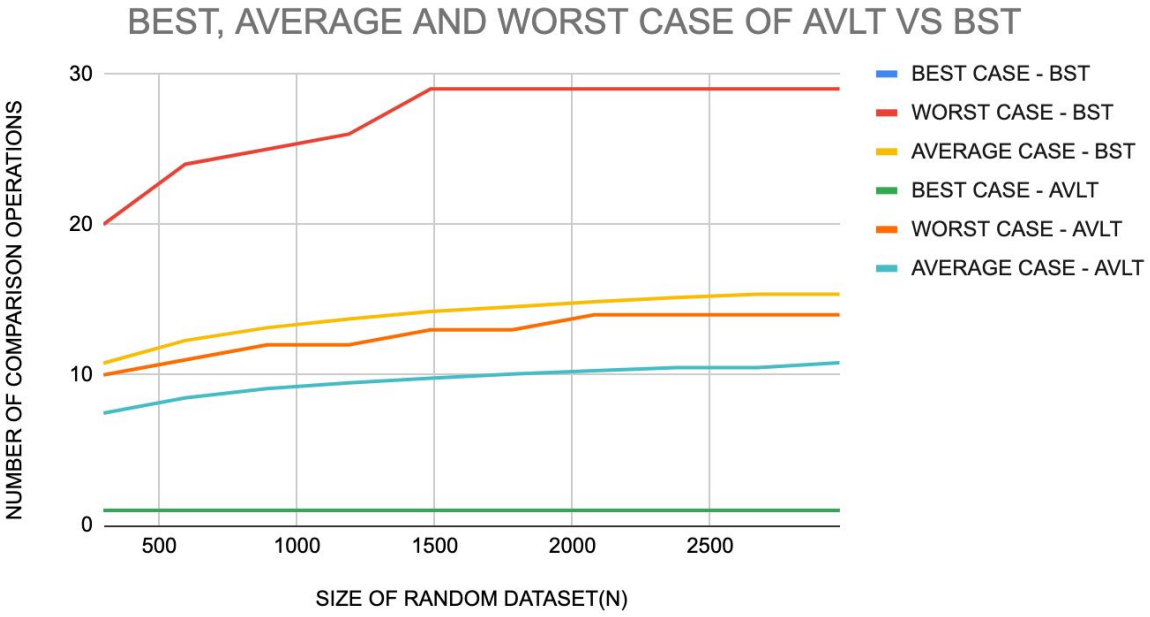
The goal of this experiment was to compare the efficiency of searching and inserting nodes into a BST vs an AVLT. The main objective was to compare the number of comparison operations required for exactly the same inputs to both applications. This way we could actually see how these data structures behave in a real-world example. In the scripts directory, there is a python script called part_5.py. This script does a number of things. Firstly, it randomized the entire dataset to produce a file called data/random_data.txt. This is to ensure that the BST does not become unbalanced as a result of the ordering of the original dataset. The randomized dataset is then split into 10 subsets. These 10 datasets were then given different sizes(n) from n = 297, 594, …, n = 2976. This way, a different tree can be created for different sizes of data, which is useful when comparing the big - O of both trees. We can see how the tree differs by increasing n.

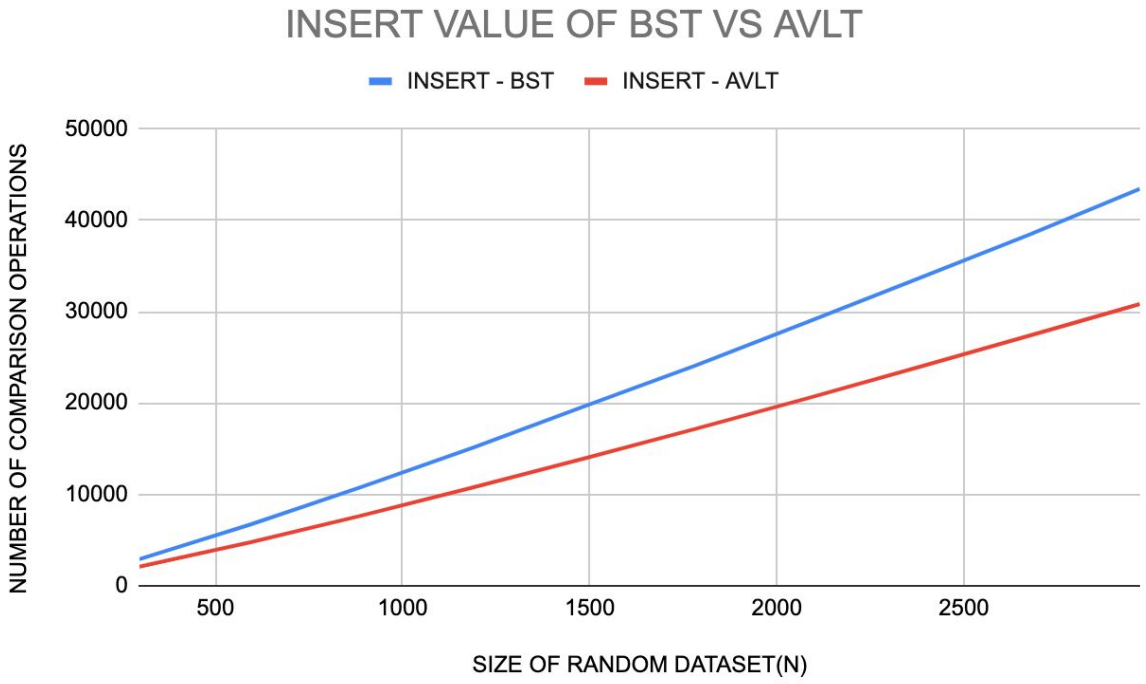
The functions run_LSAVLTApp and run_LSBSTApp carry out similar functions for both apps. For each app, the script looped through the entire list of datasets. For each dataset, every single parameter in the dataset was passed in as an argument to the app. For every dataset and argument, the output was stored in a file in the output directory. Each dataset gave a different insert and search counter value. The output was then further processed by the script to return a best, average and worst-case search counter value. These values were then included in an excel spreadsheet. All the data can be found in the output directory.

Results of the experiment

		LSBSTApp		
N	INSERT - BST	BEST CASE - BST	WORST CASE - BST	AVERAGE CASE - BST
297	2905	1	20	10.78114478
594	6704	1	24	12.28619529
891	10815	1	25	13.13804714
1188	15103	1	26	13.71296296
1485	19635	1	29	14.22222222
1782	24100	1	29	14.52413019
2079	28807	1	29	14.85618086
2376	33586	1	29	15.13552189
2673	38390	1	29	15.36213992
2970	43438	1	29	15.36213992

		LSAVLTApp		
N	INSERT - AVLT	BEST CASE - AVLT	WORST CASE - AVLT	AVERAGE CASE - AVLT
297	2088	1	10	7.461279461
594	4756	1	11	8.47979798
891	7684	1	12	9.090909091
1188	10755	1	12	9.466329966
1485	13919	1	13	9.785185185
1782	17164	1	13	10.05387205
2079	20484	1	14	10.28571429
2376	23901	1	14	10.49200337
2673	27348	1	14	10.49200337
2970	30832	1	14	10.80774411





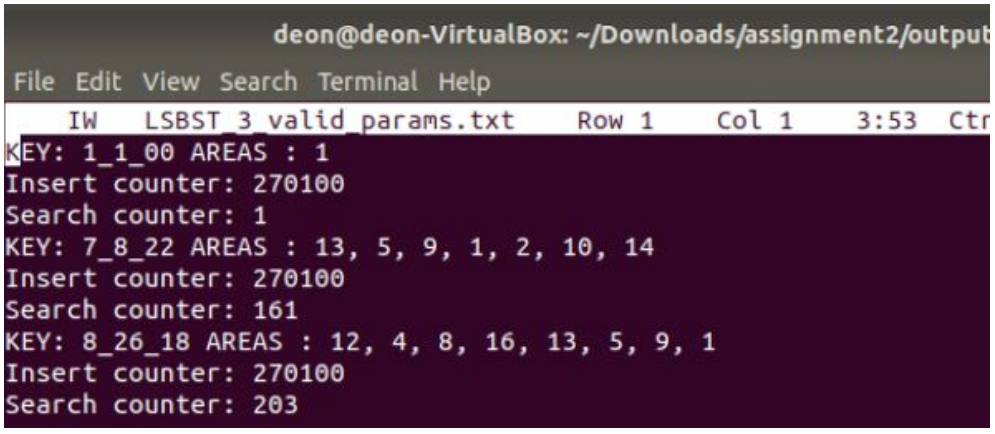
The results of the experiment yielded the above results. One thing that stood out, was that the worst-case search count for LSBST was below 30 for all of the datasets. If we look at the search count in the following page for LSBST the value is >200. This is because of part 2 and part 4 outputs on the following page are trees made out of the standard (non-randomised) dataset. The BST and AVLT made in the above experiment used a randomised dataset, which resulted in the BST being far more balanced than the BST using the ordered dataset. In both trees, the best case search count was 1, which is obvious. Regarding the average and worst-case values for a randomised dataset, the AVLT had lower values for searching and inserting. That being said, both search values only differed by a constant, which in both trees means that the trees were O(logN). Both insert values, were

O(N), which is also obvious since the length of the dataset is proportional to the number of insertions.

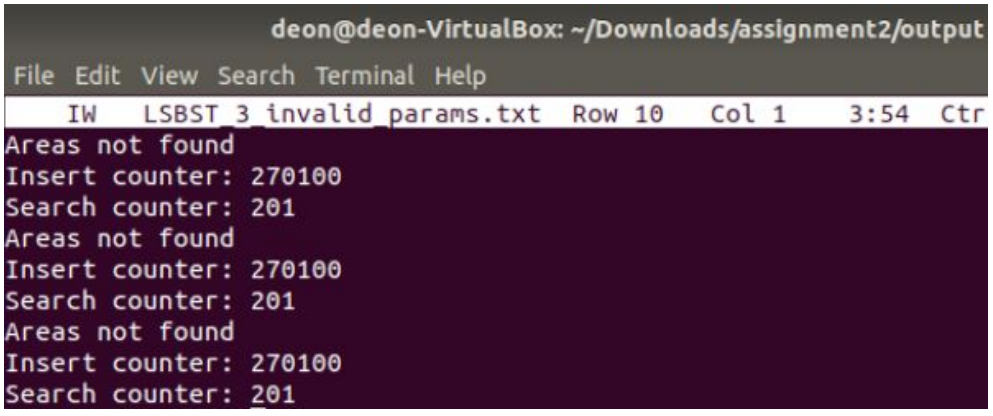
These results are however not always the case. As mentioned above, the BST is almost as balanced as the AVLT because the datasets were random. In the case of the BST with an ordered dataset, like the example below, the complexity will be O(N), which is why the search and insert values below are much higher for BST. The AVLT will always have O(logN) for search, which is why even the search values in the ordered dataset in the part 4 results below resemble the results of the random dataset. The spreadsheet can be found in the data directory.

Part 2 Test Values: LSBSTApp

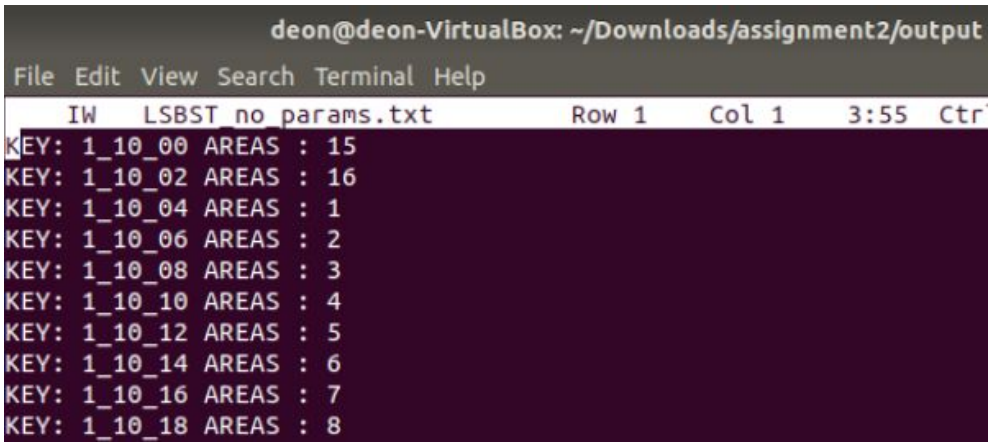
3 Valid Parameters passed into LSBSTApp

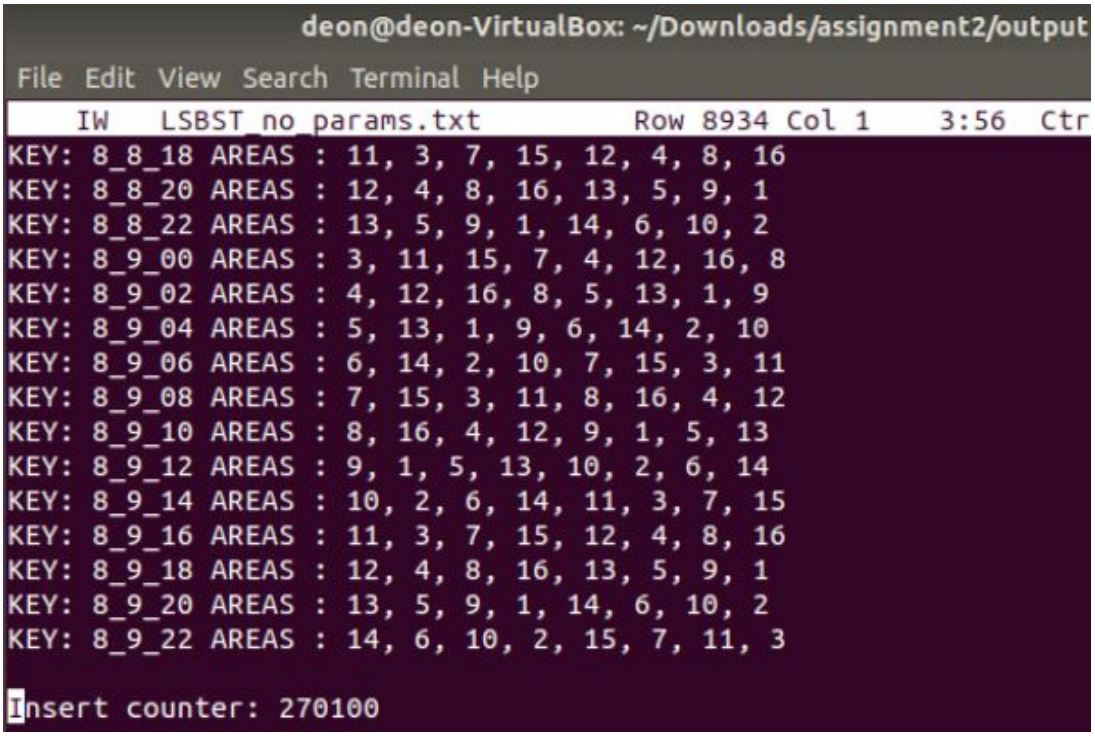


3 Invalid parameters passed into LSBSTApp



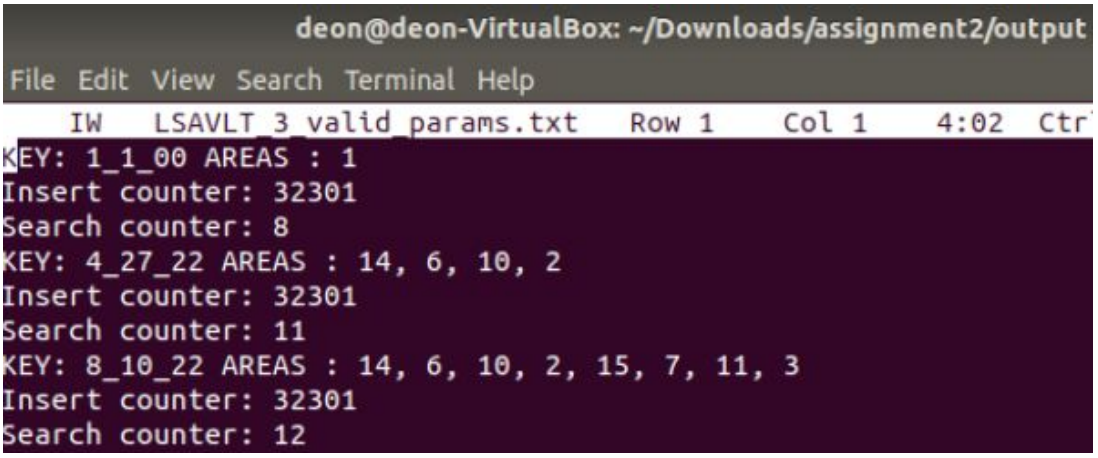
No Parameters passed into LSBSTApp (first and last 10 lines)



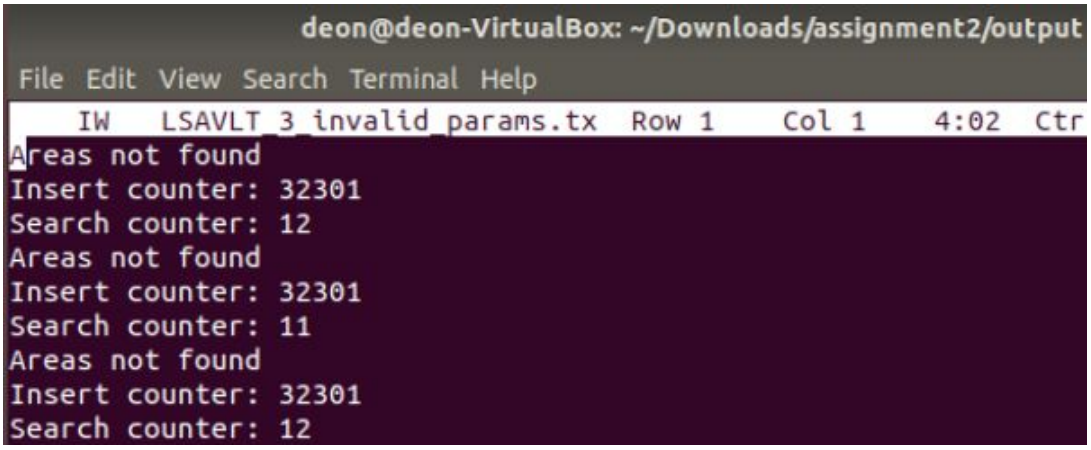


Part 4 Test Values

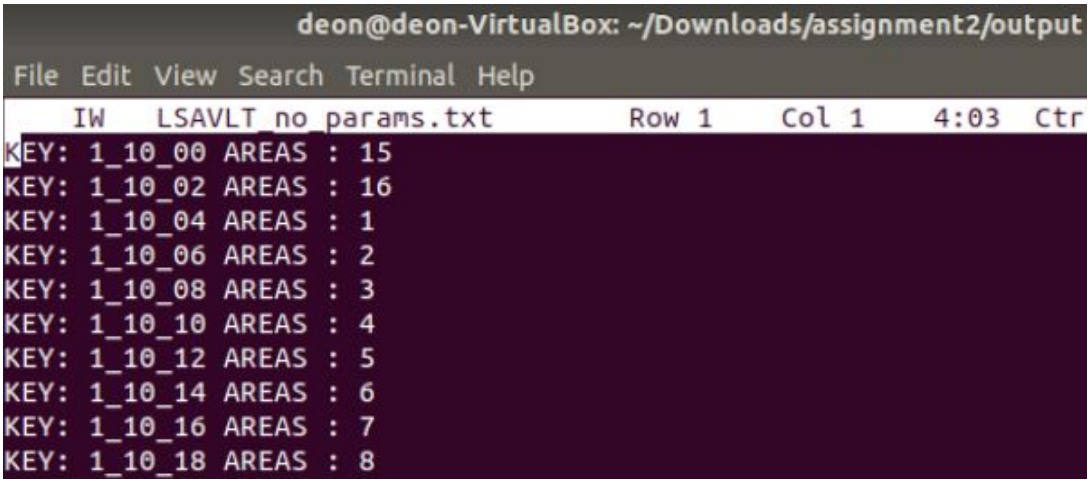
3 Valid Parameters passed into LSAVLTApp



3 Invalid parameters passed into LSAVLTApp



No parameters passed into LSAVLTApp (first and last 10 lines)



```
KEY: 8_8_16 AREAS : 10, 2, 6, 14, 11, 3, 7, 15
KEY: 8_8_18 AREAS : 11, 3, 7, 15, 12, 4, 8, 16
KEY: 8_8_20 AREAS : 12, 4, 8, 16, 13, 5, 9, 1
KEY: 8_8_22 AREAS : 13, 5, 9, 1, 14, 6, 10, 2
KEY: 8_9_00 AREAS : 3, 11, 15, 7, 4, 12, 16, 8
KEY: 8_9_02 AREAS : 4, 12, 16, 8, 5, 13, 1, 9
KEY: 8_9_04 AREAS : 5, 13, 1, 9, 6, 14, 2, 10
KEY: 8_9_06 AREAS : 6, 14, 2, 10, 7, 15, 3, 11
KEY: 8_9_08 AREAS : 7, 15, 3, 11, 8, 16, 4, 12
KEY: 8_9_10 AREAS : 8, 16, 4, 12, 9, 1, 5, 13
KEY: 8_9_12 AREAS : 9, 1, 5, 13, 10, 2, 6, 14
KEY: 8_9_14 AREAS : 10, 2, 6, 14, 11, 3, 7, 15
KEY: 8_9_16 AREAS : 11, 3, 7, 15, 12, 4, 8, 16
KEY: 8_9_18 AREAS : 12, 4, 8, 16, 13, 5, 9, 1
KEY: 8_9_20 AREAS : 13, 5, 9, 1, 14, 6, 10, 2
KEY: 8_9_22 AREAS : 14, 6, 10, 2, 15, 7, 11, 3

Insert counter: 32301
```

Git Log Summary

```
deon@deon-VirtualBox:~/Downloads/assignment2$ git log | (ln=0;
 echo $ln\: $l; ln=$((ln+1)); done) | (head -10; echo ...; tail
0: commit 793d18078b9574c8147443dfa30b97494d04f1a0
1: Author: Deon <deon4e@gmail.com>
2: Date: Mon Apr 27 11:36:12 2020 +0200
3:
4: Last commit, everything is working
5:
6: commit 127b0fc5f24ce3be4a31fc5c3ffc19796e27a8c1
7: Author: Deon <deon@Deons-MacBook-Pro.local>
8: Date: Wed Apr 22 13:24:35 2020 +0200
```