Summary of Data Structures

Two compound data structures were used to complete the assignment. The first implementation is a Binary Search Tree (BST) combined with a Linked-list and the second implementation is an AVL tree with a linked list. The data structures are used due to the structure of how each of them function. BSTs allow a quick search operation and combined with the linked list it allows for the data stored inside to be sorted. Moreover AVL trees keep all the data balanced no matter how much data is input. This ensures a consistent performance in searching and inserting data. Using trees and linked lists is allowing the solution to be optimised and scalable due to the nature of both the data structures.

Brief discussion on testing

The insertion operation for both of the implementations were chosen to compare. The reason behind choosing the insertion operation, is to compare how two different variations of trees behave when data is added to them. Understanding the behaviour is important to analyse and provide insights on the efficiency, scalability and and the balance - maintenance capabilities. The testing methodology used is using a dataset that has a lot of randomly generated data, which is being used as input for the insertion operation.

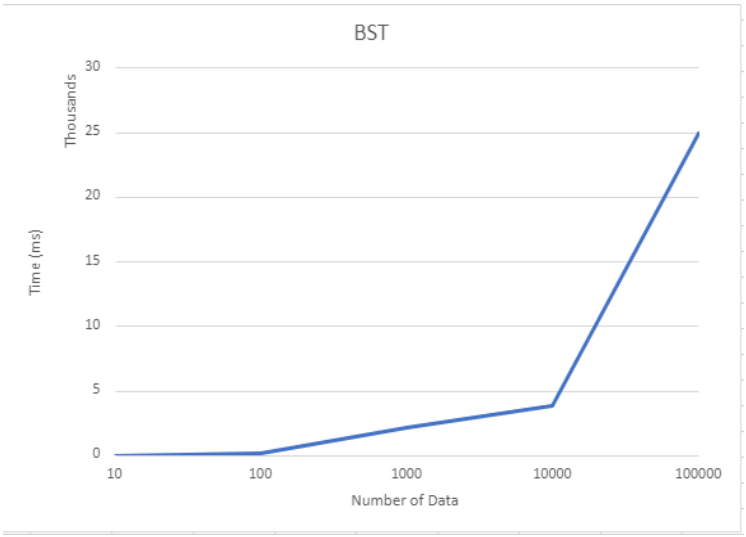
Theoretical analysis

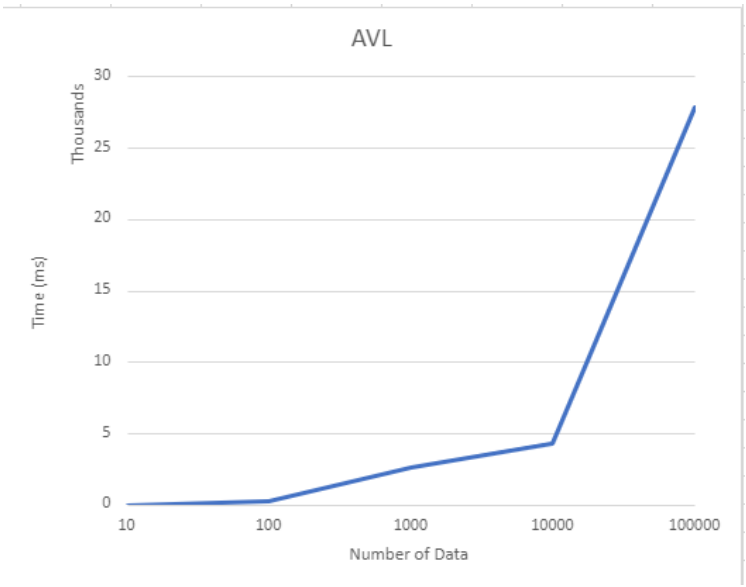
When the BST is balanced the average and worst case time complexity is O(log n), however if the tree is not balanced then the worst case can be an O(n). On the contrary AVL trees are always balanced and can consistently keep a time complexity of O(log n) as it does not degrade.

Data structure comparison

A	B	C	D	E	F
Data structure name			**Number of Data**		**Time (ms)**
AVL			10		29
			100		302
			1000		2643
			10000		4292
			100000		27865
BST			10		19
			100		183
			1000		2131
			10000		3837
			100000		24998

Elapsed time to run the solutions in tabular format.





The visuals and the data recorded show that the graph for AVL is steeper compared to BST, which signifies that for the same number of data AVL takes more time to insert. This is due to the fact that AVL trees are self balancing in nature and need to balance the node after data insertion, which takes time.

Appendix

Github link: [Eusha425/Assignment-1 (github.com)]

The files in the repository contain the header files for BST and AVL trees and the implementation files along with the respective main files. Moreover it also contains a text file which has the generated data for testing also the python script used to generate the data in the first place. However it also has an implementation and header file for another compound data structure, a hash table of linked list, but that implementation was not used and is not part of the submission. The repo has two main files named main_AVL.c and main_BST.c. To run the AVL

code, we need to rename the main method in main_BST to something like “main2” from only “main”, and vice versa.

Resources used

Some code optimised from the tutorials and ChatGPT assistance for improvising the code to fit the requirements.