**Step 3 & 4:**

Table

Description automatically generated

Waterfall chart

Description automatically generated with low confidence

Table

Description automatically generated

A picture containing waterfall chart

Description automatically generated

**Insertion:** For the AVL tree data structure, it takes almost the double time of BST data structure to insert values into the tree in ascending and descending both cases. However, with random order of data, BST tree inserts data in significant less time while it didn’t change much for the AVL tree. BST tree data structure would be the best choice if a program requires huge amount of data to be inserted.

Table

Description automatically generated

Chart, waterfall chart

Description automatically generated

Table

Description automatically generated

Waterfall chart

Description automatically generated with medium confidence

**Traversal:** As input data size increases time of traversal for both data structure increases almost parallelly. Hence, for traversal nodes in the tree would take almost similar time and choosing AVL over BST or vice versa data structure in this case, would not make any huge different.

Table

Description automatically generated

Chart, waterfall chart

Description automatically generated

Table

Description automatically generated

Waterfall chart

Description automatically generated

**Deletion:** As we can see from the graph, deleting nodes in AVL data structure is significantly time consuming than BST data structure. AVL data requires balancing the tree after a deleting a node from the tree and which makes AVL tree needing significant more time than BST. However, both data structures do much better job in saving execution time when it comes to random order input data. So, deleting data from a tree, BST would be much more efficient over the AVL tree based on the executions time.

Step 5:

Text

Description automatically generated

In the worst case, insertion of BST data structure would take Big O (n). From the previous experimental graph of different size of data and order of data shows that insertion in the BST data structure takes significant less than the counterpart data structure AVL.

Text

Description automatically generated

In the worst case, insertion of AVL data structure would take Big O (lsog n). Even though it’s less expensive in time complexity while inserting data into a tree but previous experiment shows that it takes much more than BST.

Text

Description automatically generated

Text

Description automatically generated

It takes BST and AVL data structure to traversal of it’s all element in a given tree Big O(n). Traversing in BST and AVL both are almost identical in time.

Text

Description automatically generated

In the worst case, deletion of BST data structure would take Big O (n). It’s execution time much faster than AVL tree.

Text

Description automatically generated

Deleting a node from the AVL tree takes Big O (log n). But it takes much more time to delete whole tree than BST data structure.

Step 6:

Reflection: Fort this non-linear data structure project, I have implemented BST and AVL tree data structure and analyzed their functions’ executions time. The functions that I have implemented on both data structure are insertion, traversal, and deletion. I have learned how BST and AVL tree data structure behave with different input data sizes and different orders of data. In order to testing those implemented functions’ execution time it was required to input large size of data. So, I have created a function that does create three different text-based files in order of random, ascending, and descending. Those files I have used as an input for this project. What I have learned from this project is that the results of using different size and order of data are noticeably distinct in terms of time. It’s fascinating that how random order of input data is significantly faster than other orders of data when it gets inserted or deleted from a tree. Generating graphs based on execution times gives a clear picture of which data structure is to choose without getting into technical side of the data structure. I have enjoyed doing this project throughout the whole time. To me, the most difficult part of this project was deletion of large file. I had to wait somewhere between 5 to 6 mins to empty the whole AVL tree with size of 10,000 values. I was trying to empty 100,000 values in AVL tree which was taking very large amount of time to finish its execution. I wonder how it would do in terms of execution time if I just wanted to delete root node at every iteration instead of deleting input file order. I believe, it would have been much faster than the way I have implanted because it would have saved time by not searching the target node from the huge data collections and rebalancing its tree.