**AVL vs BST**

For this project I decided to compare an AVL tree with a self-balancing BST. I’ve included below the output console window showing the timetable of the time taken for each operation to complete in milliseconds along with the same values in a table. The files are separated into three categories: small, medium and large. Each of these categories are further separated into three other categories: ascending order, descending order and random. Each file increases in number by a magnitude of 10, with small files being 100 numbers, medium files being 1,000 numbers and large files being 10,000 numbers.

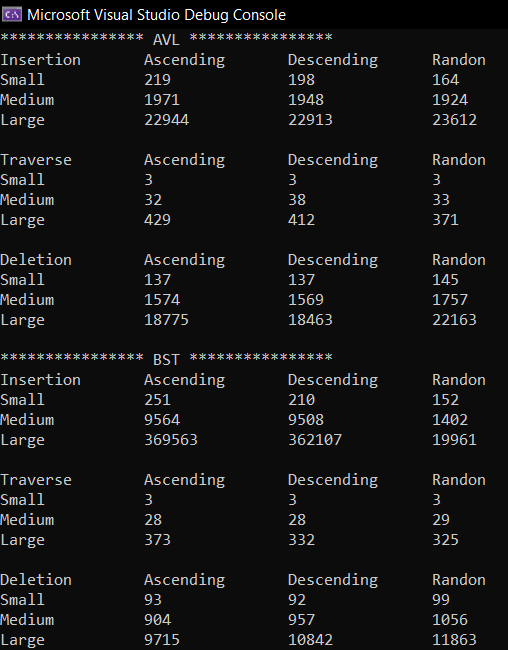


Figure . Console Window

Below in the appendix are included several graphs comparing AVL times with BST times. Figures 2-4 compare the average time taken for each operation. Figure 2 shows that the AVL tree performed slightly better than the BST with files in descending order, and much better with files in random order. On the other hand, the BST did slightly better when traversing and deleting files in descending order and even better when traversing and deleting items in random order on average.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | AVL | Ascending | Descending | Random | | Insertion | | | | | Small | 219 ms | 198 ms | 164 ms | | Medium | 1971 ms | 1948 ms | 1924 ms | | Large | 22944 ms | 22913 ms | 23612 ms | |  |  |  |  | | Traversal | | | | | Small | 3 ms | 3 ms | 3 ms | | Medium | 32 ms | 38 ms | 33 ms | | Large | 429 ms | 412 ms | 371 ms | |  |  |  |  | | Deletion | | | | | Small | 137 ms | 137 ms | 145 ms | | Medium | 1574 ms | 1569 ms | 1757 ms | | Large | 18775 ms | 18463 ms | 22163 ms | | |  |  |  |  | | --- | --- | --- | --- | | BST | Ascending | Descending | Random | | Insertion | | | | | Small | 251 ms | 210 ms | 152 ms | | Medium | 9564 ms | 9508 ms | 1402 ms | | Large | 369563 ms | 362107 ms | 19961 ms | |  |  |  |  | | Traversal | | | | | Small | 3 ms | 3 ms | 3 ms | | Medium | 28 ms | 28 ms | 29 ms | | Large | 373 ms | 332 ms | 325 ms | |  |  |  |  | | Deletion | | | | | Small | 93 ms | 92 ms | 99 ms | | Medium | 904 ms | 957 ms | 1056 ms | | Large | 9715 ms | 10842 ms | 11863 ms | |

The rest of the figures compare an operation with a given file size. For instance, Figure 5 compares the insertion operation times for BST and AVL when processing a small file size (100 numbers). For the insertion operation between Figures 5-7, the AVL tree does better for the ascending and descending files, but the BST performed just as well as the AVL tree for files in random order. For the other two operations, the BST performed better for every kind of file ordering except for the traversal of a small file (both AVL and BST performed equivalently in that case).

The insert, traversal and delete operations for the AVL tree all have a complexity of O(log(n)). The insert node traverses the tree recursively until it finds a place to insert the node. Then the balance of the tree height is checked. Depending on the structure of the tree it can be rotated right, left, left-right, or right-left. The traversal recursively goes through the left-most side of the tree before going right. If printing the preorder traversal is desired, adding a print statement before root->left and root->right would work. The delete function is called for every input read from the file. Given a key, the delete function recursively traverses through the tree depending on the numerical input value compared to the key at the present node. When found, the node is checked if there are any child nodes. If there’s only one child, it’s moved up to its parent node. If there are two, the smallest node in the right subtree is put in its place. Then the balance is checked and corrected.

The insert, traversal and delete operations for the BST also have a complexity of O(log(n)). A normal BST would have O(n) complexity for worst case scenarios. But since this is a self-balancing tree the complexity is more similar to an RB tree. It may also be important to note that the balance of the tree is checked after every 4,000 insertions and once after insertion is complete. Insertion in the BST is similar to the AVL insertion, except balancing isn’t checked with each entry. The traversal is the same. And deletion is the same except that balancing isn’t checked after each deletion.

|  |  |  |  |
| --- | --- | --- | --- |
| AVL Function | Complexity | BST Function | Complexity |
| struct Node\* insert() | O(log(n)) | struct node\* insert() | O(log(n)) |
| void preOrder() | O(log(n)) | void preOrder() | O(log(n)) |
| struct Node\* deleteNode() | O(log(n)) | struct node\* deleteNode() | O(log(n)) |

I was surprised to find that the BST performed better in most respects than the AVL tree, even with balancing. After viewing the charts created from the console output, I was also surprised to find how much better a BST insertion performs when dealing with large files of randomly assigned numbers. I thought that I had messed up in my program, to which I spent some time checking and rechecking the results. To my surprise, though, it appears to be correct. One area I had trouble with is running out of memory on the stack for my BST function. The way I went about solving this was to make the tree self-balancing (I originally was planning on doing a BST that wasn’t self-balancing). Another route I could have taken to solve this would be to go into the settings for the IDE I was using and increase the memory of the stack.

**Appendix**

**Operation Averages**

Figure . Average Insert

Figure . Average Traversal

Figure . Average Deletion

**Operation-Size Comparisons**

Figure . Small Insert

Figure . Medium Insert

Figure . Large Insert

Figure . Small Traversal

Figure . Medium Traversal

Figure . Large Traversal

Figure . Small Delete

Figure . Medium Delete

Figure . Large Delete