Justin Gou (jyg2qhc)

Lab Section: 103

The output of the results from the three testfiles suggest that AVL trees require less traversal to reach the nodes in general because the tree is always balanced. This was shown by the length of the path of the lookup functions. In the case of a generic binary search tree, the worst case for insertion, deletion, and searching is linear if, for example, the elements were passed in sorted order, since each new node would be a child of the previous tree. In comparison, an AVL tree is guaranteed to be logarithmic time since the tree is balanced after every modification.

While both BST and AVL trees use a node structure to store the information and children pointers, AVL trees store the height of the node, represented as an integer. This means each AVL node uses an extra 4*n bytes, where n is the total number of nodes (assuming a 4-byte integer). While 4 bytes is rather negligible on a small scale, it could be more significant at large sets of nodes. However, the small increase in memory usage could create a significant increase in speed on a larger scale since logarithmic time is better than linear at large scales.

The situations where AVL trees are preferred over BSTs is simply depending on the order that the nodes are added in. For example, if the nodes were added in sorted order, the binary search tree would have a height of n, where n is the number of nodes added. However, if added to an AVL tree, the height would be log(n) ± 1, which is much preferred for insertion, deletion, and finding elements.