Justin Ngo
jmn4fms
2/28/20
analysis.pdf

The different test files output proves that AVL trees are better in performance than binary search trees because the average node length used to compare speed in both data structures is shorter and better for AVL trees than binary search trees. Inserting nodes that are always greater than or less than a specific root node in a binary search tree will cause the binary search tree to become more linear and unbalanced, thus taking a longer time to run through. In contrast, inserting nodes that are always greater than or less than a specific root node in an AVL tree does not slow down retrieval time because the tree is always balancing itself.

AVL trees are a better choice than binary search trees because AVL trees prevent the worst case of a binary search tree from occurring. AVL trees are always balanced and are able to run and retrieve values faster than lists and structures that function like a list, including a linear binary search tree. The balanced AVL tree will perform at $\Theta(\log n)$ as opposed to the slower $\Theta(n)$ of a list or even an unbalanced binary search tree. Furthermore, AVL trees are much better than binary search trees at placing nodes in the same exact order such as a list because it is constantly balancing and allowing the retrieval time to be guaranteed $\Theta(\log n)$ rather than $\Theta(n)$.

Binary search trees can be preferred over AVL trees for their simplicity and less complicated implementation. If the programmer does not need to worry about balancing, the simpler implementation of a binary search tree can be preferred because it consumes less memory when it is used. For simpler implementations of each structure, an AVL tree adds extra time to balance compared to a binary search tree in which nodes can be quickly inserted without having to worry about balancing.