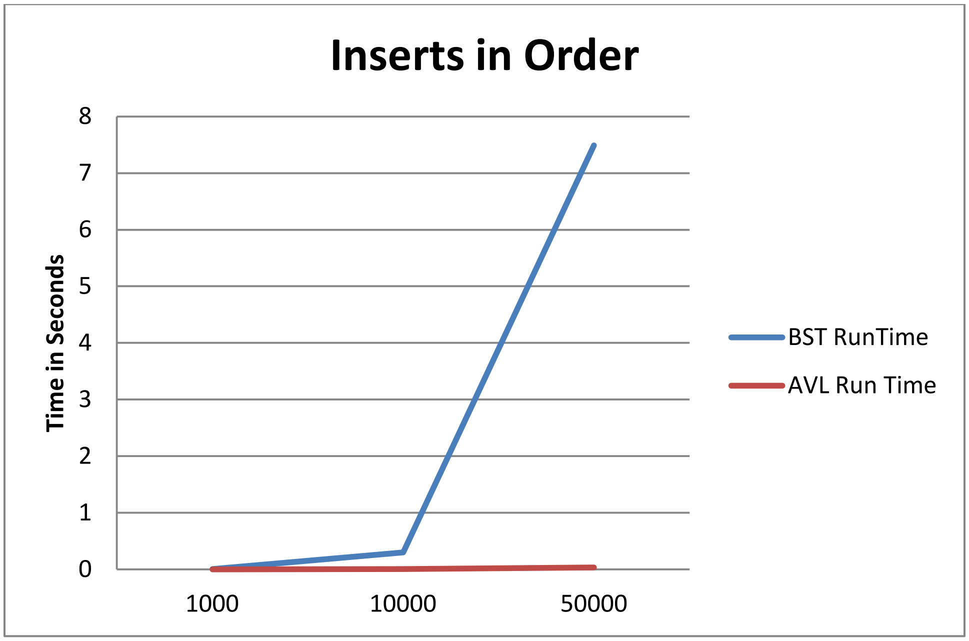
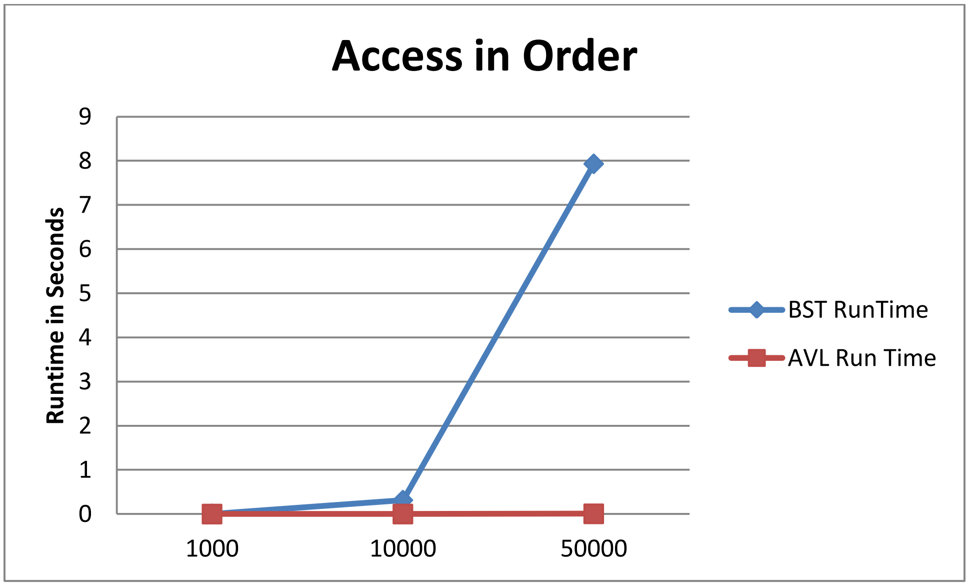
Brian Lee
3620101

CMPSC 130A PA2 Runtime Analysis

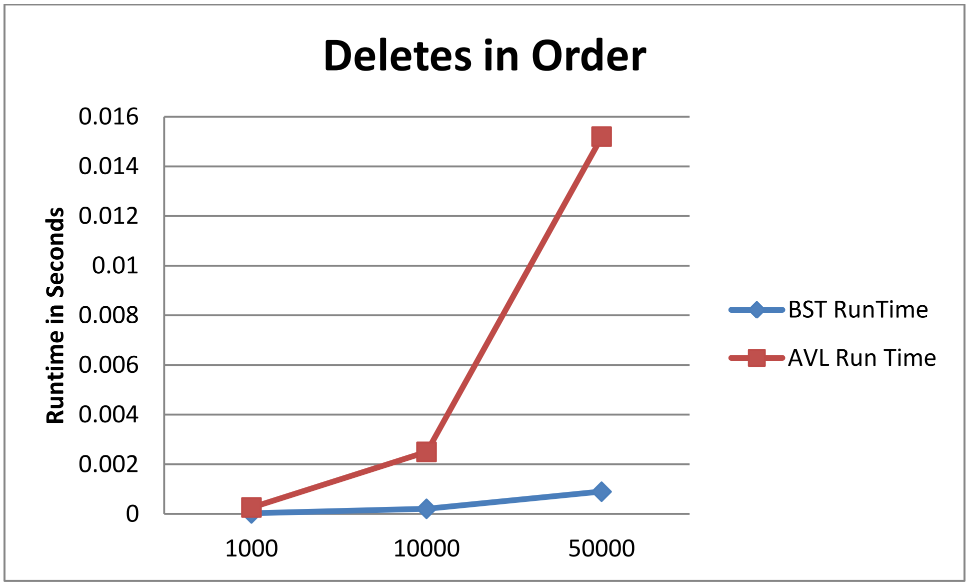
For this assignment, we were asked to do two things. First, we were to implement an AVL tree. Second, once the first task was completed, we were asked to compare our tree with the provided Binary Search Tree to see how the runtimes for the various operations stacked up.



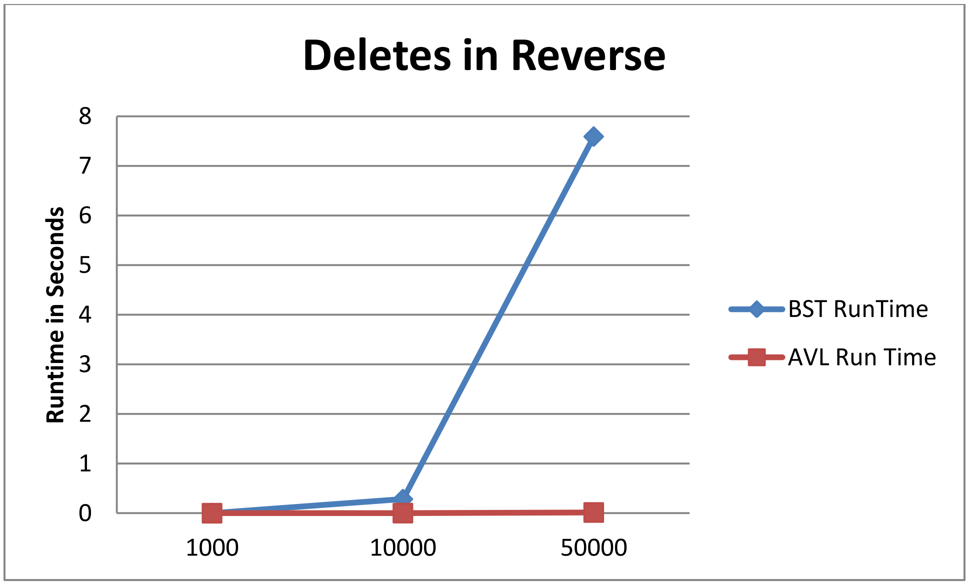
First, we inserted 1000, 10000, and 50000 integers in increasing order into both trees. As the above graph shows, the BST's runtime skyrockets upwards as the number of integers inserted increases, shooting above 7 seconds while the AVL tree barely rises above 0 seconds.



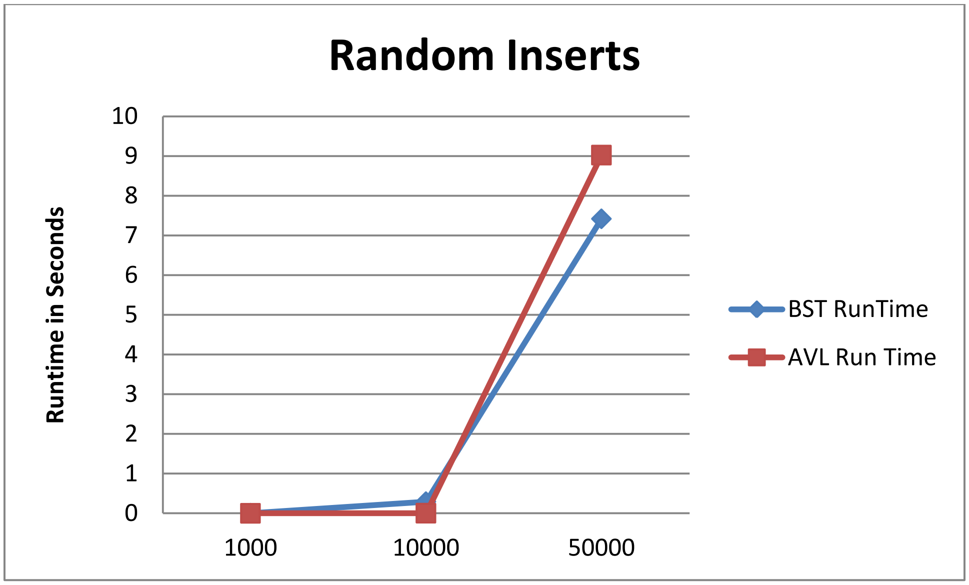
Then, we accessed every element in both trees in increasing order. Just like with the insert, the BST's runtime shoots up significantly faster than the AVL's runtime does, at almost an exponential rate. Once more, the AVL's runtime barely comes above 0 seconds while the BST's almost reaches 8 seconds.



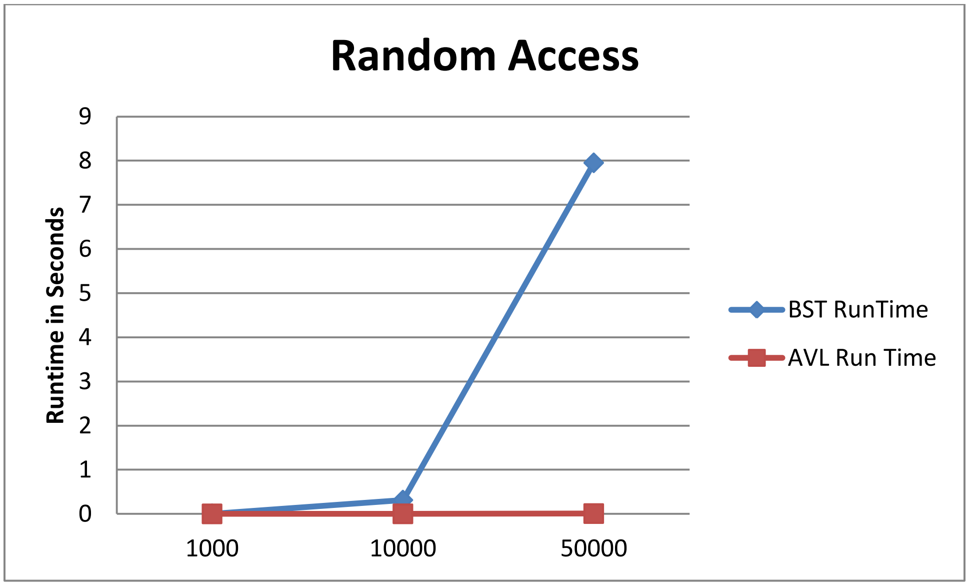
Surprisingly, it's the AVL whose runtime is greater than that of the BST when we delete all of the elements of the trees in order. However, despite the fact that it appears that the AVL tree does not handle deletions as well as the BST, it is important to note that neither tree's runtime exceeds 1 second.



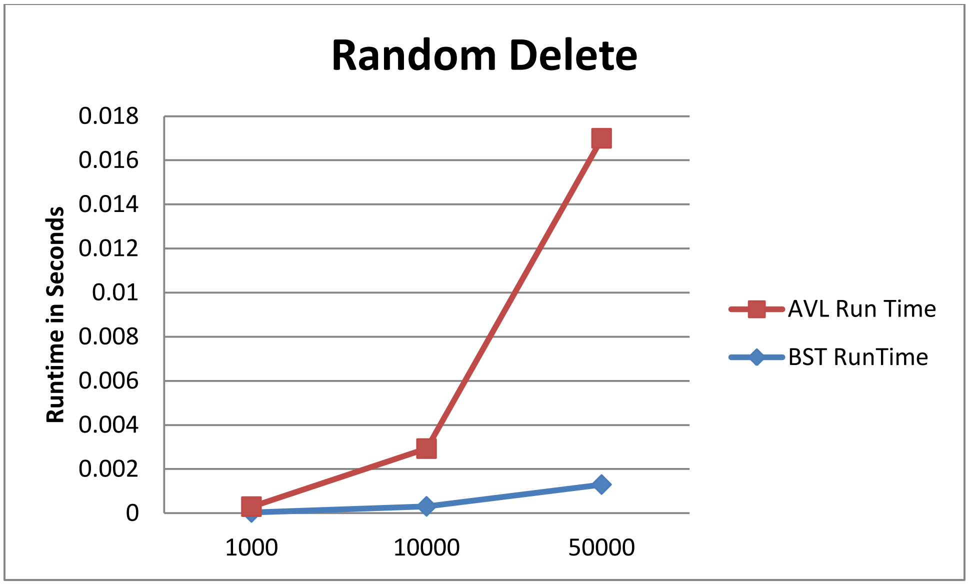
When we delete the elements of the trees in reverse, it appears that the BST again performs slower than the AVL. Again, the BST's runtime reaches above 7 seconds while the AVL's runtime does not exceed 1 second.



Now, we inserted elements into both trees in a random order. This marks the first case we looked over where the AVL tree has reached over 1 second with its runtime. It seems like the AVL tree's runtime is rising quicker than the BST's. However, given how close the two runtimes still are, a few more tests may be necessary before anything conclusive is to be said in this case.



After inserting in a random order, we then accessed every element in a random order as well. As we can see here, again, the BST's runtime has almost reached 8 seconds while the AVL tree's runtime has barely risen above 0.



And finally, we deleted every element in random order. Unlike with the insert, there is no denying that the AVL tree's runtime is growing faster than the BST's. However, it is also important to note that neither tree's runtime for this case has risen above 1 second.

It is understandable that, in random cases, the AVL tree's runtime grows much faster than the BST's. After all, the AVL not only has to insert or delete the elements, it also has to maintain the tree's balance to ensure that each branch is, at most, 1 step different from the neighboring branches. This is also the case with reverse deletions, while inserting and deleting in order does not require as much effort on the AVL tree's part as the other cases. However, because of this trait, accessing elements in an AVL is much faster than it is in a BST. In a BST, there isn't an effort to curtail the heights of the different branches. So, depending on where the element in the BST was placed, finding said element can take a significant amount of time. In short, while the AVL can be more costly to build and take down, it generally performs better than the BST and will consistently perform better when it comes to accessing the information held.