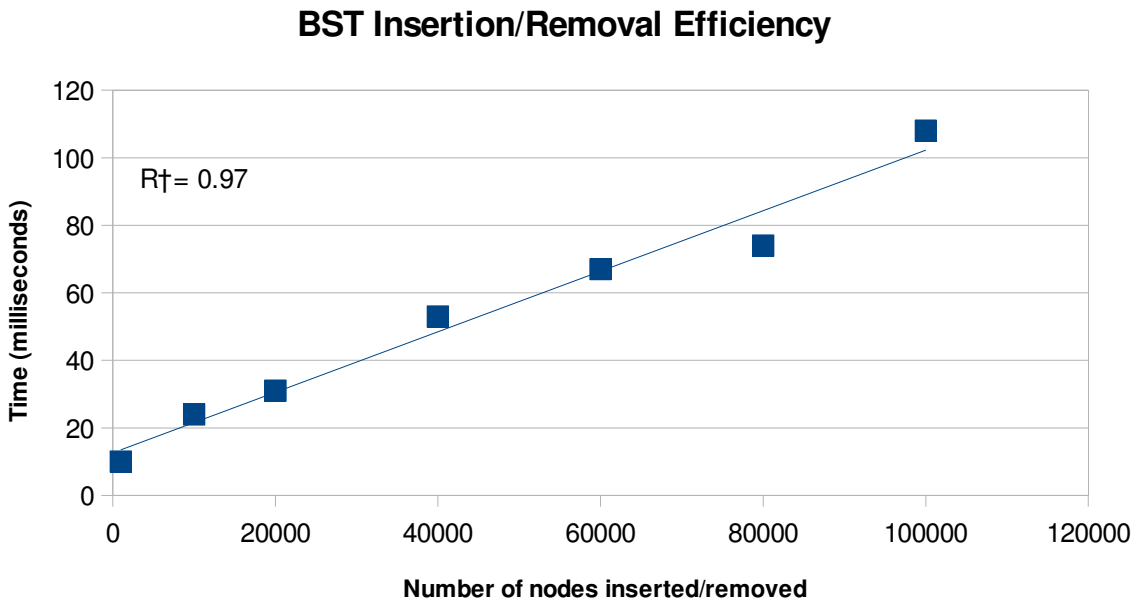
BST vs AVL Tree Efficiency

Ankit Gandhi & John Moon
CS146 – September 18, 2015

The first part of this project was a little tricky. The BST wasn't too difficult to implement using the algorithms given in class. The AVL tree was a little trickier... the rotation methods are fairly complicated. Debugging the AVL tree with the tester was interesting. We noted that with a certain rotation we were losing data and it seemed that the left subtree was moving to the right subtree. Sure enough, upon investigation, we found that we were using a setRight method instead of a setLeft method!

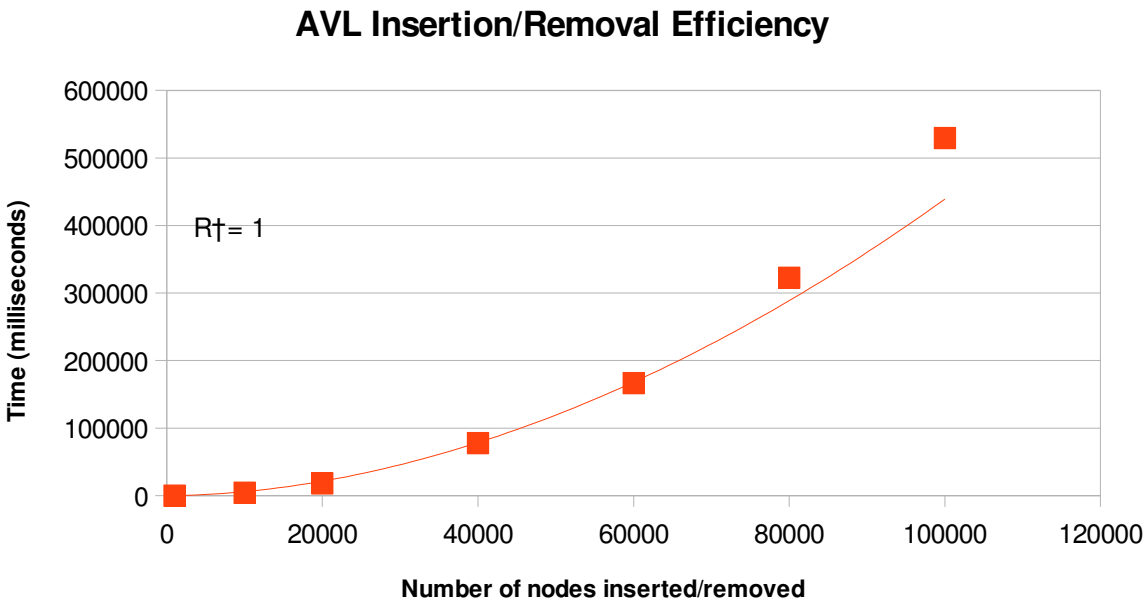
The second part of this project wasn't extremely difficult to implement, but it was a lengthy process to interpret the results of the data our testing program was outputting. Luckily our trees were solid enough to get some interesting efficiency results out of them!

When it comes to inserting and removing data, of course the BST did a better job. It didn't need to pull those rotation methods every time it became unbalanced. The results can be seen here:



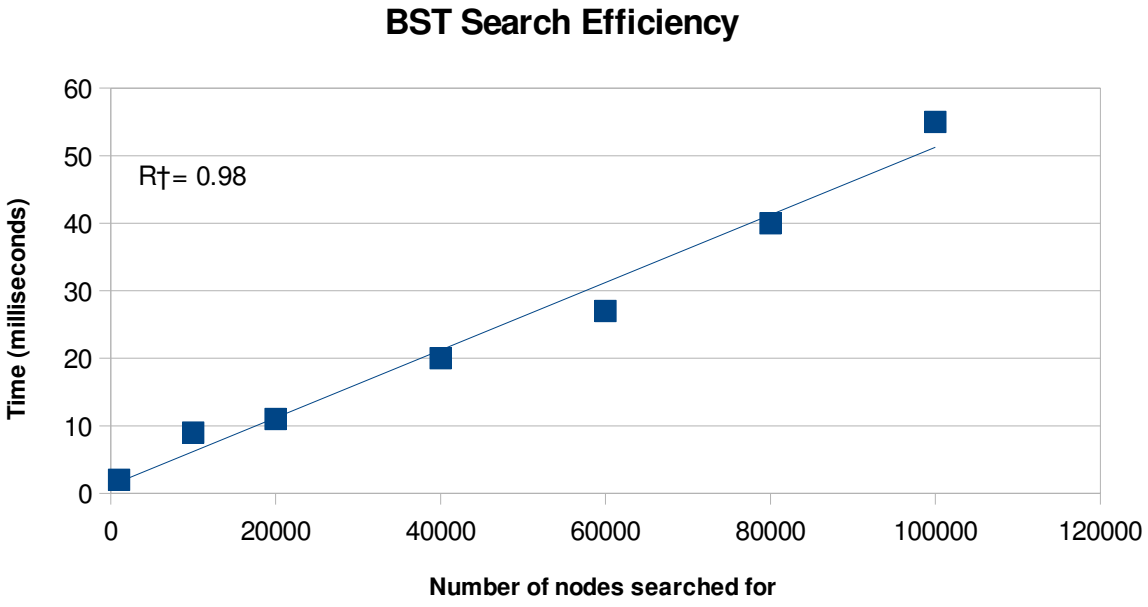
That's a linear trendline with very strong correlation coefficient of 0.97. This indicates that the insertion/removal efficiency is approximately O(n).

Now, here's what the AVL insertion/removal looked like:



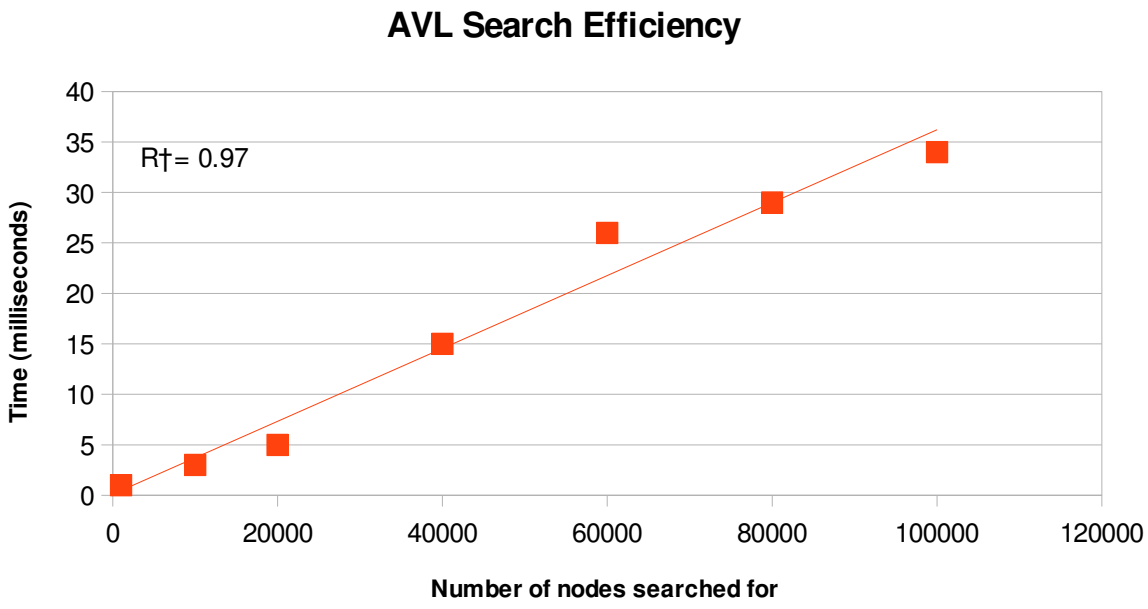
Amazingly, we actually get a correlation coeeficient of 1 with the AVL tree! This means that the efficiency of this process is about $O(n^2)$. You'll also notice that the time it took to complete gets insanely large compared to the BST.

Now, when we look at the BST search efficiency, we see about the same correlation as before (O(n)):



And with the AVL tree search efficiency, we see a less stable correlation... a few more data

points would be nice, but the AVL tree was inneficient enough that when N got above 100,000, it crawled:



So this also turned out to have about a linear efficiency (O(n)), though in general the searches on the AVL tree took less time than the BST.

So, the question was to empirically estimate the ratio where an AVL tree has better performance than a BST for a mixture of insertions and searches. Basically, since the AVL tree has such horrible performance on large N for insertions and removals, there has to be a small number of those... then it has great performance on large numbers of searches. So, essentially, the best scenario for an AVL tree is a small number of insertions and a large number of searches.