Project 2 is an exercise in the implementation and run-time analysis of BST (Binary Search Tree) and AVL Trees. As a short recap, BSTs are special trees with 2 conditions: nodes are ordered, and each node has at most 2 children. AVL trees are an extension of BSTs with the additional requirement that they are self-balancing; every insertion or removal requires a balance check.

Each **part of the program is run from a Driver\_part1 and Driver\_part2 respectively**. Much of the code for this project for BST, AVL construction and presentation has been adopted and modified from:

* my previous data structures and algorithms from Michael Loceff at Foothill College
* CS 146 material from Professor Ron Mak at San Jose State University

The corresponding algorithms from the implementation match those shown in the CS 146 Fall 2015 lectures. In addition, the key algorithms of balancing by rotation, insertion and removal have been verified to work by working the problems by hand with trees on the whiteboard.

For part 1 of the assignment, the BST and AVL tree are verified to work correctly. They are each instantiated, and nodes are inserted until about height 5. The results can be seen in **Appendix** A.

For part 2 of the assignment, performance of each tree is analyzed for different but very large n size values. Part 2.1 tests different numbers of insertions, 2.2 tests searching for k random integers, 2.3 analyzes the effects of both insertion and searching. The table below shows the results of the timings and the analysis. Red shows the raw times gathered from each run. Pink shows the calculations for Part 2.1, Light Green for 2.2, and Turquoise for 2.3 respectively.

1. Part 2.1 asks how does the growth of T\_BST(n) compare with T\_AVL(n)?

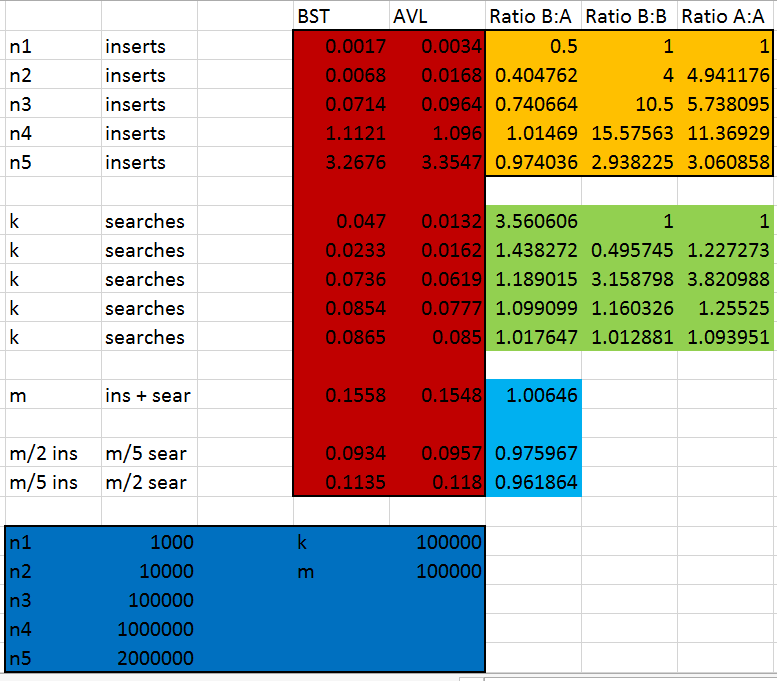
Firstly looking at the ratios of n\_i to n\_i-1, there is a **log(n) growth rate for both BST and AVL trees**, as a 10 factor increase of n leads to less than 10 times the growth. However, at the larger n numbers the ratio seems to increase. Comparatively, **BST run-times are less than AVL run-times**, but seem to even up at n4 = 1000000.

1. Part 2.2 asks to compare the growth rates of searching both n-node trees for the k random integers.

In this case, k is chosen to be 100000 random integers from 1 to 100000. As k is run through different trees of n-node size, the **growth rates for searching for increasing n shrinks, indicating log(n) growth for both BST and AVL trees.** Notably, except for small n it seems to occur at the same pace for both BST and AVL. Also, the **AVL run-time growth rate is smaller than for BST indicating that AVL is more efficient**.

1. Part 2.3 asks to perform random m insertions and searches on the n-node BST and AVL trees, and empirically estimate the ratio where AVL trees has better performance than BST.

In the interest of time, 3 tests were done with each using n5 = 2000000. For equal m insertions and deletions, performance is essentially identical (with a run-time ratio of 1.0065). The same seems to occur with mixing m/5 and m/2 insertions and searches on both trees. Testing with n3 = 100000 (not shown below) shows similar results. In conclusion however, **identical run-times for all cases is likely wrong** and probably reflects a limitation or bug in my code implementation. **For the average cases, AVL trees have O(log n) search and O(1) insert runtimes, while BST trees have O(log n) search and insert times** which should indicate results different than what I have. Expected behavior would indicate that **AVL trees should perform better on search** (as the tree is balanced O(log n) is the worst case, compared to O(n) in the BST worst case due to a heavily imbalanced tree). **BST trees should perform better on insertions, given that both trees are “relatively” balanced, as AVL self-balancing creates some overhead**. In all cases, a **correctly implemented AVL tree should always be better than BST**, as the worst cases are all O(log(n)) versus O(n) for BSTs.



**Appendix** A: Part 1 raw output

Formatted printed trees do not show up well in MS Word. See attached file *Project1\_part1\_output.txt*

**Appendix** B: Part 2 raw output

Elapsed Time for BST with n1 : 0.0017 seconds.

Elapsed Time for BST with n2 : 0.0068 seconds.

Elapsed Time for BST with n3 : 0.0714 seconds.

Elapsed Time for BST with n4 : 1.1121 seconds.

Elapsed Time for BST with n5 : 3.2676 seconds.

Elapsed Time for Avl with n1 : 0.0034 seconds.

Elapsed Time for Avl with n2 : 0.0168 seconds.

Elapsed Time for Avl with n3 : 0.0964 seconds.

Elapsed Time for Avl with n4 : 1.096 seconds.

Elapsed Time for Avl with n5 : 3.3547 seconds.

Elapsed Time for BST n1 tree with k searches : 0.047 seconds.

Elapsed Time for BST n2 tree with k searches : 0.0233 seconds.

Elapsed Time for BST n3 tree with k searches : 0.0736 seconds.

Elapsed Time for BST n4 tree with k searches : 0.0854 seconds.

Elapsed Time for BST n5 tree with k searches : 0.0865 seconds.

Elapsed Time for Avl n1 tree with k searches : 0.0132 seconds.

Elapsed Time for Avl n2 tree with k searches : 0.0162 seconds.

Elapsed Time for Avl n3 tree with k searches : 0.0619 seconds.

Elapsed Time for Avl n4 tree with k searches : 0.0777 seconds.

Elapsed Time for Avl n5 tree with k searches : 0.085 seconds.

Elapsed Time for BST with m insert and searches : 0.1558 seconds.

Elapsed Time for Avl with m insert and searches : 0.1548 seconds.

Elapsed Time for BST with m/2 insert and m/5 searches : 0.0934 seconds.

Elapsed Time for Avl with m/2 insert and m/5 searches : 0.0957 seconds.

Elapsed Time for BST with m/5 insert and m/2 searches : 0.1135 seconds.

Elapsed Time for Avl with m/5 insert and m/2 searches : 0.118 seconds.