Name: Jiwon Cha (jc4va)
Date: 10/12/2017
Lab section: 104

My testfile4.txt:
23 2198 12382 2498 2 4 8323 84 289 1238 12 3828 123 3616 4124 234 2345241 1234 6346 23452 12 64 626 321

Why the test file works:
My test file is a list of 24 numbers, and the average depths of nodes of AVL tree and binary search tree are clearly different; 2.41667 (AVL) and 4.04167(BST). Even though the order of the list is random, the cycle of the list is that the numbers get larger. Therefore, the right subtree of binary search tree would be bigger than the left tree. However, because AVL tree constantly balances the tree by rotating, AVL tree has less average depths.

Results

1. Testfile1.txt

	AVL TREE	**BINARY SEARCH TREE**
Initially	• 19 nodes created • 2 single rotations, 2 double rotations • Average node depth: 2.26316	• 19 nodes created • Average node depth: 3.15789
Operation: look up 'einstein'	• 2 left links followed • 2 right links followed	• 4 left links followed • 2 right links followed
Operation: look up 'we'	• 0 left links followed • 2 right links followed	• 0 left links followed • 0 right links followed
Operation: look up 'solve'	• 0 left links followed • 0 right links followed	• 1 left links followed • 1 right links followed

2. Testfile2.txt

	AVL TREE	BINARY SEARCH TREE
Initially	• 16 nodes created • 9 single rotations, 0 double rotations • Average node depth: 2.5	• 16 nodes created • Average node depth: 6.0625
Operation: look up 'flown'	• 2 left links followed • 2 right links followed	• 1 left links followed • 5 right links followed
Operation: look up 'higher'	• 0 left links followed • 0 right links followed	• 0 left links followed • 7 right links followed
Operation: look up 'ever'	• 3 left links followed • 1 right links followed	• 1 left links followed • 4 right links followed

3. Testfile3.txt

	AVL TREE	BINARY SEARCH TREE
Initially	• 13 nodes created • 1 single rotations, 2 double rotations • Average node depth: 2.23077	• 13 nodes created • Average node depth: 3.23077
Operation: look up 'landscape'	• 1 left links followed • 1 right links followed	• 2 left links followed • 1 right links followed
Operation: look up 'zany'	• 0 left links followed • 2 right links followed	• 0 left links followed • 0 right links followed
Operation: look up 'surreal'	• 1 left links followed • 3 right links followed	• 2 left links followed • 3 right links followed

4. Testfile4.txt

	AVL TREE	BINARY SEARCH TREE
Initially	• 24 nodes created • 6 single rotations, 5 double rotations • Average node depth: 2.41667	• 24 nodes created • Average node depth: 4.04167
Operation: look up '32'	• 2 left links followed • 1 right links followed	• 4 left links followed • 4 right links followed
Operation: look up '234524'	• 4 left links followed • 0 right links followed	• 4 left links followed • 0 right links followed
Operation: look up '2345'	• 2 left links followed • 2 right links followed	• 2 left links followed • 3 right links followed

When is AVL tree preferable to BST? What is the cost incurred in an AVL implementation?

AVL tree is preferable to binary search tree when the right subtree is much bigger than the left subtree in binary search tree, and vice versa. In the worst case scenario, binary search tree could result in a singly linked list; the time complexity of find() operation for such binary search tree is O(n). AVL tree was devised to address this problem by balancing itself whenever a new node is added. Testfile2 represents the situation where the input is a sentence with alphabetically ordered words, and therefore producing a singly linked list. Thanks to the rotation, the time complexity of AVL find() is O(log n). Therefore, AVL tree would be preferred when the input would produce a singly linked list or when the search operation is frequently used.

However, the insertion and removal of AVL tree are slower than binary search tree. In many cases, insertion requires rotations, and as the tree gets bigger, many rotations may be required for the tree to be balanced. Moreover, removal in AVL tree is O(log n) time, but it may still require the whole tree to be balanced again, which may significantly slow down the overall process. Therefore, when the overall process involves many insertions and deletions and when few search operations are used, AVL tree clearly has a disadvantage in terms of time complexity and memory. The fact that each node in AVL tree needs to keep track of its balance factors and rotate itself means that AVL tree takes up more memory than binary search tree.

To conclude, AVL has advantage over binary search tree because of its fast search time, but it has slower deletion and insertion time than binary search tree.