CSE 332 Data Abstractions, Winter 2013
Homework 3

Due: **Friday, February 1, 2013** at the BEGINNING of lecture. Your work should be readable as well as correct. You should refer to the written homework guidelines on the course website for a reminder about what is acceptable pseudocode. You will notice that this assignment has FOUR Fantastic questions due Friday February First!! Please write your **section** at the top of your homework.

Problem 1: AVL Insertion

Show the result of inserting 13, 8, 5, 9, 4, 6, 12, 2, 1 and 3 in that order into an initially empty AVL tree. Show the tree after each insertion, clearly labeling which tree is which.

Problem 2: Verifying AVL Trees

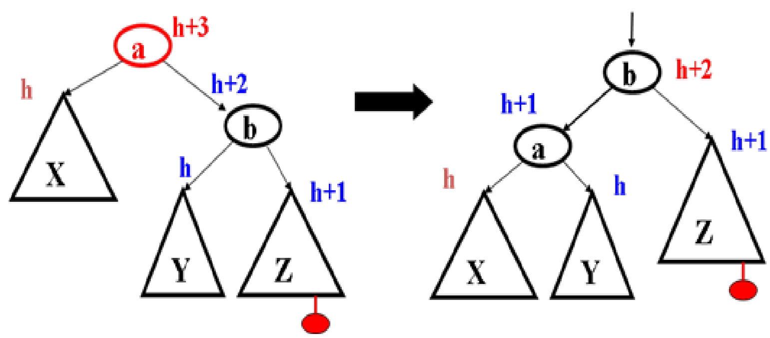
Give pseudocode for a linear-time algorithm that verifies that an AVL tree is correctly maintained. Assume every node has fields: key, data, height, left, and right and that keys can be compared with <, = =, and >. The algorithm should verify all of the following:

- The tree is a binary search tree.
- The height information stored in each node is correct.
- Every node is balanced.

Your code should throw an exception if one of these trees is violated; if the tree is a valid AVL tree, no exception should be thrown.

Problem 3: AVL Rotation Scenarios

The diagram below shows the general case for performing a case 4 (right-right) rotation; an insertion has taken place in subtree Z, and an imbalance has been detected at node 'a'. Assume subtree Z has height h before the insertion, and height h+1 after the insertion, as shown below. Argue that subtrees X & Y must also have height h prior to the insertion; that is, show that it is not possible for either to have height h-1 or h+1.



Problem 4: B-Tree Insertion

Show the result of inserting 28, 12, 17, 4, 31, 34, 8, 14 & 16 in that order into an initially empty B tree with M = 3 and L = 2. (Recall the text, lecture, and this problem call a B tree what many call a B+ tree.) Show the tree after each insertion, clearly labeling which tree is which. In an actual implementation, there is flexibility in how insertion overflow is handled. However, in this problem, follow these three guidelines:

- Always use splitting (not adoption).
- Split leaf nodes by keeping the smallest 2 elements in the original node and putting the 1 largest element in the new node.
- Split internal nodes by keeping the 2 children with the smaller values attached to the original node and attach the 2 children with the larger values to the new node.

Have fun!