CSC 2111 **Spring 2013**

Name(s): _____

Lab 14: Balanced Binary Search Trees (AVL-Trees)

Objective: To work more with binary search trees, particularly a well-known one called AVL trees, which takes advantage of a search tree that is balanced.

Files: *None*

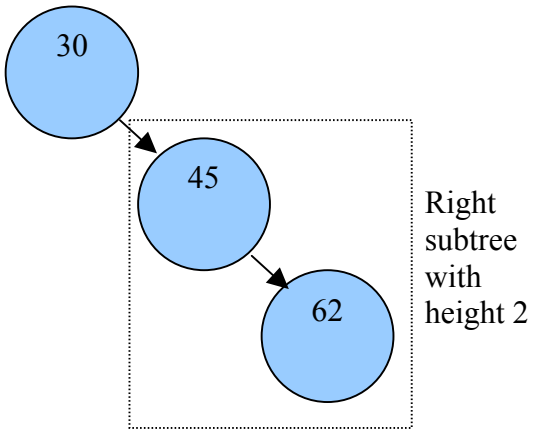
Introduction

We know that Binary Search Trees (BSTs) have the annoying property that when values are inserted into the tree in certain orders, the tree becomes imbalanced. Additionally, we cannot achieve perfect balance unless the tree has exactly 2n - 1 nodes (for some n). Fortunately, we know that search time is reduced (as is insertion and deletion time) if the BST is nearly balanced. In fact, if our tree is only off by a little bit, we still get searches, inserts, and deletes in O(logN) time.

The AVL tree data structure is a type of BST that will stay nearly balanced. That is, AVL trees will have the height of one subtree being greater than the height of the other subtree by no more than 1. The question is, "How do we build an AVL tree so the balance is at most off-by-one?"

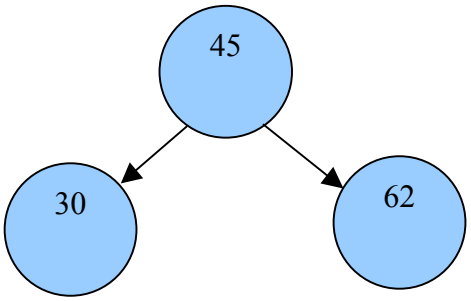
Defining AVL Trees

Note that the following tree is off-by-two.

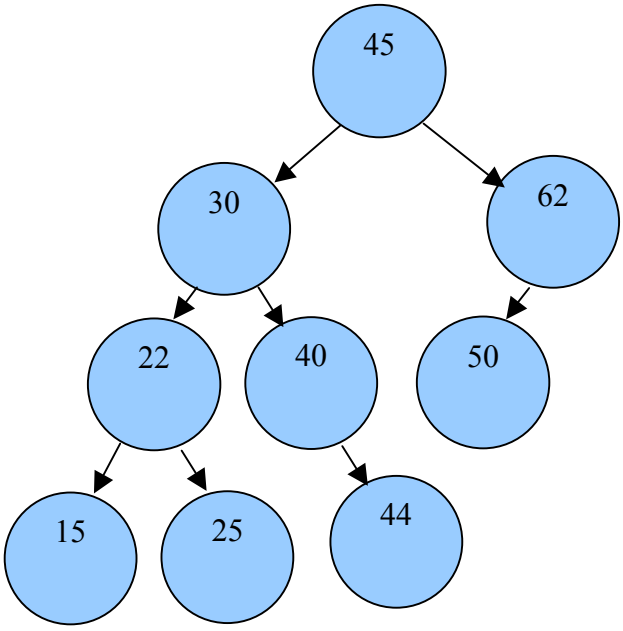


The height of the left subtree is 0 (by definition an empty tree has height 0) and the height of the right subtree is 2.

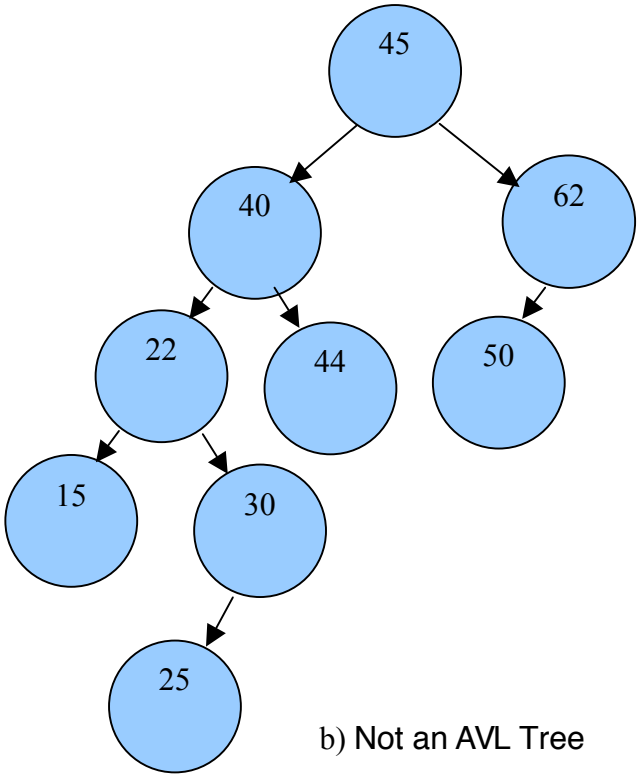
We can re-arrange the tree to bring it into balance:



Here is an example of two trees one of which is an AVL-tree and the other is not.



a) AVL Tree

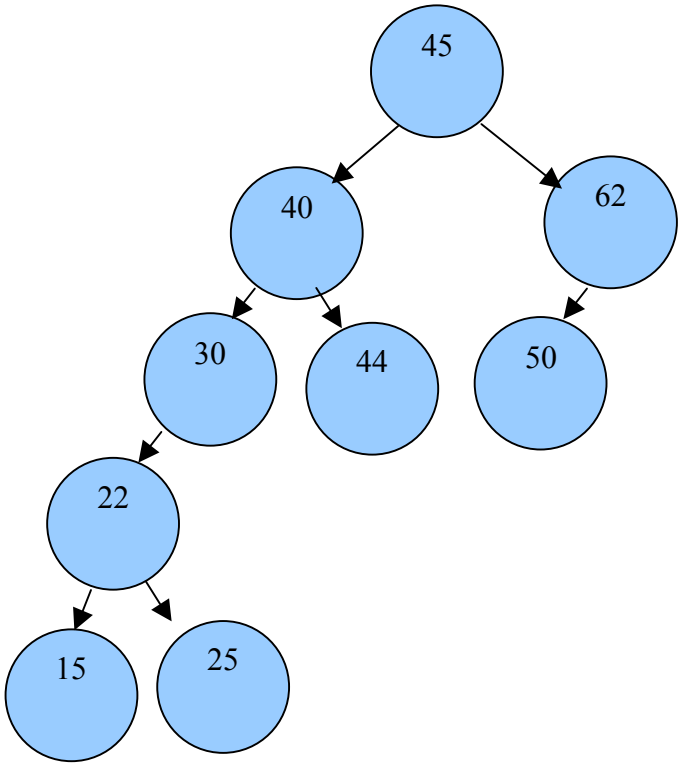


b) Not an AVL Tree

Why is the tree in (b) not an AVL-tree? <u>Be specific</u>:

Rebalancing

How can we rearrange (b) into an AVL-tree? We must apply a "left rotation about 22" followed by a "right rotation about 40." The tree in below is pictured below after a left rotation about 22.



After rotation about 22

A subsequent right rotation at 40 will balance the tree and change it into a) :