AVL Trees: Insertion

Computer Science Department
University of Central Florida

COP 3502 – Computer Science I

Motivation for AVL Trees

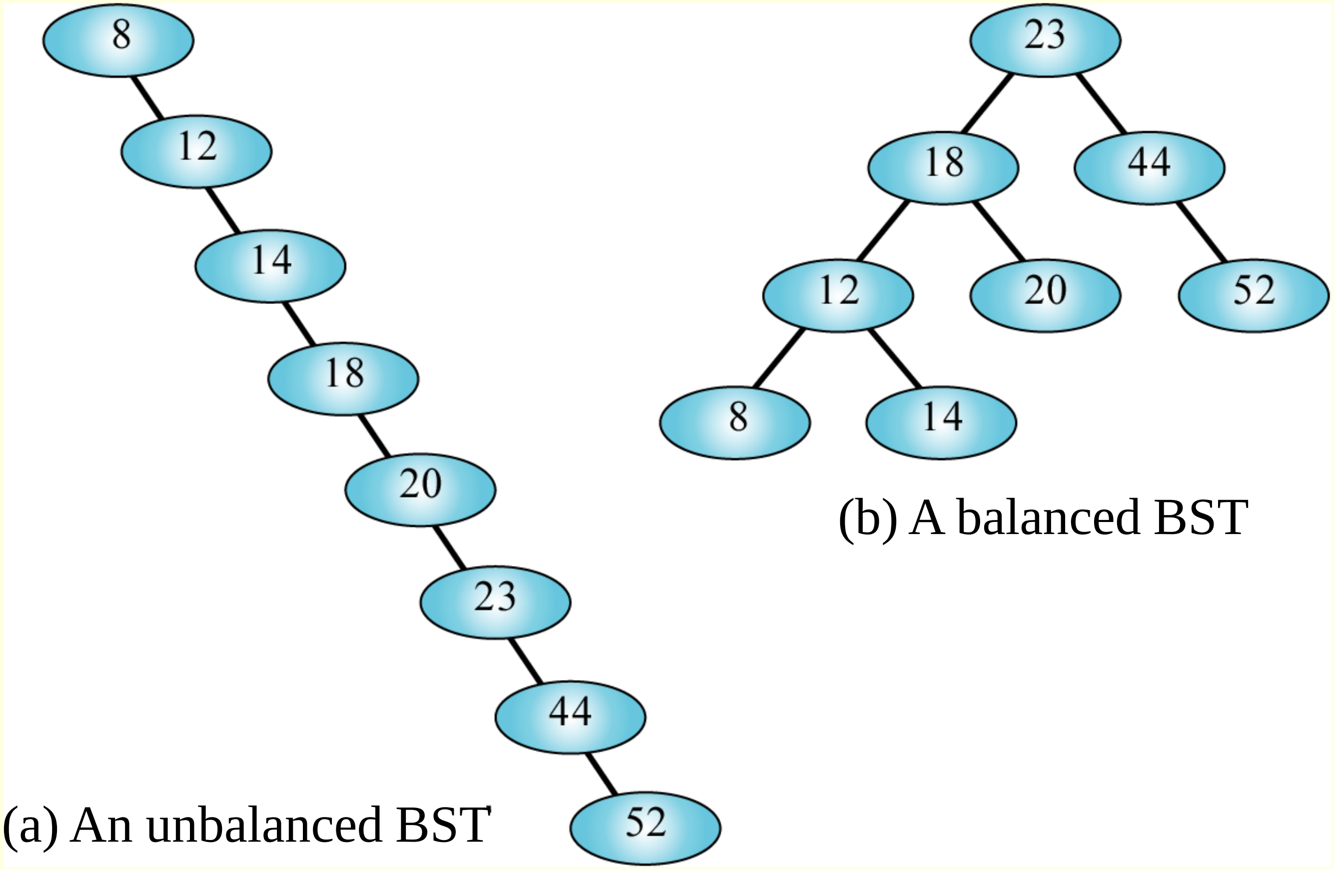
- Recall the basics of Binary Search Trees
  - The goal of a BST is to provide O(log n) lookup, insertion, deletion, etc.
  - However, this goal is only accomplished on a "complete" binary tree
    - a tree where all levels are filled with the possible exception of the last level, which is filled from left to right
    - Given a complete BST, the height of the tree is approximately log n, where n is the number of nodes
  - Remember:
    - If a BST is not complete, the height is NOT necessarily logn

Motivation for AVL Trees

- Recall the basics of Binary Search Trees
  - The height of a BST depends on the order of insertion
  - Example:
    - Inserting values 1, 2, 3, 4, 5, 6, and 7 into an initially empty BST results in what?
    - Each new values ends up going to the “right” of the previous value
    - So we end up with a completely right-skewed tree
    - This “tree” has degenerated into a linked list with respect to the running time of operations

Motivation for AVL Trees

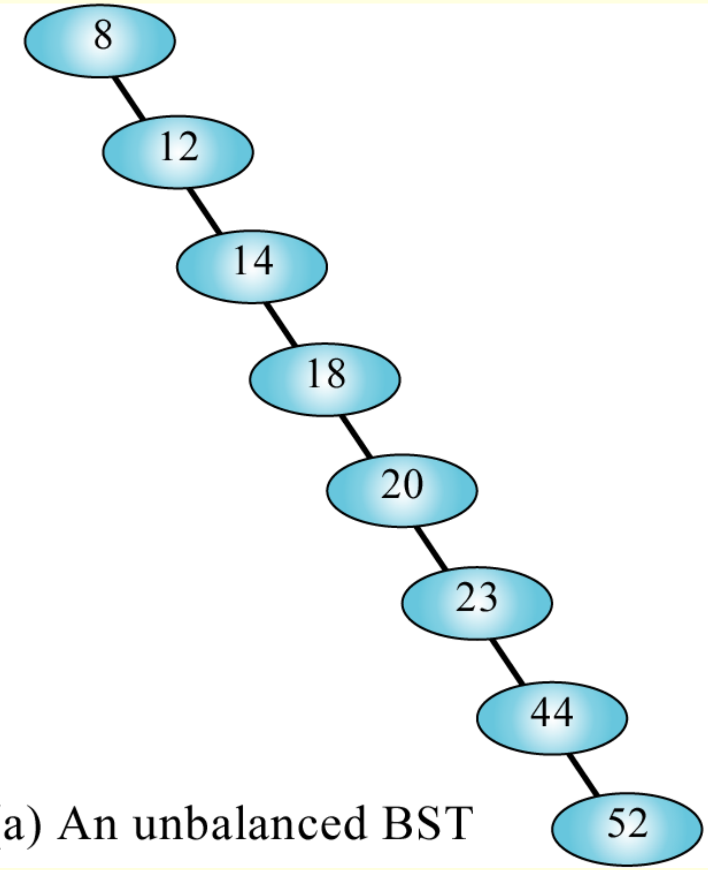
- Recall the basics of Binary Search Trees



(a) An unbalanced BST

(b) A balanced BST

Motivation for AVL Trees



(a) An unbalanced BST

- This "tree" is just a linked list in binary tree clothing.
- It takes 2 tests to locate 12, 3 to locate 14, and 8 to locate 52.
- Hence, the search effort for this binary tree is O(n).

Motivation for AVL Trees

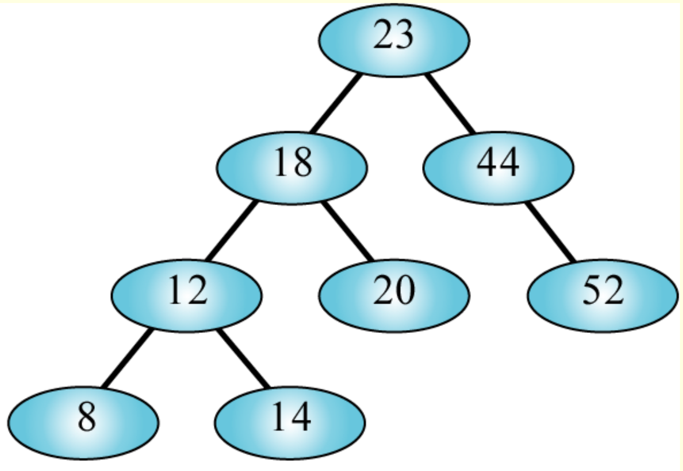
- Balanced BST
  - We want to maintain balance in our BSTs
  - Is there a way, regardless of the insertion order of elements, to maintain this balance?
    - To guarantee a height of log(n)?
  - Basically, can we keep this balance?
  - Short answer: yes!
  - AVL Trees:
    - G.M. Adelson-Velskii and E.M. Landis
    - Published their algorithm in 1962 in a paper entitled "An algorithm for the organization of information."

AVL Trees

- AVL Tree
  - Definition:
    - An AVL tree is a BST in which the heights of the subtrees, of any given node, differ by no more than 1
    - For EVERY node in a BST, you must check the height of the left and right subtree of that node
    - If the height of those subtrees differ by no more than 1, then that BST is an AVL tree
  - Thus, an AVL tree is a balanced BST

AVL Trees

- AVL Tree



(b) An AVL tree

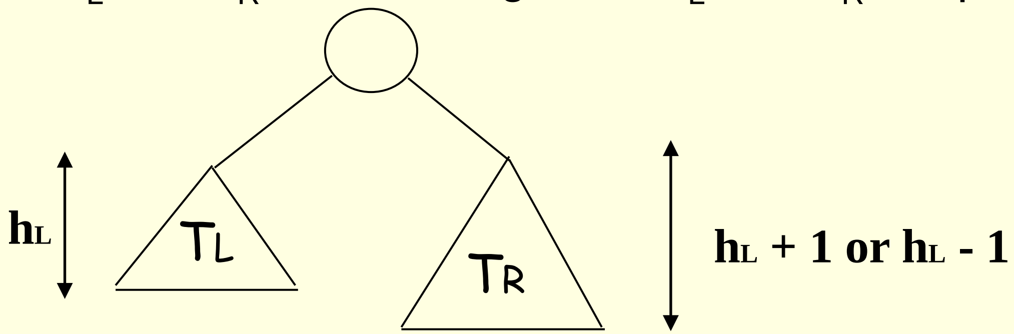
- This BST is an AVL tree.
- It takes 2 tests to locate 18, 3 to locate 12, and 4 to locate 8.
- Hence, the search effort for this binary tree is $O(\log_2 n)$.

AVL Trees

- AVL Tree
  - For a tree with 1000 nodes, the worst case for a completely unbalanced tree is 1000 tests.
    - Again, degenerating to a linked list
  - However, the worst case for a balanced tree is 10 tests.
    - HUUUUUGE difference
  - Hence, balancing a tree can lead to significant improvements.

AVL Trees

- AVL Trees: Formal Definition
  1) All empty trees are also, by definition, AVL trees
  2) If T is a non-empty BST with T_L and T_R as its left and right subtrees, respectively, then T is an AVL tree if and only if:
     1) T_L and T_R are also AVL trees
     2) $|h_L – h_R| <= 1$
        - where h_L and h_R are the heights of T_L and T_R, respectively



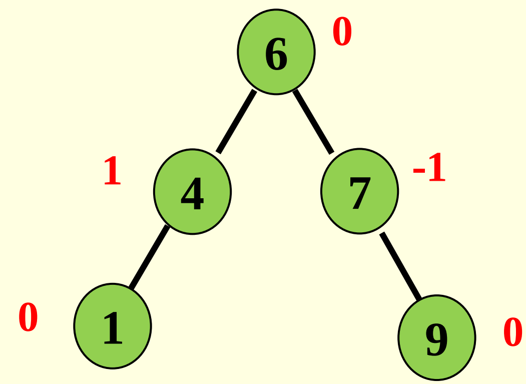
AVL Trees

#hL for node containing 1 = -1
hR for node containing 1 = -1
So, BF for node containing 1 = -1 - (-1) = 0

#hL for node containing 7 = -1
hL for node containing 7 = 0
So, BF = -1-0 = -1

- AVL Tree
  - AVL trees are height-balanced BSTs
  - All nodes in an AVL tree have a Balance Factor (BF)
  - Balance factor of a node = height of the left subtree minus the height of the right subtree
    - BF = hL – hR
    - or BF = hR – hL
  - An AVL tree can have only balance factors of -1, 0, or 1 at every node
  - For every node in a BST, the height of the left and right subtrees can differ by no more than 1

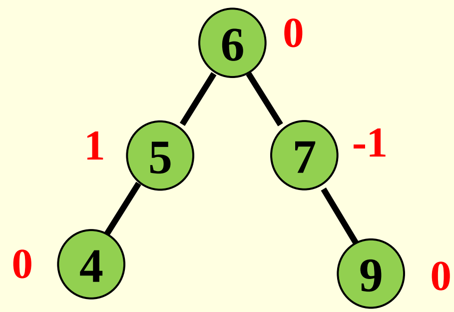
An AVL Tree



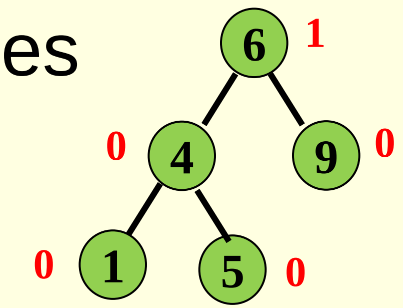
Red numbers are Balance Factors

AVL Trees

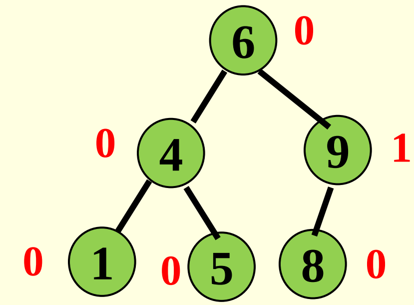
- AVL Trees: Examples



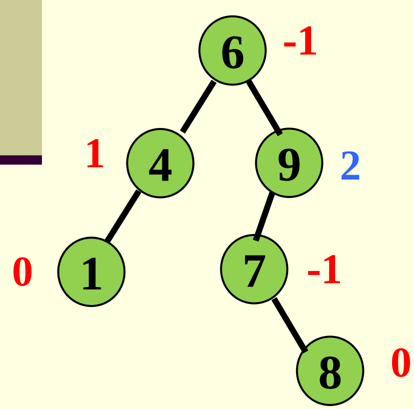
An AVL Tree



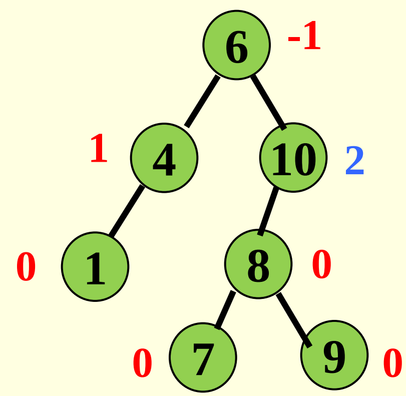
An AVL Tree



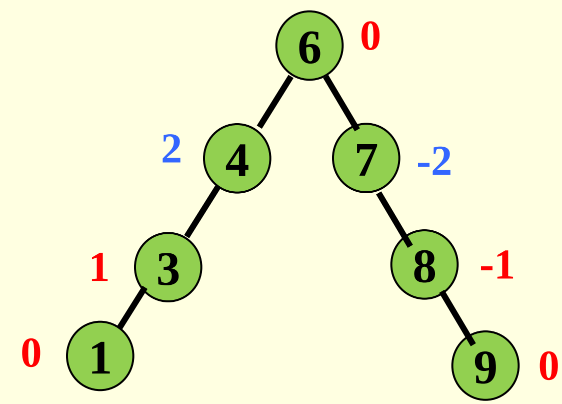
An AVL Tree



Non-AVL Tree



Non-AVL Tree

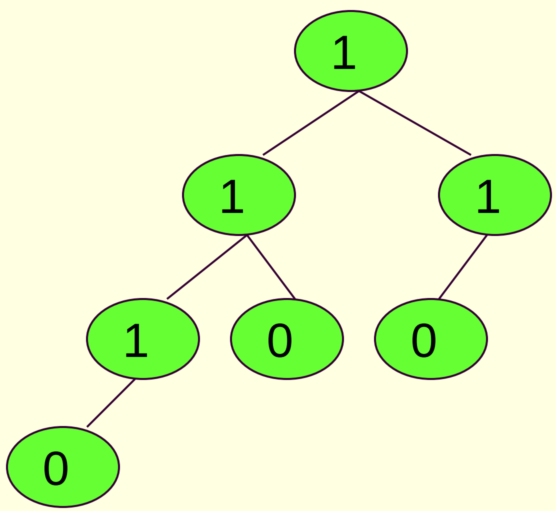


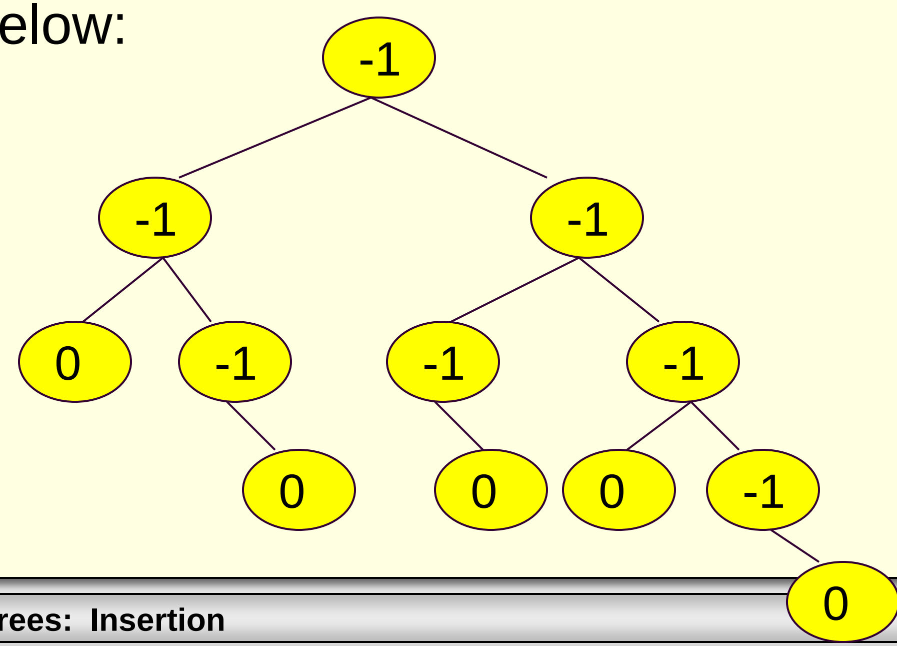
Non-AVL Tree

Red numbers are Balance Factors

AVL Trees

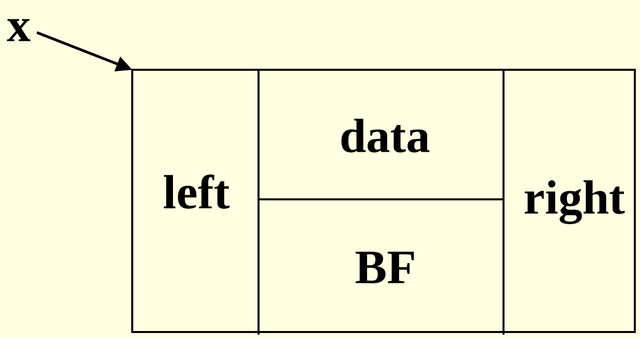
- Skewed AVL Trees
  - Notice that the definition of an AVL tree does NOT require that all leaf nodes be on the same level or even adjacent levels
    - As such, it is possible to construct AVL trees that are quite skewed as shown below:





AVL Trees

- AVL Trees: Implementation
  - To implement an AVL tree, simply associated a BF with each node, “x”

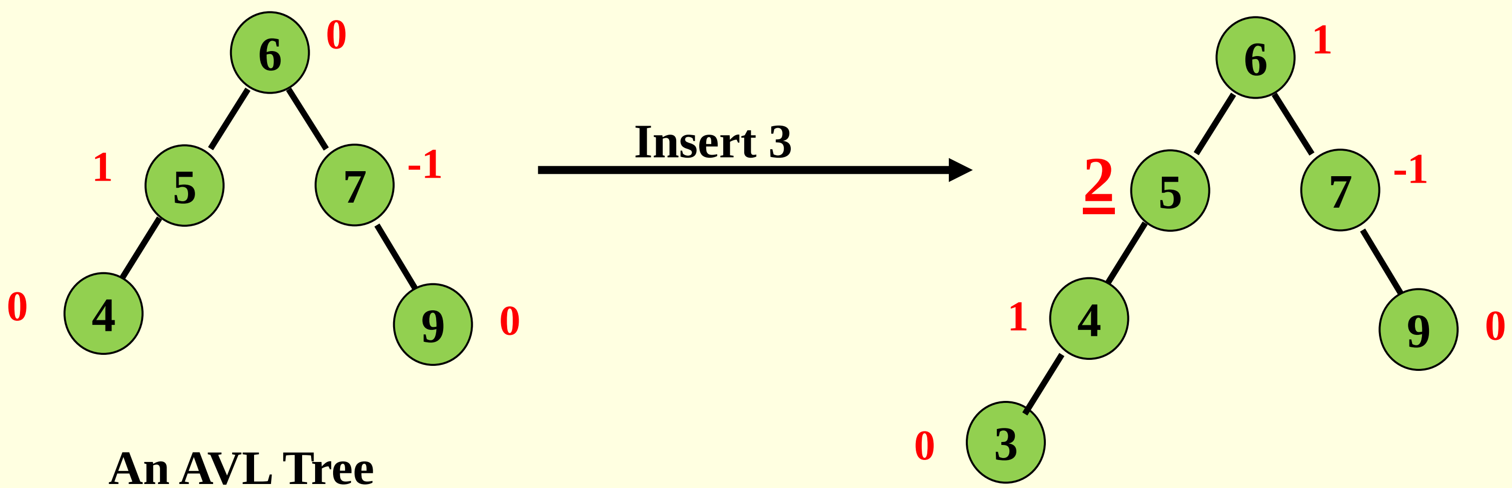



```
struct AVLTreeNode{
        int data;
        int BF;
        struct AVLTreeNode *left;
        struct AVLTreeNode *right;
};
```

  - $x\text{->}bf = h_L - h_R$
  - Again, in an AVL-tree, BF can be one of {-1, 0, 1}

AVL Trees

- AVL Trees: Good News & Bad News
  - Good News
    - Search is O(log n) = O(height)
  - Bad News
    - Insert and delete may cause the tree to be unbalanced



An AVL Tree

No longer an AVL Tree

AVL Trees

- Insertion into an AVL Tree
  - Insertion into an AVL tree is just like inserting into a standard BST
    - You simply do a search, going left or right at every step, in the tree until you find the correct leaf node
    - You then insert in either the left or right child of that node
  - Once the new node is inserted, the balance MUST be checked and restored if the tree has become unbalanced
    - It often turns out that the new node can be inserted without affecting the height of the subtree
      - If this happens, then the balance of the root will not change

AVL Trees

- Insertion into an AVL Tree
  - Once the new node is inserted, the balance MUST be checked and restored if the tree has become unbalanced
    - Even if the insertion caused one of the subtrees to increase in height, it may be that the shorter of the subtrees changed in height.
      - So only the balance factor of the root will change
  - The only case that causes difficulty:
    - Inserting a new node into a subtree of the root, which is taller than the other subtree, and the height of the taller subtree increases
    - So one subtree will have a height 2 more than the other

AVL Trees

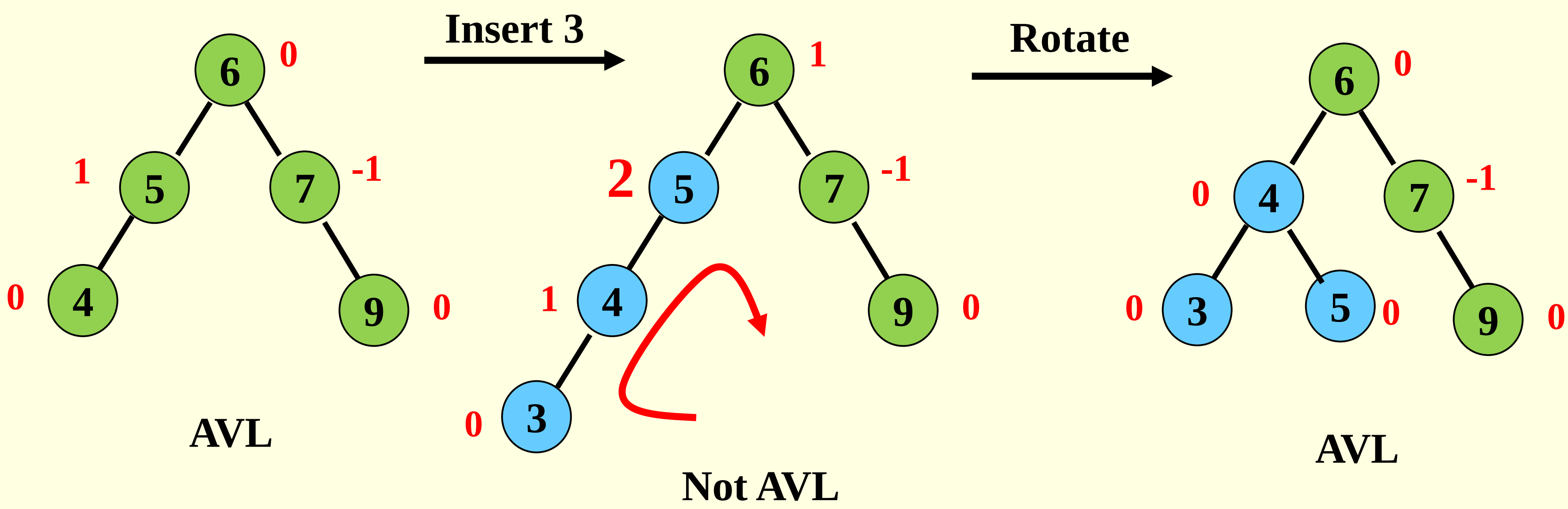
- Insertion into an AVL Tree
  - Thus, an AVL tree can become unbalanced due to an insertion in one of four ways:
    - (two of which are symmetric to the others)
    1) Inserting a new node into the right subtree of a right child
    2) Inserting a new node into the left subtree of a left child
       - This is the symmetric case
    3) Inserting a new node into the left subtree of a right child
    4) Inserting a new node into the right subtree of a left child
       - This is the symmetric case
    - The first two cases are easier to handle (as the require only one rotation), so we will go over them first

AVL Trees

- Restoring Balance in an AVL Tree
  - Problem
    - Inserting a new node may cause the BF of some node, on the path from the root to the insertion point, to become 2 or -2
  - Solution:
    - First insert the node following typical rules of a BST
    - Then, from that insertion point, **BACK UP towards the root**, updating the BFs of all nodes along the path to root
    - If a node ends up with a BF of 2 or -2, you must adjust the tree by rotating around deepest such node

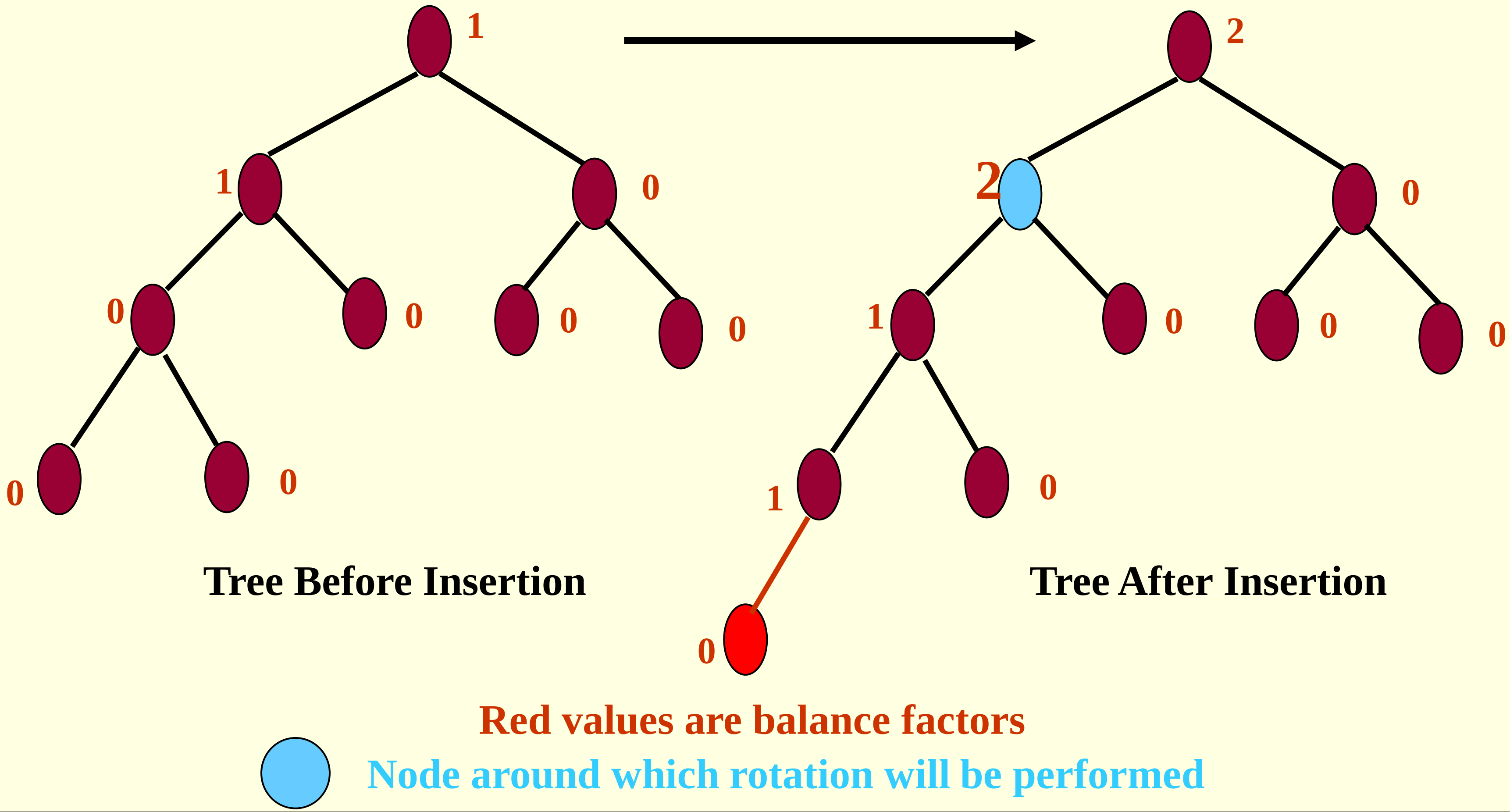
AVL Trees

- Restoring Balance in an AVL Tree



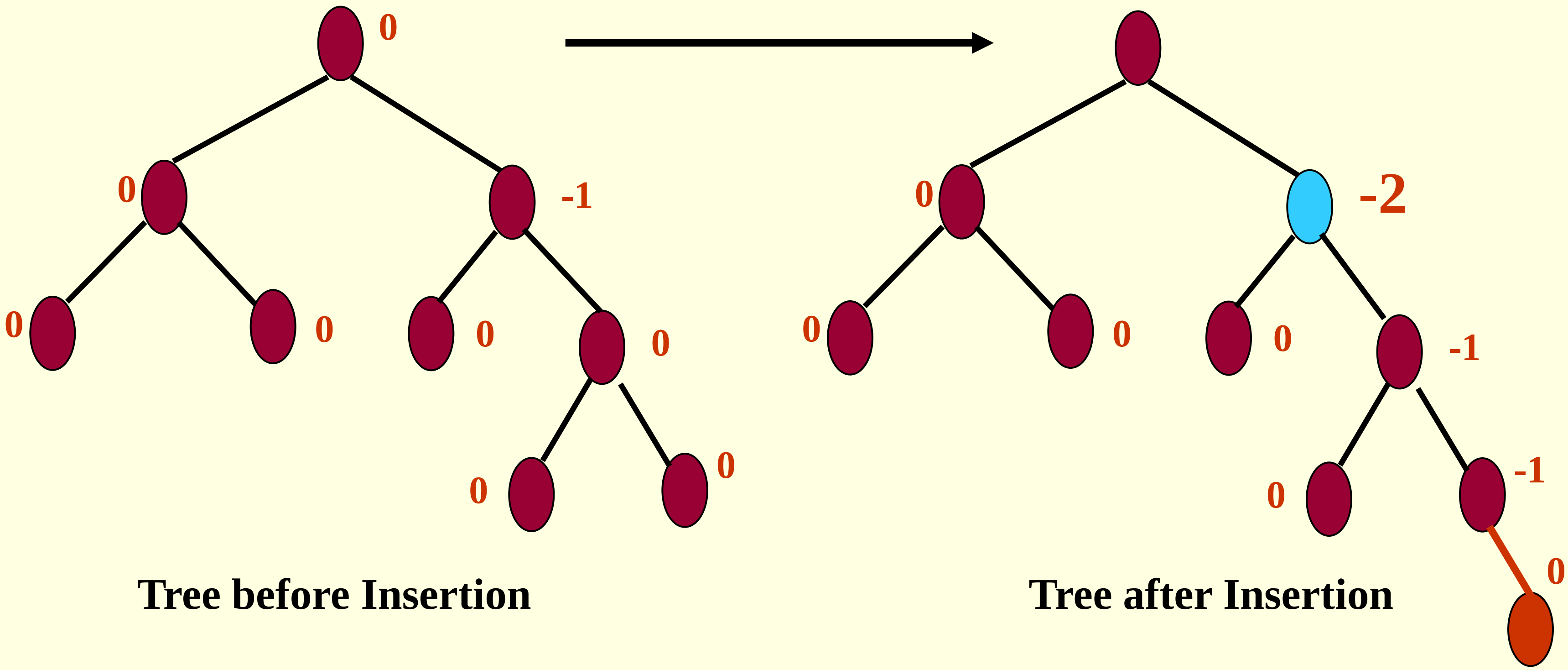
AVL Trees

- Four Cases of Imbalance: LL Imbalance

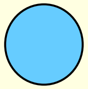


AVL Trees

- Four Cases of Imbalance: RR Imbalance



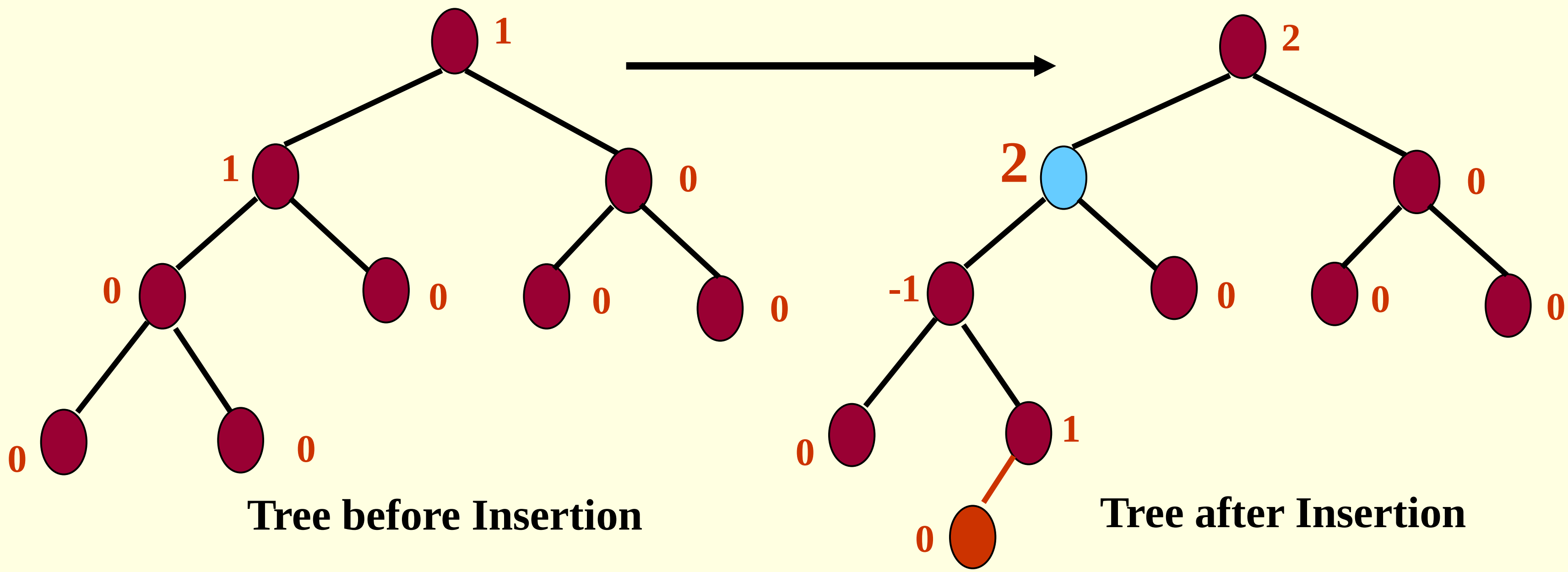
Red values are balance factors



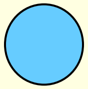
Node around which rotation will be performed

AVL Trees

- Four Cases of Imbalance: LR Imbalance



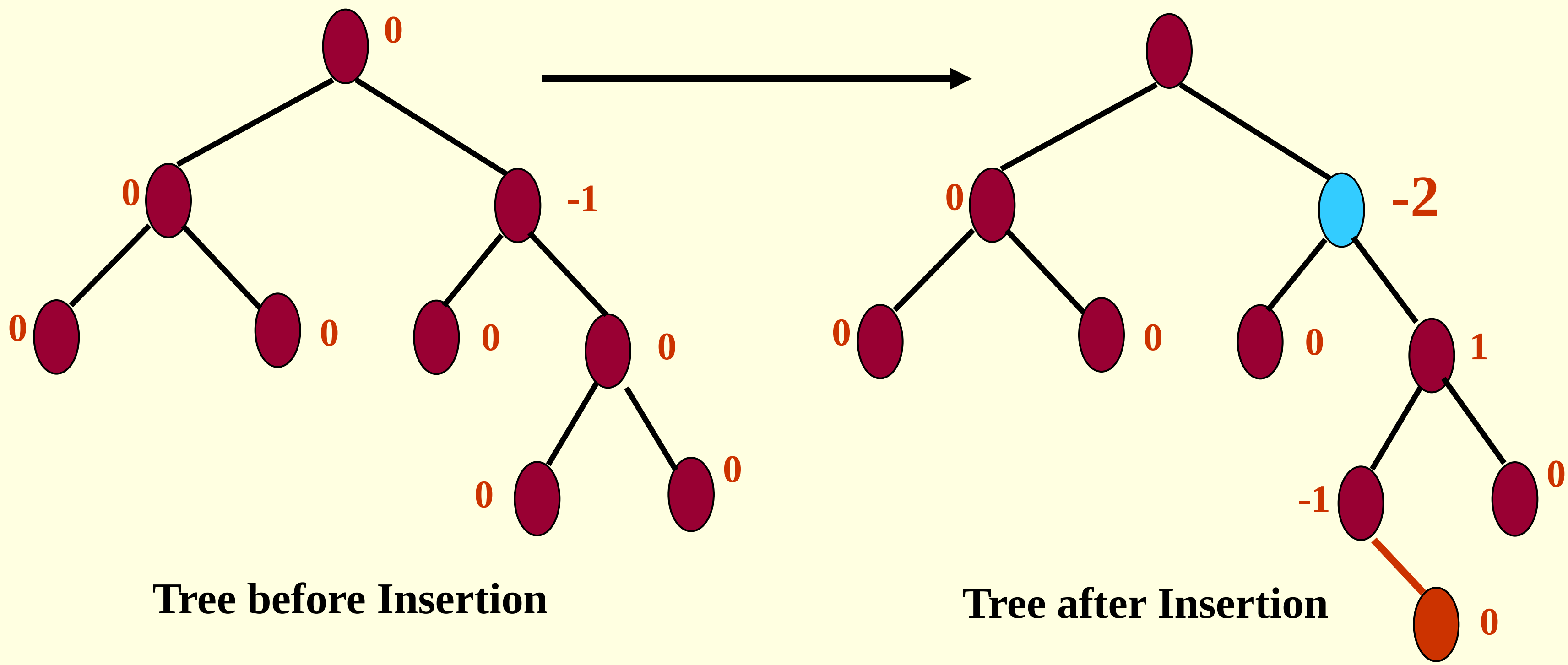
Red values are balance factors



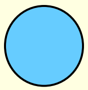
Node around which rotation will be performed

AVL Trees

- Four Cases of Imbalance: RL Imbalance



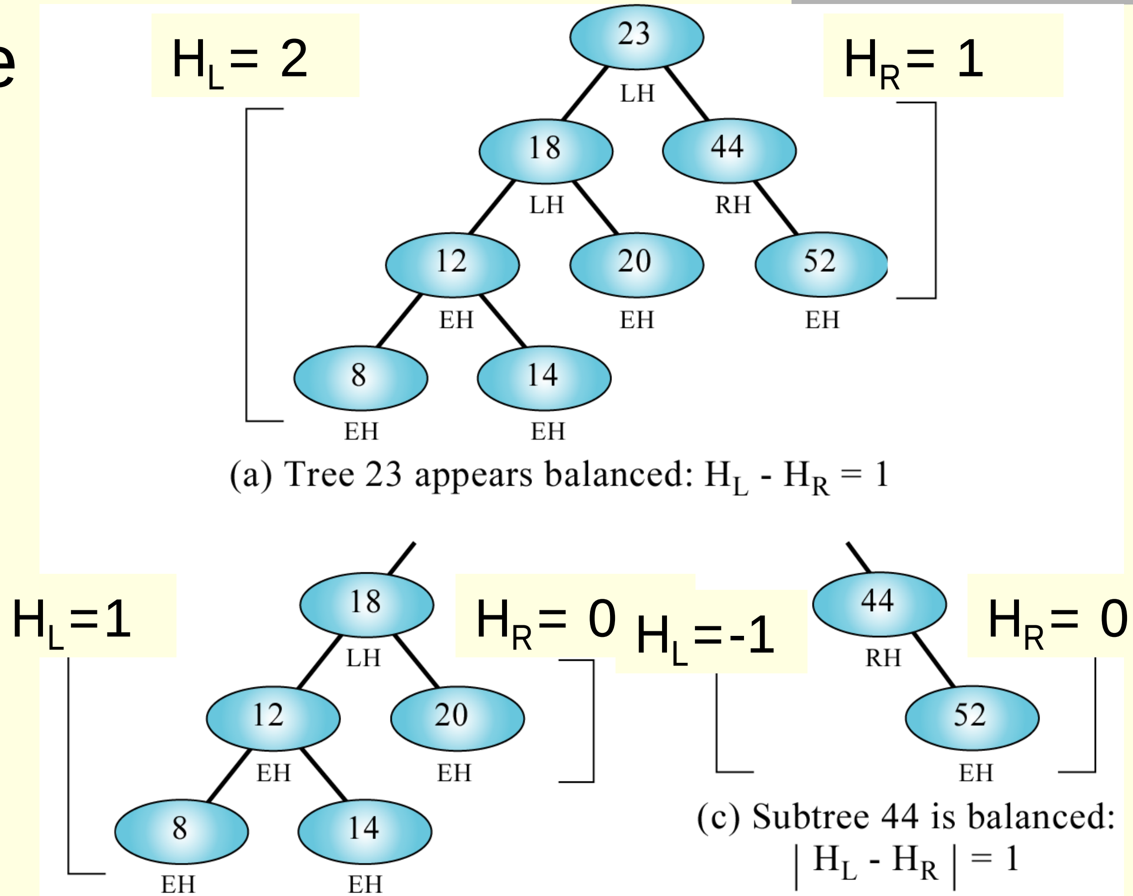
Red values are balance factors



Node around which rotation will be performed

AVL Trees

- AVL Balance Factor:



(a) Tree 23 appears balanced: $H_L - H_R = 1$

(b) Subtree 18 appears balanced: $H_L - H_R = 1$

(c) Subtree 44 is balanced: $|H_L - H_R| = 1$

- We will learn more about Insertion technique in another pdf/ppt file.
- To align with the next pdf, some slides are removed from the original pdf file

AVL Trees

- Balancing AVL Trees:
  - Whenever we insert a node into a tree or delete a node from a tree, the resulting tree may become unbalanced.
  - When we detect that a tree has become unbalanced, we must rebalance it.
  - AVL trees are balanced by rotating nodes either to the left or to the right.

AVL Trees

- Insertion into AVL Trees (Summary)
  - We insert following standard rules of a BST
  - Then we trace back up to the root of the tree
  - As we back out of the tree, constantly check the balance factor of each node
  - When a node is out of balance, we balance it and continue backing up out of the tree
  - Note:
    - Not all inserts will produce an out of balance tree

AVL Trees

- Summary of AVL Trees:
  - Arguments for using AVL trees:
    1) Search/insertion/deletion is **O(log N)** since AVL trees are **always balanced**.
    2) The height balancing adds no more than a constant factor to the speed of insertion.
  - Arguments against using AVL trees:
    1) Requires extra space for balancing factor
    2) It may be OK to have a partially balanced tree that would give performance similar to AVL trees without requiring the balancing factor
       - Splay trees (something we won't be covering in CS1)

AVL Trees: Insertion

Computer Science Department
University of Central Florida

COP 3502 – Computer Science I