6 - AVL Trees

Joseph Afework
CS 241

Dept. of Computer Science
California Polytechnic State University, Pomona, CA

Agenda

- Intro
- AVL Tree Rules
- AVL Node Rotations
- AVL Operations
- Overhead
- Examples

Reading Assignment

- Read Chapter 27 - Balanced Search Trees
  - Chapter 27 (Read about: **AVL**, Red-Black Trees, B-Trees)

AVL Tree Rules

- **AVL Tree:**
  - A type of binary search tree (remember BST rules/restrictions)
  - Every node (in addition to having a key value) has a value called the **balance factor** for the node.
  - For the tree to remain **balanced,** the **balance factor** for any given node may **NOT** exceed <u>**1**</u>.

Balance factor = height(node.right) - height(node.left))

Balanced vs Unbalanced

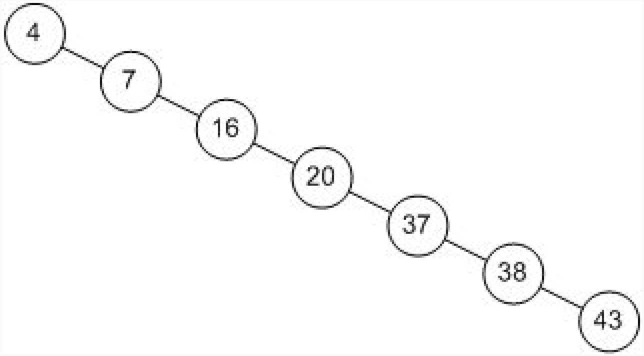
- An AVL tree is said to be **unbalanced** if the balance factor for the node is greater than 1.
- In a balanced tree AVL tree, every node has a balance factor of :
  - (**-1, 0 or 1**).

- **Remember**:
  - The balance factor for EVERY node must follow this rule

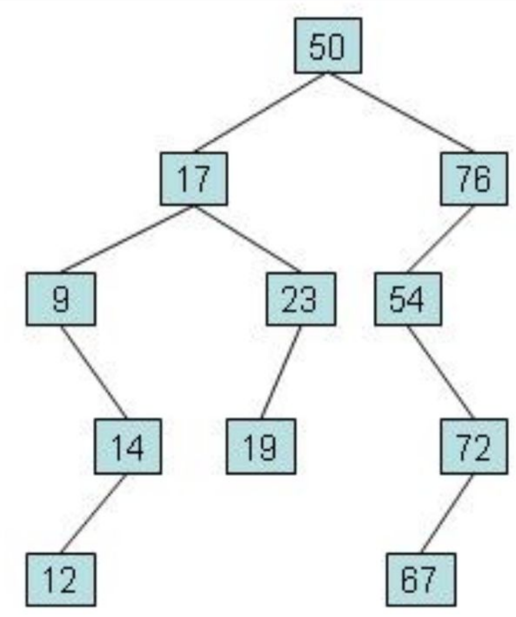
Remember

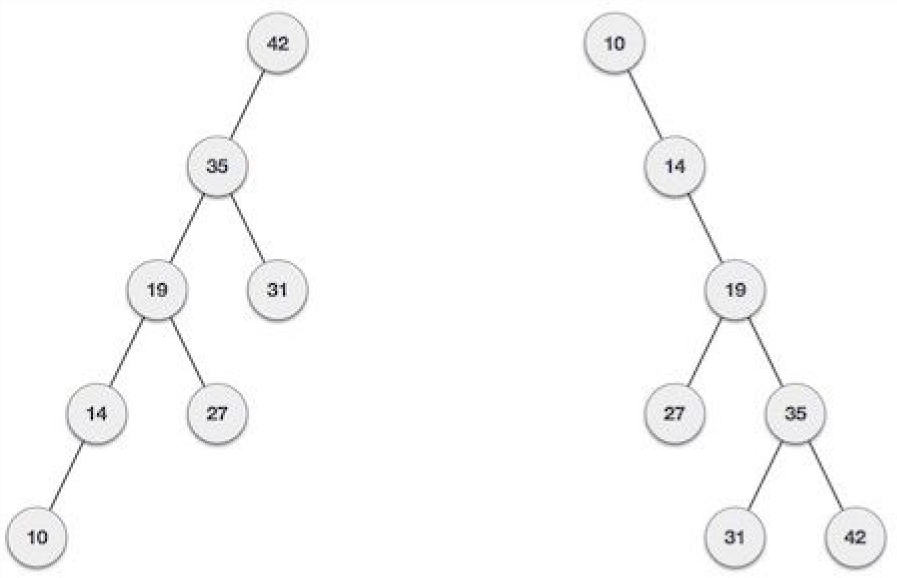
Worst Representation for a Binary Search Tree:

- resembles a linked lists....

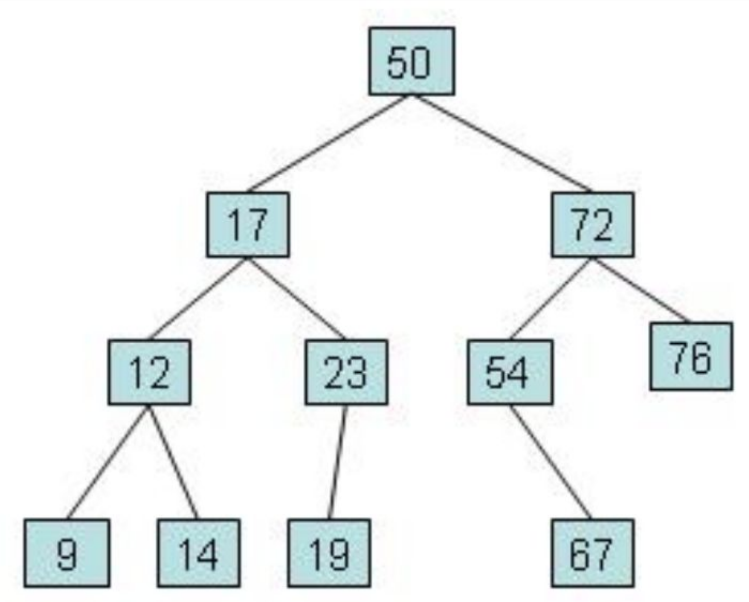


Unbalanced Trees



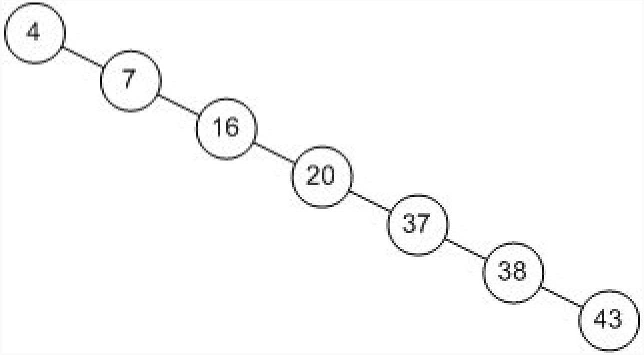


Balanced Tree



Unbalanced Trees

- An AVL tree can become unbalanced during:
  - Insertion (adding a node)
  - Deletion (removing a node)
- The rules for a Binary Search Tree do NOT prevent the creation of an unbalanced tree.
  - Ex. (Inserting a series of ordered terms)



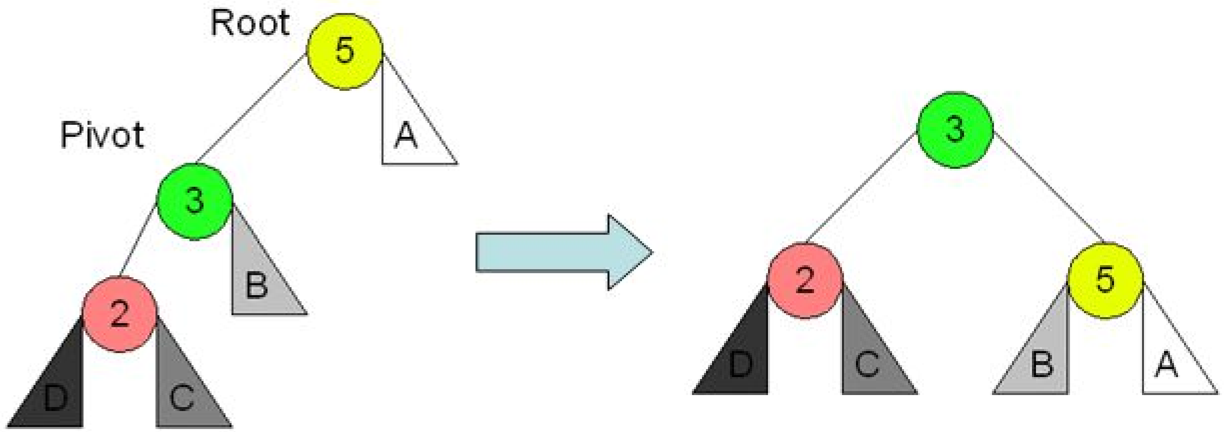
AVL Process

1. Perform an operation: (according to BST rules)... Insert or Delete node
2. Traverse from the node of the operation (inserted node, or deleted node), and compute the balance factor for each node moving upwards to the root of the tree.
3. When you encounter the first violation of the balance factor rule, **perform a rotation operation** to rebalance that node.
4. Once the node is rebalanced (passes balance factor rule), continue moving upwards until the root node is reached.
5. Repeat from Step 2 until root node passes balance factor rule.

Rotation Operations

- 4 types of rotation operations:
  - Single Rotations:
    - Left-Left Case = Right Rotation
    - Right-Right Case = Left Rotation
  - Double Rotations:
    - Left-Right Case = Left Rotation + Right Rotation
    - Right-Left Case = Right Rotation + Left Rotation

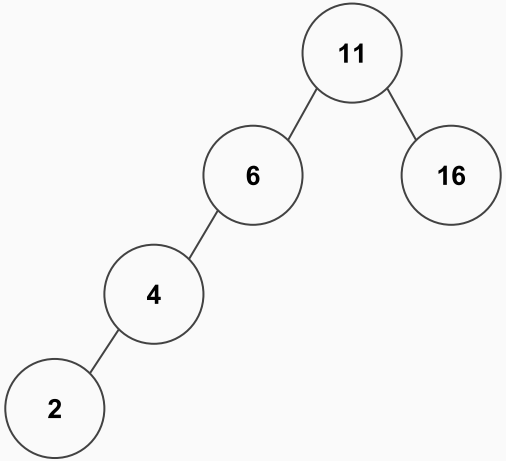
Left-Left Case



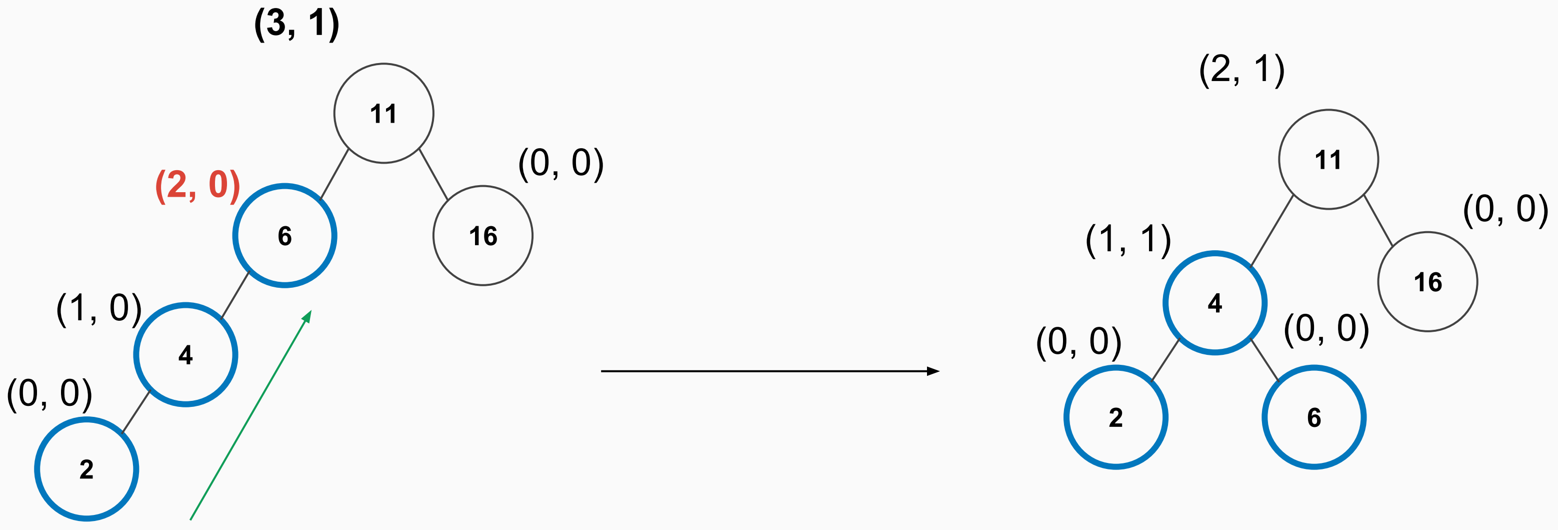
Left-Left Case: Right Rotation

ICE 6.1 Left-Left

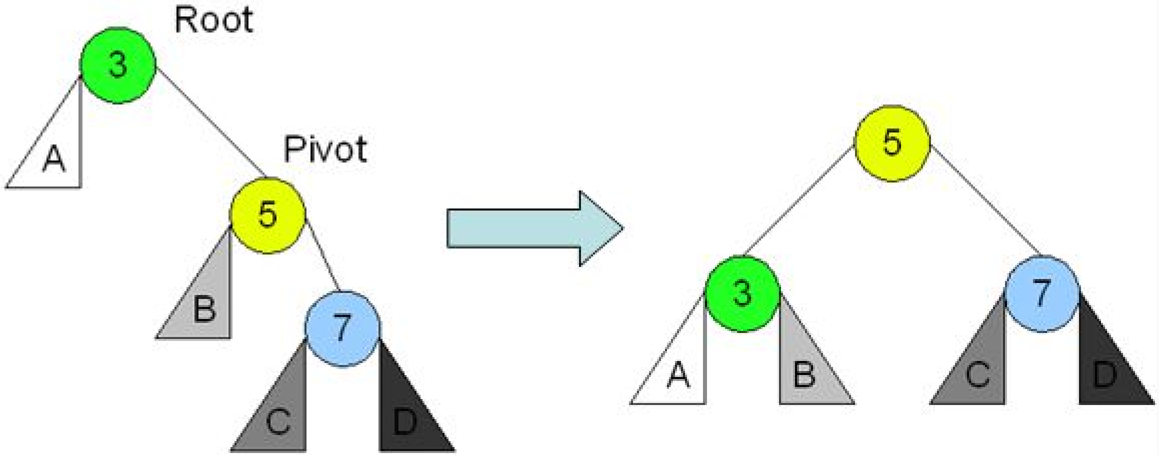
Perform a Left-Left Rotation:



Left-Left Example



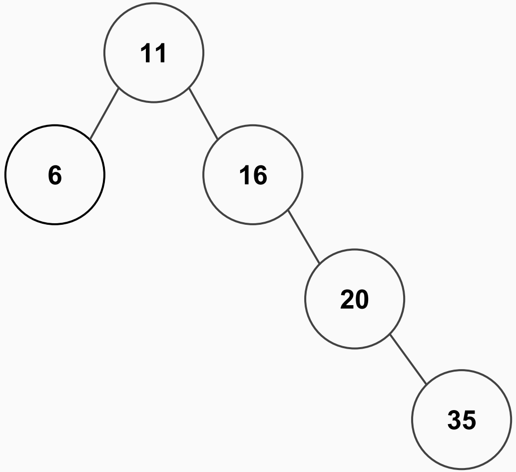
Right-Right Case



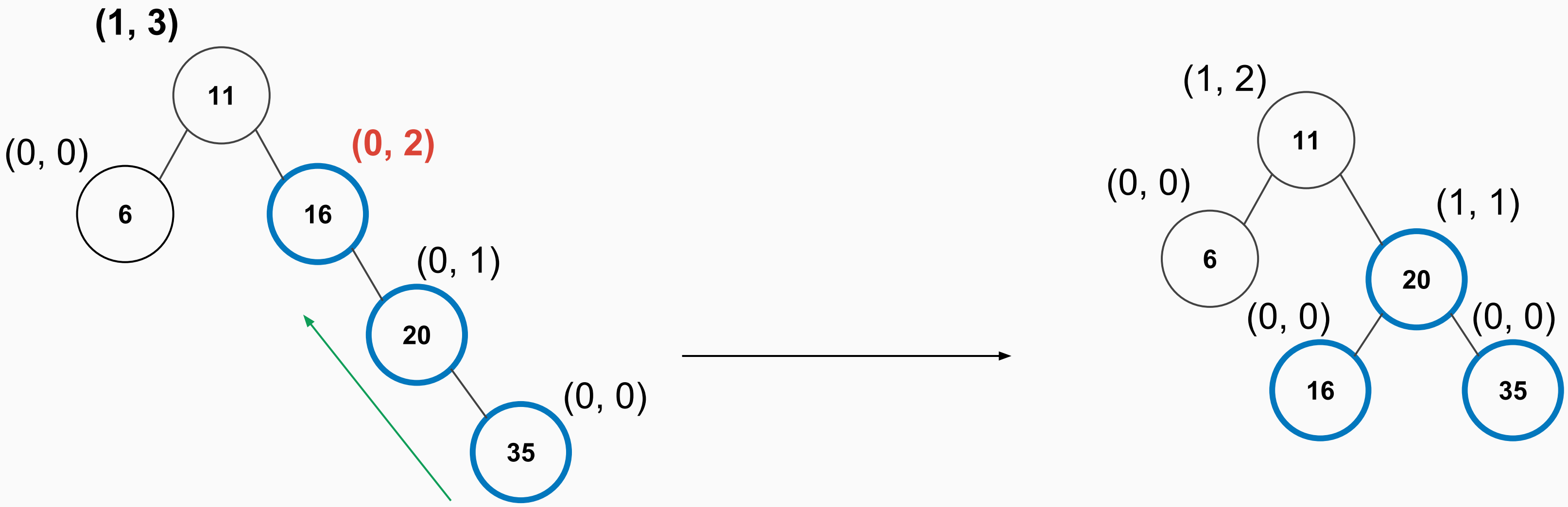
Right-Right Case: Left Rotation

ICE 6.2 Right-Right

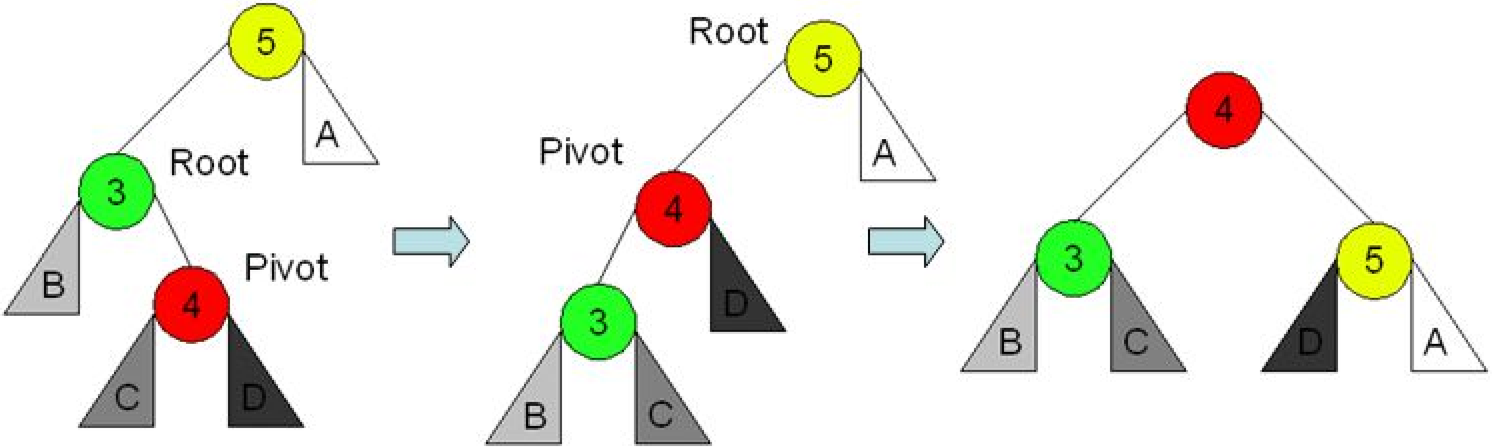
Perform a Right-Right Rotation:



Right-Right Case



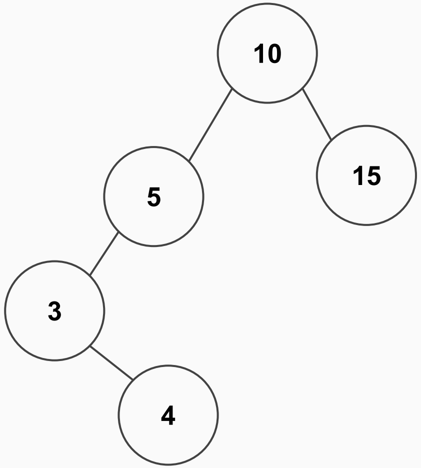
Left-Right Case



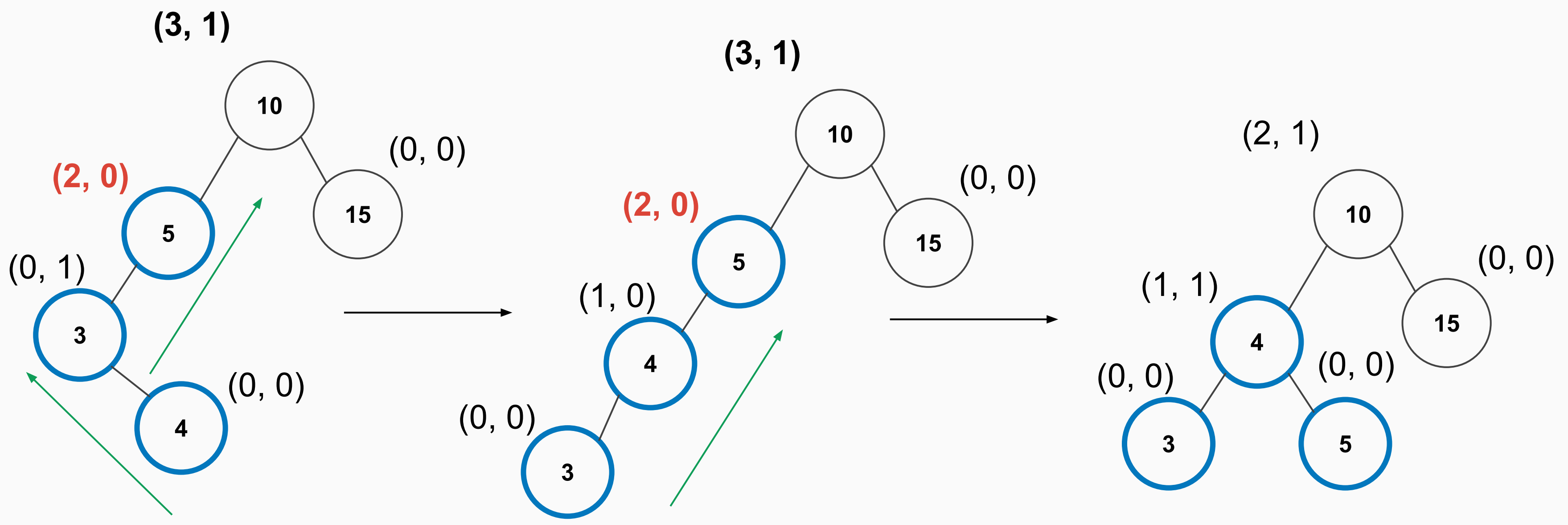
Left-Right Case: Left-Rotation and then Right-Rotation

ICE 6.3 Left-Right

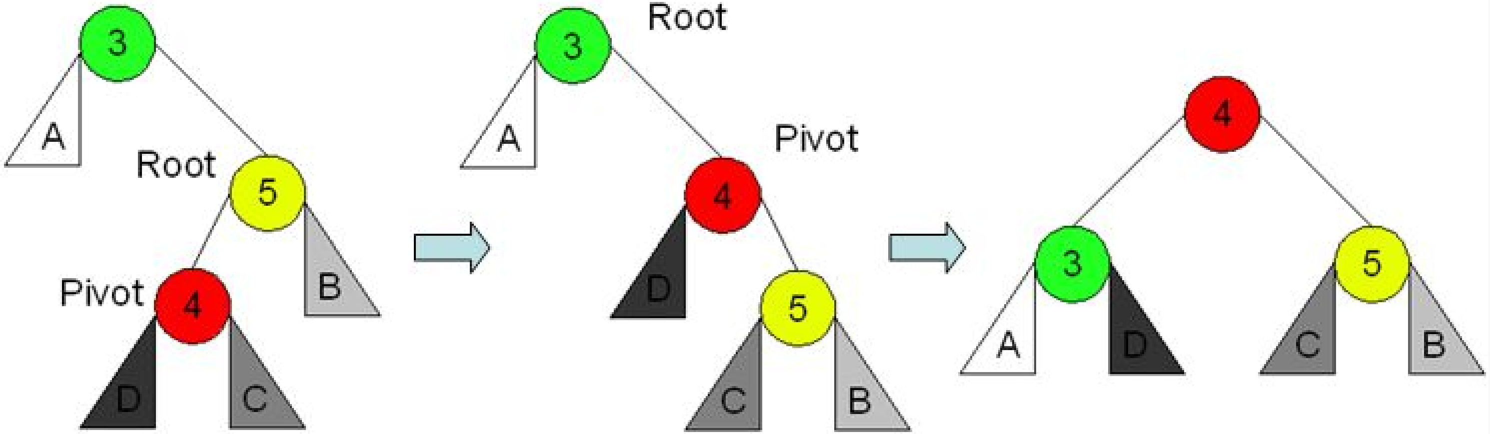
Perform a Left-Right Rotation:



Left-Right Example



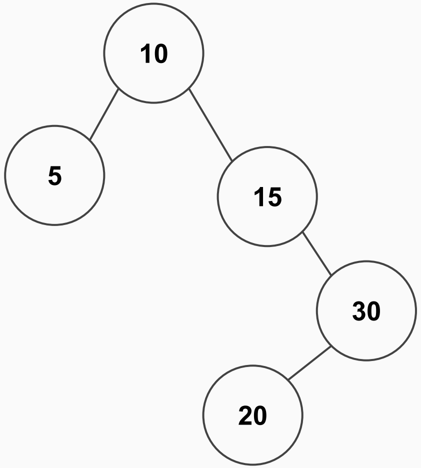
Right-Left Case



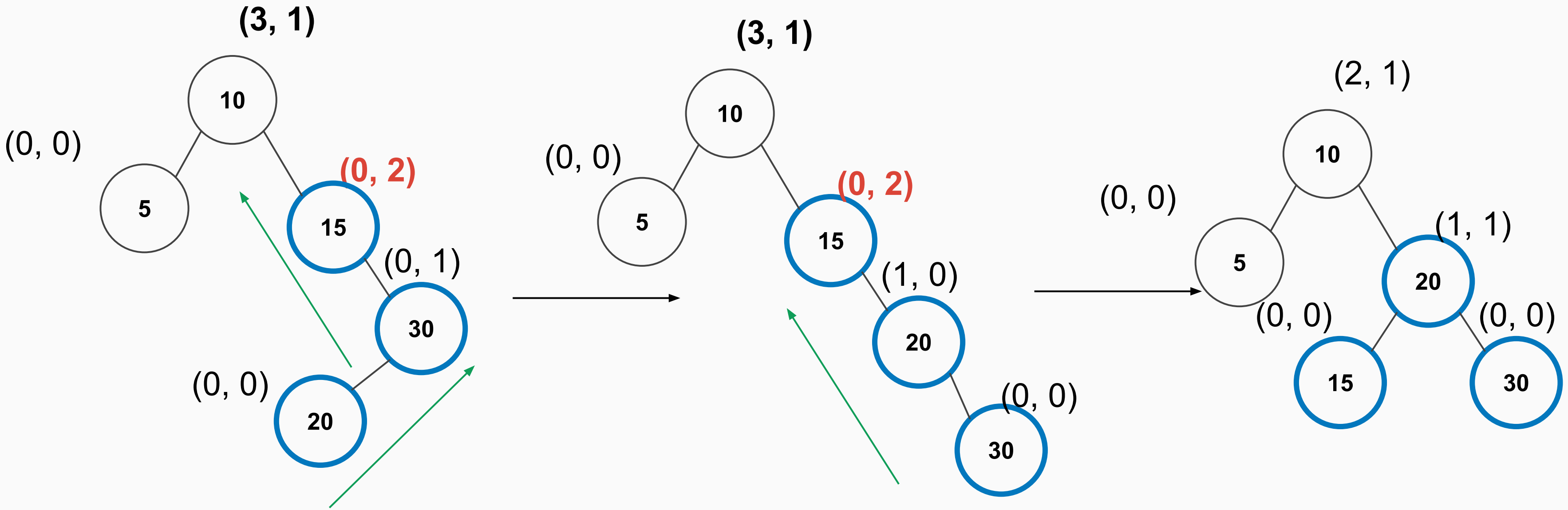
Right-Left Case: Right-Rotation and then Left-Rotation

ICE 6.4 Right-Left

Perform a Right-Left Rotation:



Right-Left Example



Time Complexity

- Insertion: log(n) even with rotations.... Assuming rotations take O(1)
  - Maximum of log(n) rotations
  - log(n)+log(n) = 2log(n) =.... O(log(n))
- Deletion: log(n)
- Search: log(n) - Binary Search Tree algorithm

ICE 6.5 AVL Trees

Instructions:

1. Construct an AVL Tree with the following terms: 15, 20, 24, 10, 13, 7, 30, 36, and 25.
2. Remove 24 and 20 from the above tree.

Note: Remember to check to see if AVL rotation operations are needed

Resources

https://www.cs.usfca.edu/~galles/visualization/Algorithms.html