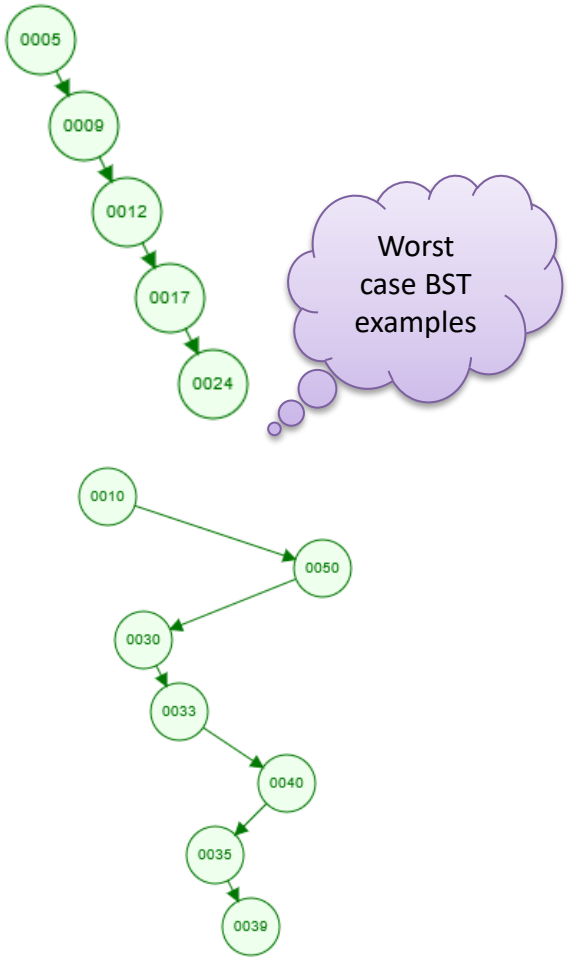
AVL Tree

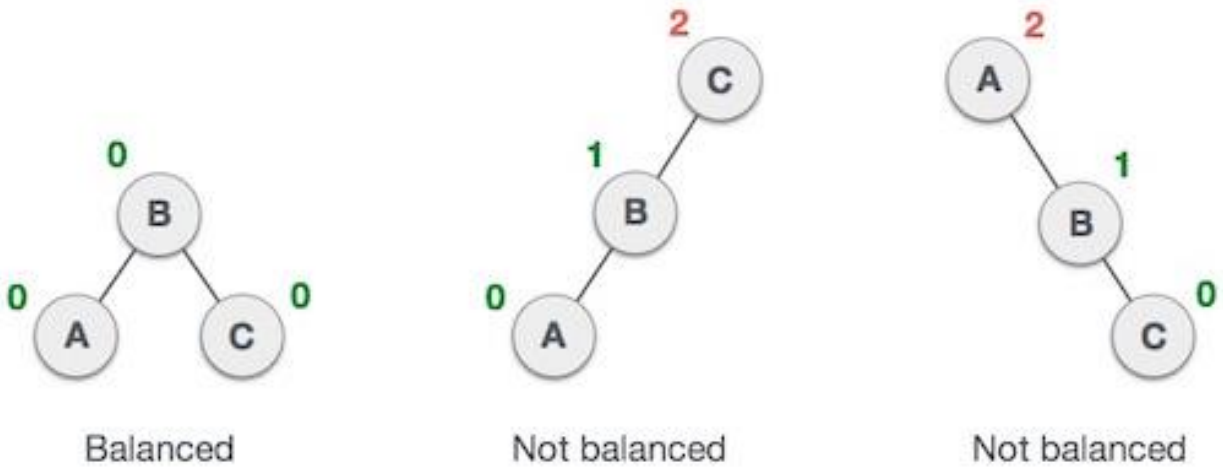
Introduction

- The worst case performance of BST is closest to linear search algorithms, that is O(n).
- In real-time data, we cannot predict data pattern and their frequencies. So, a need arises to balance out the existing BST.
- AVL tree is a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees cannot be more than one for all nodes.
- AVL tree checks the height of the left and the right sub-trees and assures that the difference is not more than 1. This difference is called the Balance Factor.



Source: https://www.tutorialspoint.com/data_structures_algorithms/avl_tree_algorithm.htm
https://www.geeksforgeeks.org/avl-tree-set-1-insertion/

Introduction



- Here we see that the first tree is balanced and the next two trees are not balanced
- In the second tree, the left subtree of C has height 2 and the right subtree has height 0, so the difference is 2. In the third tree, the right subtree of A has height 2 and the left is missing, so it is 0, and the difference is 2 again. AVL tree permits difference (balance factor) to be only 1.

BalanceFactor = height(left-subtree) − height(right-subtree)

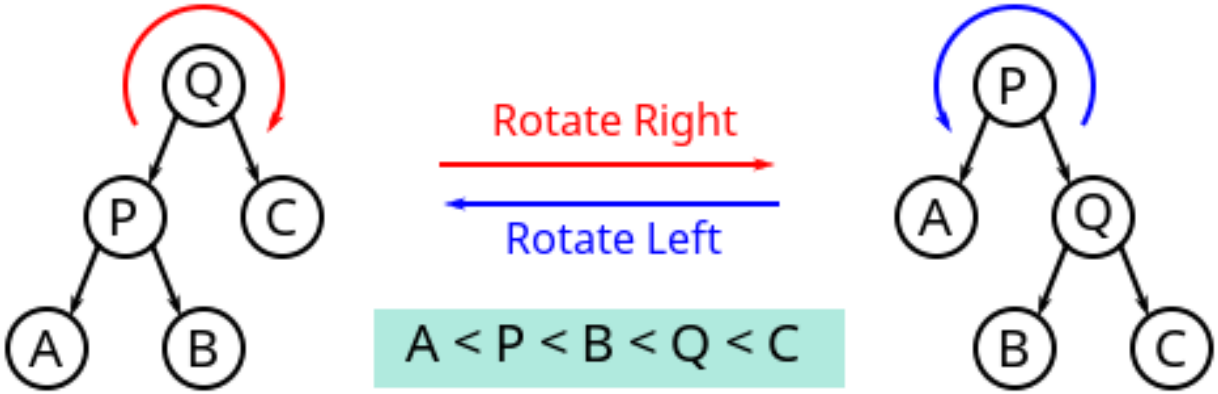
AVL Rotations

- To balance itself, an AVL tree may perform the following four kinds of rotations

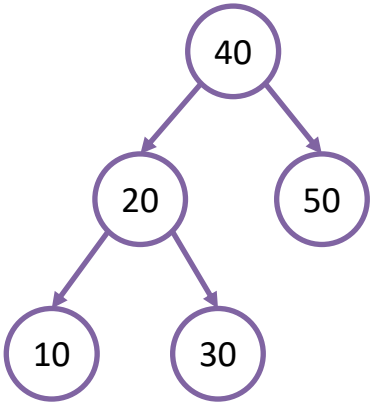
- Left Rotation LR
- Right Rotation RR
- Left-Right rotation LRR
- Right-Left rotation RLR

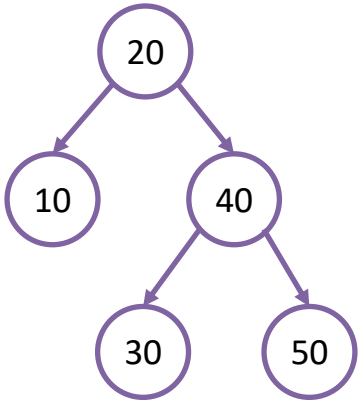
- The first two rotations are single rotations and the next two rotations are double rotations.
- To have an unbalanced tree, we at least need a tree of height 2.

Rotations



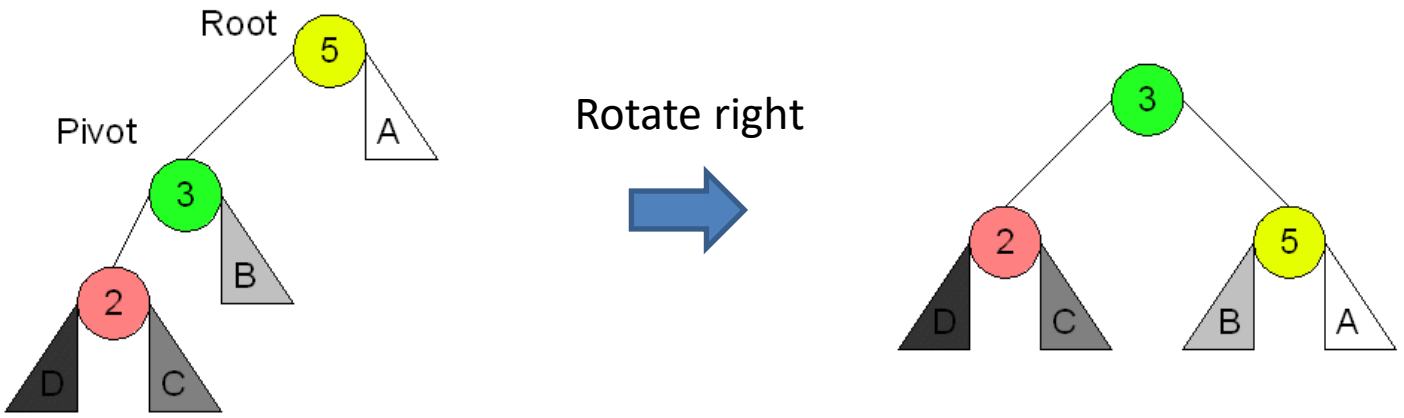
- Example:



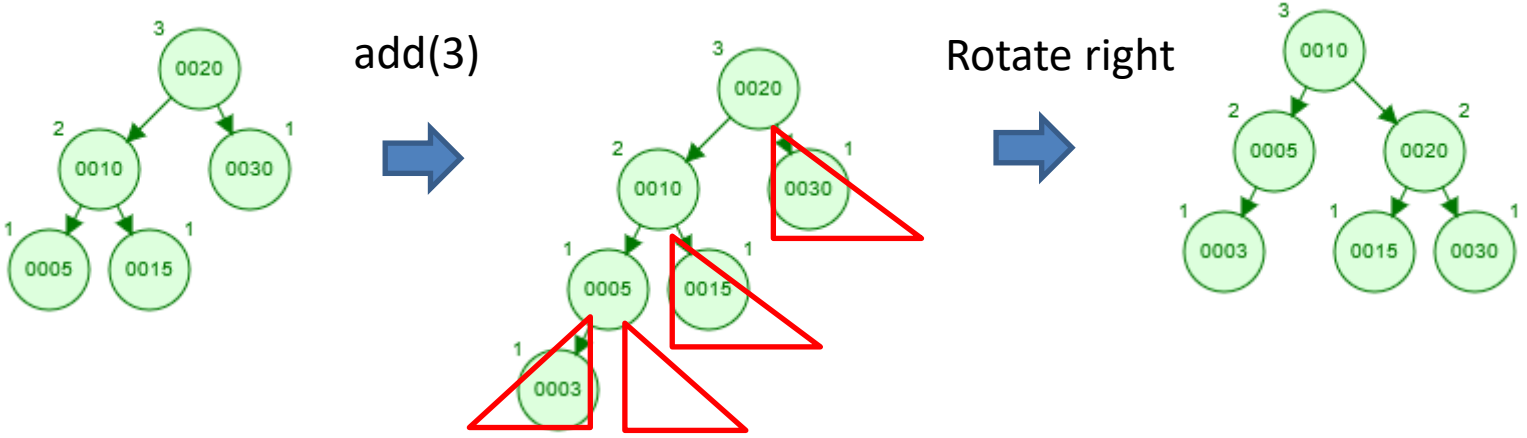


Right Rotation

Left Left Case

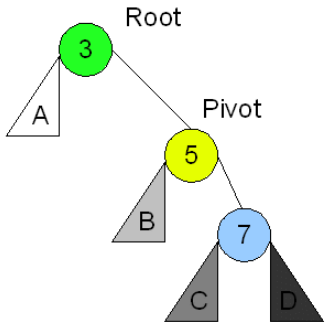


Example:

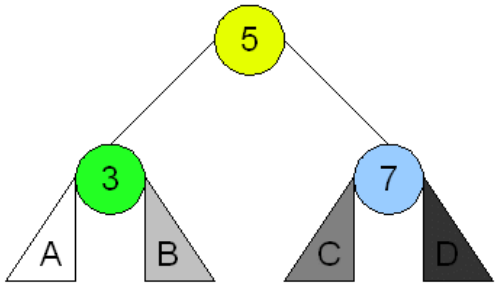


Left Rotation

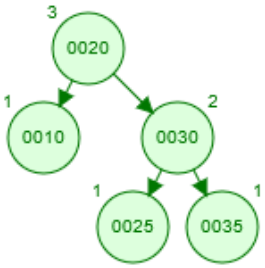
Right Right Case



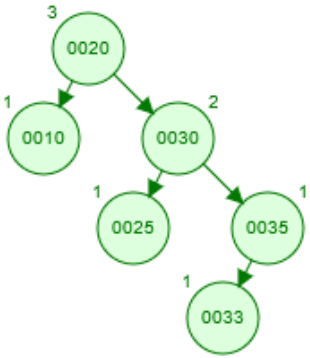
Rotate left



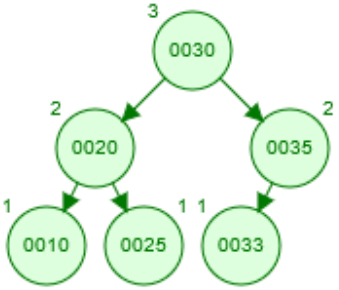
Left Rotation



add(33)

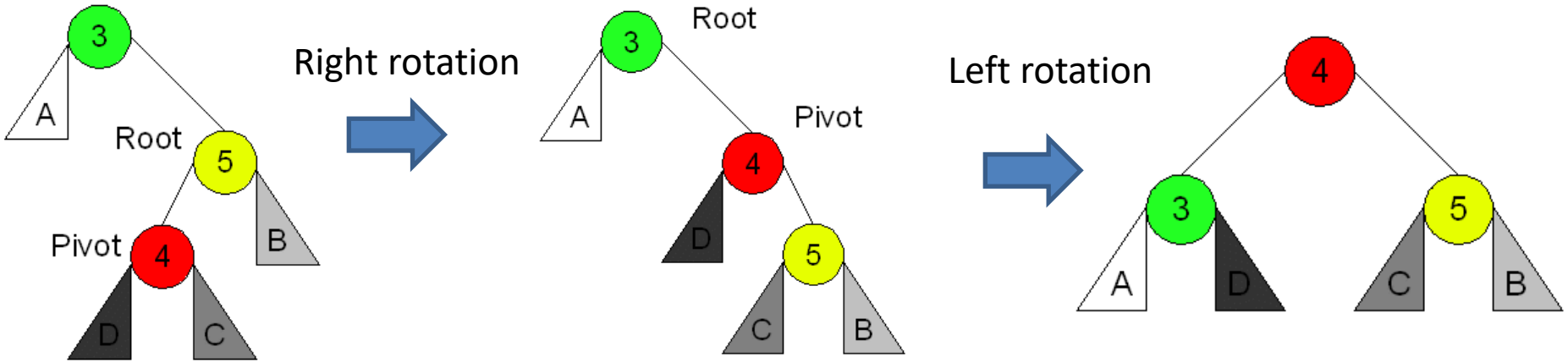


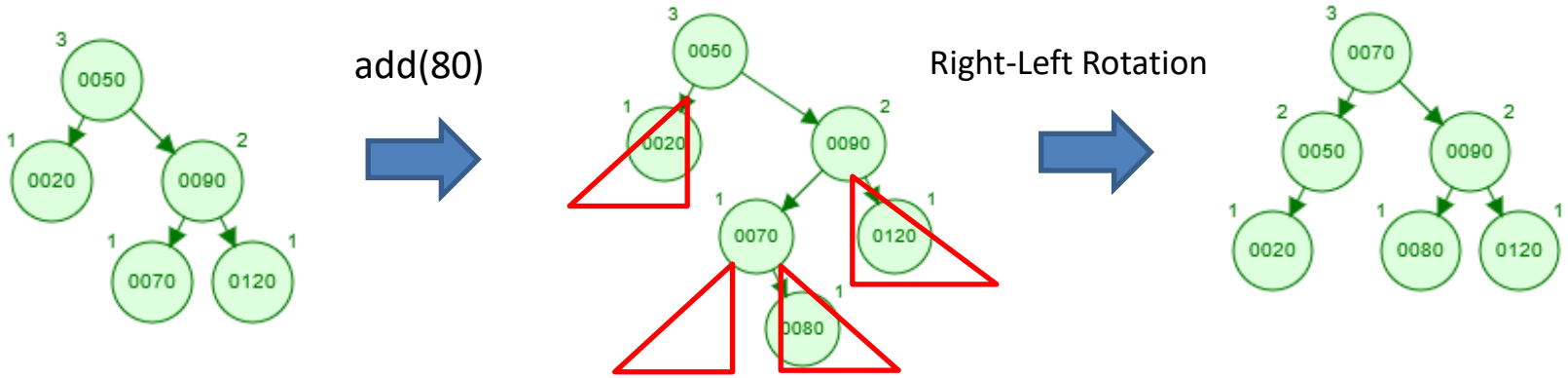
Rotate left



Right Left Rotation

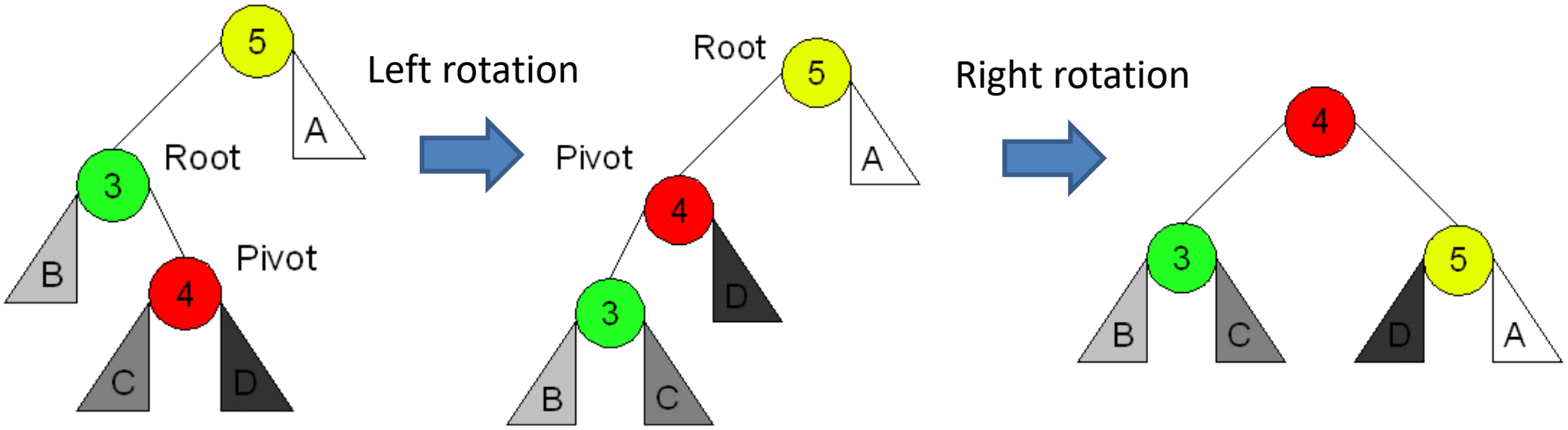
Right Left Case

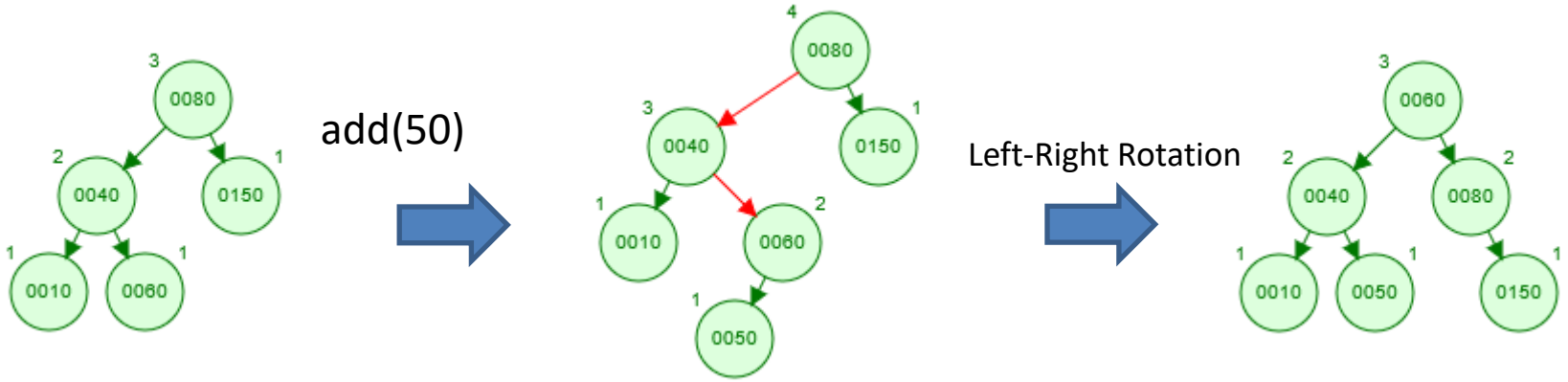




Left Right Rotation

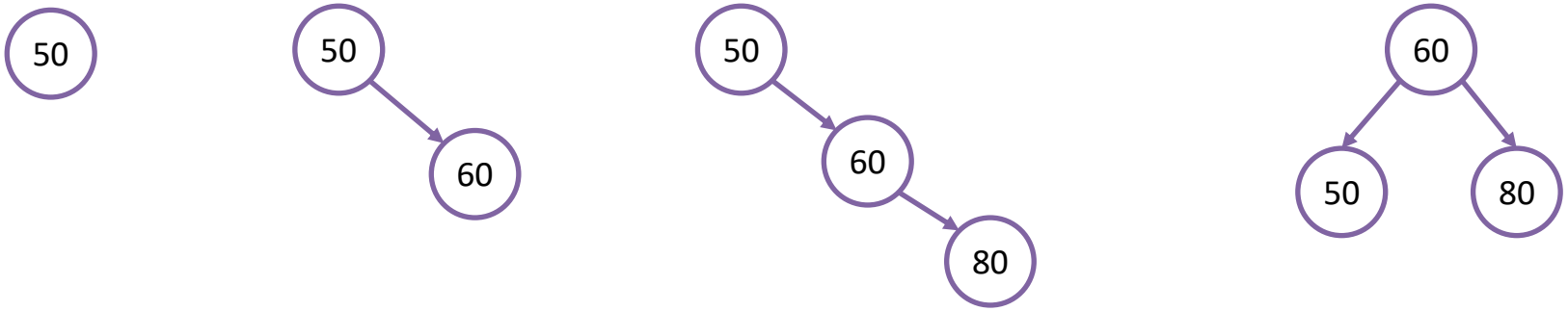
Left Right Case





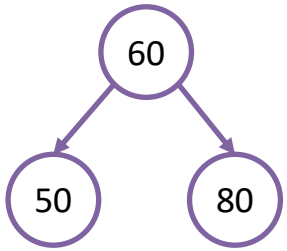
Example

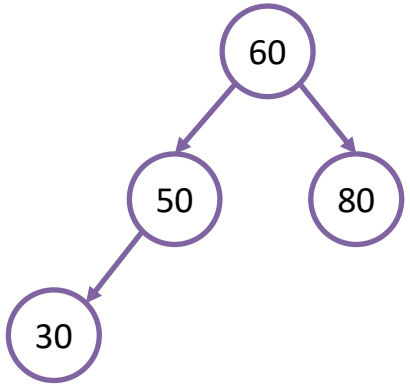
- Add below elements to a AVL tree
- A={50,60,80,30,20,40,70}

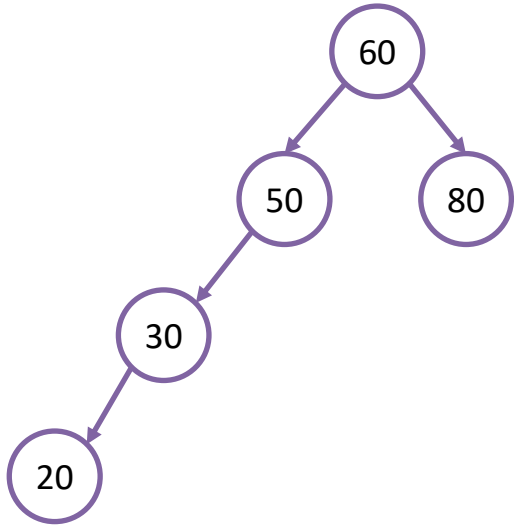


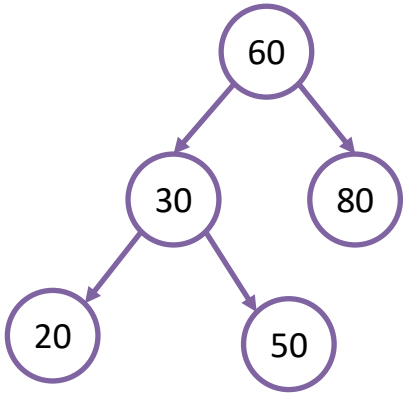
Example

- A={50,60,80,30,20,40,70}



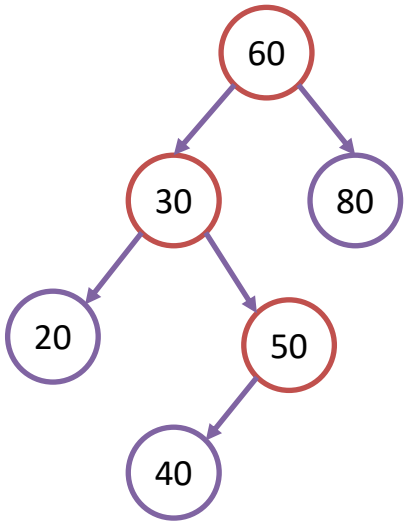


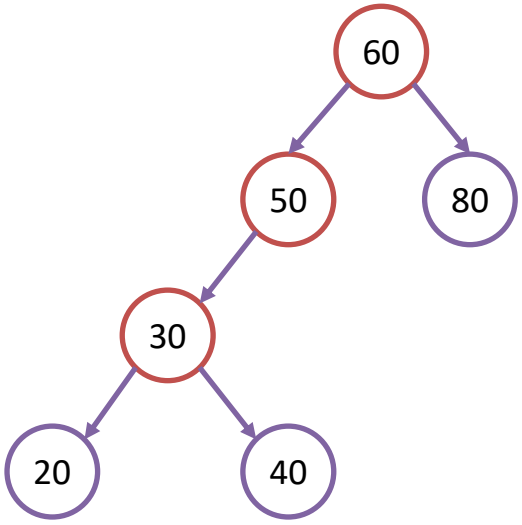


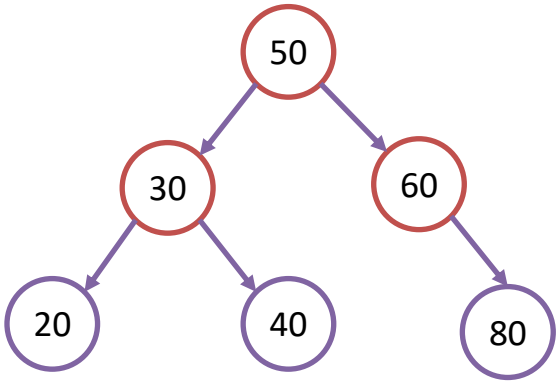


Example

- A={50,60,80,30,20,40,70}

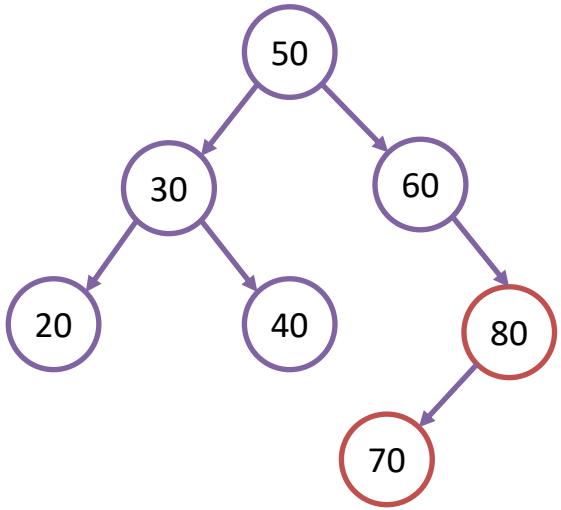


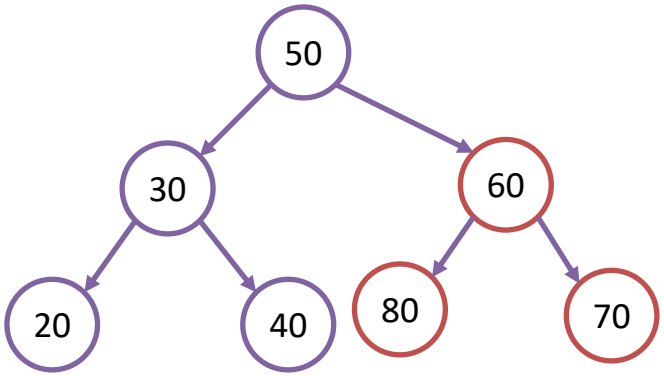


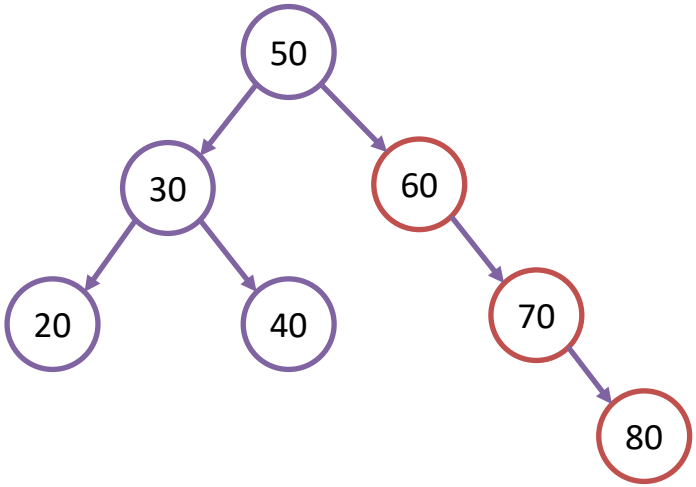


Example

- A={50,60,80,30,20,40,70}







AVL complexity

- Average cases
- Insert : $O(\log_2 n)$
- Search : $O(\log_2 n)$
- Remove: $O(\log_2 n)$
- Memory space: O(n)

- Disadvantages:
  - Frequent rotations
  - Complex structure