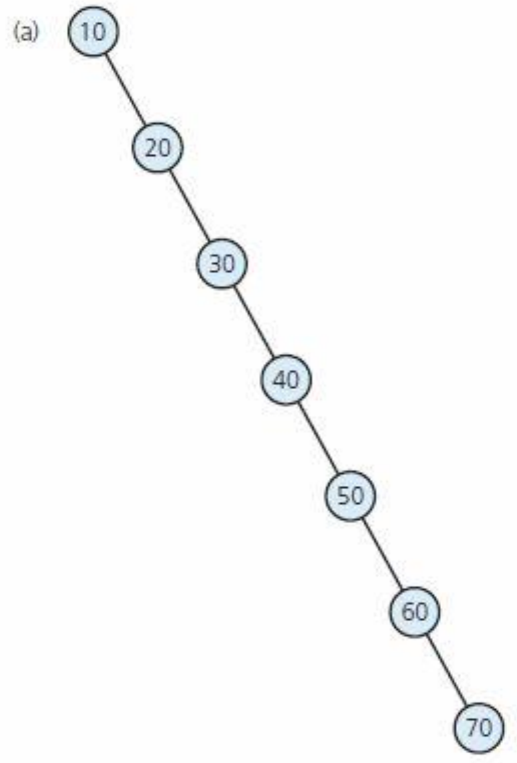
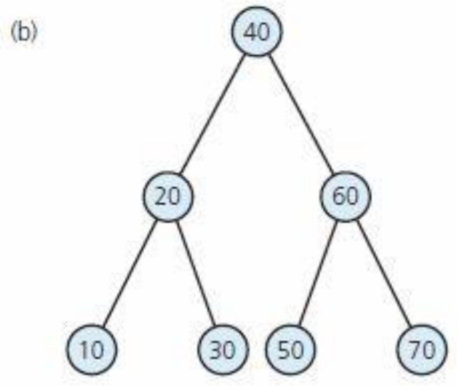
AVL Tree

- Problem with BST?
- BST are fast if they're **shallow**.
- Problems occur when one branch is **much longer** than the other.

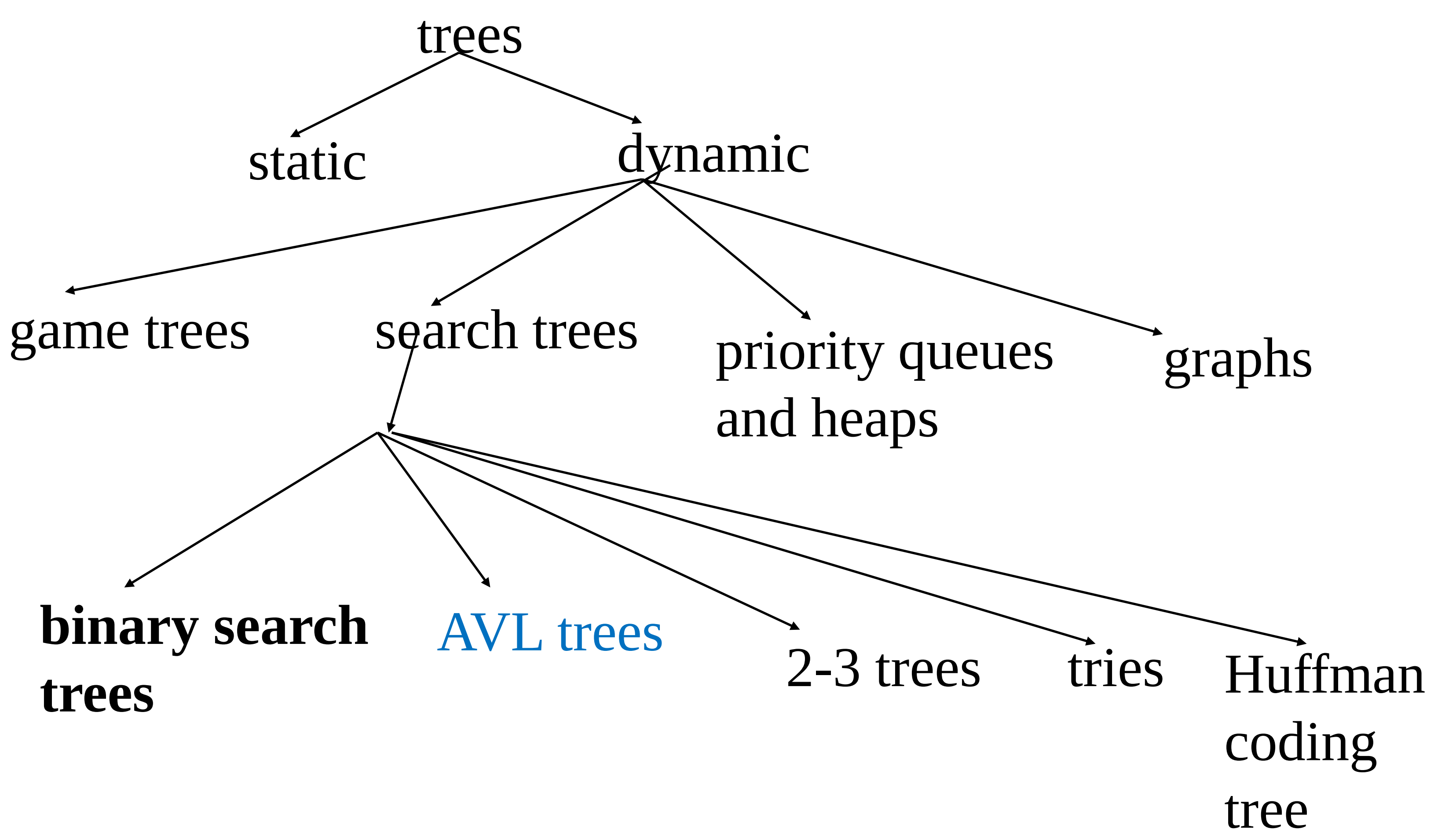
A binary search tree of maximum height





a binary search tree of minimum height

Types of Trees

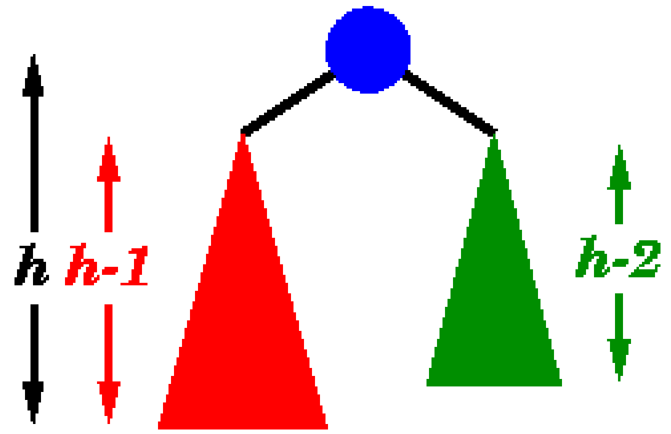


AVL Tree is…

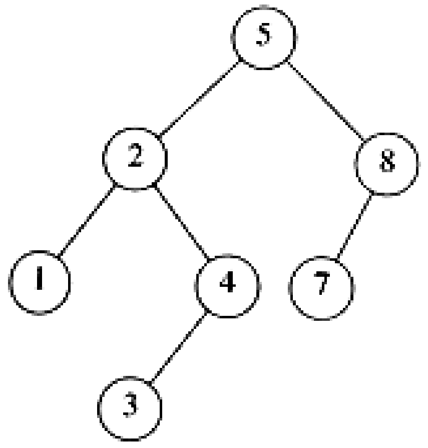
- named after **A**delson-**V**elskii and **L**andis
- the first **dynamically balanced trees** to be propose
- **Binary search tree** with **balance condition** in which the **sub-trees** of each node can differ by **at most 1** in their height

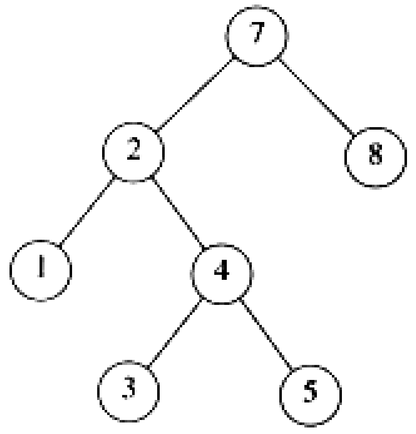
An AVL tree has the following properties:

1. **Sub-trees of each node can differ by at most 1 in their height**
2. Every sub-trees is an AVL tree



AVL tree?



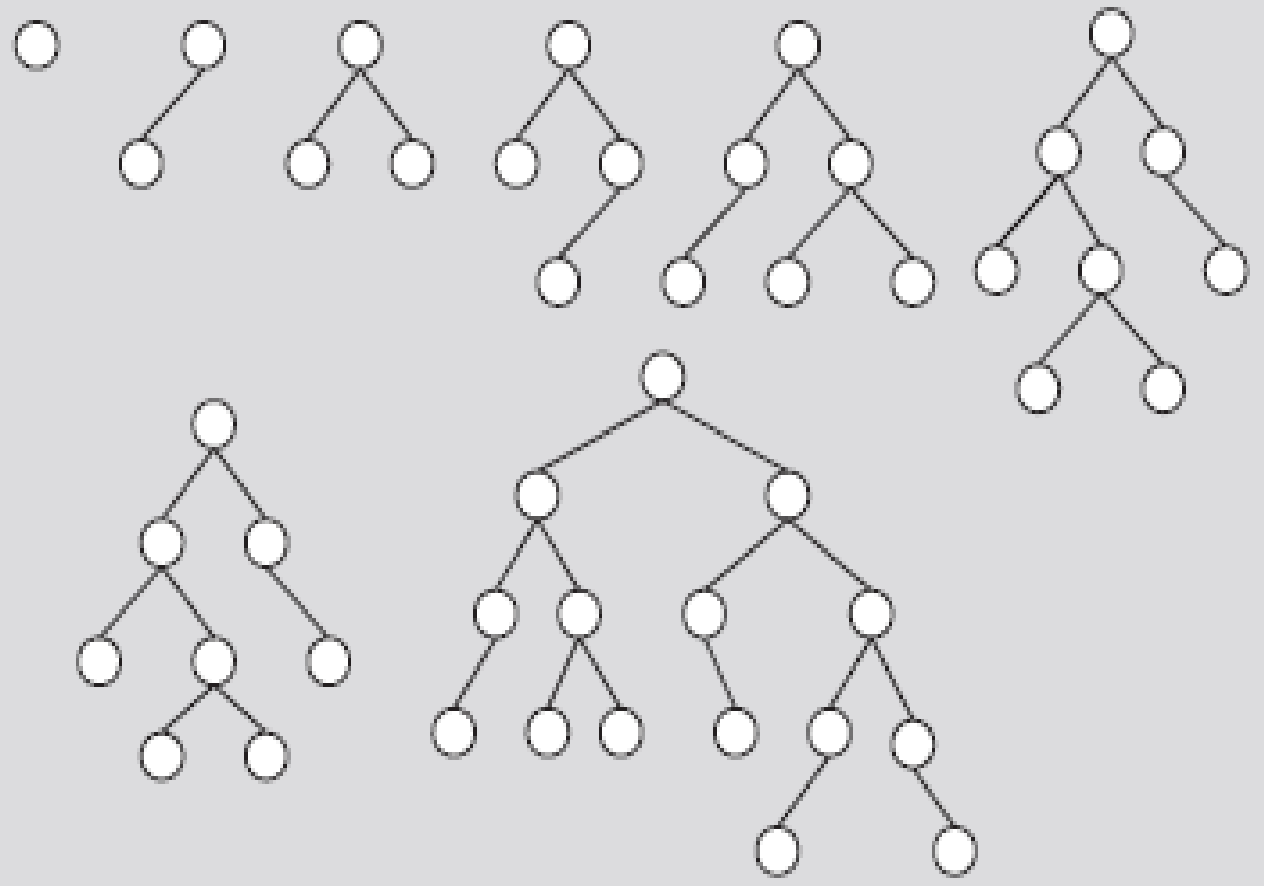


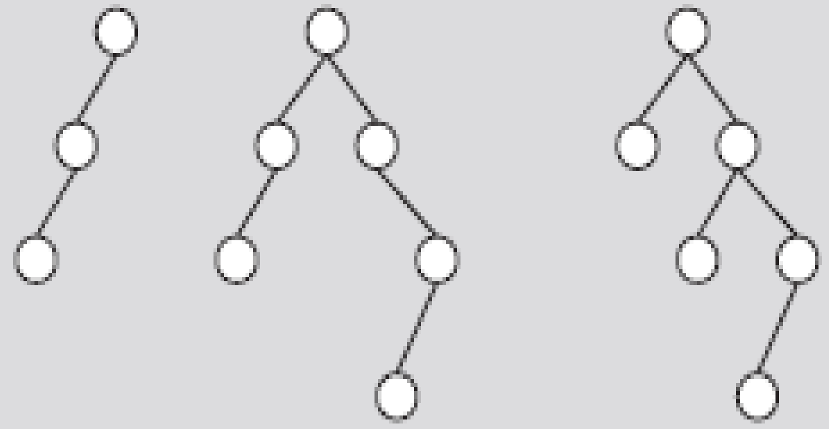
YES

Each left sub-tree has height 1 greater than each right sub-tree

NO

Left sub-tree has height 3, but right sub-tree has height 1


(a) AVL trees

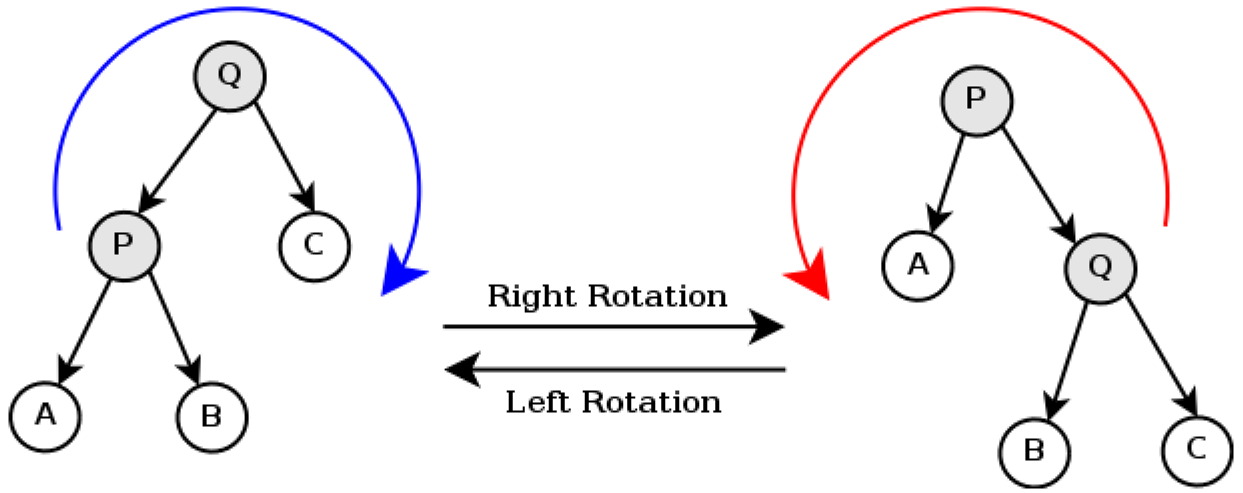

(b) Non-AVL trees

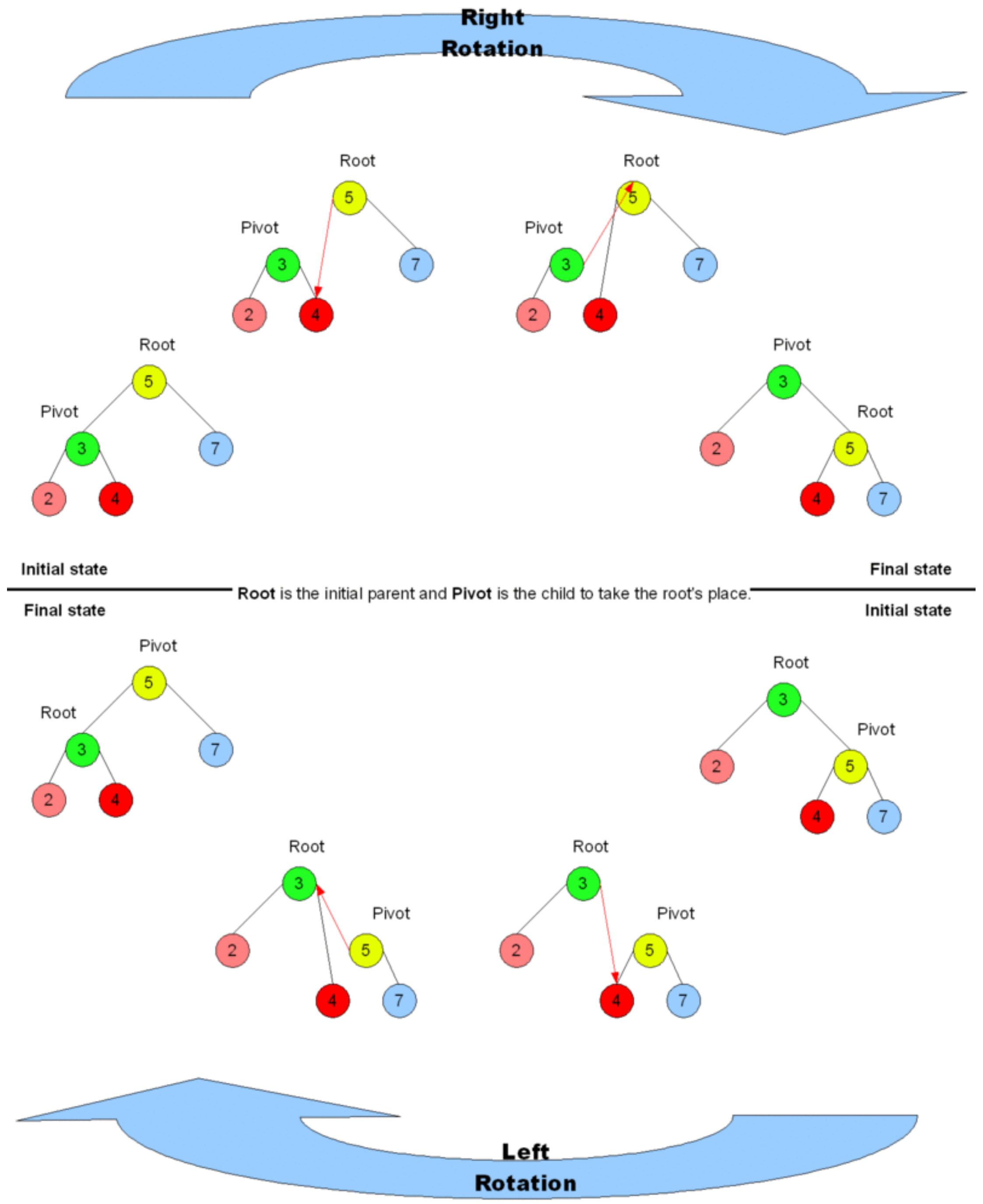
Insertion and Deletions

- Operations on AVL trees (e.g., search) are implemented as binary trees, except insertion and deletions which may perform additional computations.
- One more thing is associated with each node x in the AVL tree =>
- **balanceFactor** = height(left subtree) - height(right subtree)
  - OR: balanceFactor = height(right subtree) - height(left subtree)
- The balance factor of any node of an AVL tree is **(-1,0, or +1)**.
- The temporarily recomputed balance factor of a node after an insertion will be in the range **[−2,+2]**.
- **After inserting a node**, it is necessary to **check** each of the node's ancestors (in sequence, parent , grandparent etc) for consistency.
- **If a node is unbalanced,** perform **single or double rotation/reconstruction** to correct balance.

Tree rotation

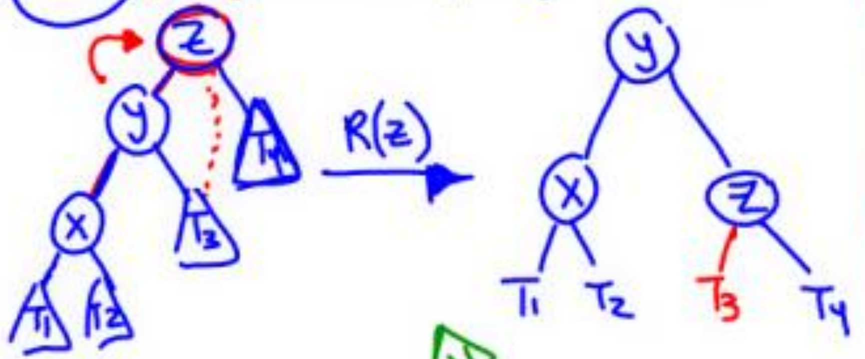
- An operation on a binary tree that changes the structure without interfering with the order of the elements.
- A tree rotation **moves one node or subtree up** in the tree and **one down**.
- It is used to change the shape of the tree, and in particular to decrease its height by moving smaller subtrees down and larger subtrees up, resulting in improved performance of many tree operations.



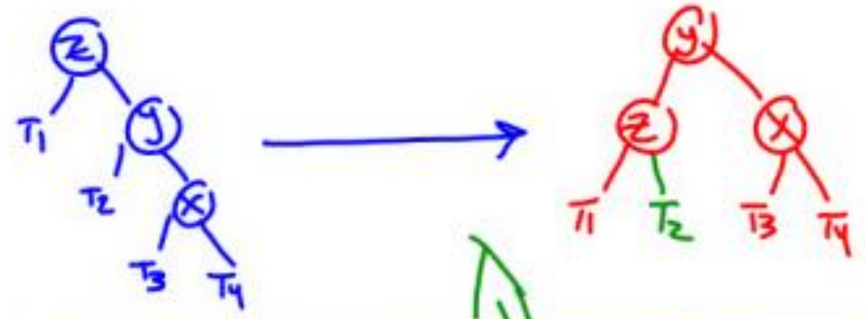


Root is the initial parent and **Pivot** is the child to take the root's place.

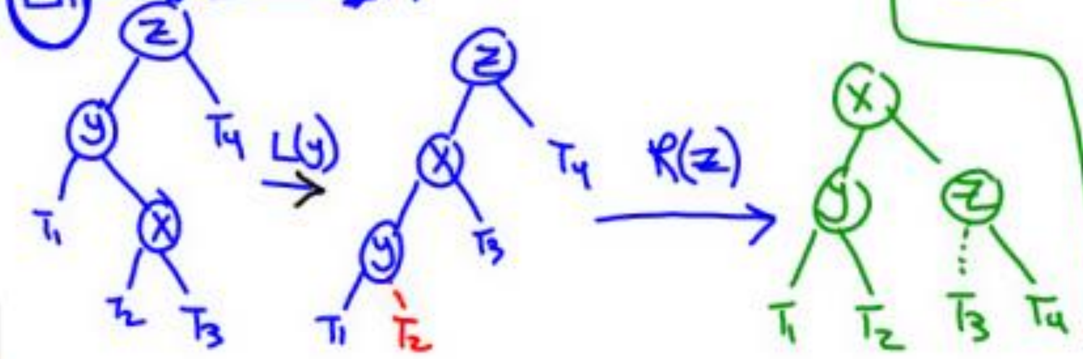
① LL → Right Rotation (single Rotation)



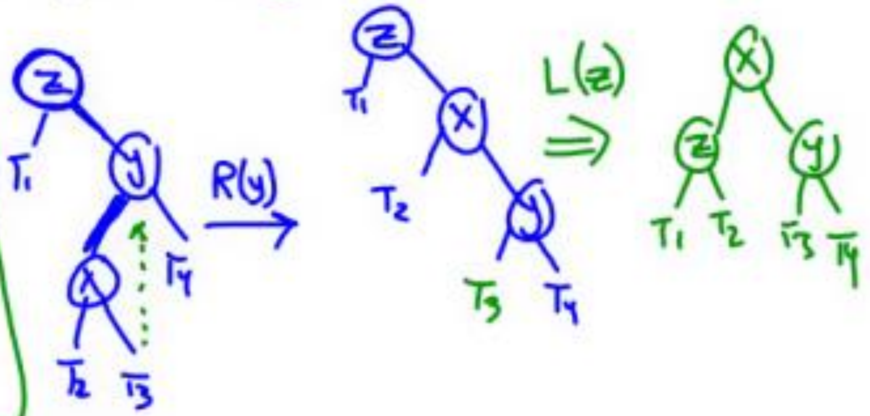
③ RR Left Rotation (single Rotation)

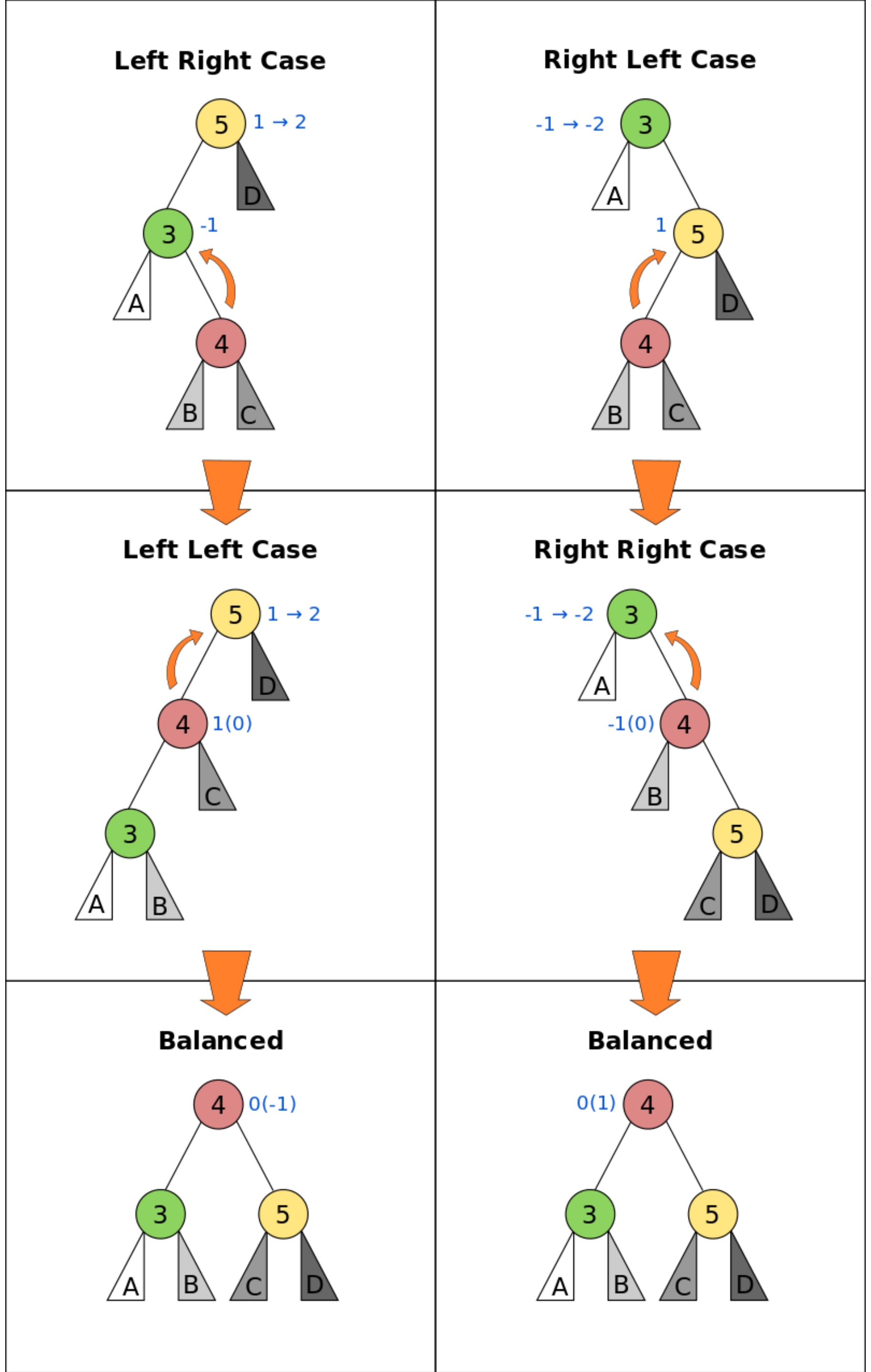


② LR → L → R



④ RL → R → L





Summary:

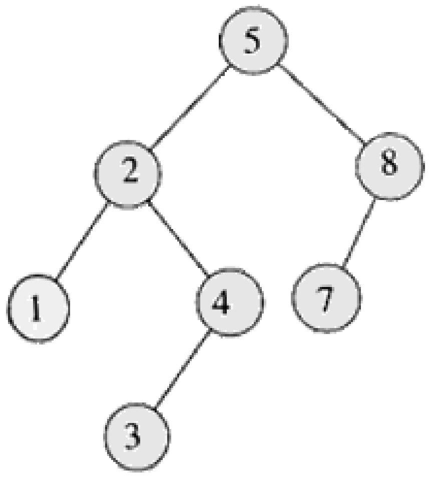
1. LL -> R (node)
2. RR-> L (node)
3. LR-> L (subtree) -> R(node)
4. RL-> R (subtree) -> L(node)

A blue number next to a node denotes possible balance factors (those in parentheses occurring only in case of deletion).

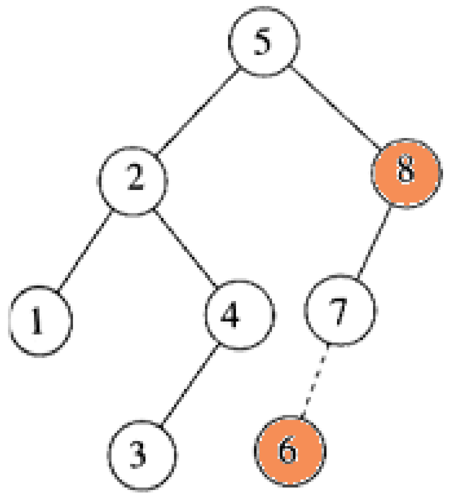
Rotation steps

- IF tree is **right** heavy (height differ by more than 1)
  - IF tree's right subtree is left(L) heavy **(RL)**
    - **R** rotation to the subtree and then **L** rotation to the node (double rotation)
  - ELSE IF tree's right subtree is right(R) heavy **(RR)**
    - **L** rotation to the unbalanced node (single rotation)
- ELSE IF **tree** is left heavy
  - IF tree's left subtree is right heavy **(LR)**
    - **L** rotation to the subtree and then **R** rotation to the node (double rotation)
  - ELSE IF tree's left subtree is left heavy **(LL)**
    - **R** rotation to the unbalanced node (single rotation)
- NOTE: During a rotation, the moved children must be connected to a **parent and not another child**.

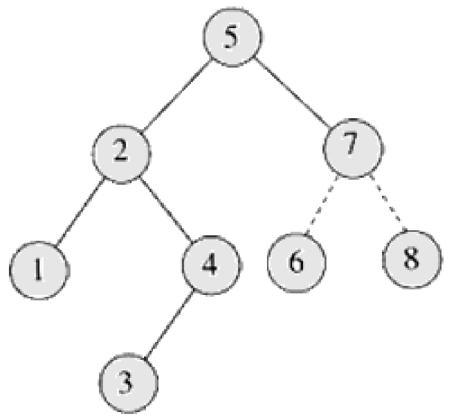
Insertion



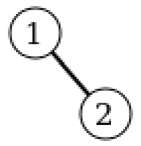
Insert 6

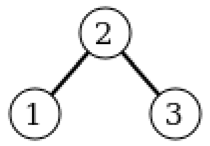


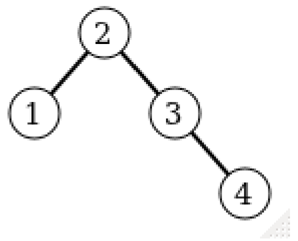
Imbalance at 8 Perform rotation with 7

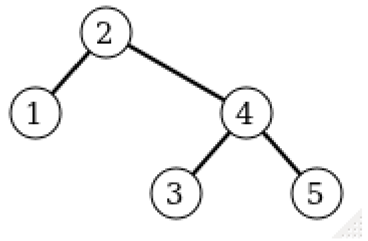


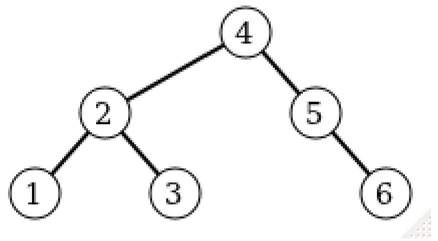
Insert 1, 2, 3, 4, 5, 6, 7 into an AVL Tree

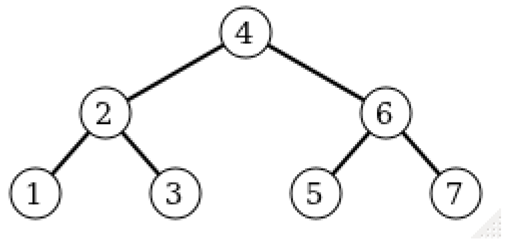




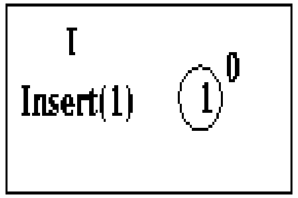


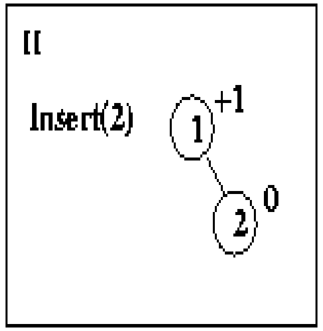


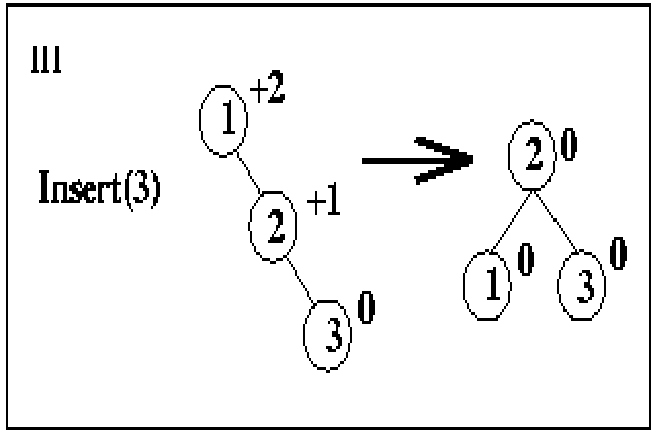


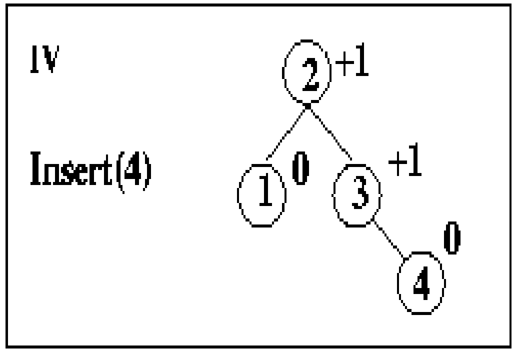


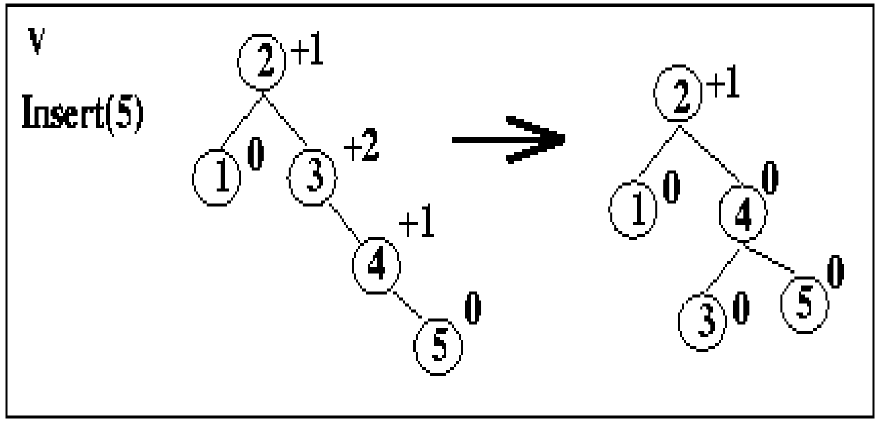
Insert 1, 2,3, 4, 5, 0,7, 6 into an AVL Tree

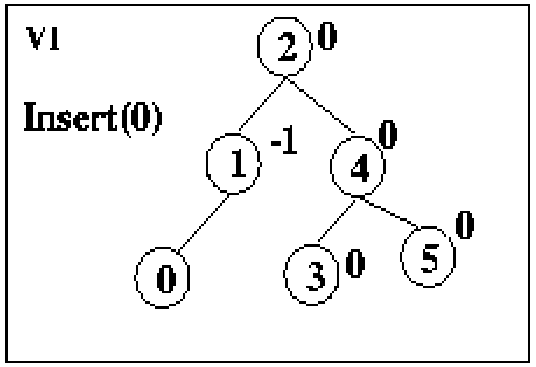


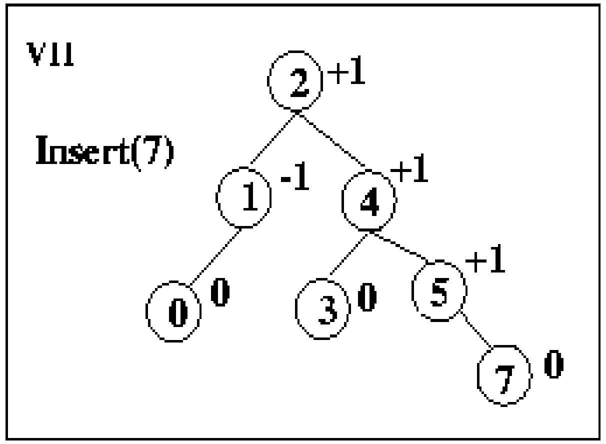


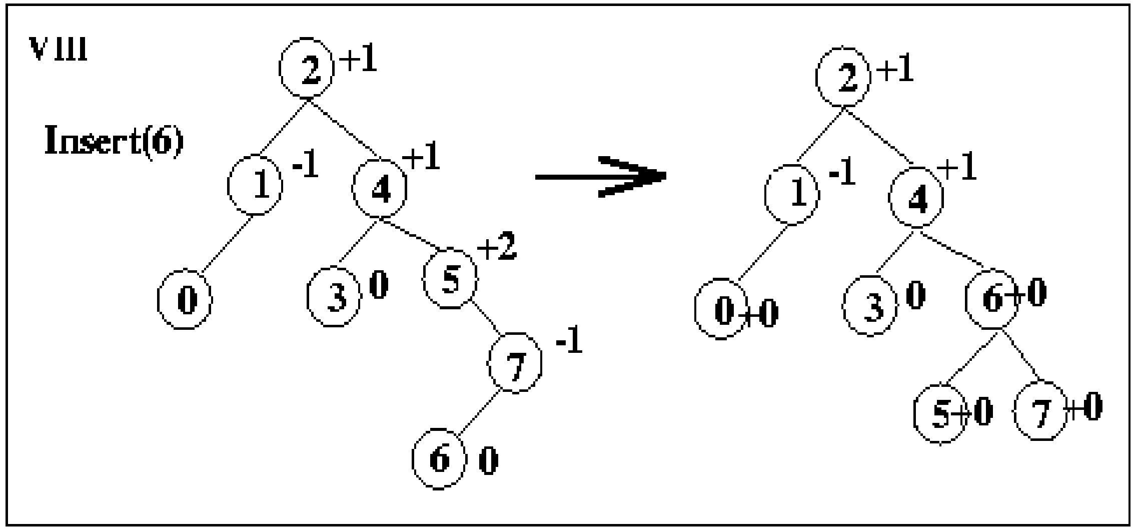




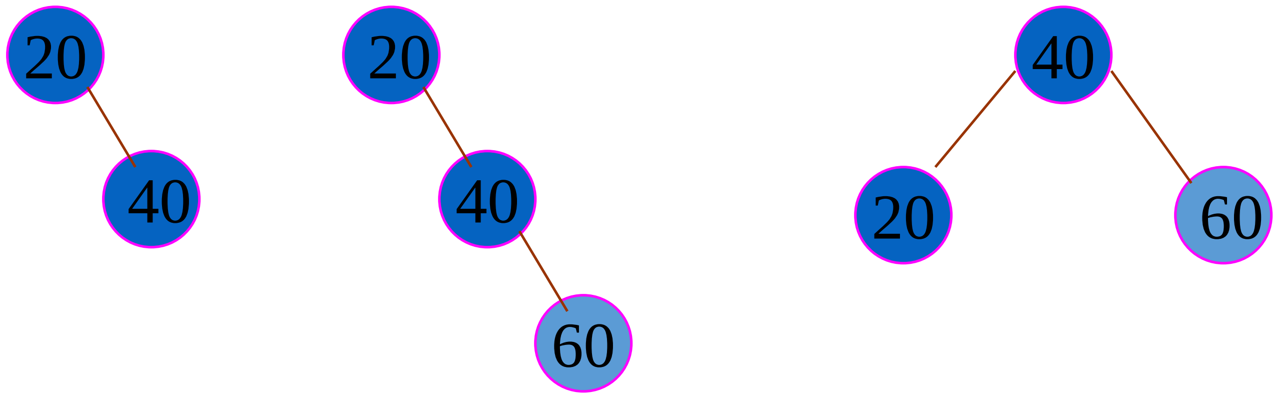



VI
Insert(0)
2 0
1 -1
4 0
0
3 0
5 0

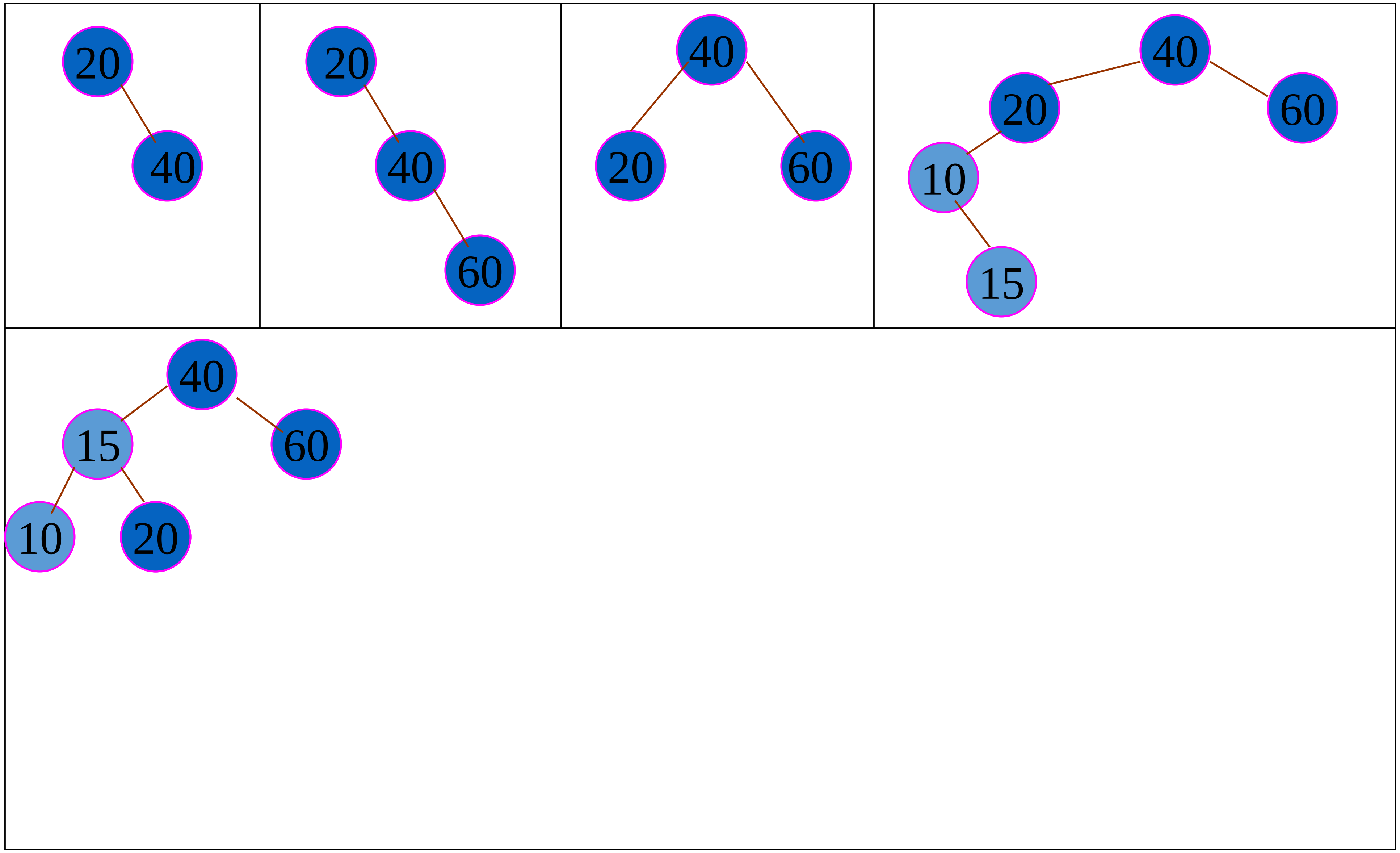

VII
Insert(7)
2 +1
1 -1
4 +1
0 0
3 0
5 +1
7 0


VIII
Insert(6)
2 +1
1 -1
4 +1
0
3 0
5 +2
7 -1
6 0
2 +1
1 -1
4 +1
0 +0
3 0
6 +0
5 +0
7 +0

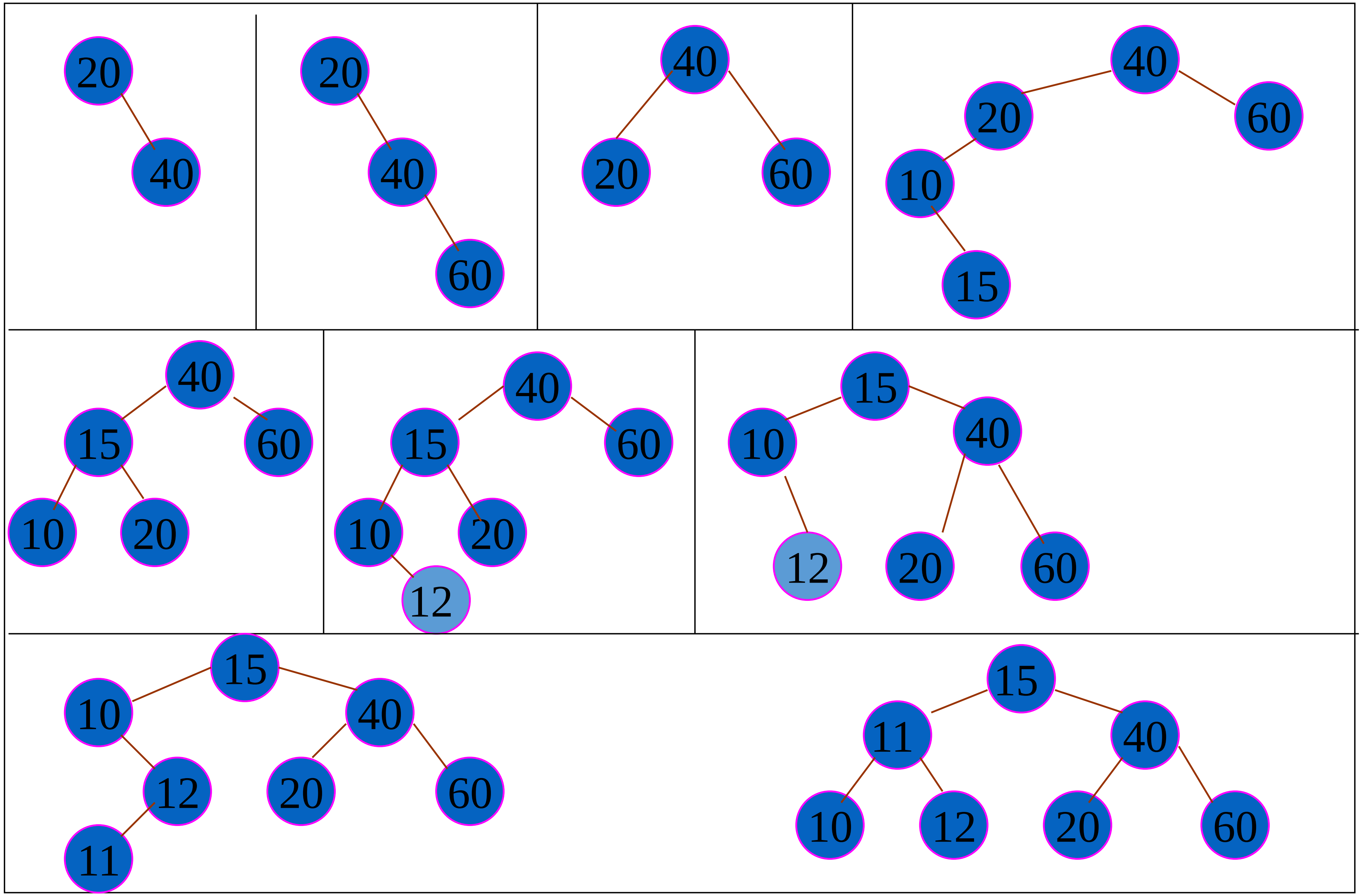
Example of insertion (with rotation): 20 40 60



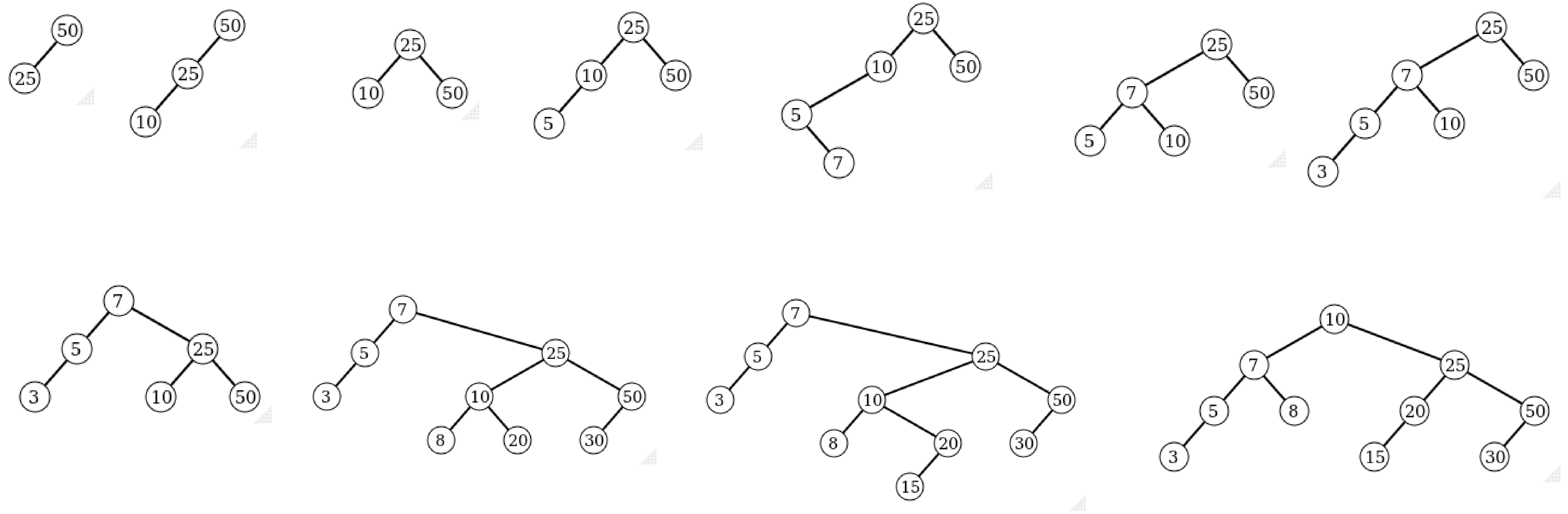
Example of insertion (with rotation): 20 40 60 **10 15**

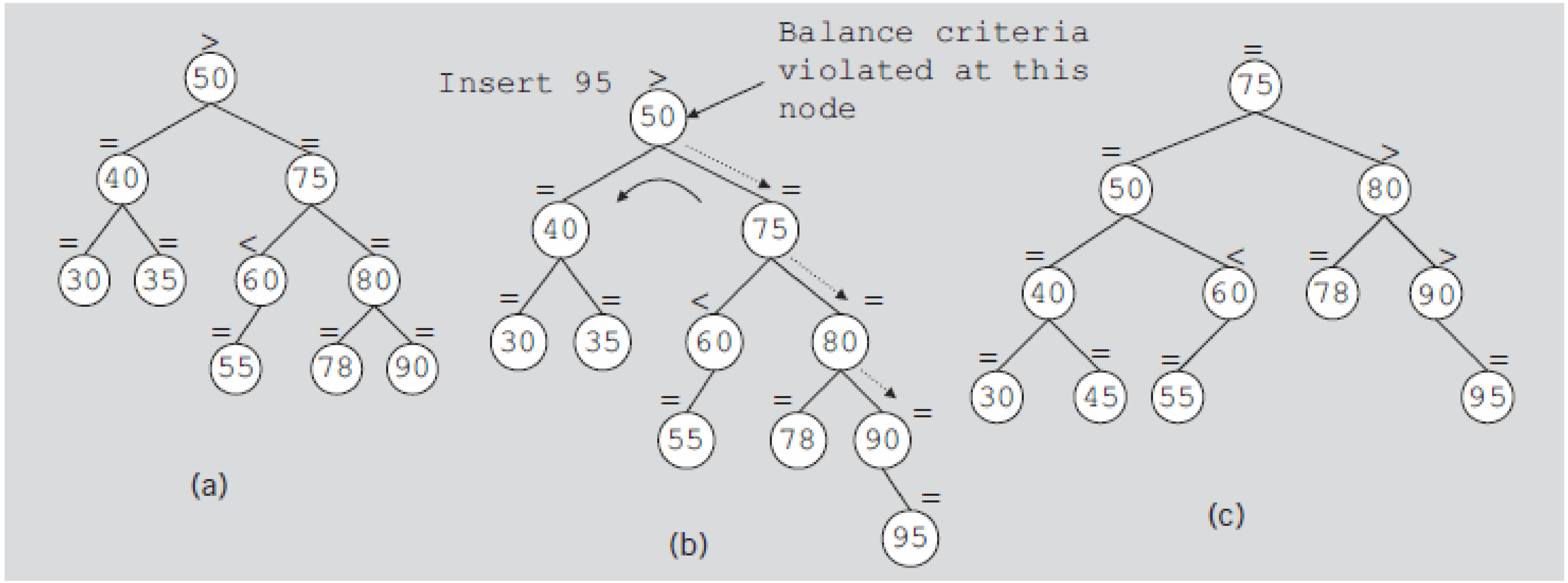


Example of insertion (with rotation) 20 40 60 10 15 **12 11**

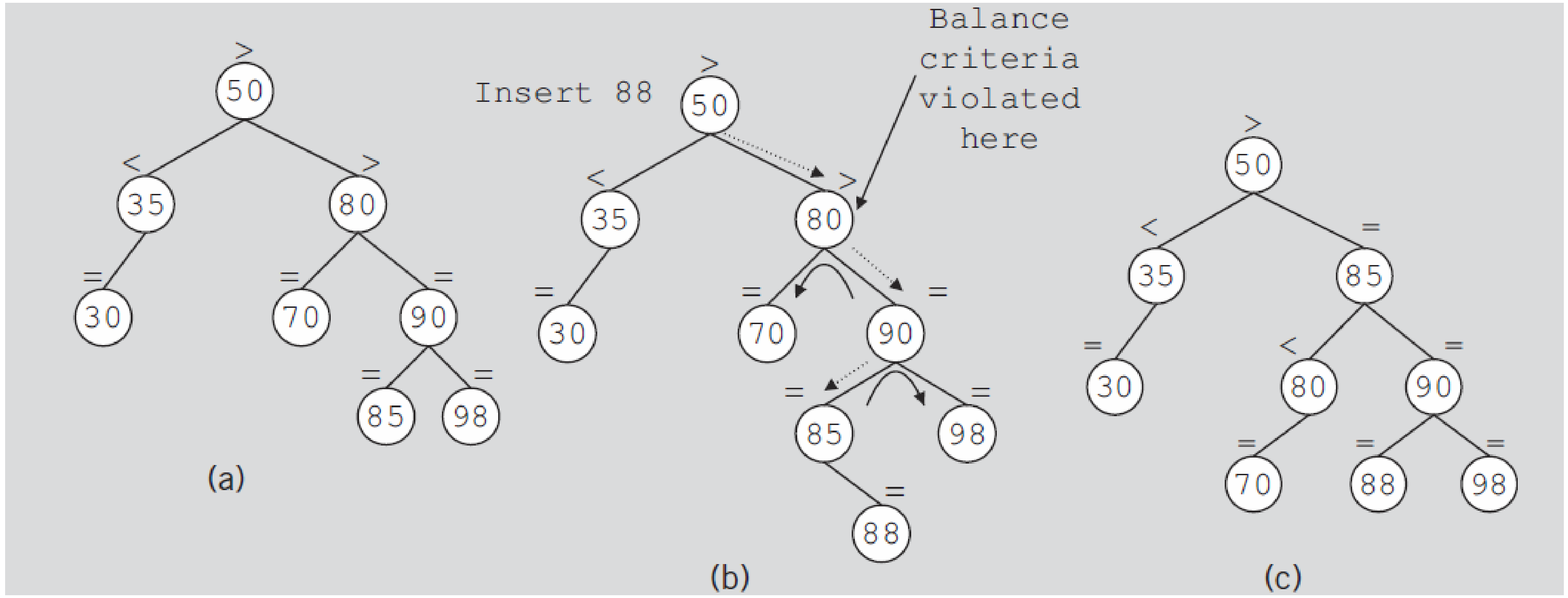


Insert 50, 25, 10, 5, 7, 3, 30, 20, 8, 15 into an AVL Tree



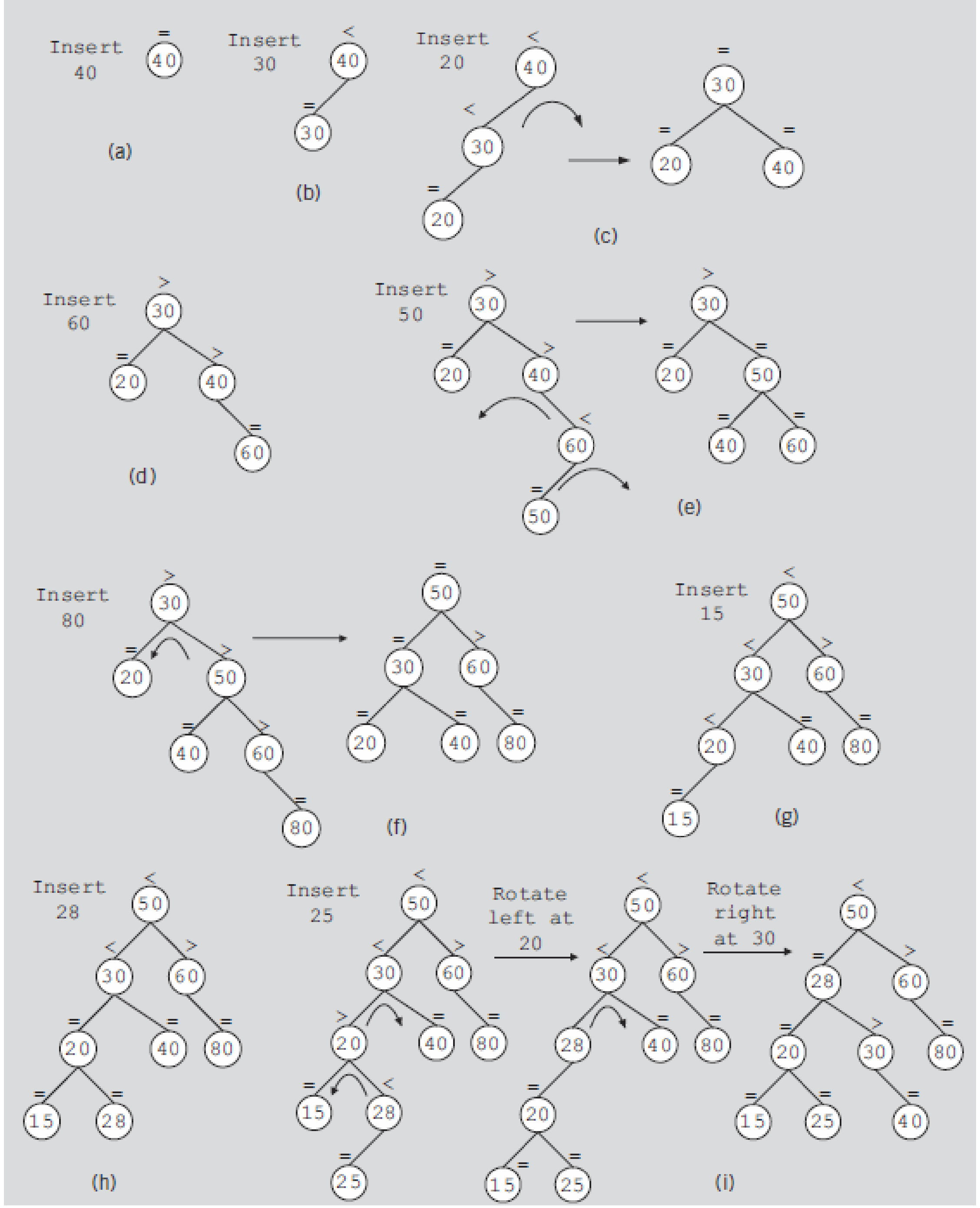


Insert 95
Balance criteria violated at this node
(a)
(b)
(c)



Insert 88
Balance criteria violated here
(a)
(b)
(c)

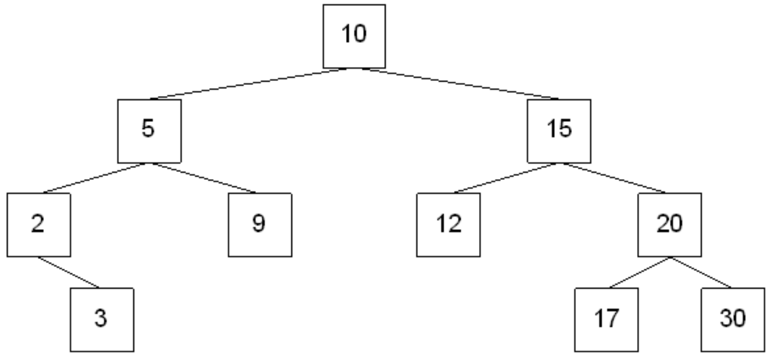
Insert 40, 30, 20, 60, 50, 80, 15, 28, 25 into an AVL Tree



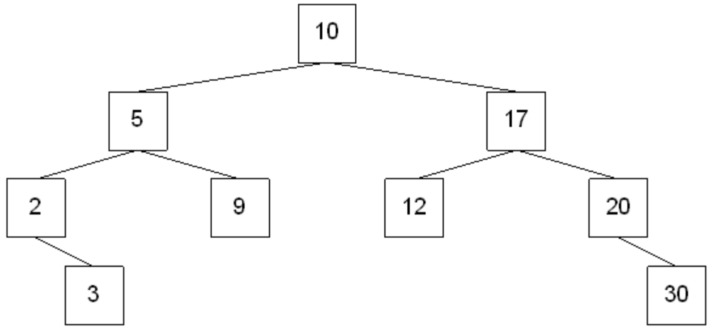
Deletion from AVL Trees

- We first do the normal BST deletion:
  - **0 children:** just delete it
  - **1 child:** delete it, connect child to parent
  - **2 children:** put predecessor/successor in your place, delete predecessor/successor
- After deleting the node, the resulting tree might no longer be an AVL tree. As in the case of insertion into an AVL tree, and the “height” at the top could decrease or increase by 1.
- Which nodes’ heights may have changed:
  - **0 children:** path from deleted node to root
  - **1 child:** path from deleted node to root
  - **2 children:** path from deleted successor leaf to root
- **Similar rotations to insert**, but in case there is an imbalance at a node and the left subtree height is equal to right subtree height, then perform **either rotation** (single or double rotate).

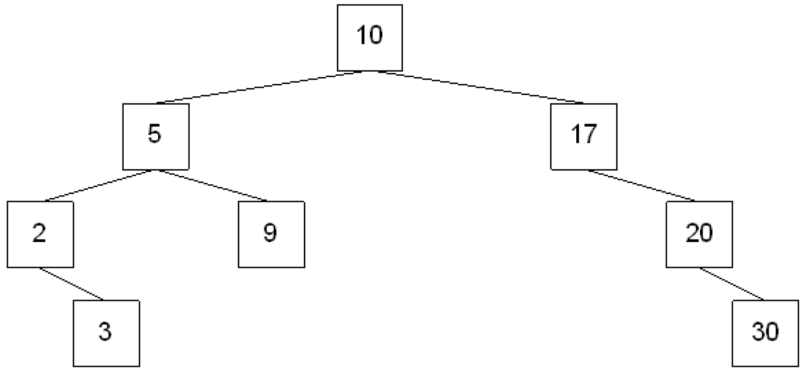
Delete 15 and 12 using INORDER SUCCESSOR logic



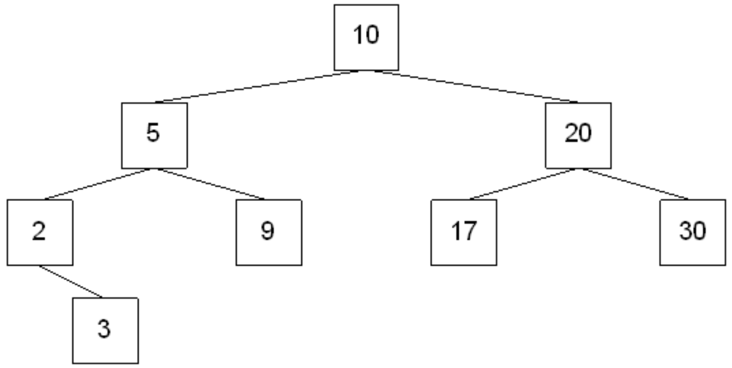
(1)



(2)

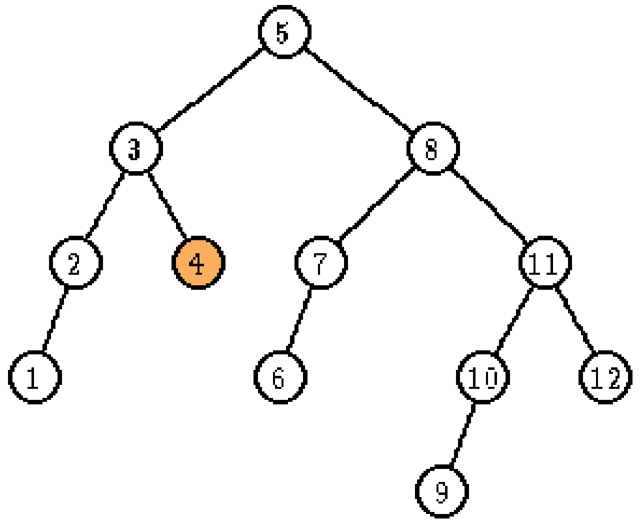


(3)

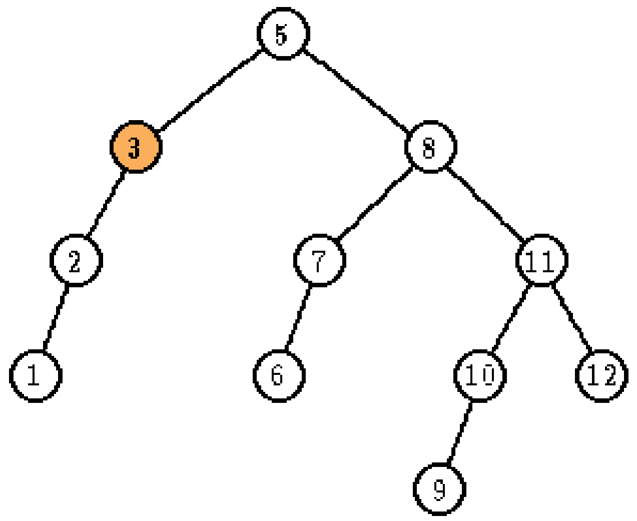


(4)

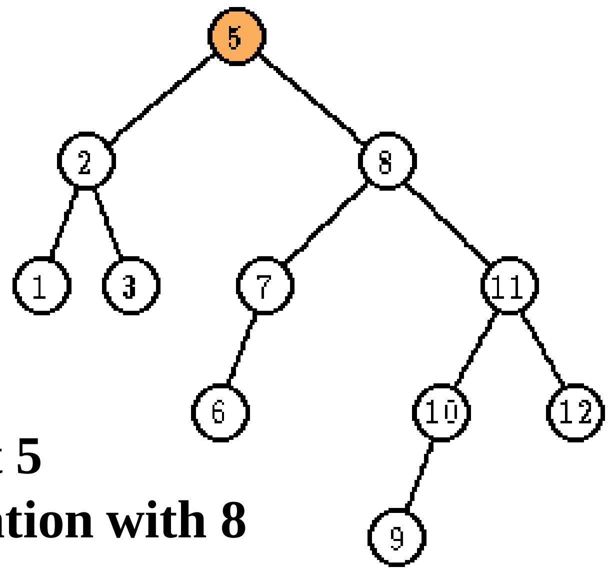
Deletion



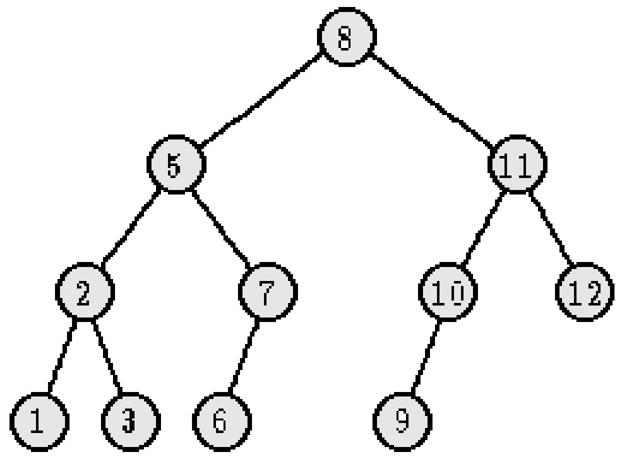
Delete 4



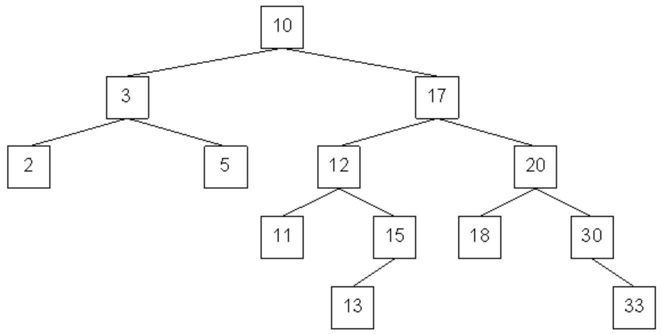
Imbalance at 3
Perform rotation with 2

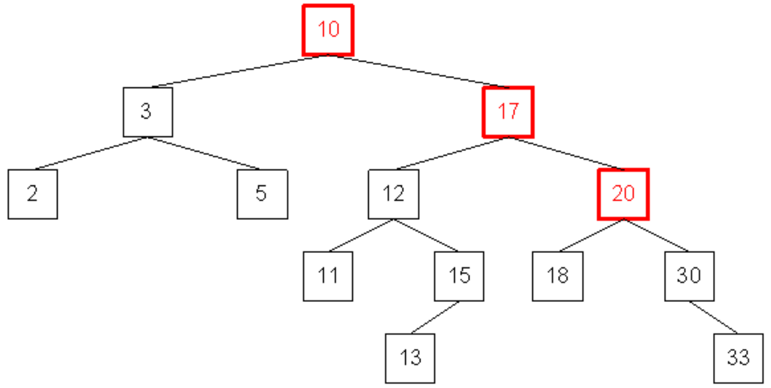


Imbalance at 5
Perform rotation with 8

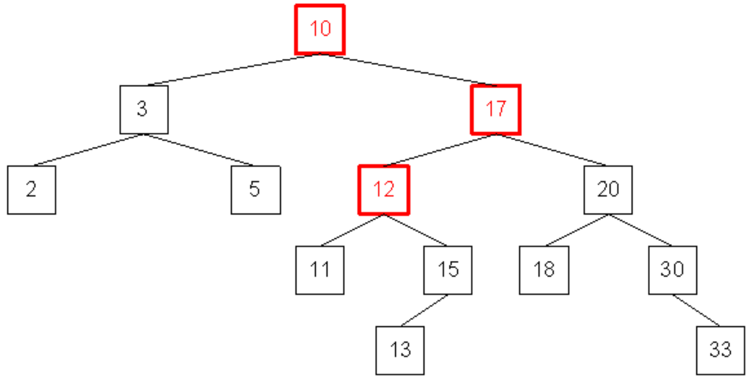


What type of rotation should be performed to fix the imbalance?

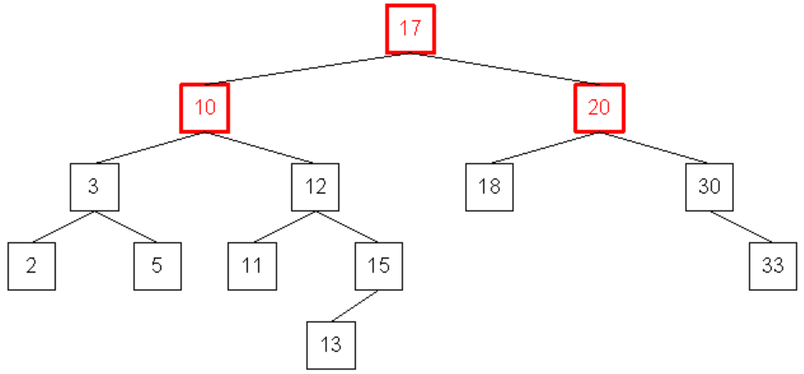




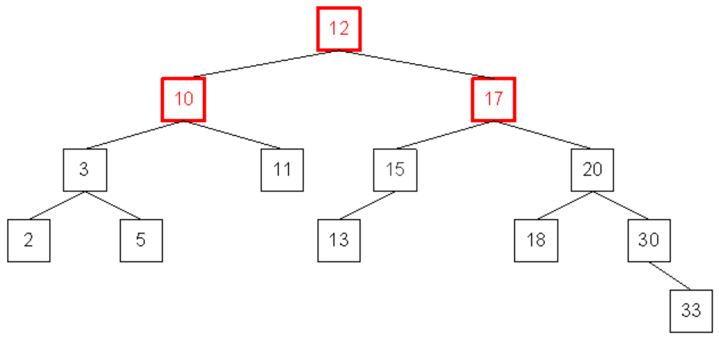
(1) A single left rotation with the highlighted nodes



(2) OR a right-left rotation with the highlighted nodes



After the single left rotation



After the right-left rotation

Pros and Cons of AVL Trees

- **Arguments for AVL trees:**
  - Fast search, insert, delete operations because trees are always balanced.
  - The height balancing adds no more than a constant factor to the speed of insert and delete.
- **Arguments against AVL trees:**
  - More **space** for **height** field
  - Rebalancing takes a little **time** -> slower than ordinary BST on random data
  - Most large searches are done in database-like systems on disk and use other structures (e.g., B-trees).

Useful links

- AVL Tree animation
  - https://www.cs.usfca.edu/~galles/visualization/AVLtree.html
  - http://www.cs.armstrong.edu/liang/animation/web/AVLTree.html
- AVL Tree Insertion https://www.youtube.com/watch?v=rbg7Qf8GkQ4
- The AVL Tree Rotations Tutorial http://www.cise.ufl.edu/~nemo/cop3530/AVL-Tree-Rotations.pdf
- AVL Tree insert algorithm http://www.geeksforgeeks.org/avl-tree-set-1-insertion/
- AVL Tree delete algorithm http://www.geeksforgeeks.org/avl-tree-set-2-deletion/