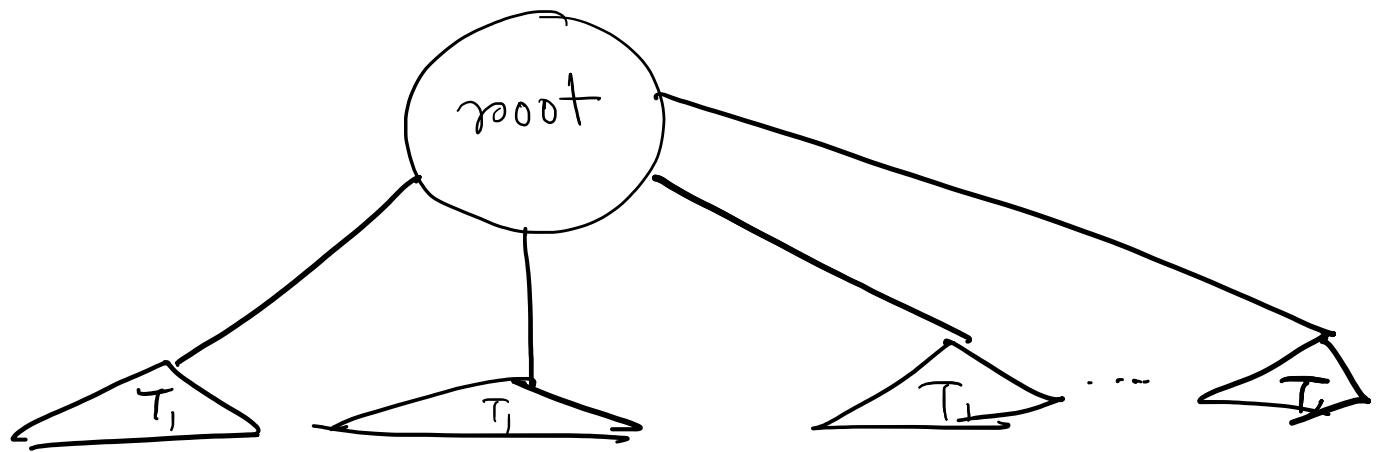


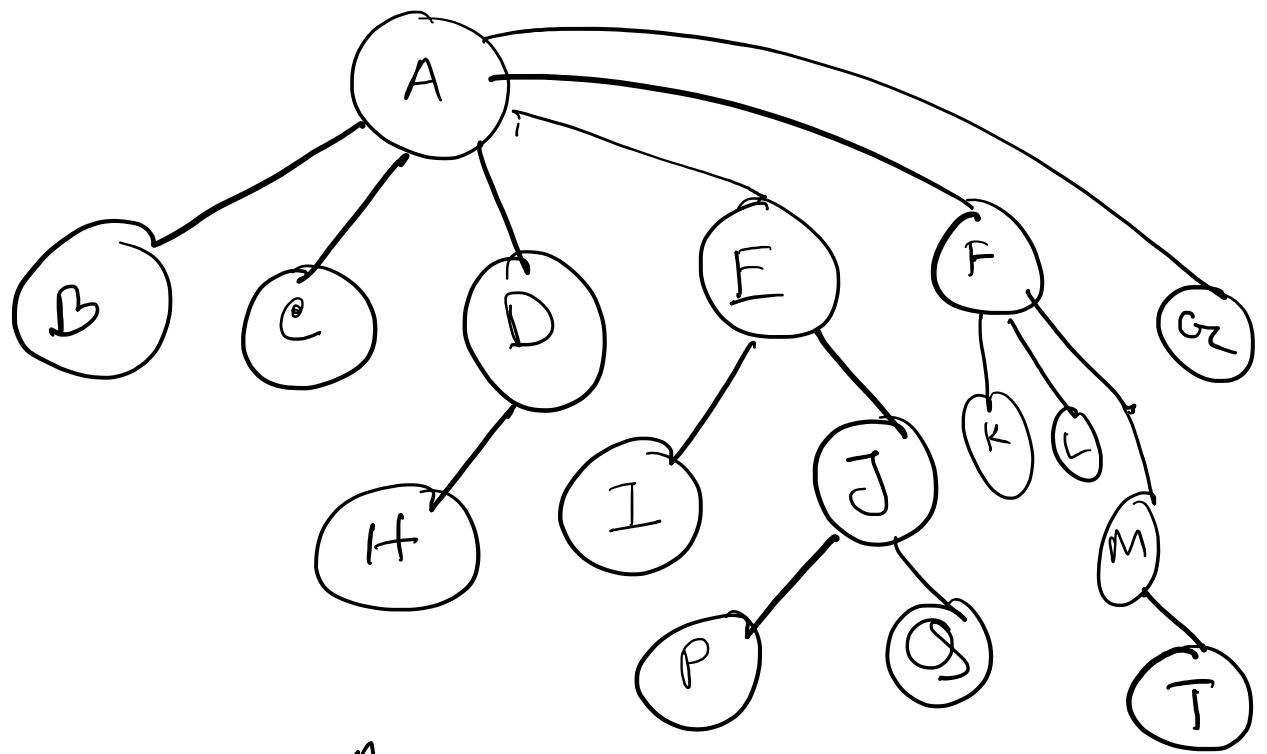
# Chapter 4

Tree



Collection of nodes and edges.

Reursively,  
a root and its subtrees



$\text{Parent}(D) = A$

$\text{children}(D) = \{H\}$

$\text{children}(E) = \{I, J\}$

$\text{Ancestors}(J) = \{E, A\}$

$\text{Decendents}(F) = \{K, L, M, T\}$

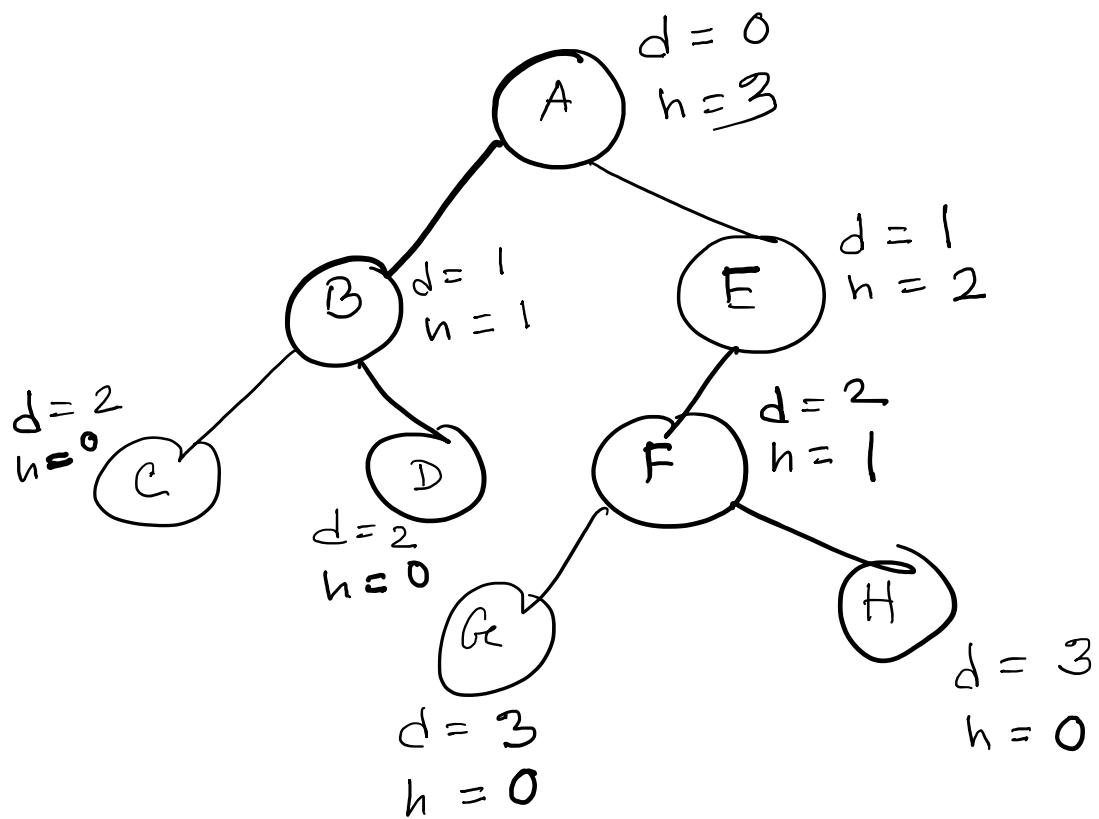
$\text{Siblings}(B) = \{C, D, E, F, G\}$

## Definitions

- Leaf node: node having no children
- Size: total number of nodes in tree
- $\text{path}(s, d) = \{n_1, n_2, n_3, \dots, n_k : n_1 = s, n_k = d, \text{parent}(n_{i+1}) = n_i \text{ for } 1 \leq i \leq k\}$
- length of a path: is the number of edges along the path
- Level: all nodes at a given depth share a level.

→ Depth: distance of node  $n$  from root in terms of the number of edges

→ Height: maximum distance of node  $n$  from its farthest leaf.



→ Width: number of nodes on the level containing the most nodes.

# Binary Tree

$$\min \text{ height} = \lceil \log_2 n \rceil$$

$$\max \text{ height} = n - 1$$

$n$  = total number of nodes

Array representation

$$\text{parent}(i) = (i-1)/2$$

$$\text{left}(i) = 2i + 1$$

$$\text{right}(i) = 2i + 2$$

Tree traversal:

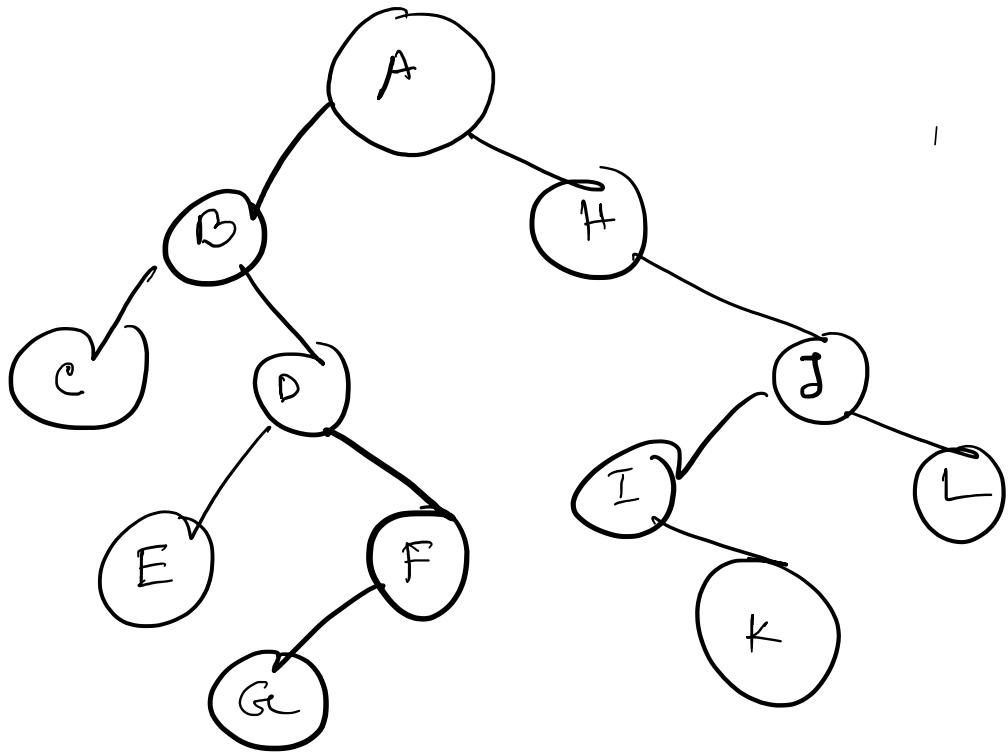
→ Breadth-first: visit nodes level by level, from left to right

→ Depth-first:

→ pre-order: root, left, right

→ in-order: left, root, right

→ post-order: left, right, root



Breadth-first: A B H C D J E F I L G K

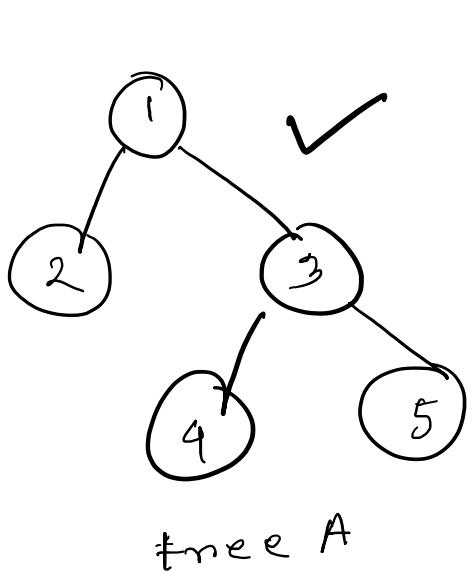
Pre-order: A B C I E F G H J K L

In-order: C B E D G F A H I K J L

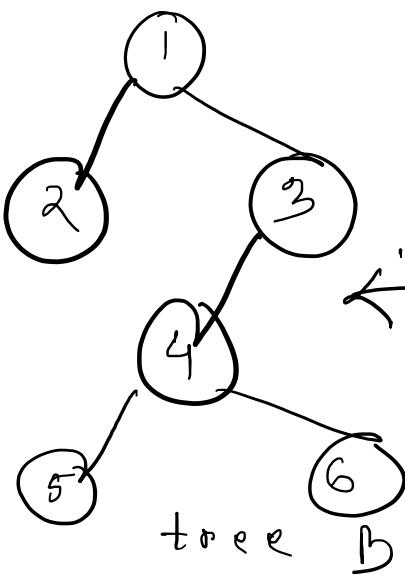
Post-order: C E G F D B K I L J H A

## Full Binary Tree:

Each interior node (non-leaf)  
node has two children



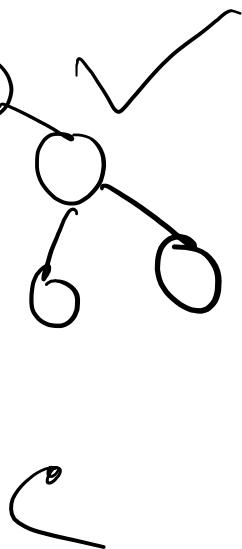
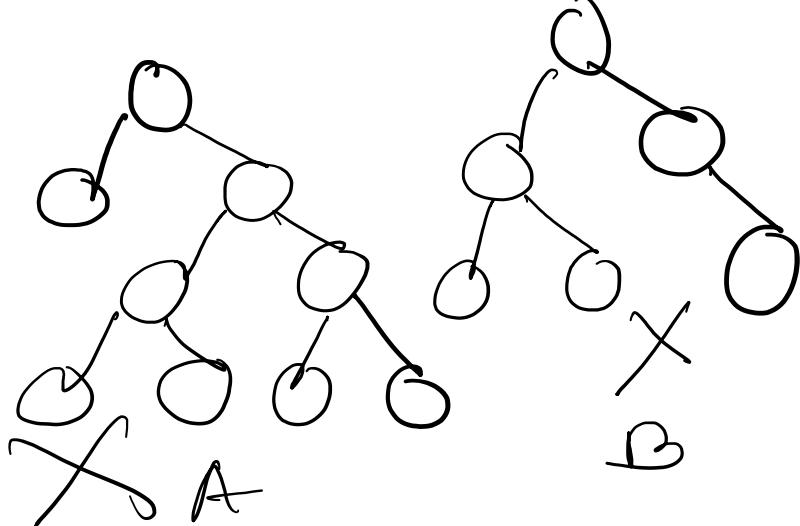
tree A



not a  
Full B. Tree

## Perfect Binary Tree

A full binary tree with all  
leaves at the same level.



C

# Binary Search Tree

## Operations

① contains

↳ boolean

② find Min / find Max

↳ for min keep going to the left

↳ for max keep going to the right

③ insert

↳ if  $x$  is not found in the tree (`contain` returns false),  
insert in appropriate path

④ delete

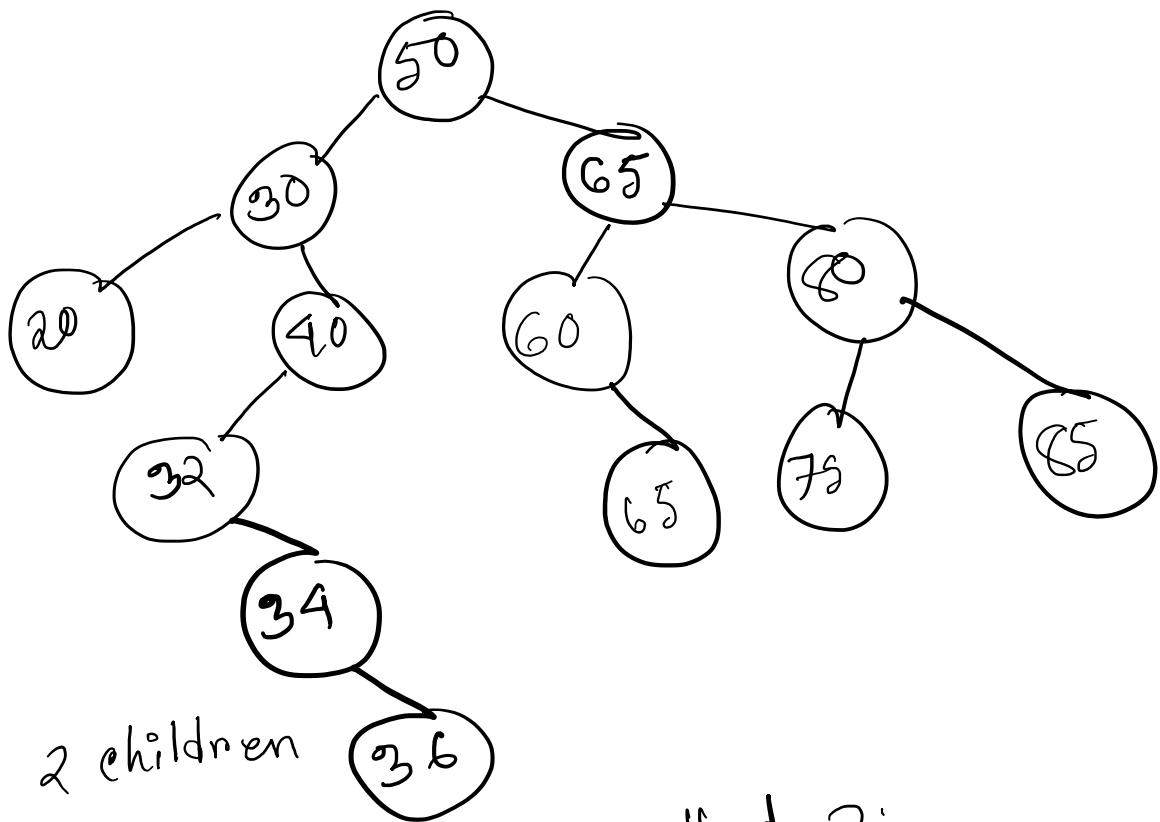
3 cases

case 1: leaf node

→ delete the node

case 2: 1 child

↳ delete the node and connect with the parent of the deleted node.



Case 3: 2 children

method 1:

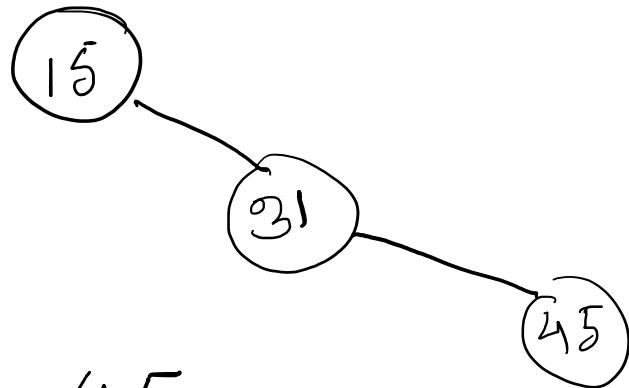
- Find min of the right subtree
- copy the value in target node
- remove duplicate

method 2:

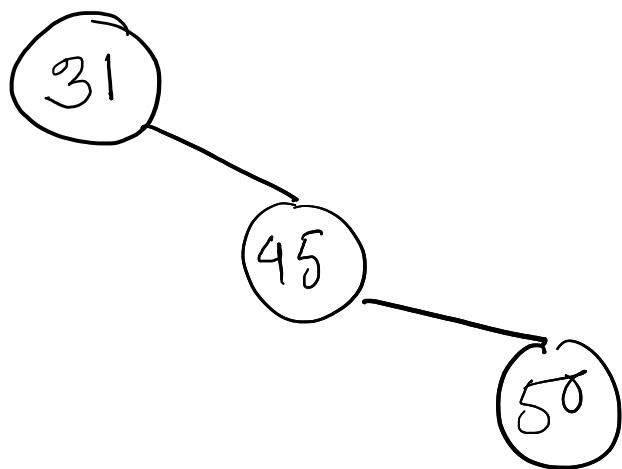
- Find max of the left subtree
- and c) same as method 1.

# Balancing

15, 31, 45



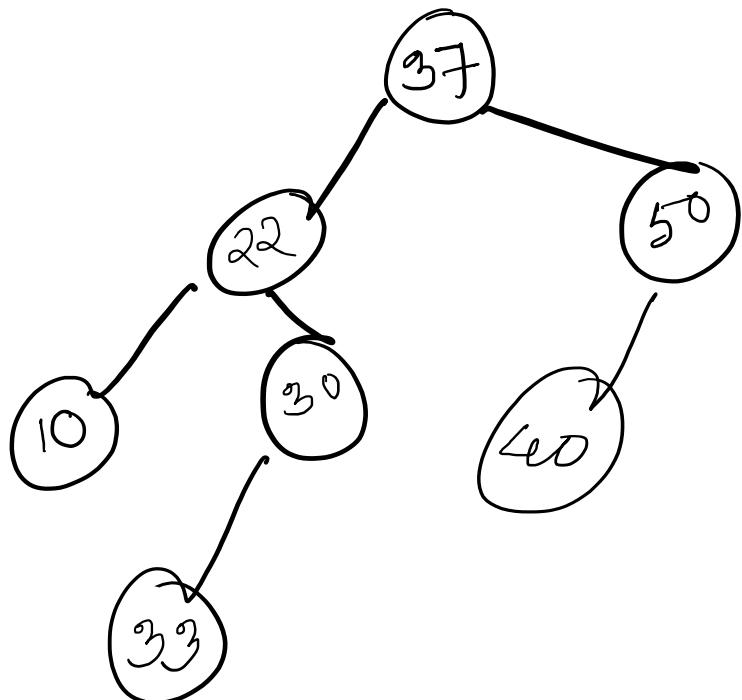
31, 15, 45



# AVL Trees

A binary search tree where for every node the height of the left and right subtrees can differ by at most 1.

AVL = Adelson-Velskii and Landis

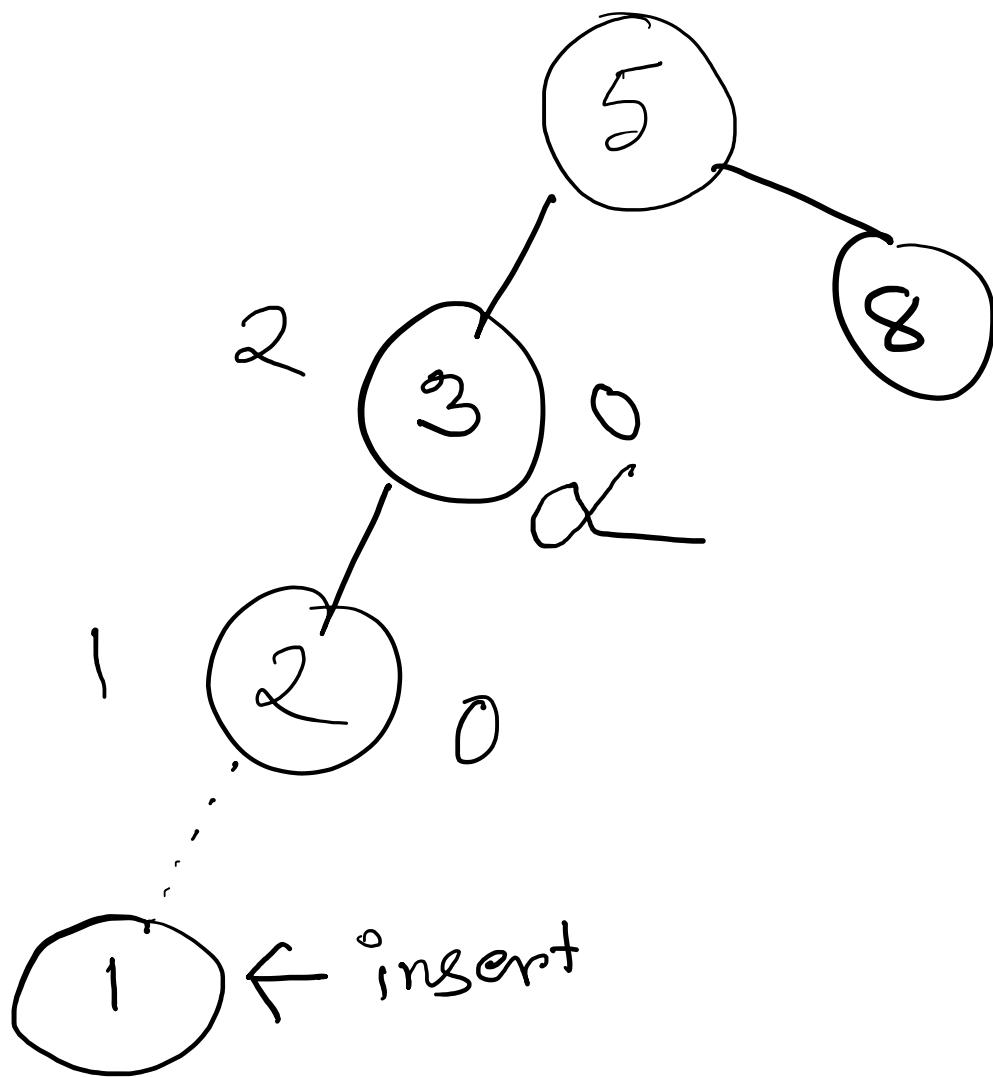


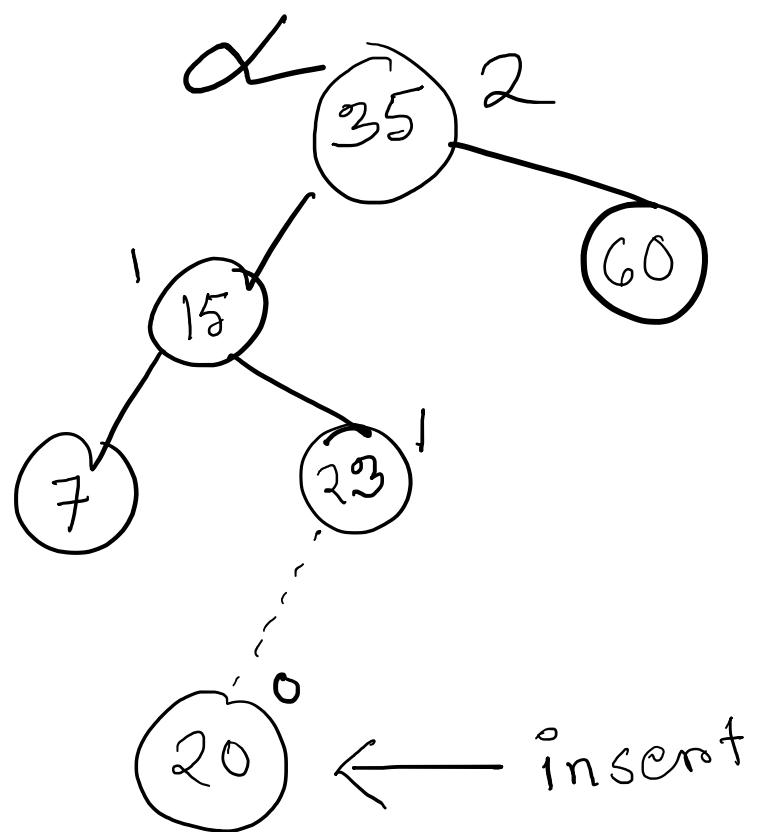
Four cases of unbalanced

insertions

---

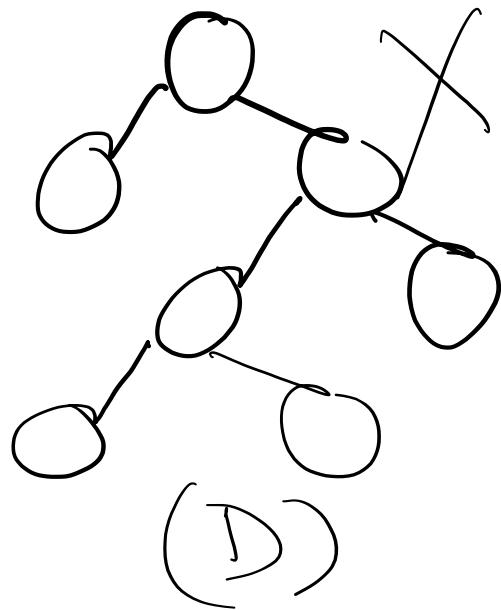
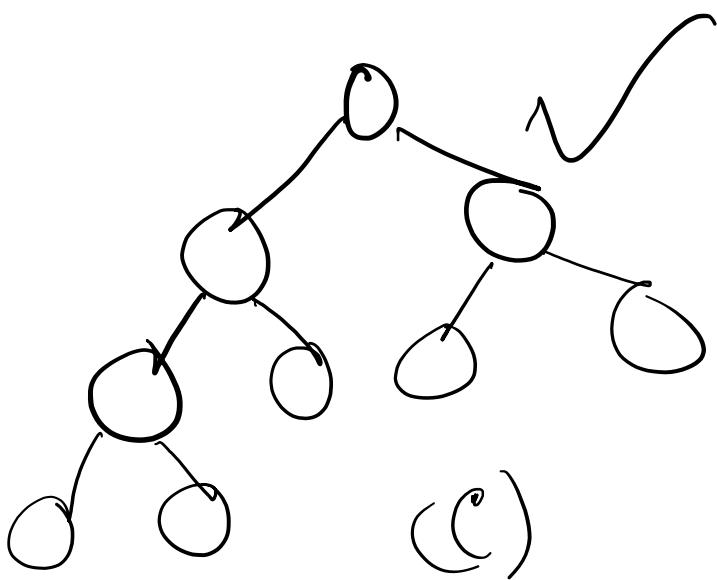
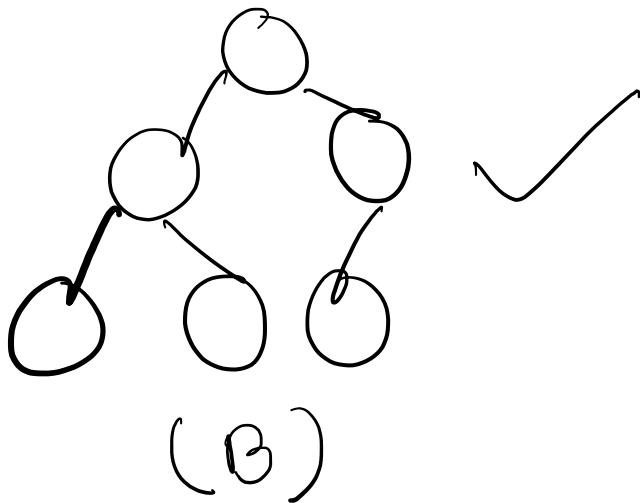
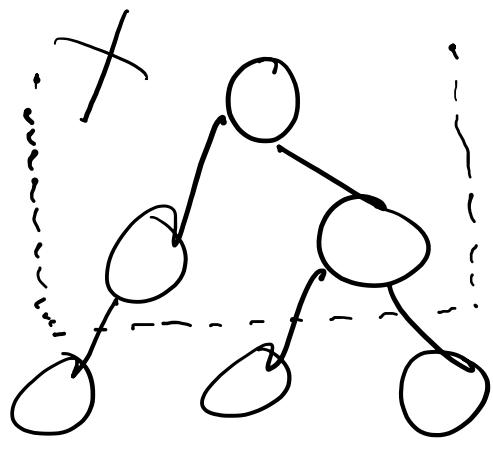
Let  $\alpha$  be the node  
that become unblanced



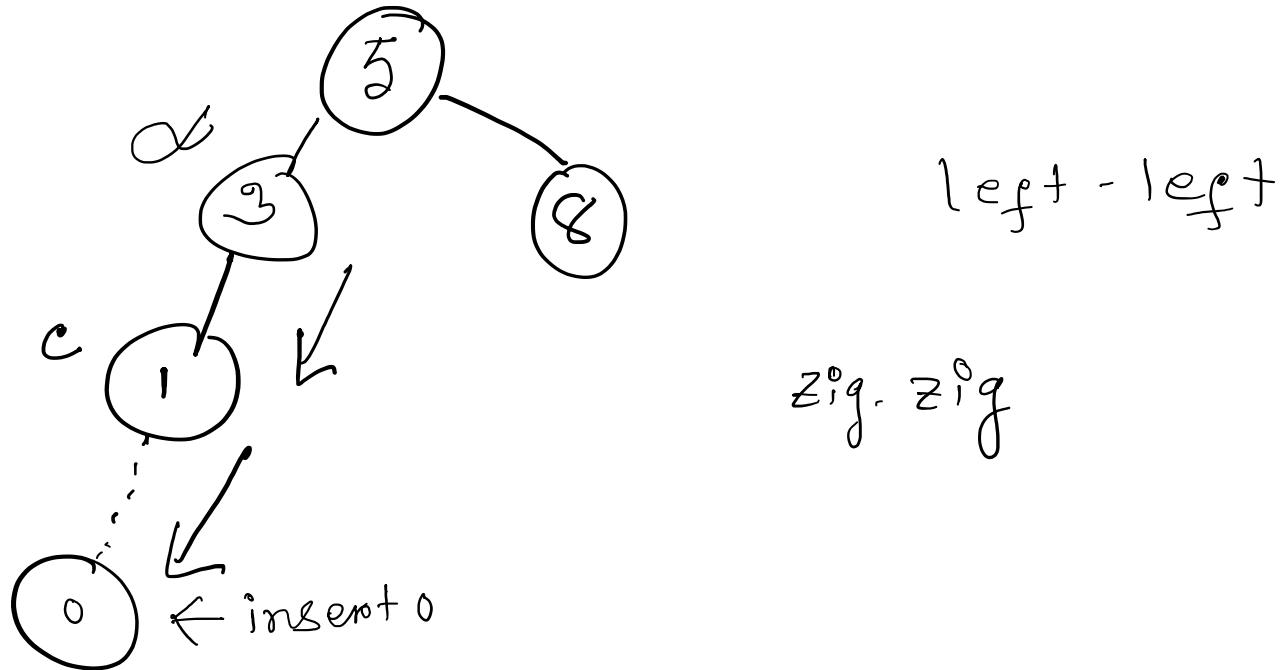


# Complete Binary Tree

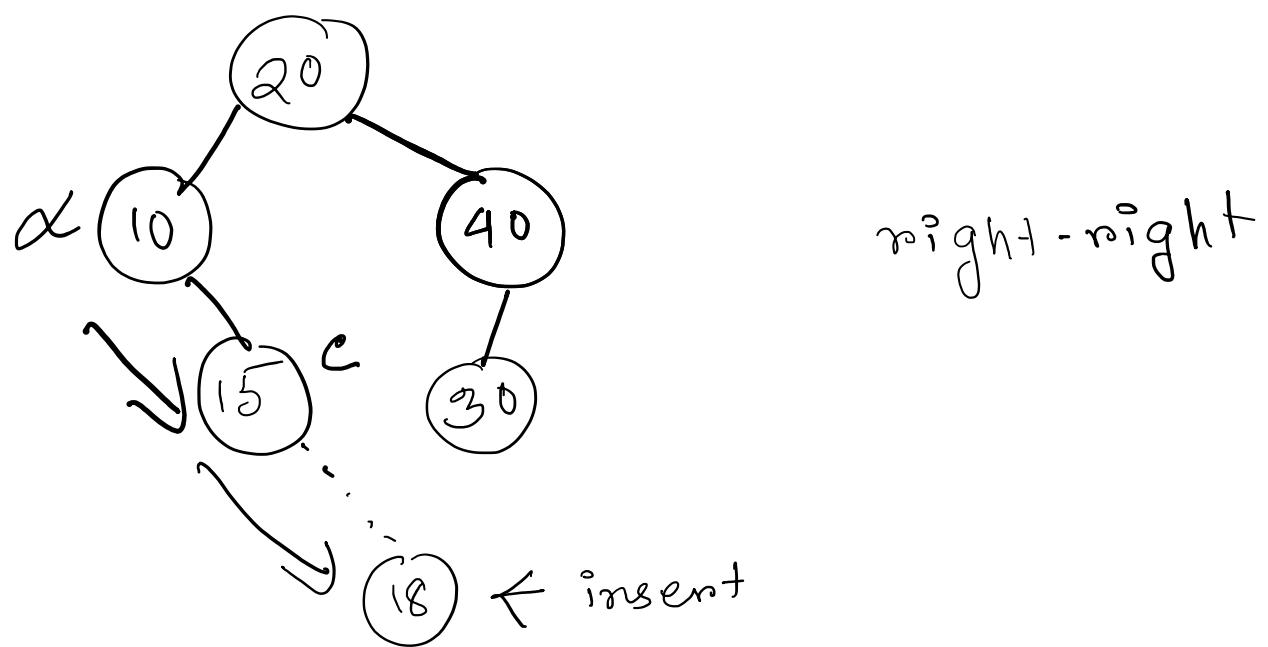
A binary tree of height  $h$ , where the tree is a perfect binary tree down to height  $h-1$  and the nodes at the lowest level are filled from left to right.



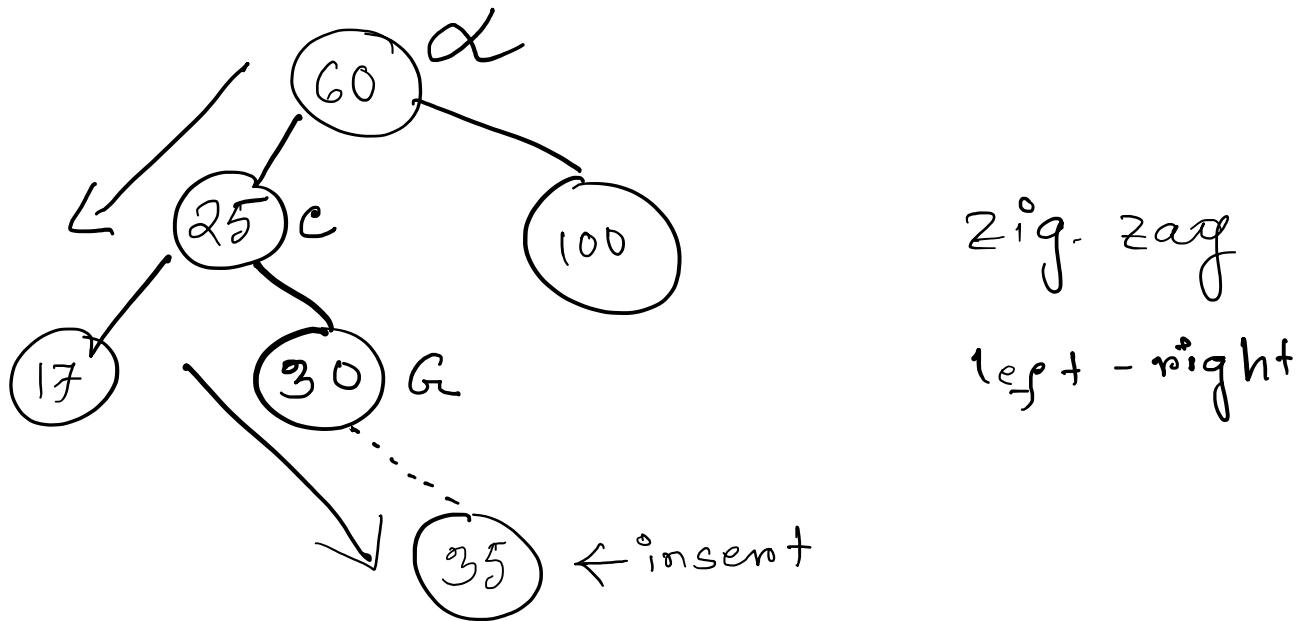
Case 1: insertion into the left subtree of the left child ( $c$ ) of  $\alpha$



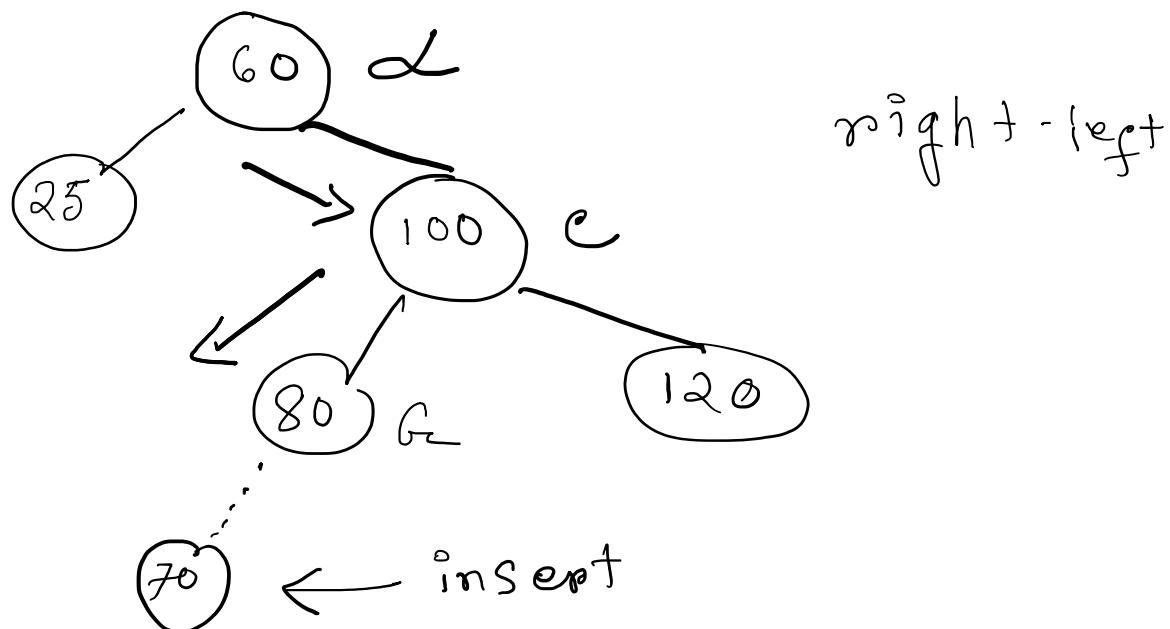
Case 4: Insertion into the right subtree of the right child of  $\alpha$



Case 2: insertion into the right subtree ( $G_e$ ) of the left child ( $c$ ) of  $\alpha$ .



Case - 3: insertion into the left subtree of the right child ( $c$ ) of  $\alpha$ .

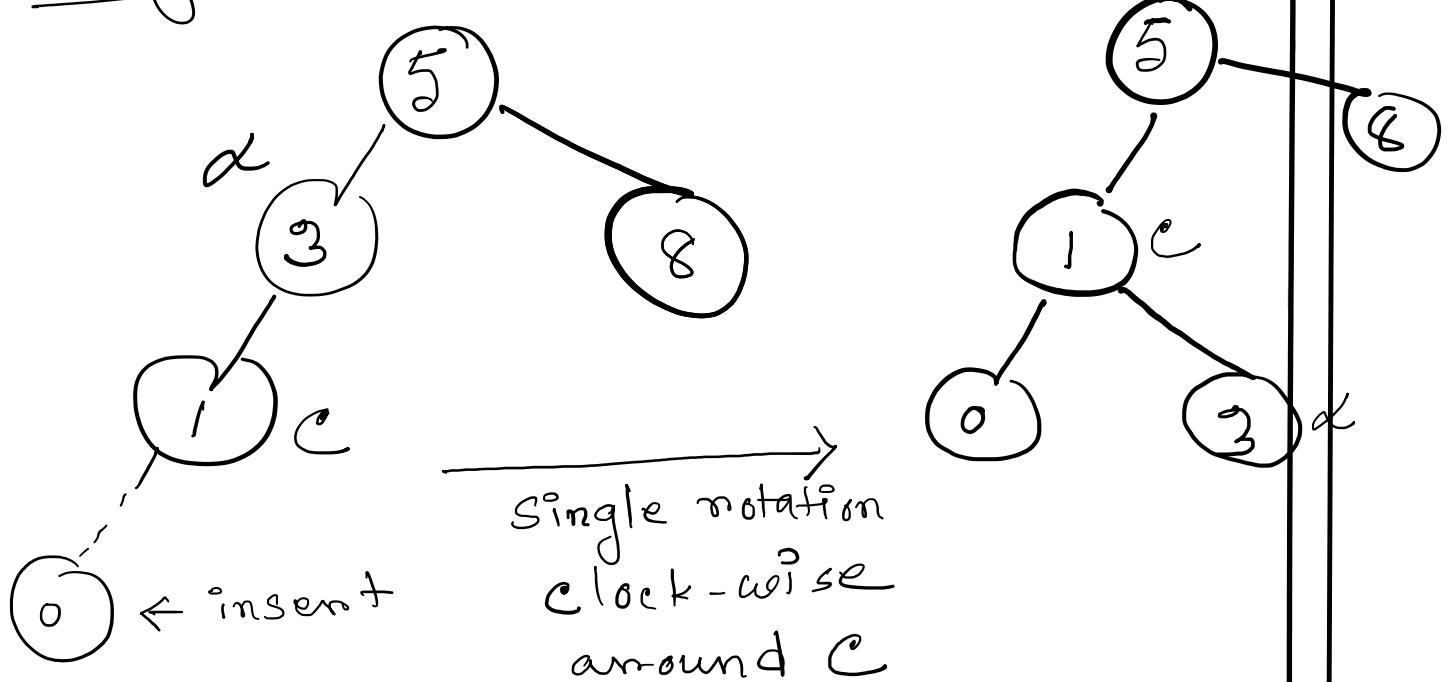


# Balancing Strategies:

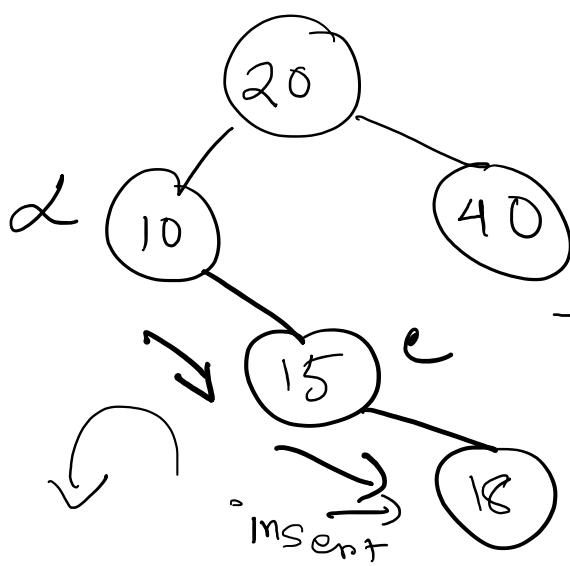
1. Single Rotation: For case 1 and 4 (left-left or right-right)

2. Double-Rotation: For case 2 and 3 (left-right or right-left)

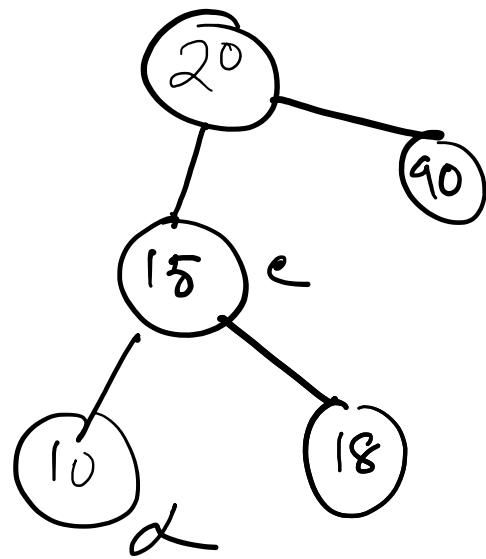
## Single Rotation:



## Case 1

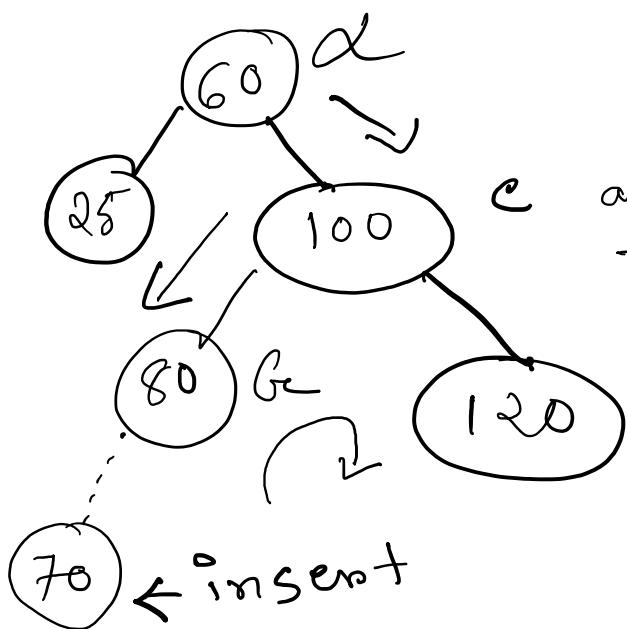


Single notation  
 counter-clock  
 around c

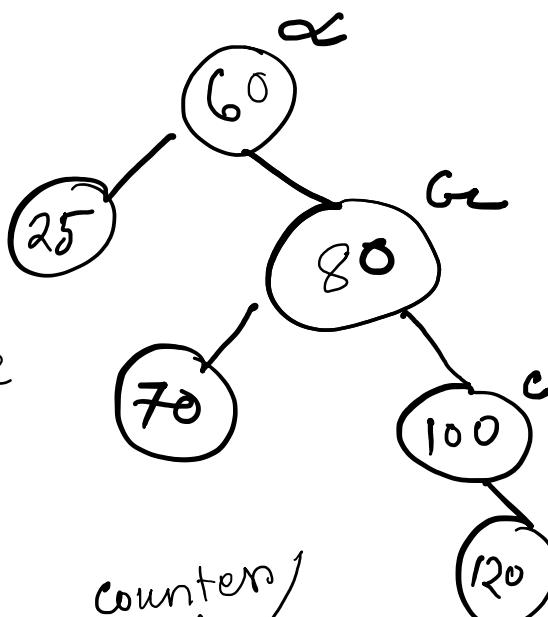


Case 4

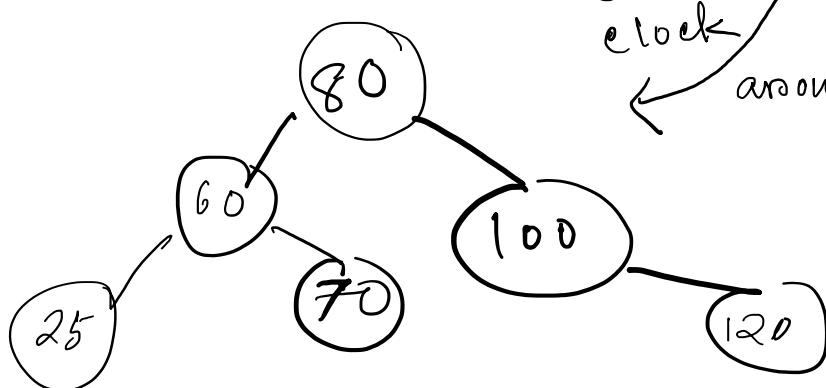
Double Rotations:

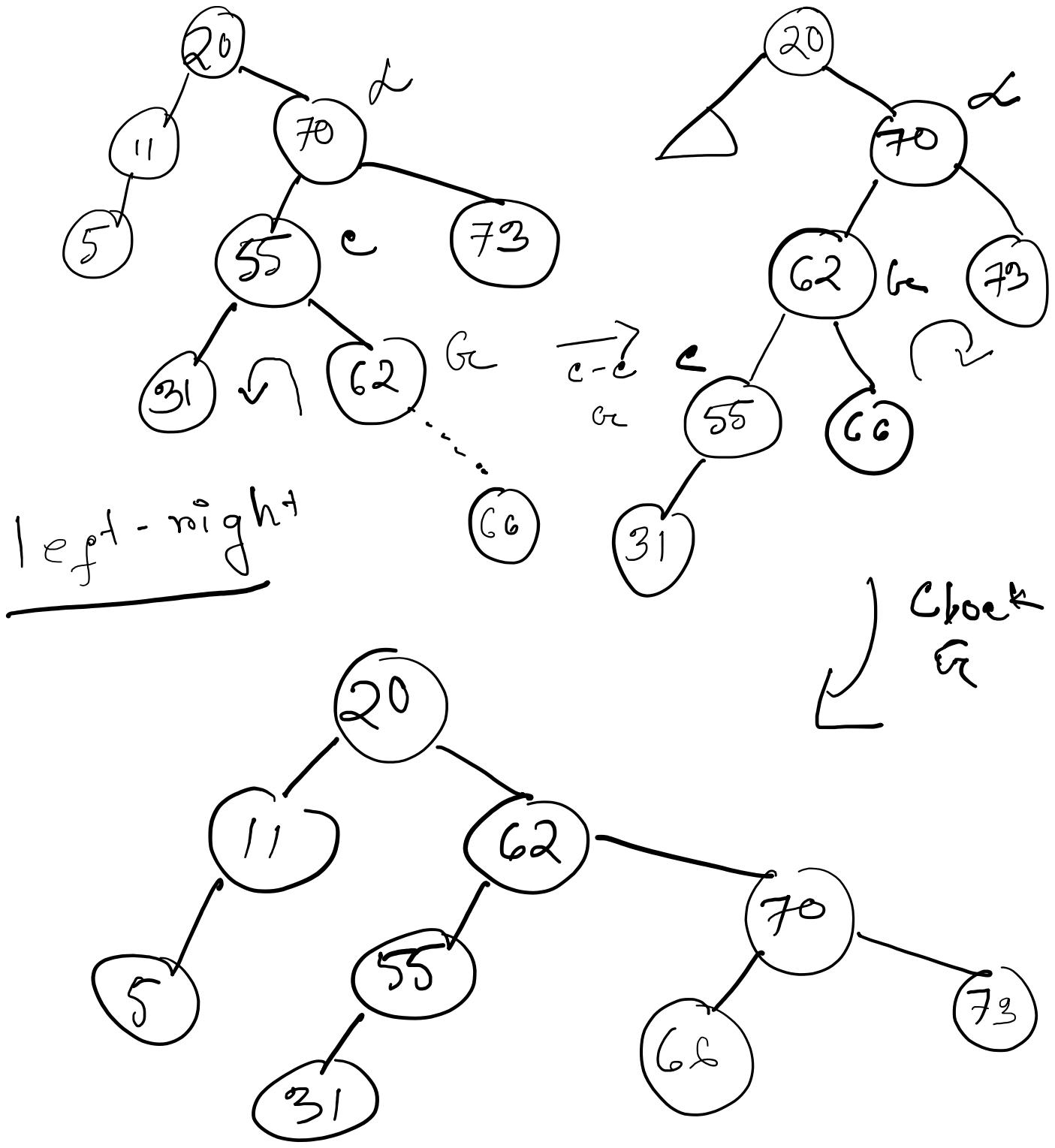


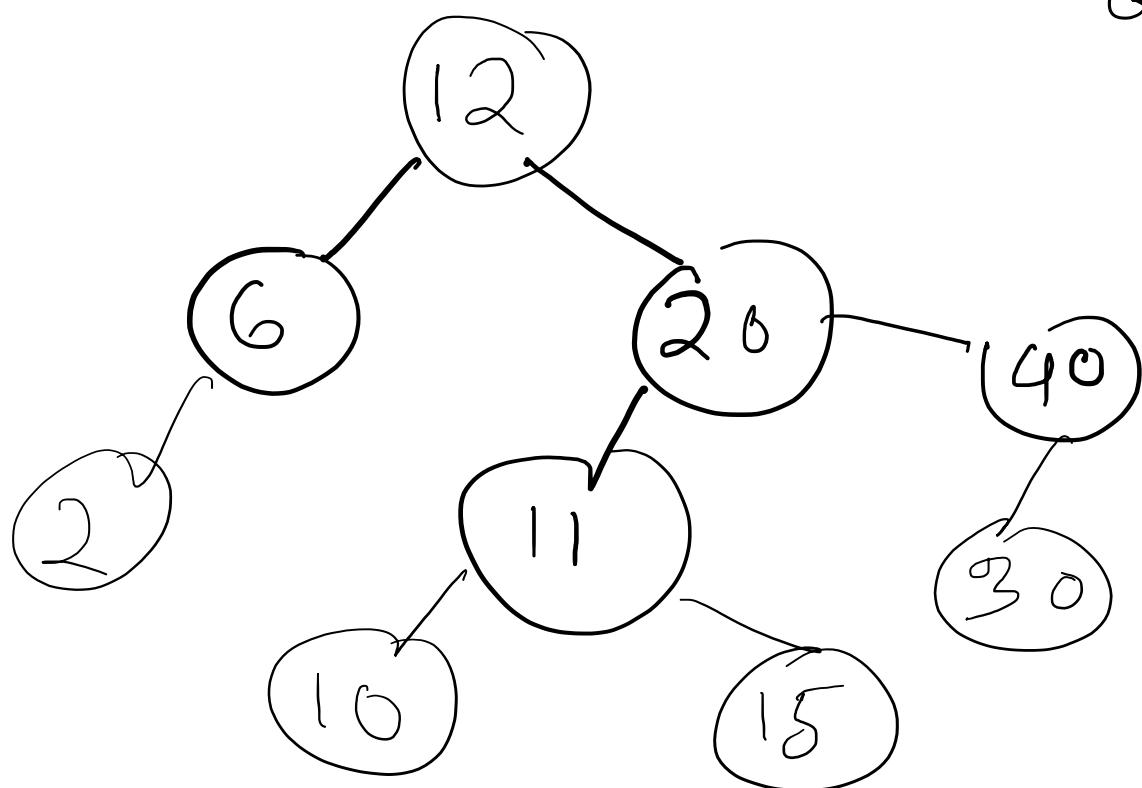
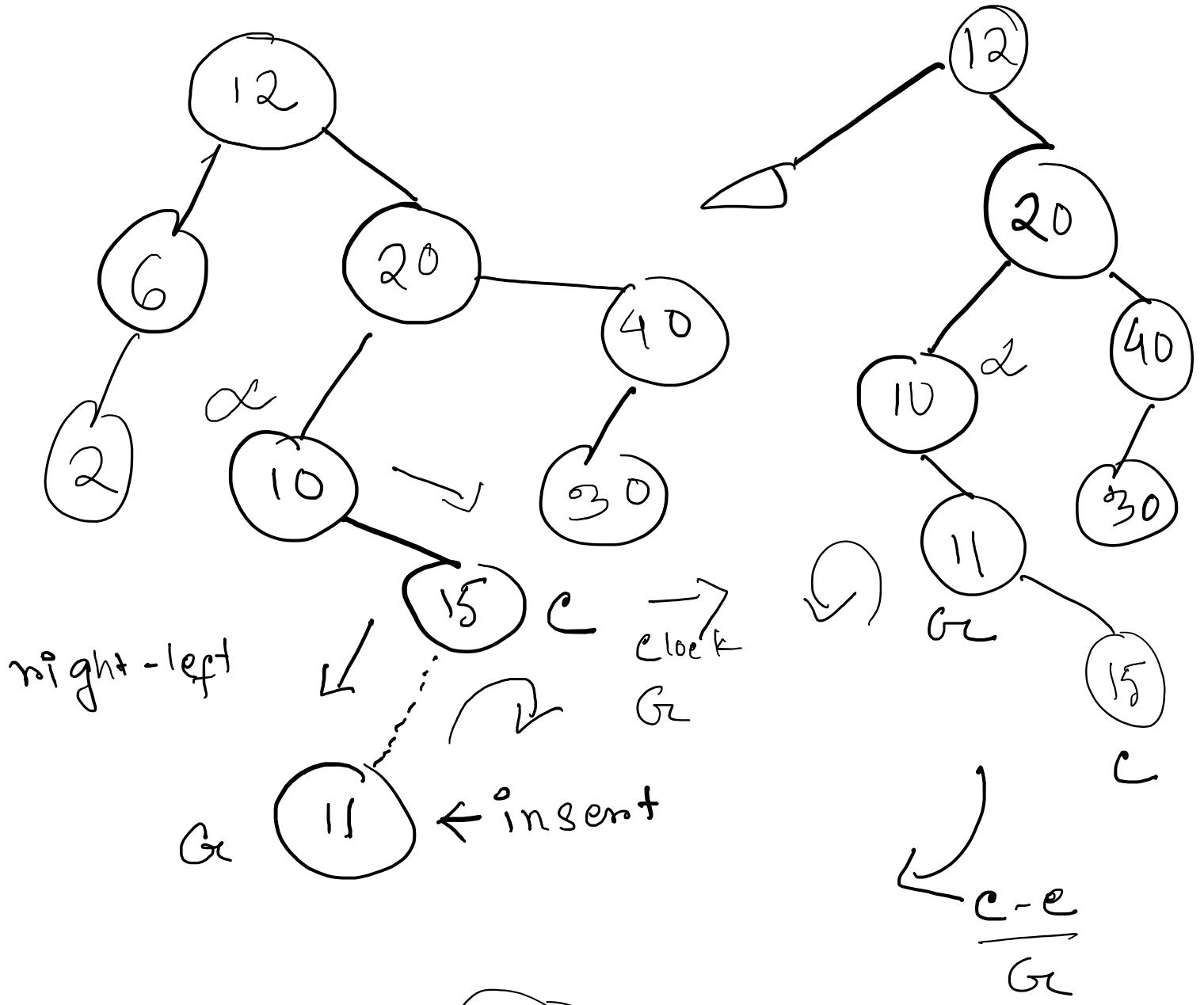
around g  
 clockwise



case 3





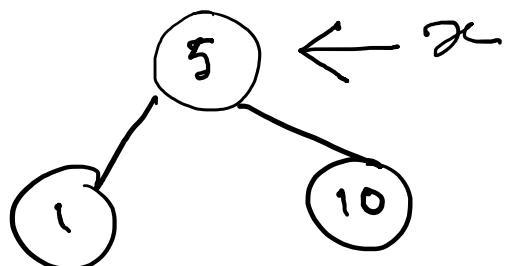


# Splaying

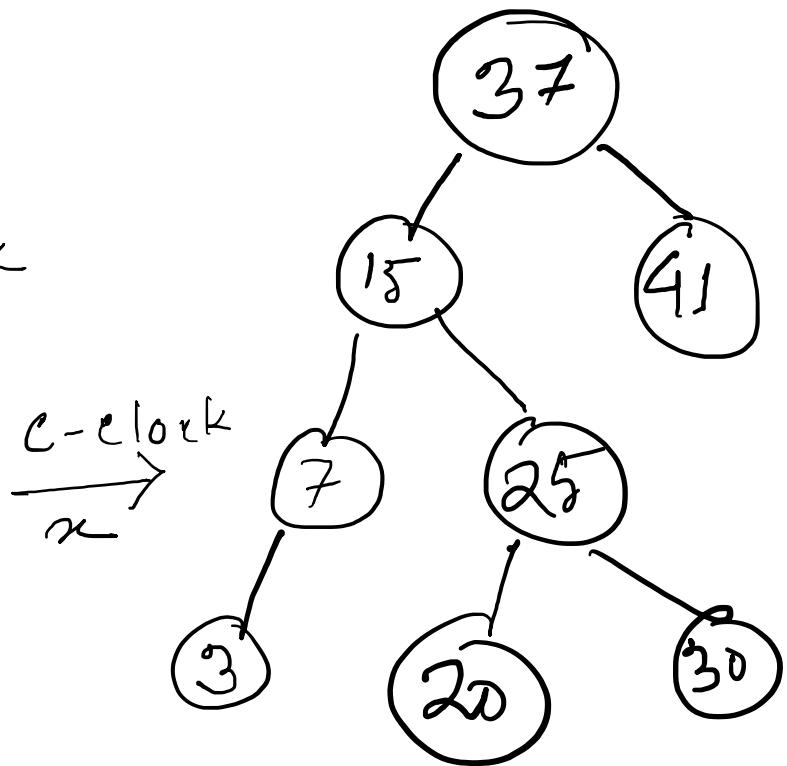
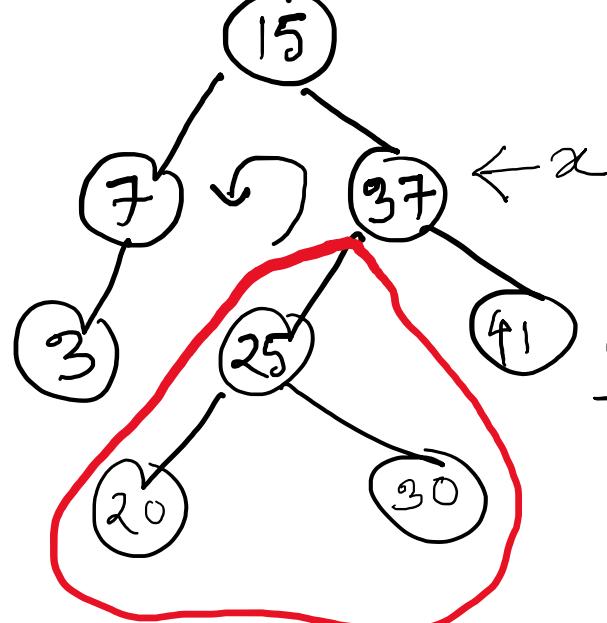
The basic idea of Splay tree is that after a node is accessed, it is pushed to the root by a series of AVL rotations.

Let the last accessed node be  $x$ .

case 1:  $x$  is the root of the tree, do nothing.



Case 2: Parent of  $x$  is the root of the tree.  
 Simply note  $x$  and the root

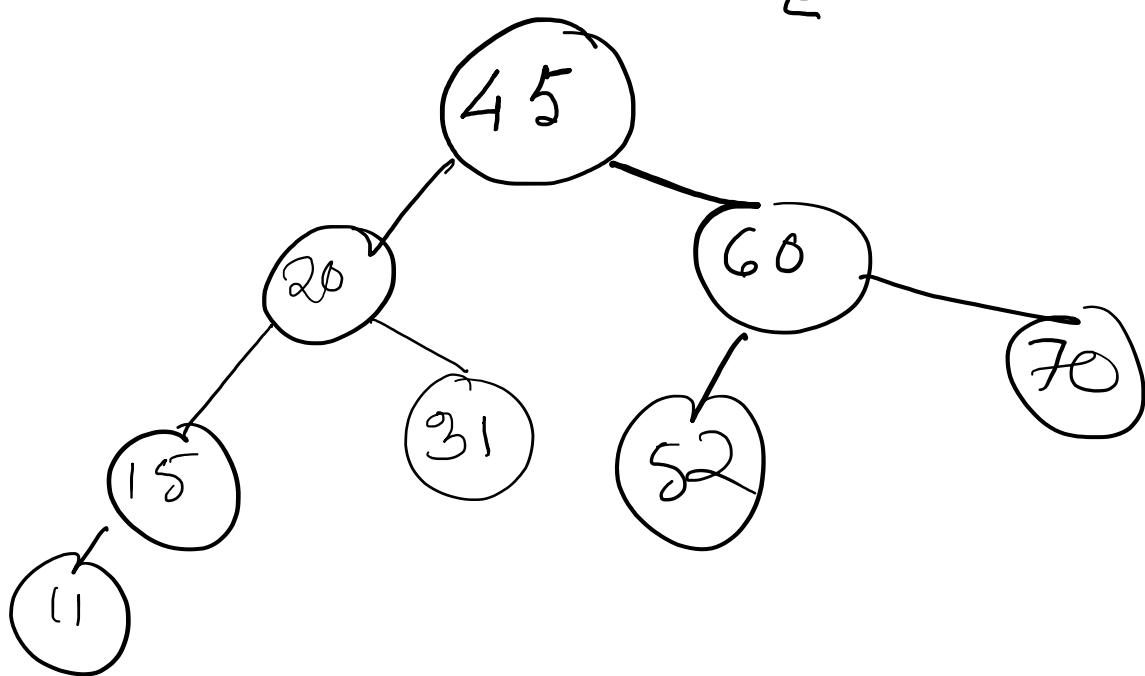
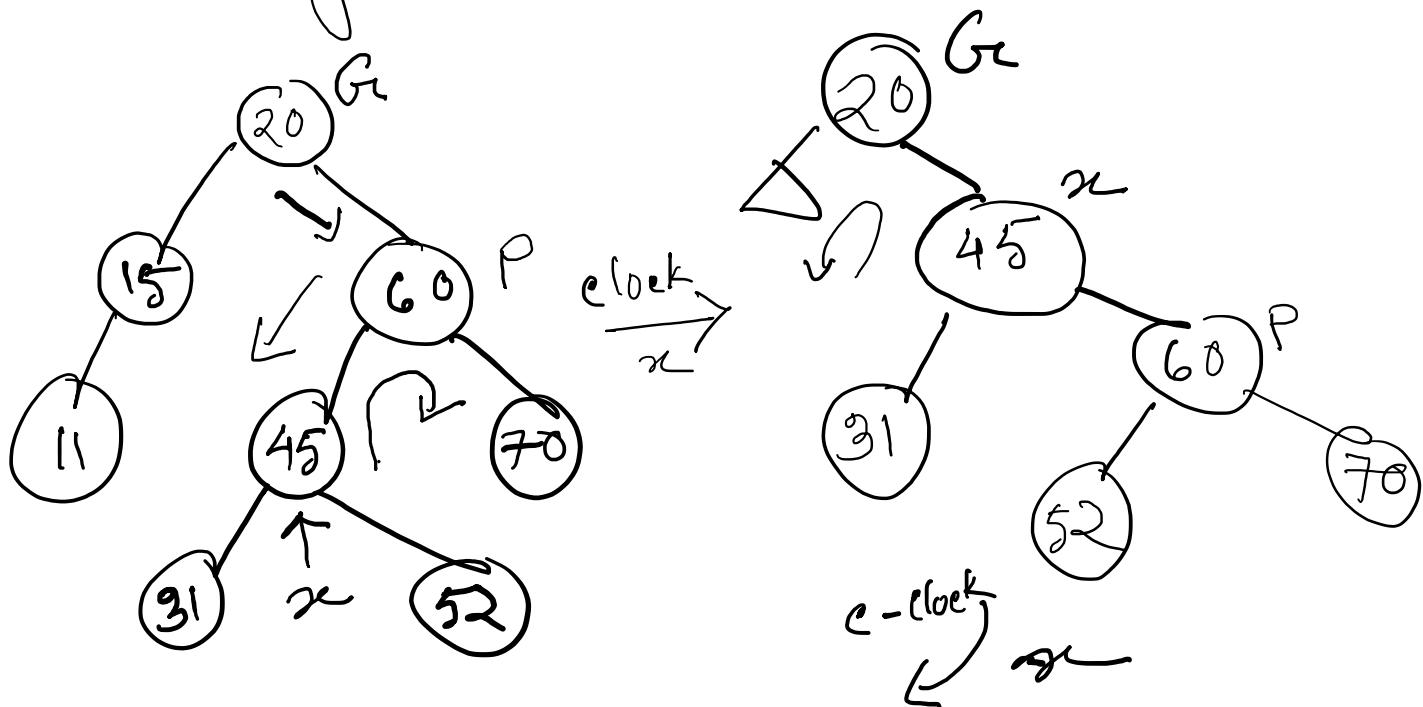


Case 3: Parent of  $x$ , denoted as  $P$  is not the root of the tree.  
 So,  $x$  also have a grand-parent.

Two cases -

(a) Zig-Zag:  $x$  is right child  
and  $P$  is the left child (or vice-versa)

Perform a double rotation  
exactly like AVL tree. (around  $x$ )



(b) Symmetric:  $x$  and  $P$  both are left children (on, right).  
 Rotate twice downwards starting from  $G_e$ .

