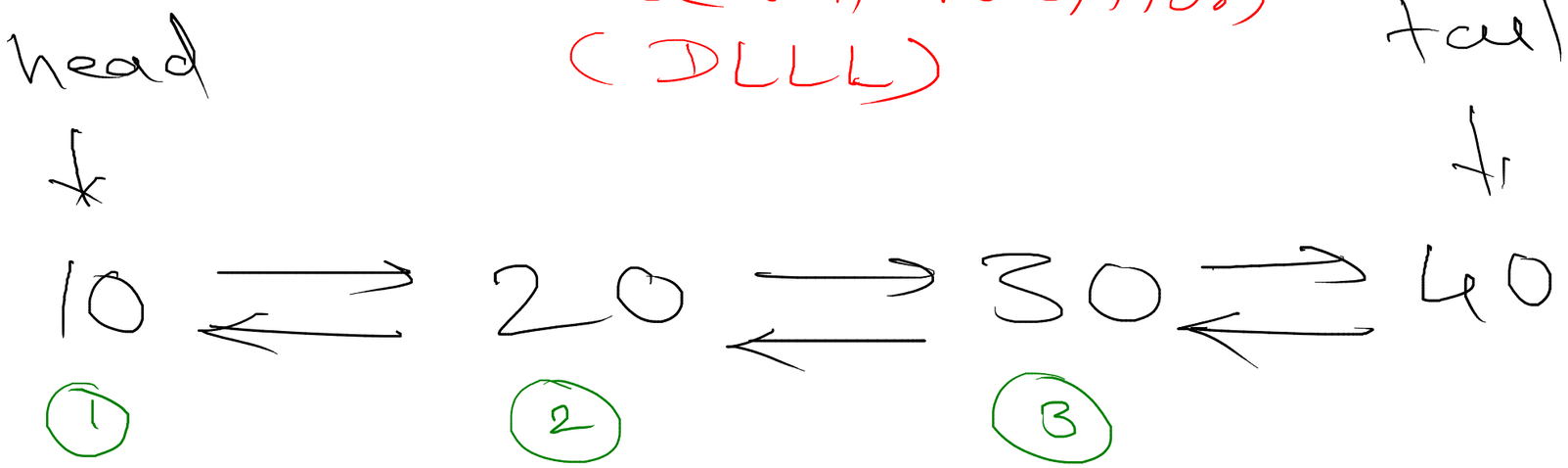
Traverse till Position (DLLL)

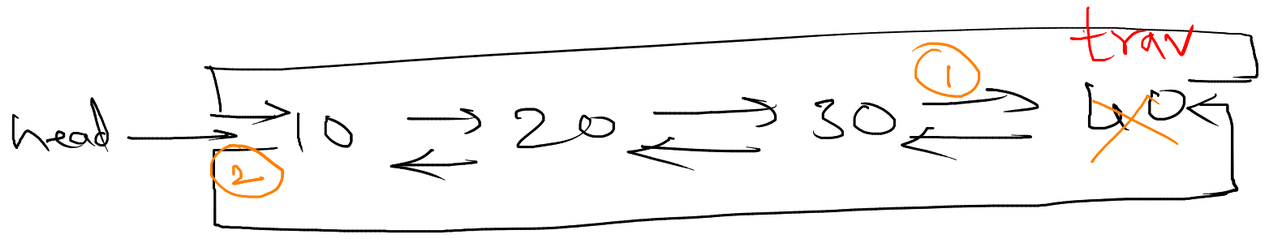



```
Node trav = head;
for(int i=1; i < pos-1; i++)      // 3-1=2
    trav = trav.next;
```

trav	i
&10	1
&20	2

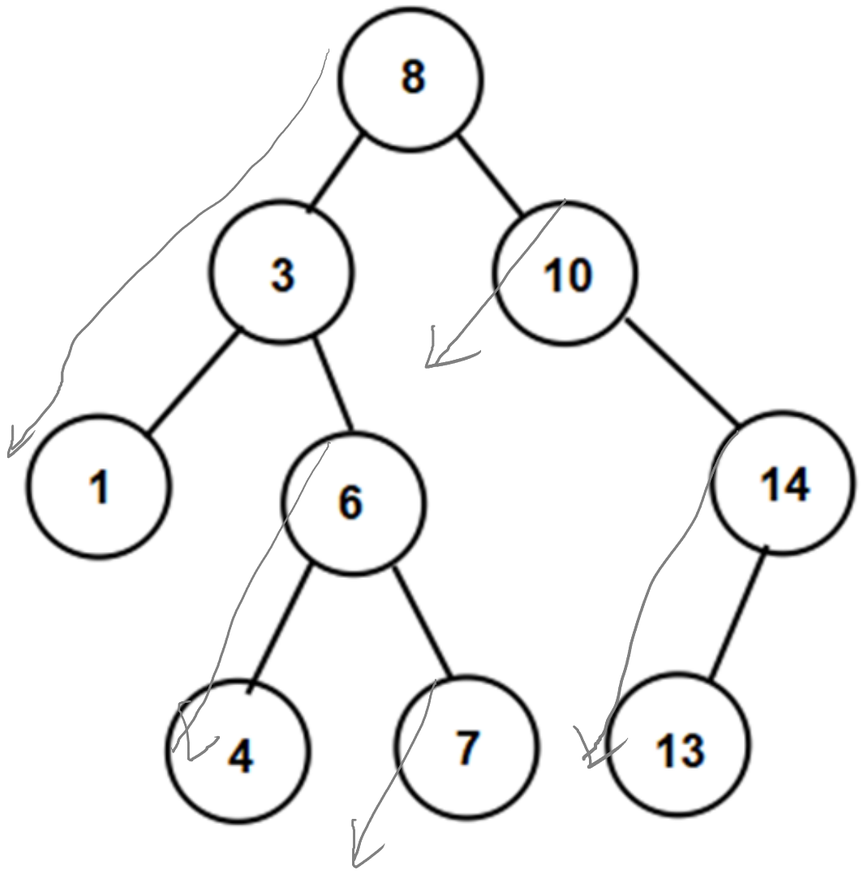
DCLL

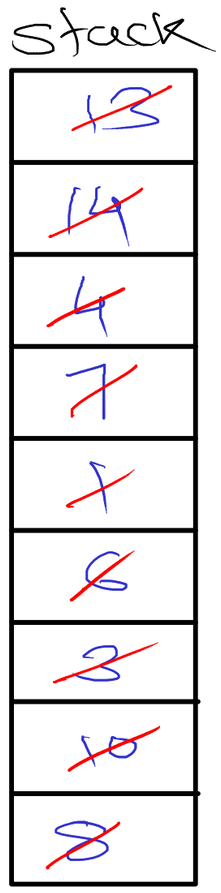
Delete from Last Position



1. pos-1 → next = pos+1
2. pos+1 → prev = pos-1

BST - DFS (Depth First Traversal)

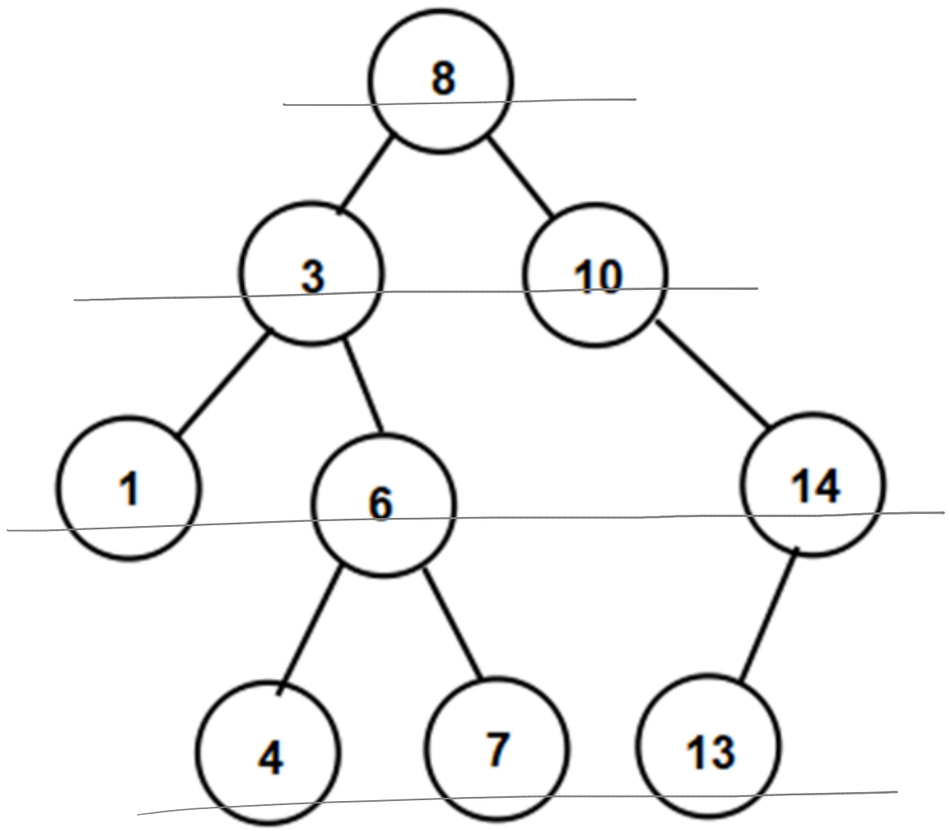


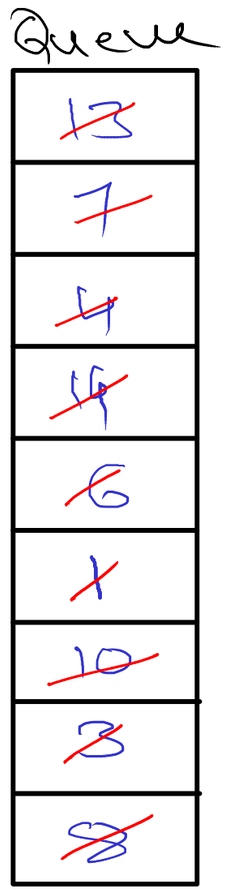


//1. push root on stack
//2. pop one node from stack
//3. visit(print) node
//4. if right exist, push it on stack
//5. if left exist, push it on stack
//6. while stack is not empty
//repeat ste 2 to 5

DFS Traversal : 8, 3, 1, 6, 4, 7, 10, 14, 13

BST - BFS (Bredth First Search)

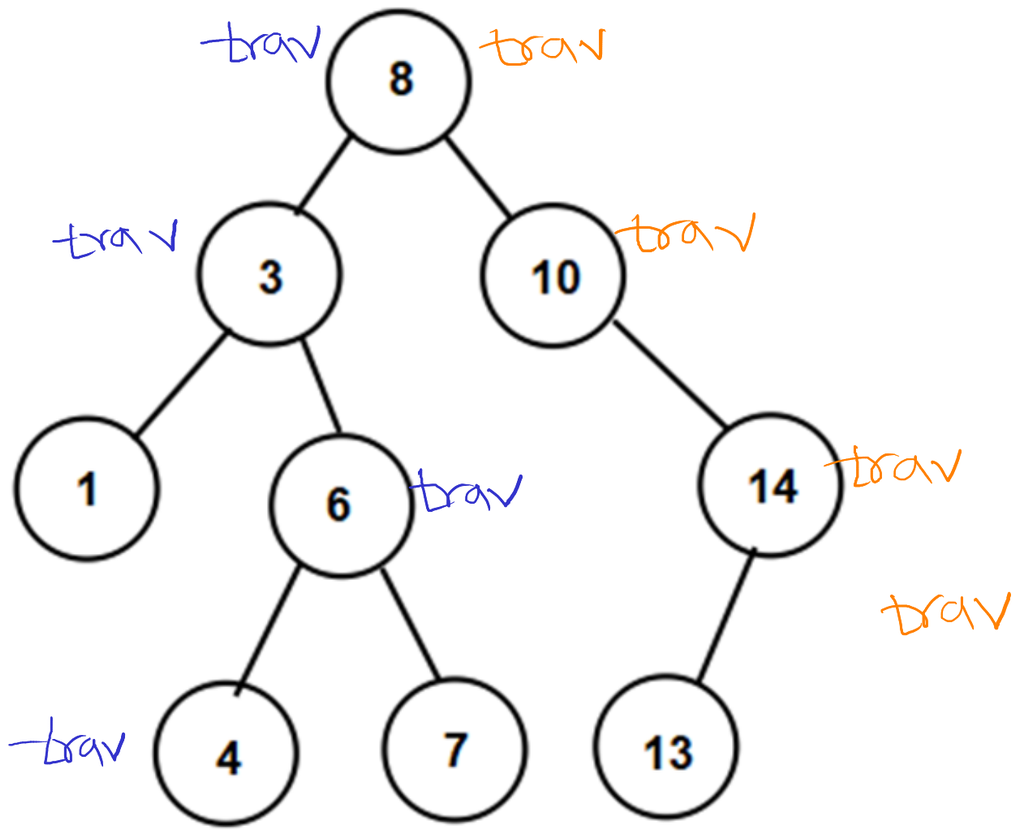




//1. push root on queue
//2. pop one node from queue
//3. visit(print) node
//4. if left exist, push it on queue
//5. if right exist, push it on queue
//6. while queue is not empty
//repeat ste 2 to 5

BFS Traversal: 8, 3, 10, 1, 6, 14, 4, 7, 13

BST - Binary Search

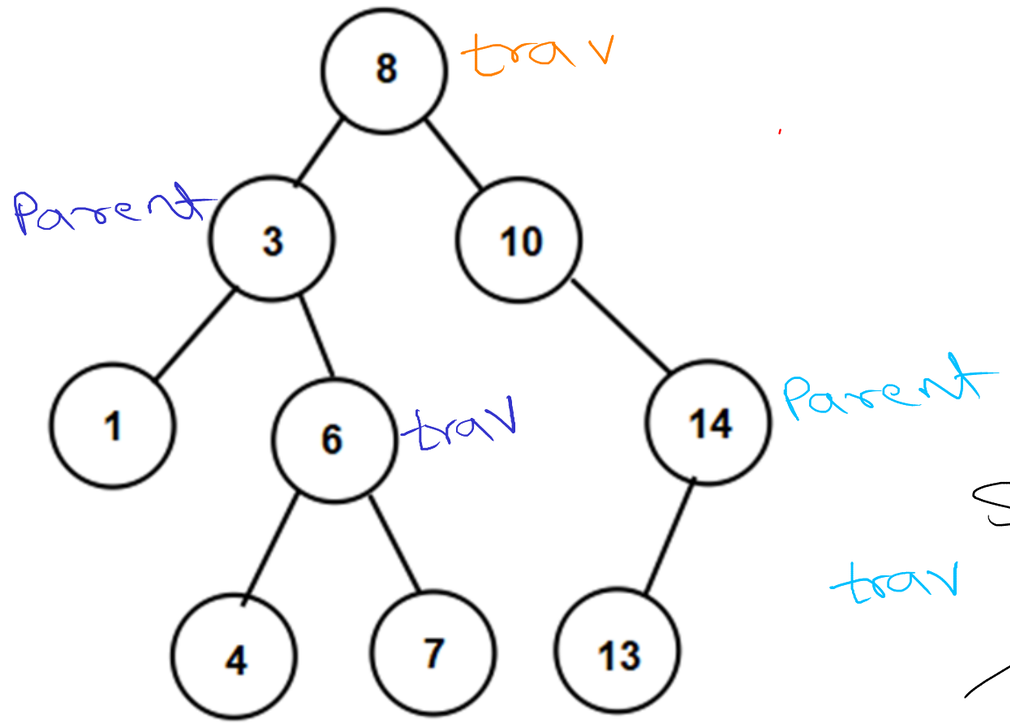


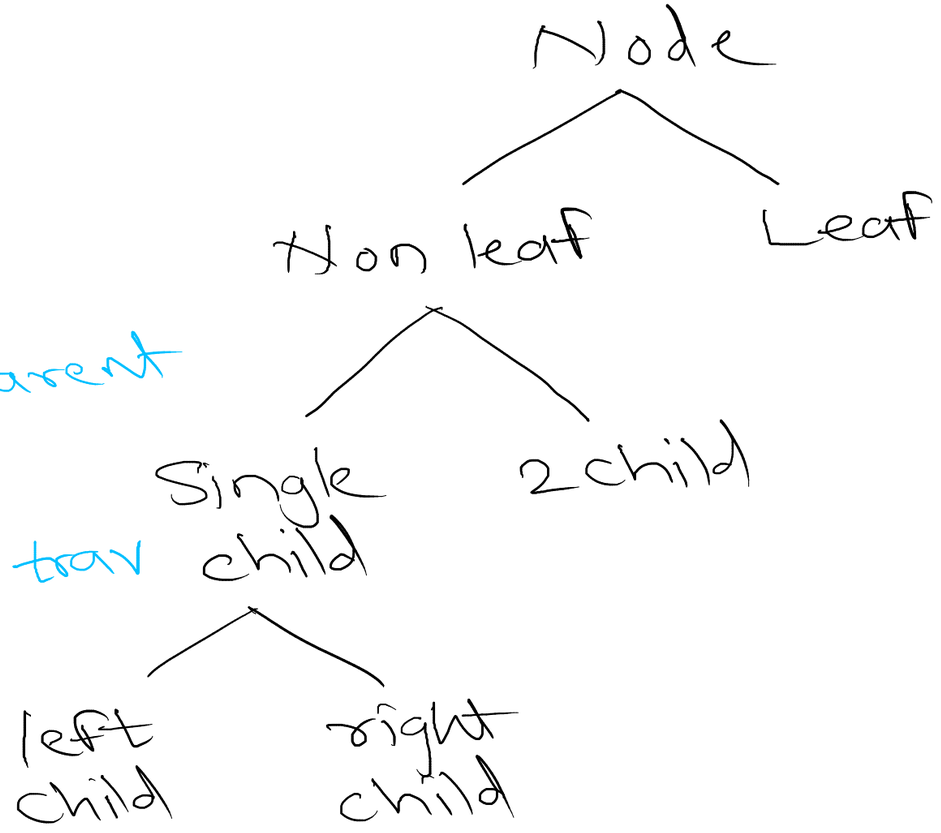
//1. start from root
//2. if key is equal to current data
//return current node
//3. if key is less than current data
// search key into left of current node
//4. if key is greater than current data
// search key into right of current node
//5. repeat step 2 to 4 till leaf nodes

key=4, key found
key=15, key not found

BST - Delete Node

Parent = null



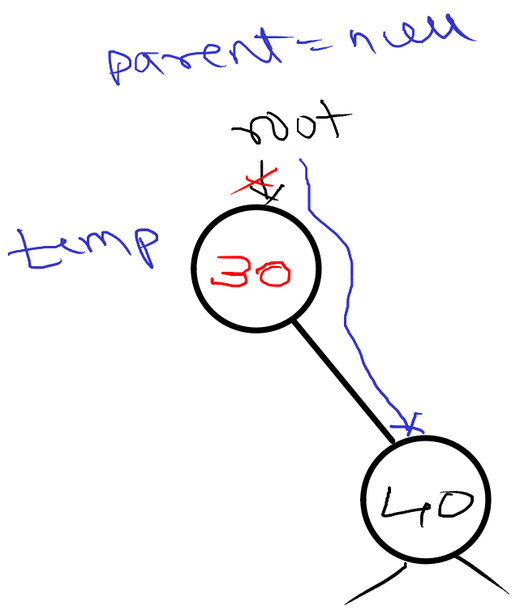


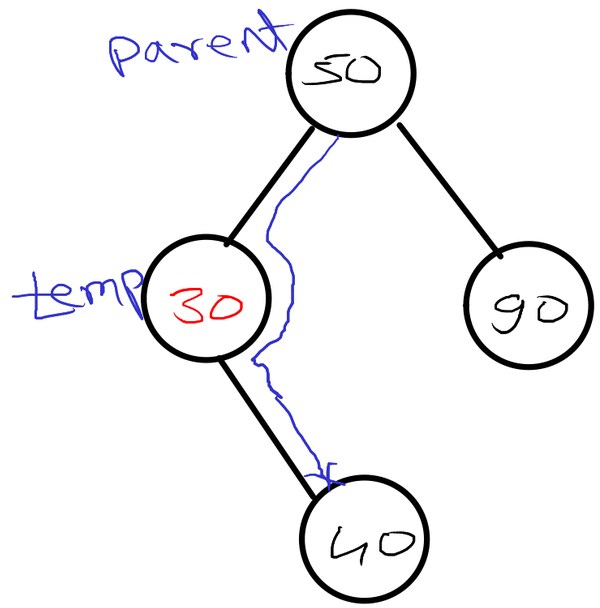
key = 6

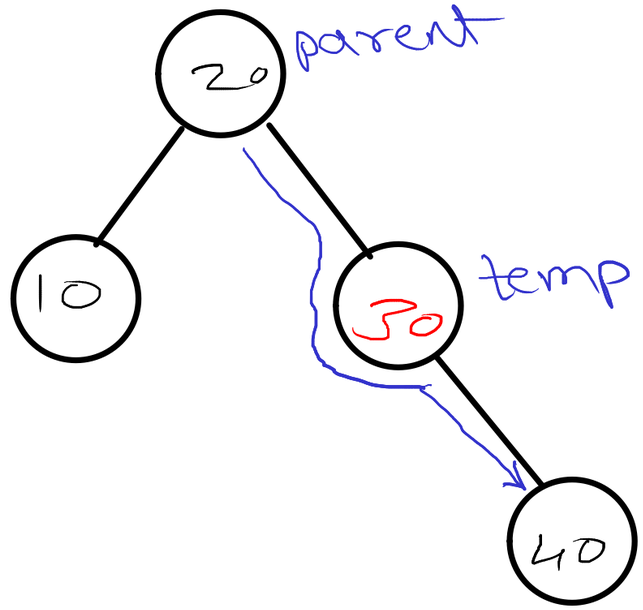
key = 8

key = 15

BST - Delete node which has single child (right child)

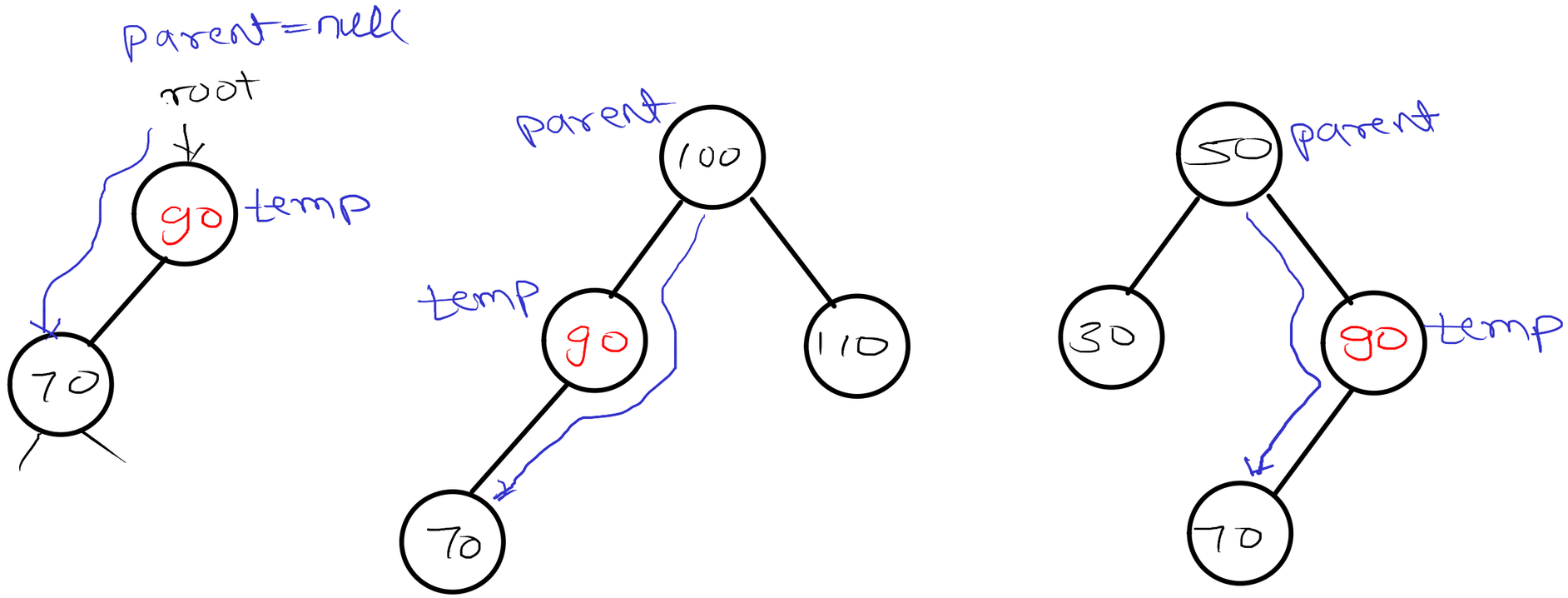






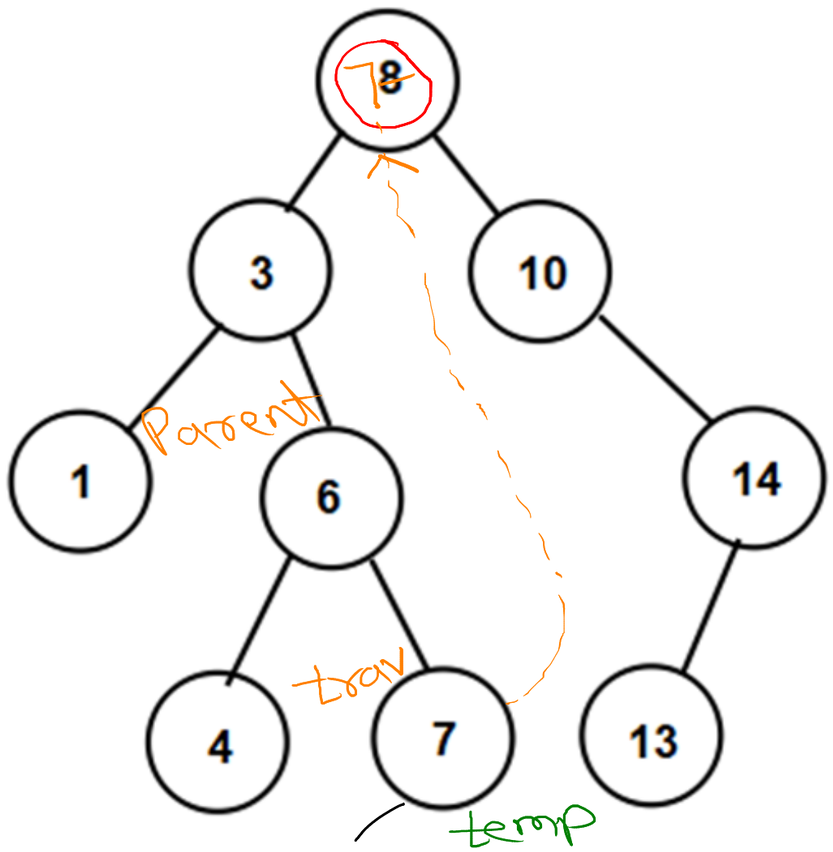

```
if(temp.left == null){
        if(temp == root)
                root = temp.right;
        else if(temp == parent.left)
                parent.left = temp.right;
        else  //if(temp == parent.right)
                parent.right = temp.right;
}
```

BST - Delete node which has single child (left child)




```
if(temp.right == null){
        if(temp == root)
                root = temp.left;
        else if(temp == parent.left)
                parent.left = temp.left
        else //if(temp == parent.right)
                parent.right = temp.left;
}
```

BST - Delete node which has two childs




```
if(temp.left != null && temp.right != null){
    //1. find inorder predecessor
    Node trav = temp.left;
    parent = temp;
    while(trav.right != null){
        parent = trav;
        trav = trav.right;
    }
    //2. update data by data of predecessor
    temp.data = trav.data;
    //3. mark trav to be deleted
    temp = trav;
}
```

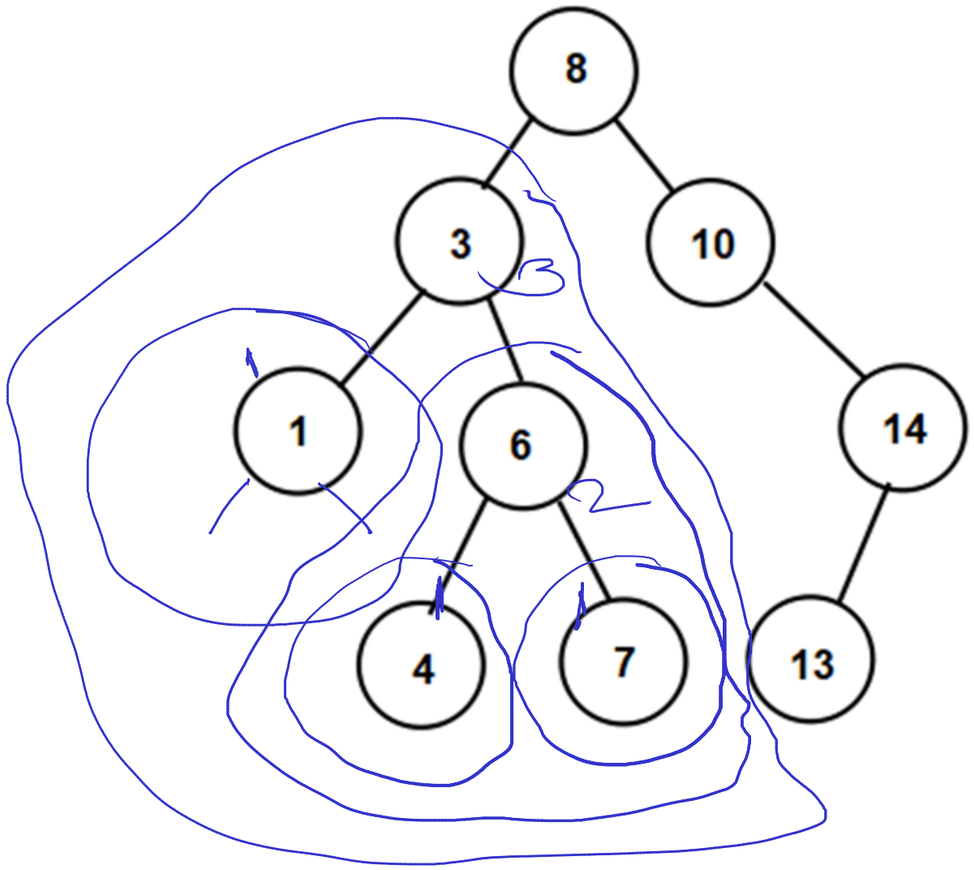
Inorder Traversal : 1 3 4 5 7 8 10 13 14

Inorder predecessor → **left extreme right**

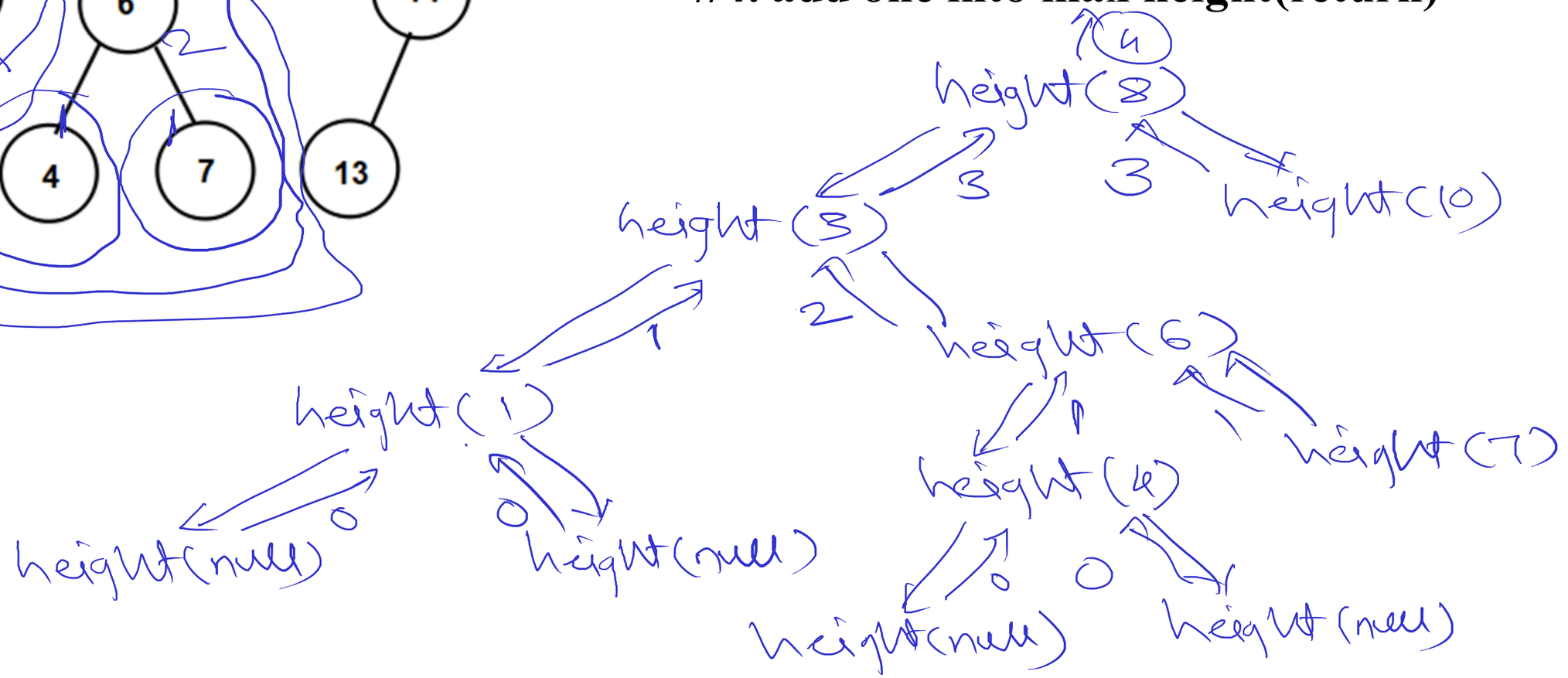
Inorder successor → **right extreme left**

BST - Height

Height of tree = MAX(Height(left sub tree), Height(right sub tree)) + 1

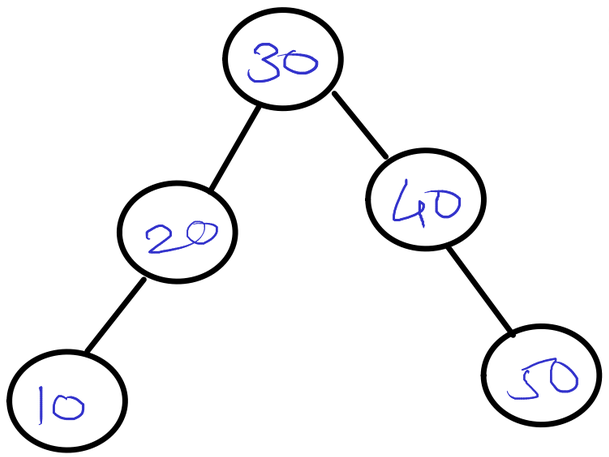



```
//0. if left or right sub tree is absent
                //then return 0
//1. find height of left subtree
//2. find height of right subtree
//3. find max height
//4. add one into max height(return)
```



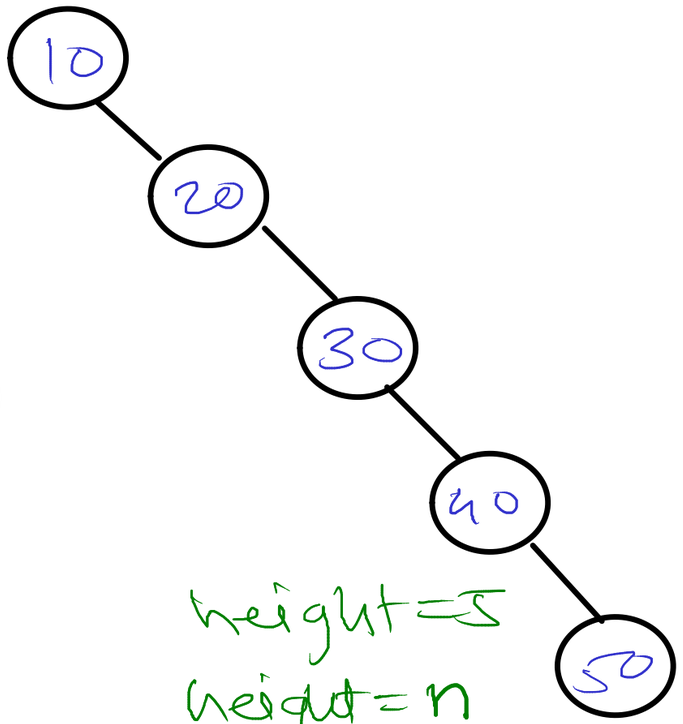
Skewed BST

Keys : 30, 40, 20, 50, 10



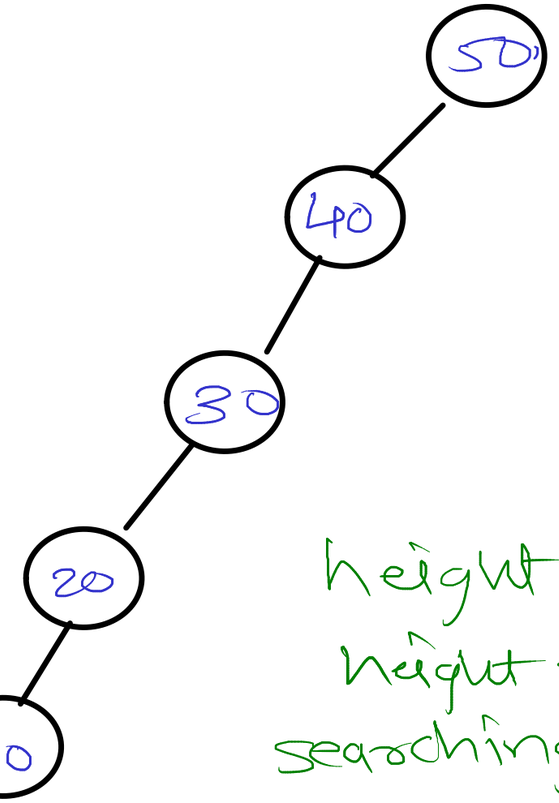
height = 3
height = log n
searching = O(log n)

Keys : 10, 20, 30, 40, 50



height = 5
height = n
searching = O(n)

Key : 50, 40, 30, 20, 10



height = 5
height = n
searching = O(n)

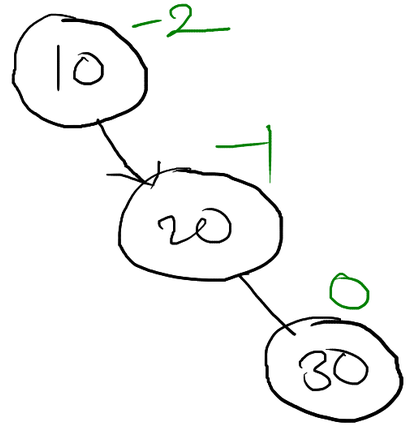
- **if tree is growing in only one direction, such tree is known as skewed BST**
- **if tree is growing in only right direction, such tree is known as right skewed BST**
- **if tree is growing in only left direction, such tree is known as left skewed BST**

Balanced BST

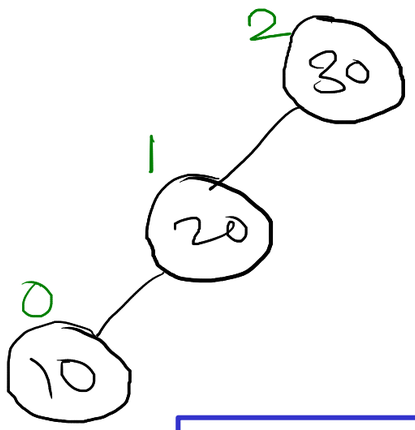
Balance Factor = height(left sub tree) - height(right sub tree)

- tres is balanced if balance factor of all the nodes is either -1, 0 or +1
- balance factor = {-1, 0, +1}

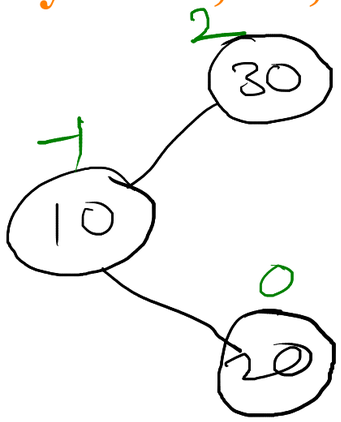
Keys : 10, 20 ,30



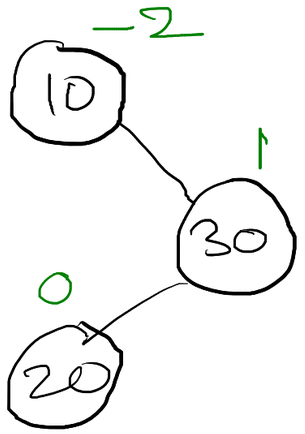
Keys : 30, 20, 10



Keys : 30, 10, 20

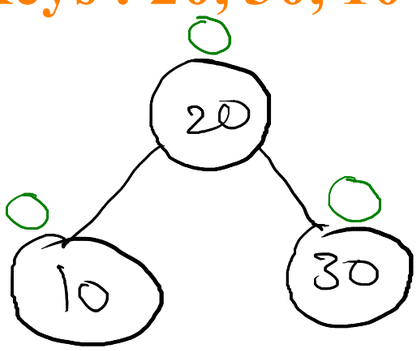


Keys : 10, 30, 20



Keys : 20, 10, 30

Keys : 20, 30, 10

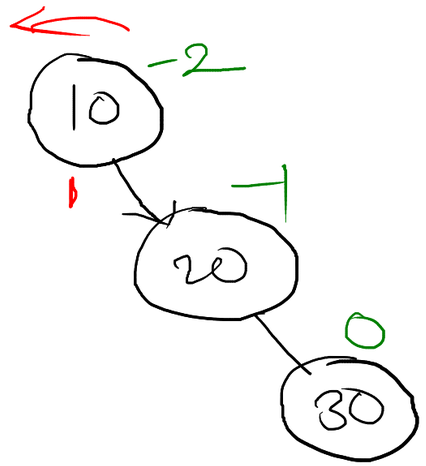


Balanced BST

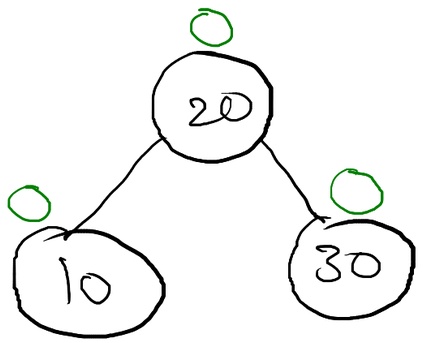
Rotations

RR Imbalance

Keys : 10, 20 ,30

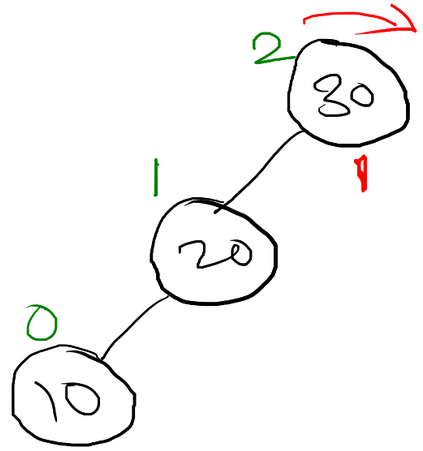


Left Rotation

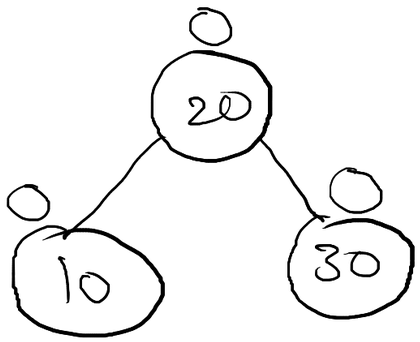


LL Imbalance

Keys : 30, 20, 10



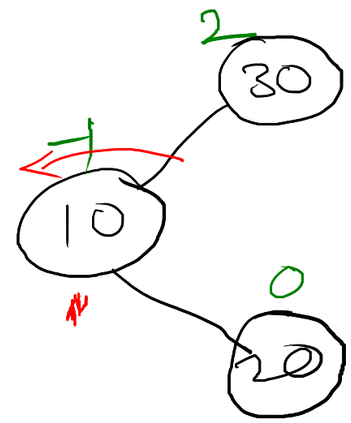
Right Rotation



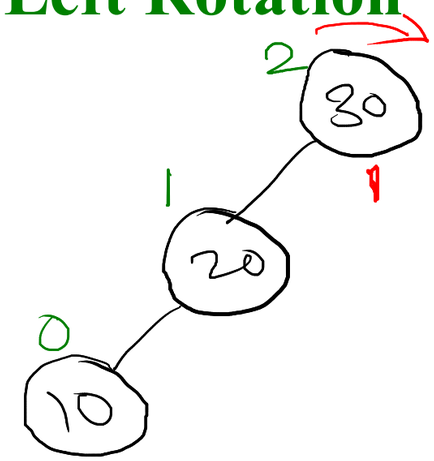
Single Rotation

LR Imbalance

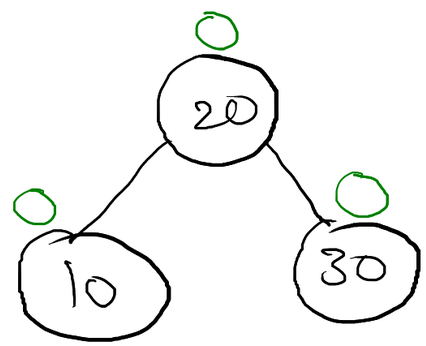
Keys : 10, 30, 20



Left Rotation

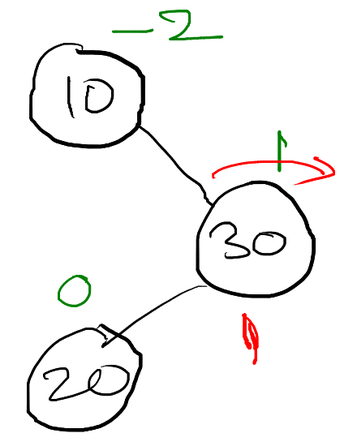


Right Rotation

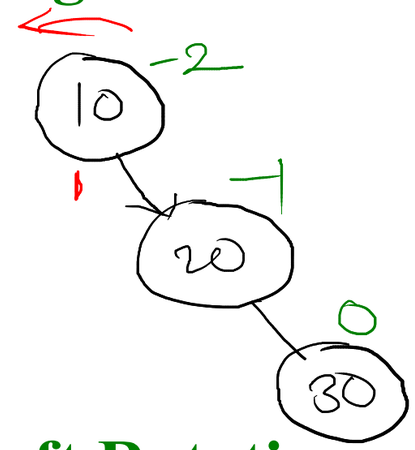


RL Imbalance

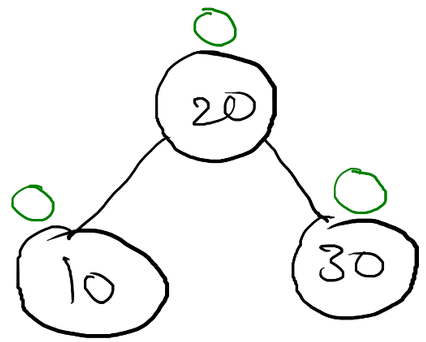
Keys : 30, 10, 20



Right Rotation



Left Rotation



Double Rotation