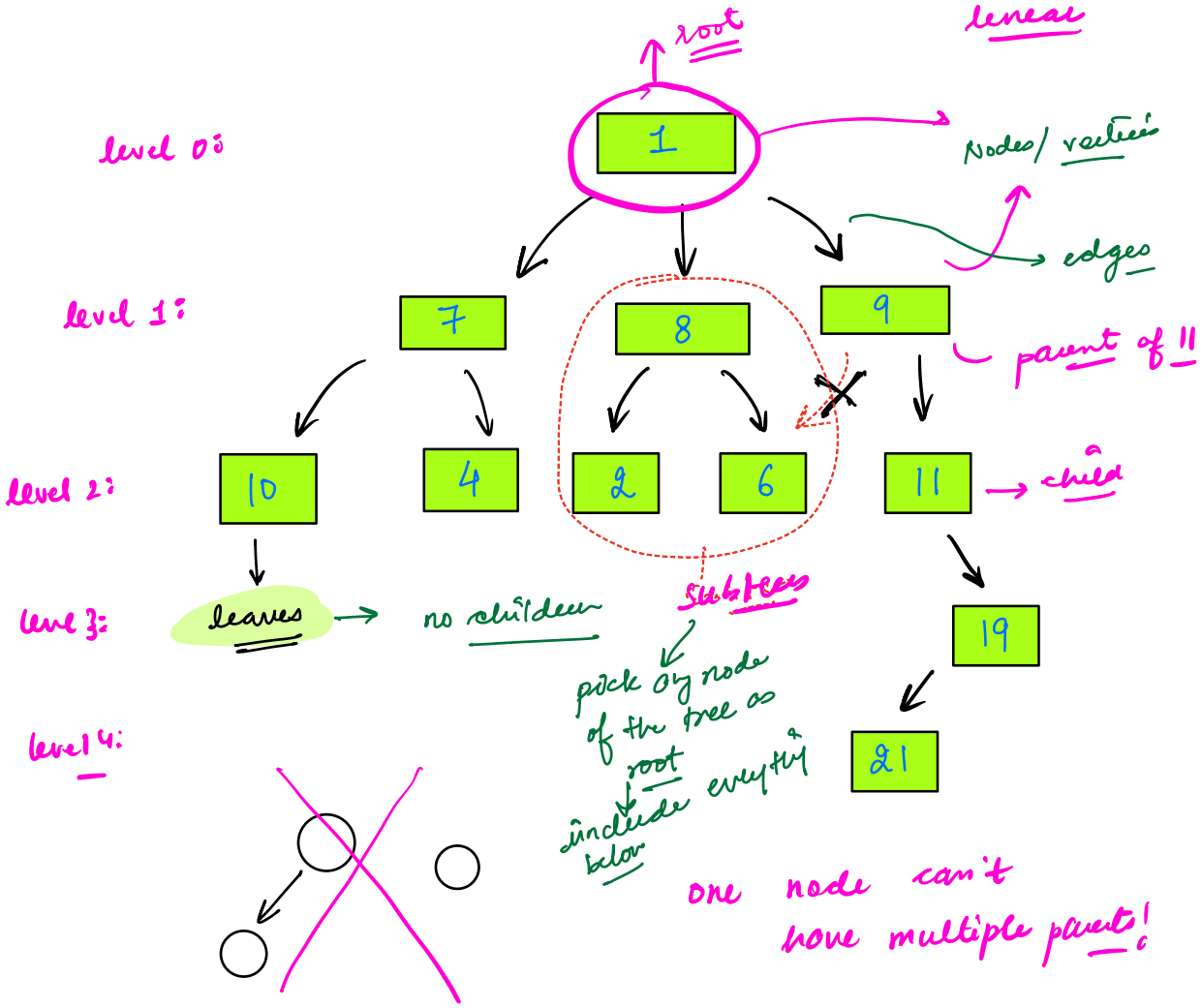
Data structures :- array, LL, maps, stack, queue

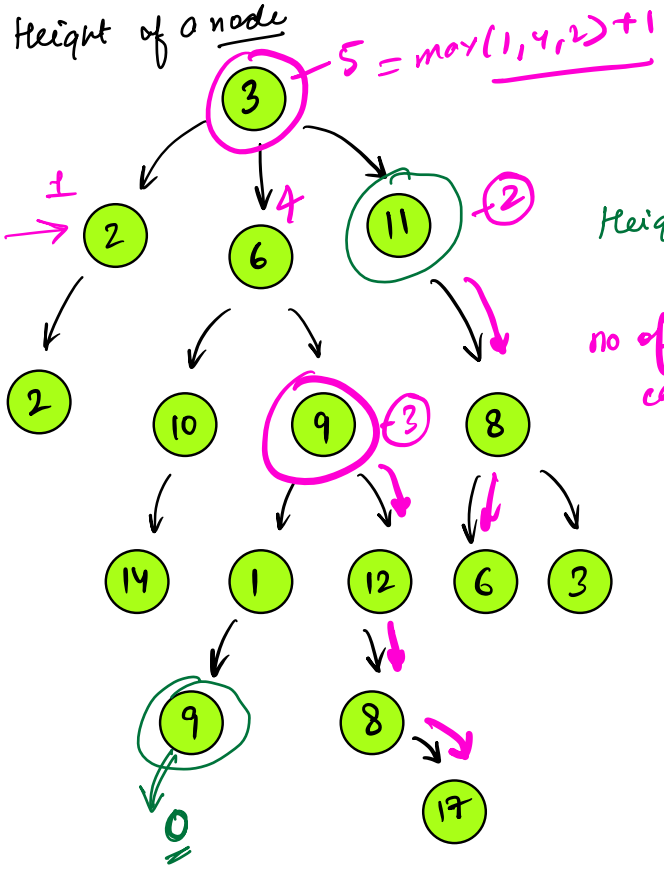
linear



one node can't have multiple parents!

n nodes ≡ n-1 edges

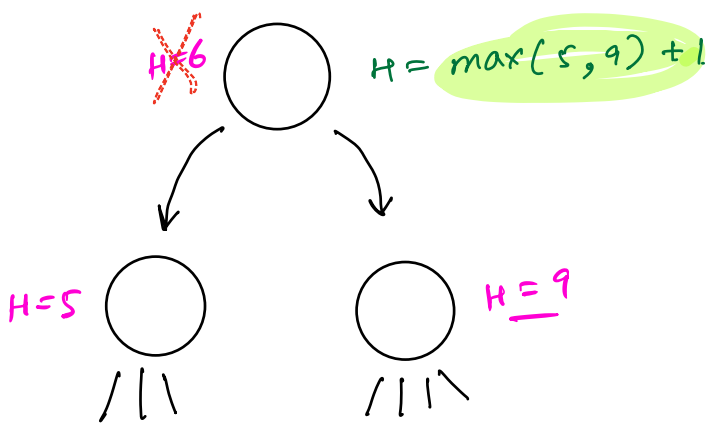
Height of a node



Height = distance from the node to the farthest leaf in its subtree

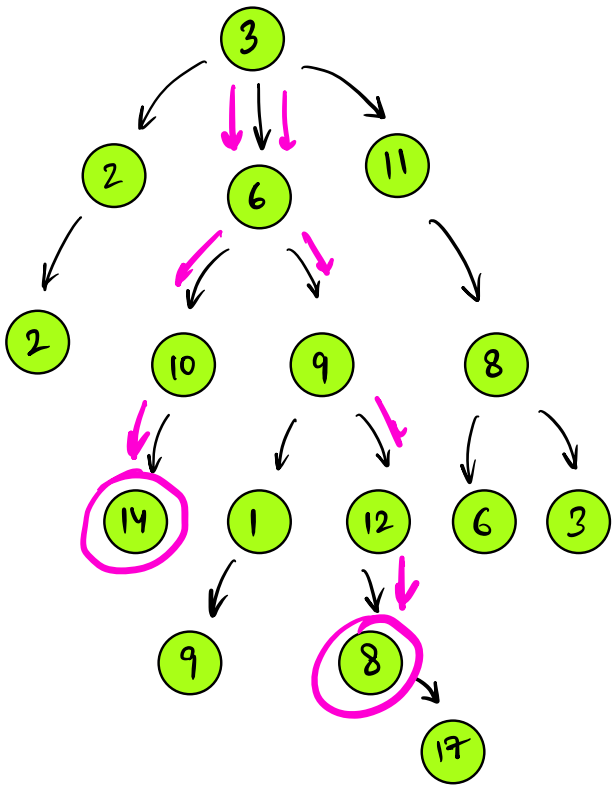
no of edges

Height of tree = Height of root



Height of a node = max(heights of children) + 1

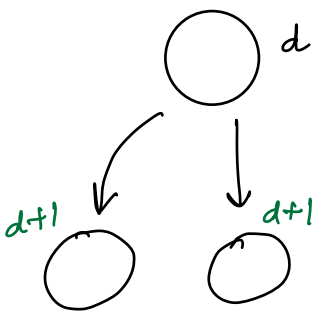
depth



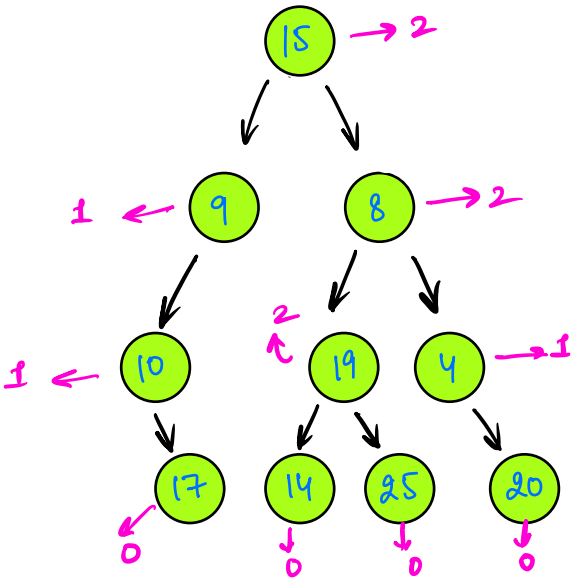
depth = distance b/w root & node

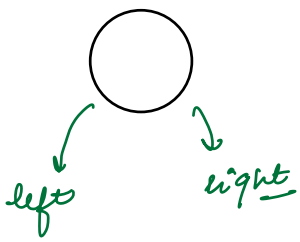
||

level

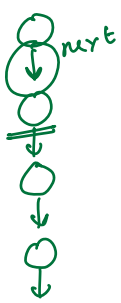


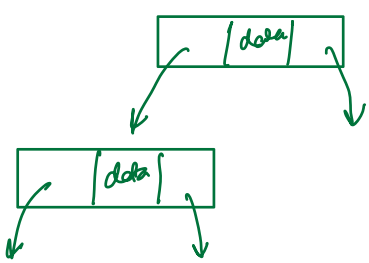
Binary Trees ≤ 2 children 0, 1, 2



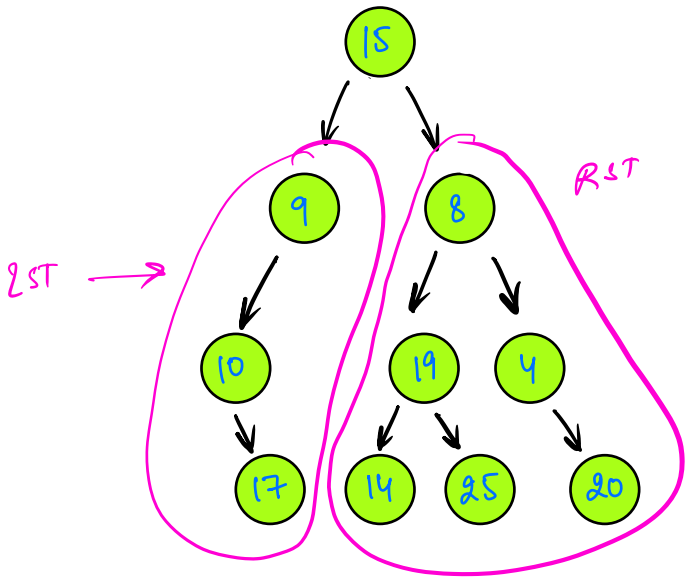



```
class Node {
    int data;
    Node left;
    Node right;
    Node parent;
}
```





recursion

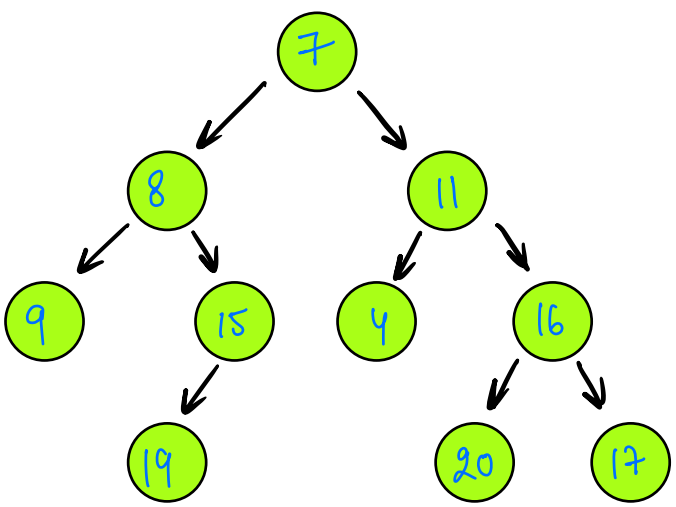


Traversals

preorder

inorder

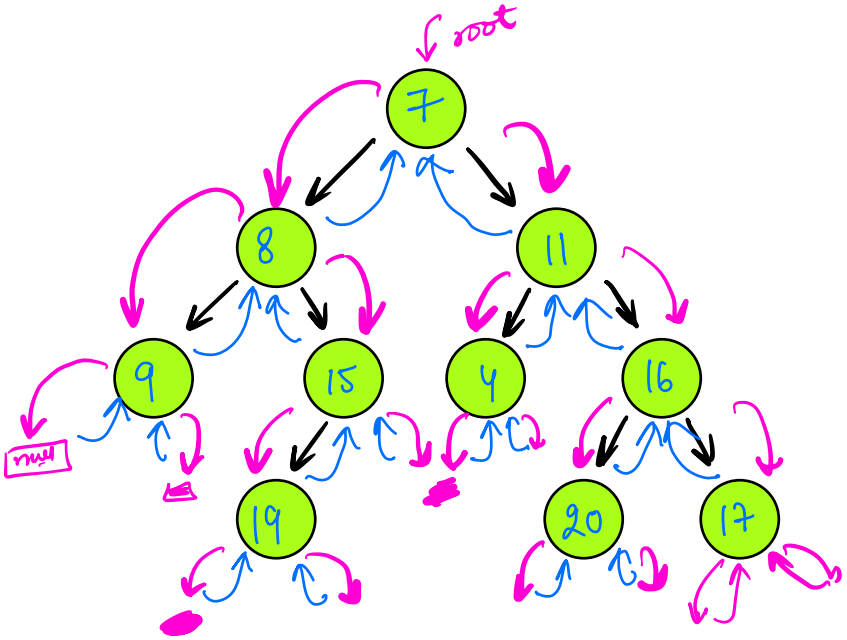
postorder

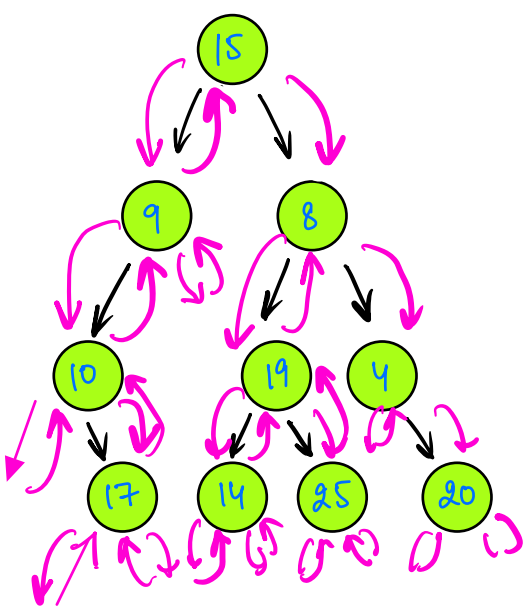


Inorder Traversal

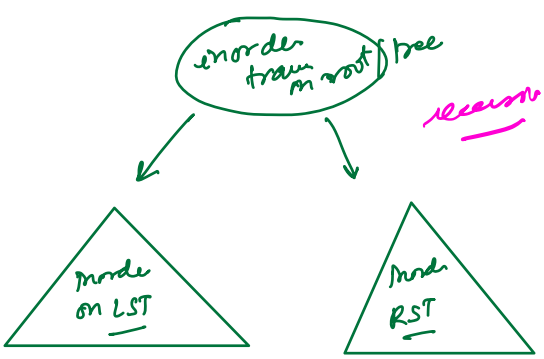
left
root
right

9 8 19 15 7 4
11 20 16 17



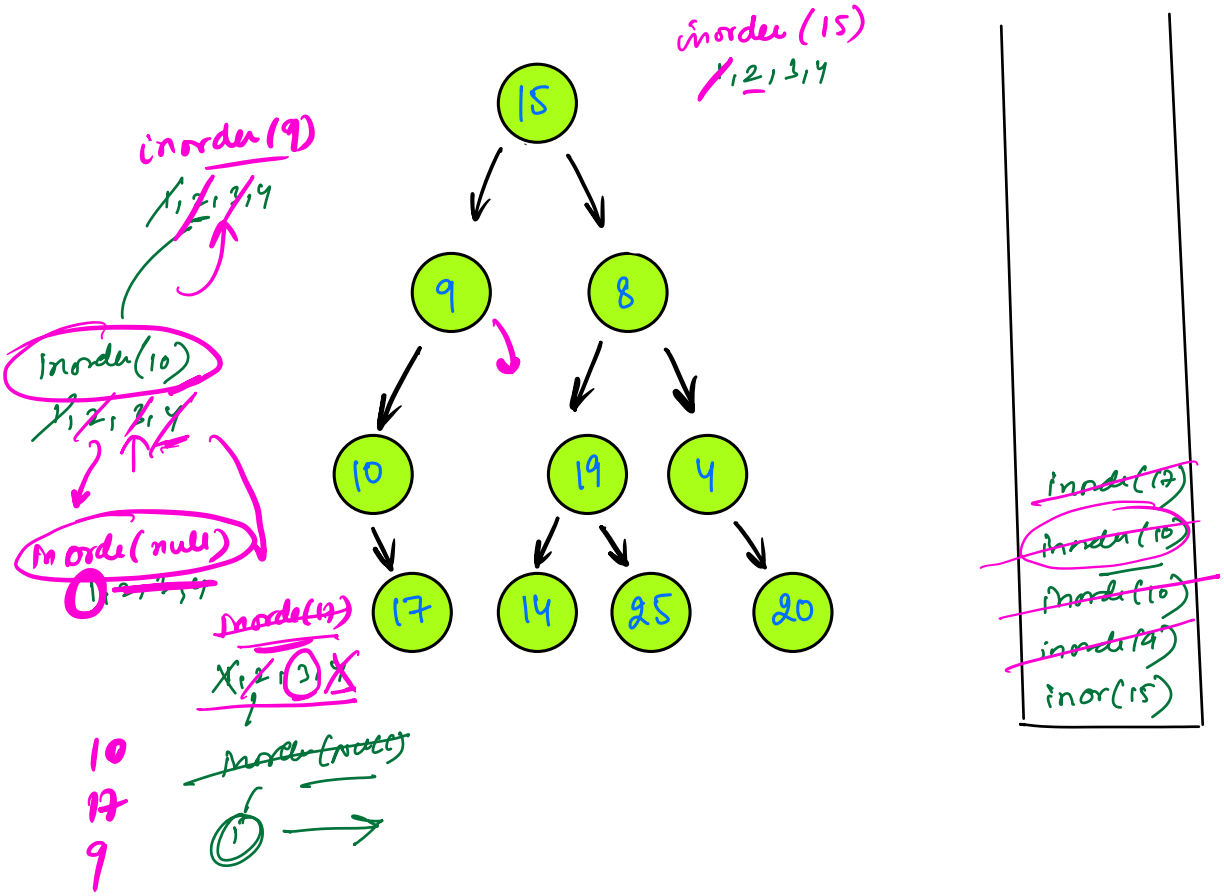


10 17 9 15 14 19 25 8 4 20



→ traverse your tree with root given in node

```
void inorder( Node root)
{
    if( root==null) return;
    inorder( root.left);
     print( root.data);
     inorde( root.right);
}
```



- preorder

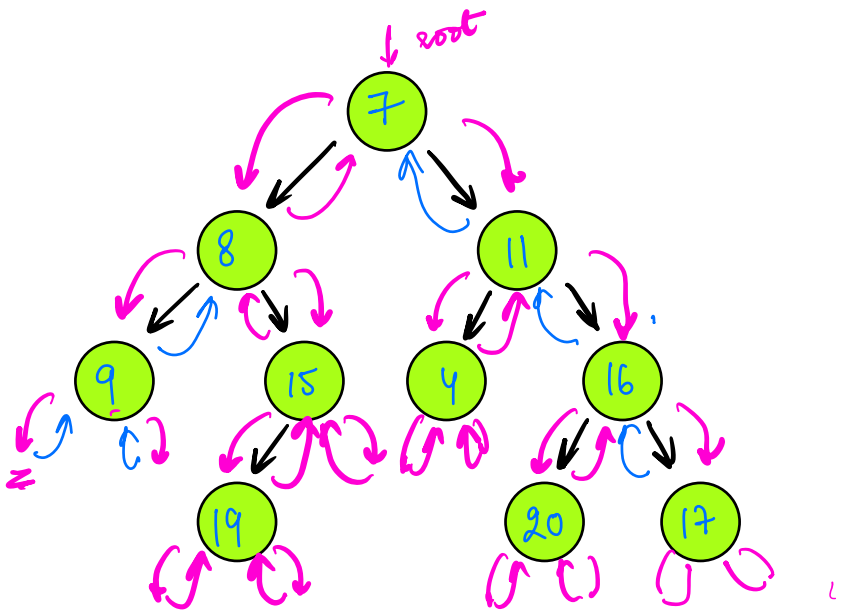
Root

Left

Right

7 8 9 15 19

11 4 16 20 17



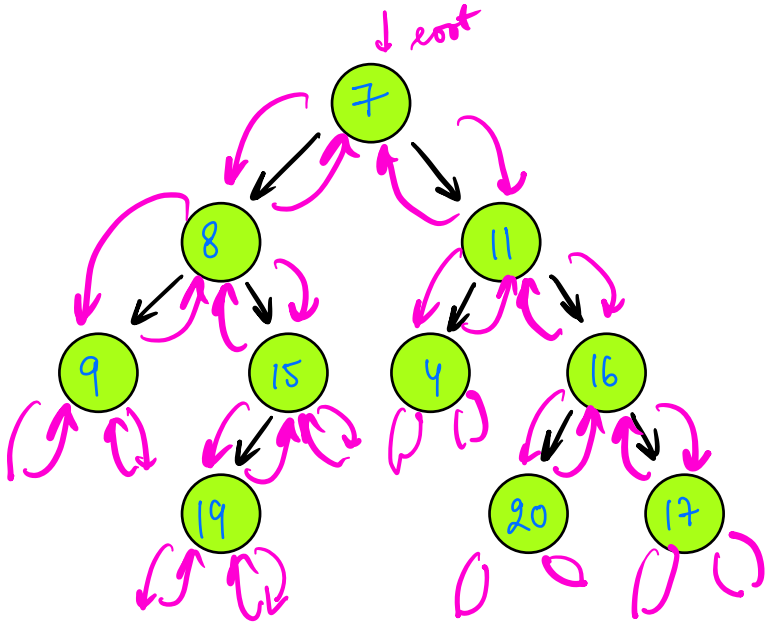

```
void preorder ( Node root)
{
    if( root ==null) return;
        print ( root.data);
        preorder( root.left);
        preorder ( root.right);
}
```

call by reference/ all

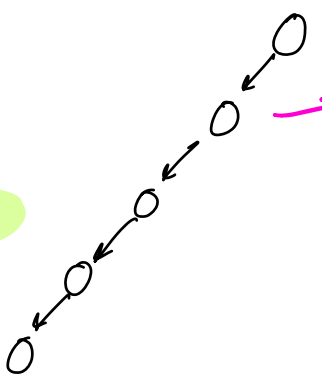
postorder

Left
Right
Root

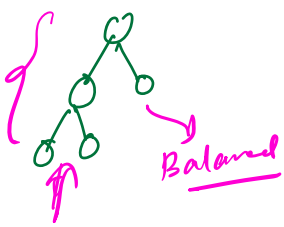
9 19 15 8 4
20 17 16 11 7



T.C: O(N)
S.C: recursive stack
↓
O(n) O(height)



skewed tree



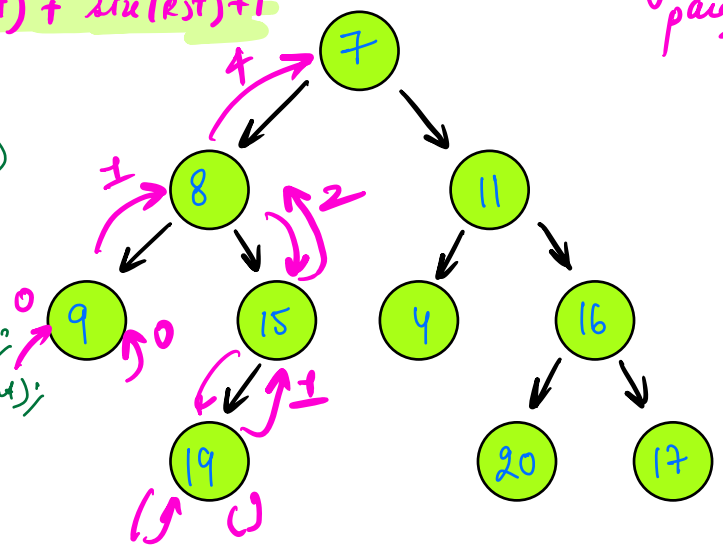
size of the tree :- total count of nodes

size(tree) = size(LST) + size(RST) + 1

global parameter

```
int size(Node root)
{
    if(root==NULL)
        return 0;
    int l = size(root.left);
    int r = size(root.right);

    return l+r+1;
}
```



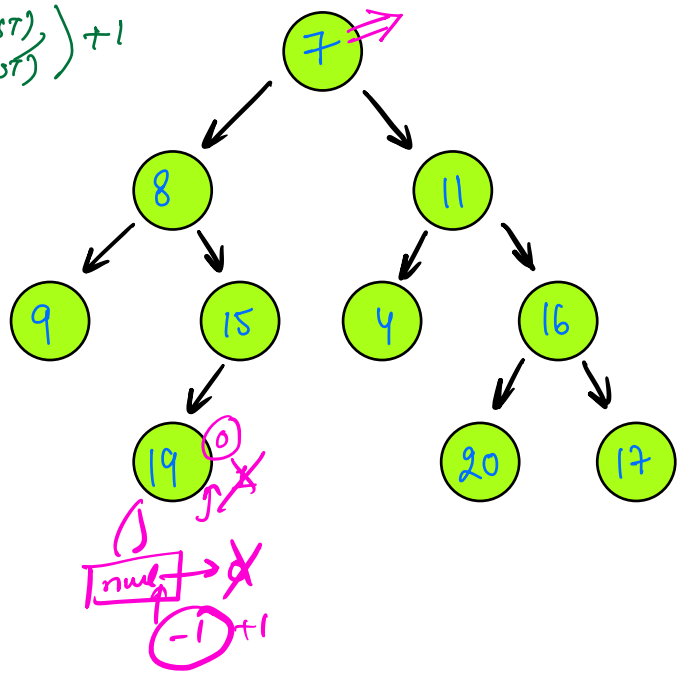
o height of the Tree

height(root) = max(height(LST), height(RST)) + 1

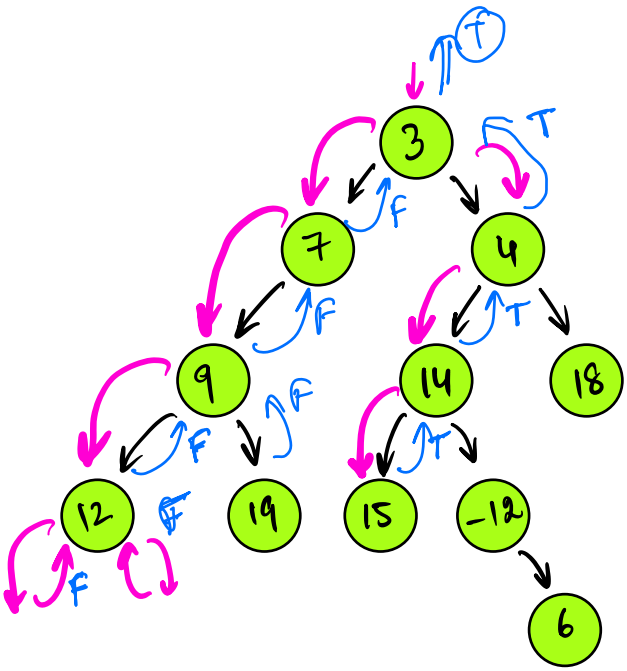
```
int height( Node root)
{
    if( root == null) return -1;

    int l = height( root.left);
    int r = height( root.right);

    return max( l, r) + 1;
}
```



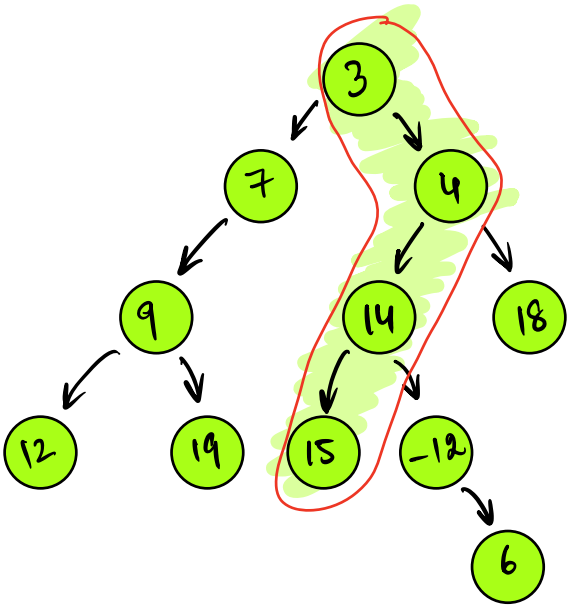
Search element K in tree




```
search(root, k) =
    root.data == k ||
        search(root.left, k) ||
        search(root.right, k)
```

```
bool search(root, k)
{
    if(root == NULL)
        return false;

    return
     root.data == k  ||
     search(root.left, k) ||
     search(root.right, k);
}
```

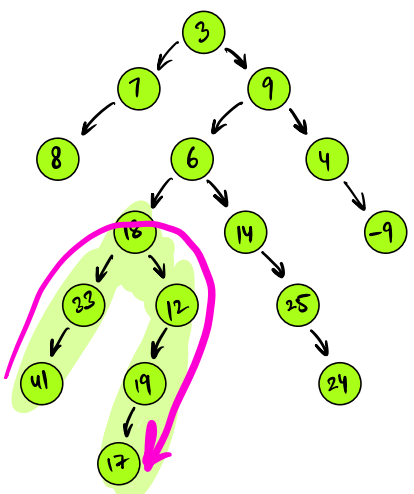


ans: 15 14 4 3

reverse(ans);

```
bool path( root, k, pathlist)
{
    if( root==null) return false;

    if( root->data==k)
    {
        pathlist.insert( root->data);
        return true;
    }
    if( path(r->left, k, list) || path(r->right, k, list))
    {
        pathlist.insert( root->data);
        return true;
    }
    return false;
}
→ reverse (pathlist);
```



41 → 17