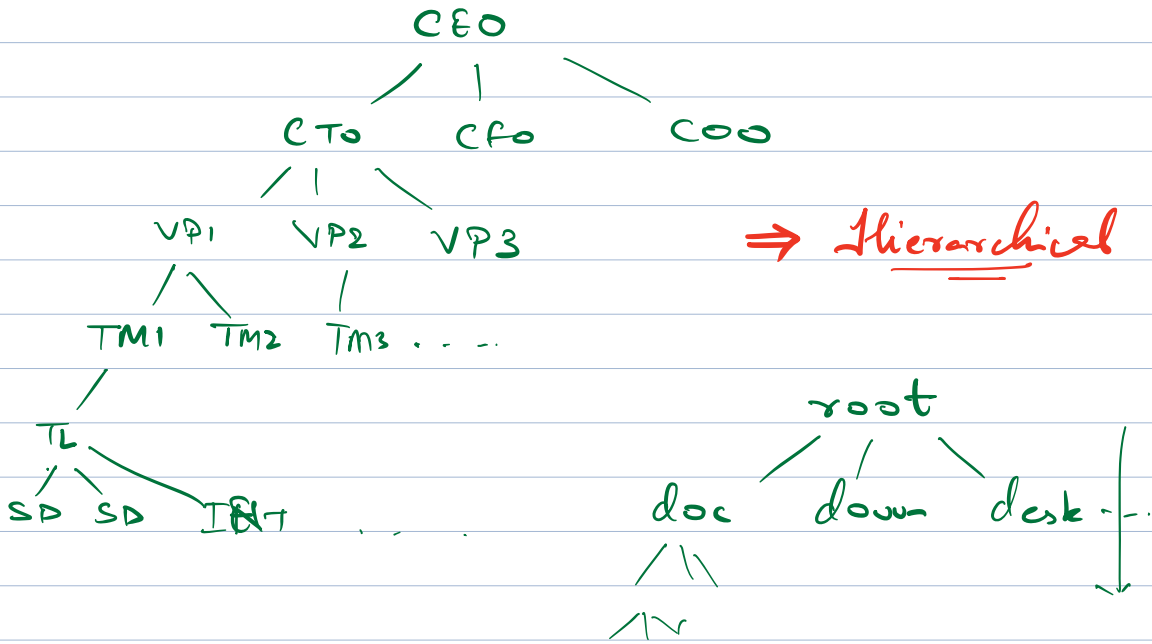
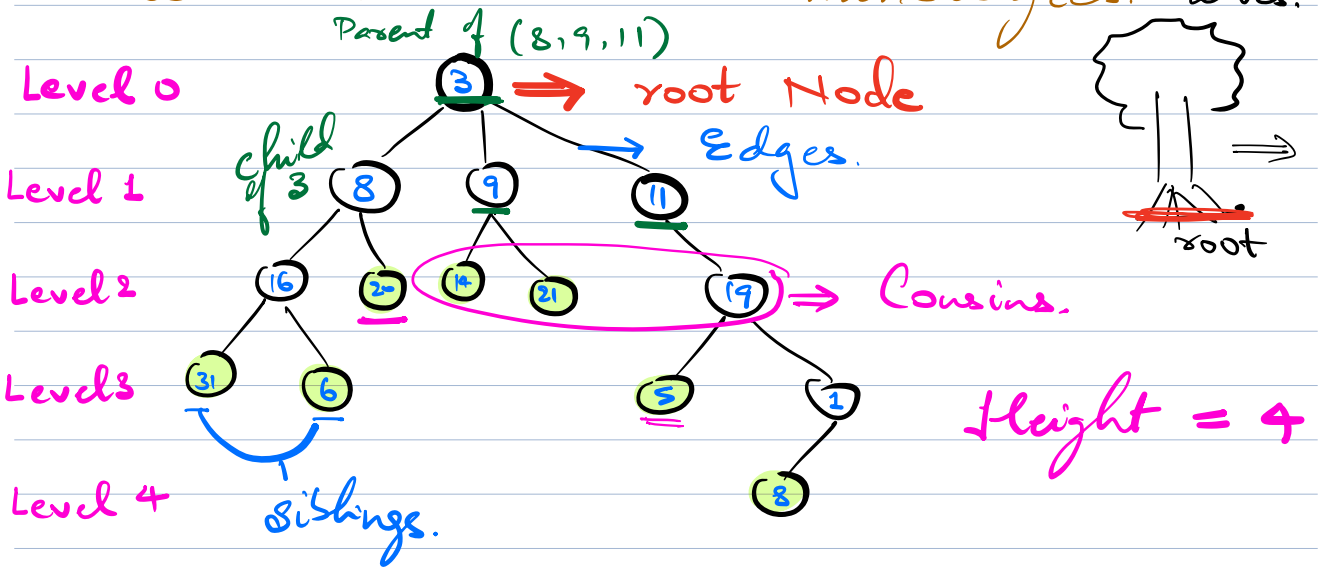


⇒ Hierarchical

Tree Basics & Terminologies.



Root : No parent
Leaf : No children.

Height Of A Tree

Height of Tree : (No. of edges)
Distance b/w root node & its farthest leaf

Height of a Node : Distance b/w node & its farthest leaf.

Height (9) = 1

Height (11) = 3

Height (5) = 0

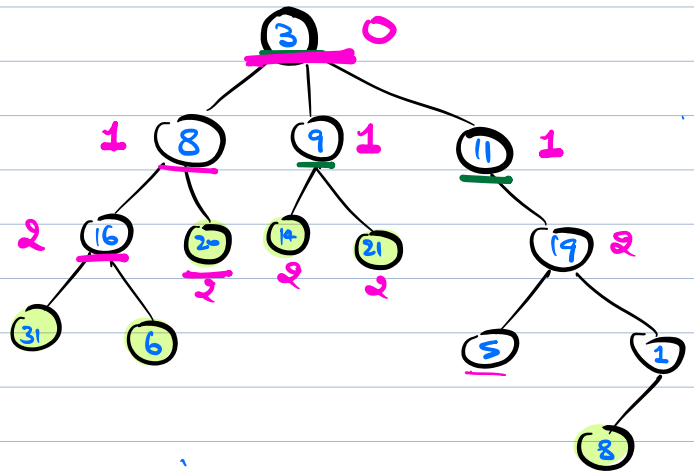
Height of all leaf nodes = 0

Height (root Node) = Height of Tree.

Height of any node = 1 + max (child node heights)

Depth Of A Node

⇒ Distance of the node from root Node



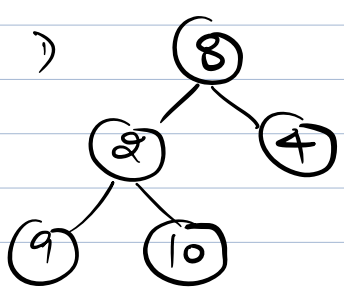
Depth of a node = d ⇒ Depth of its child node

d + 1

Binary Tree

⇒ Every node can have atmax 2 children

1)

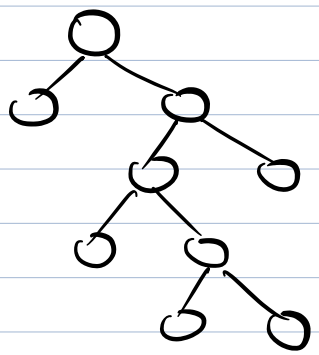


2)

Types of BT.

1) Full Binary Tree

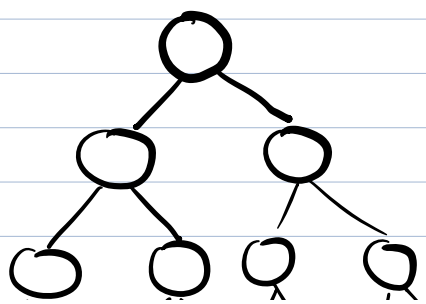
Node — 0 child
     — 2 children



2) Perfect Binary Tree

⇒ All the levels of the BT are completely filled

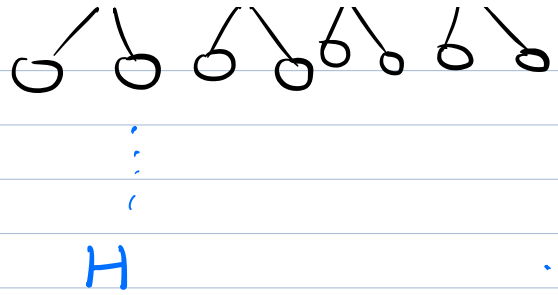
0
1
2



$1 = 2^0$

$2 = 2^1$

$4 = 2^2$



$8 = 2^3$

2^3

H

2^H

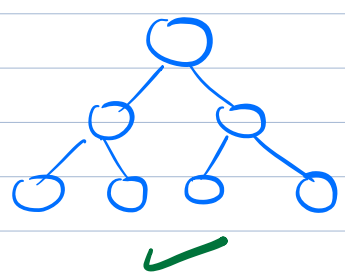
$2^0 + 2^1 + 2^2 \ldots\ldots 2^{H-1}$

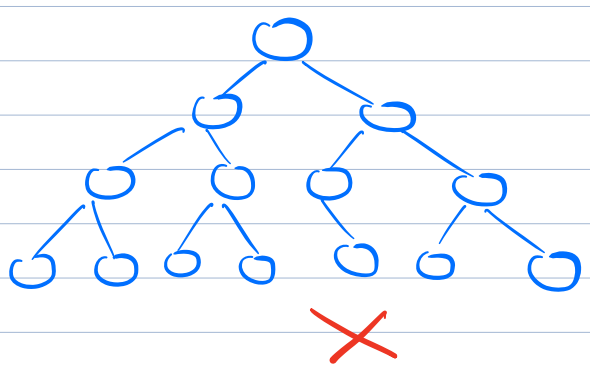
$$\frac{(1)\left(2^{H+1}-1\right)}{(2-1)} \Rightarrow 2^{H+1}-1$$

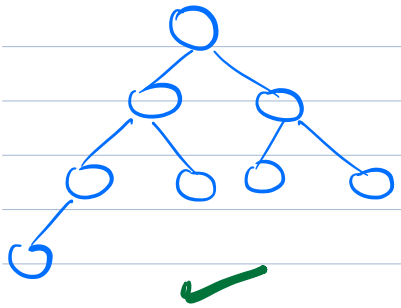
3) Complete Binary Tree

⇒ All the levels are completely filled except the last level.

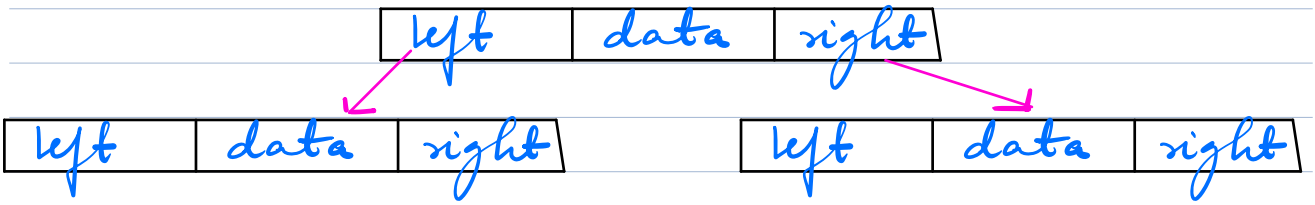
⇒ In the last level the nodes are present as left as possible (left alligned)





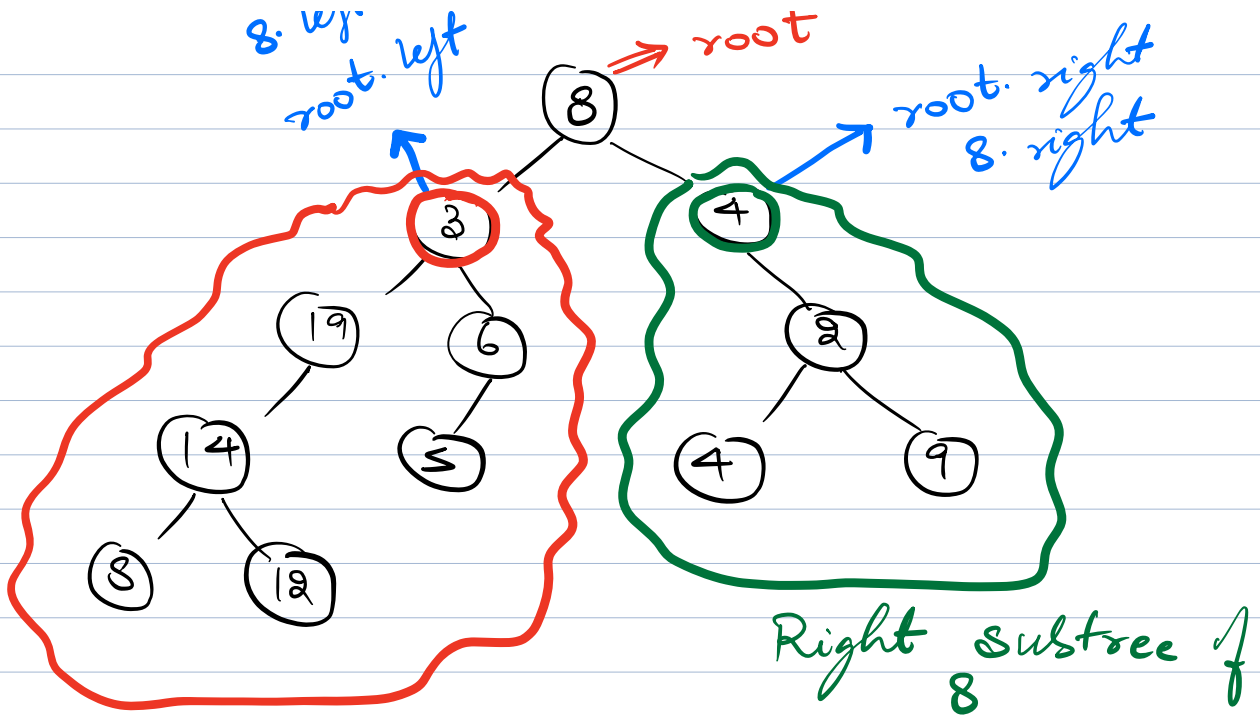


Node Structure



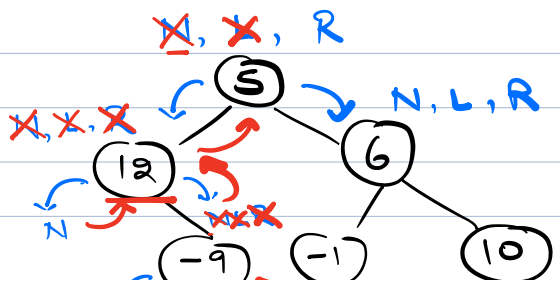

```
class Node {

    int data;
    Node left;
    Node right;

    Node (x) {
        this.data = x;
        left = NULL;
        right = NULL;
    }

}
```

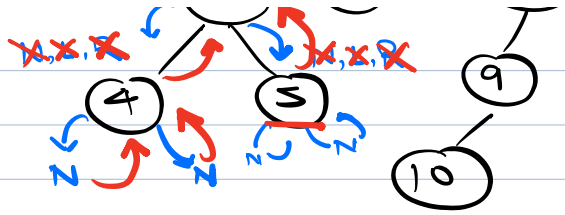
constructor



Tree Traversal

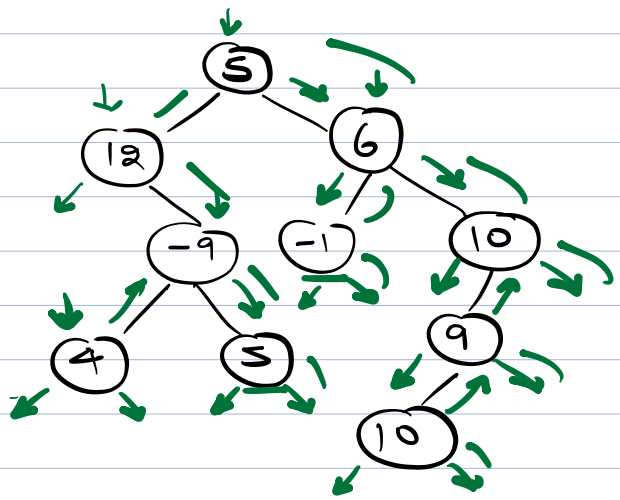
1) Pre Order — Node, LST, RST
2) In Order — LST, Node, RST
3) Post Order — LST, RST, Node





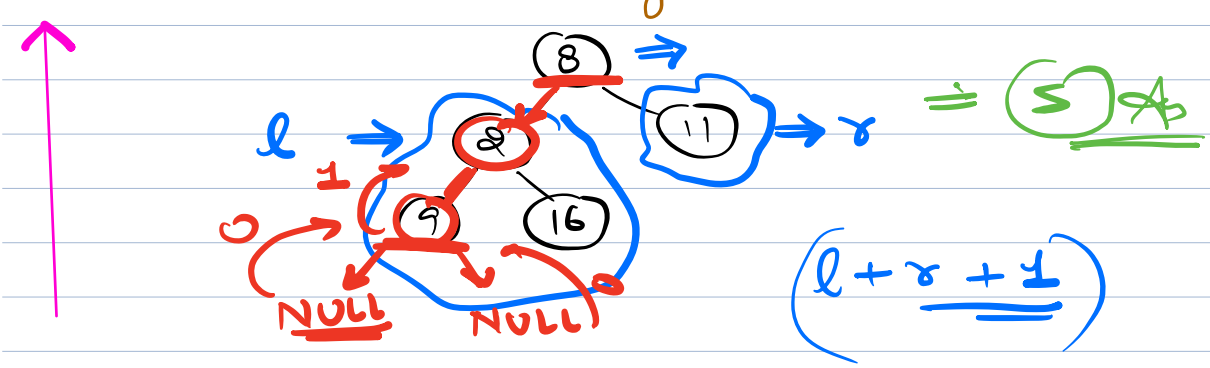
Pre : 5, 12, -9, 4, 5, 6, -1, 10, 9, 10.

L, N, R



In: 12, 4, -9, 5, 5, -1, 6, 10, 9, 10

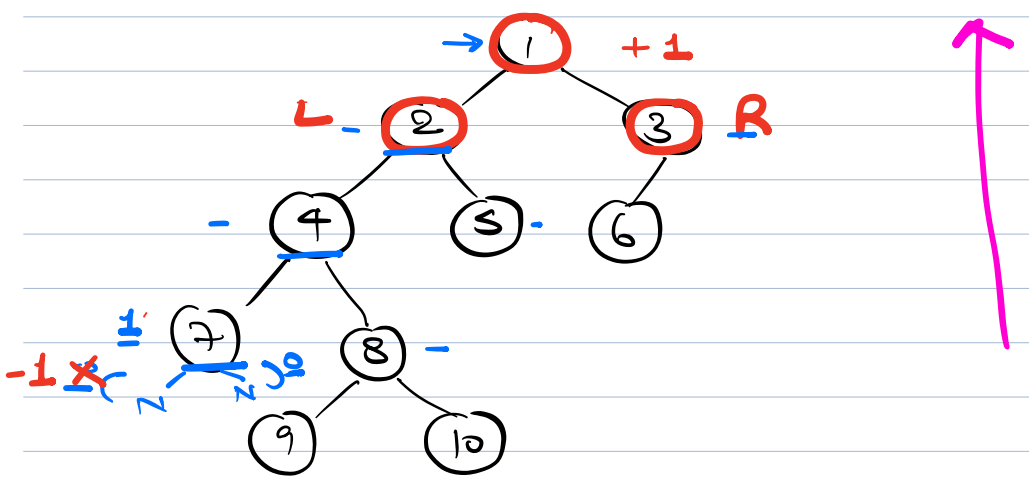
Q Given the root node of a BT. Count the no. of nodes.



```
int size ( Node root ) {
1   if (root == NULL) { return 0; }

2 L int l = size( root.left);
3 R int r = size (root.right);
4 N return ( l + r + 1);
}
```

Post Order Traversal

Q Given the root node of a BT. find the Height



```
int height (Node root) {
1   if (root == NULL) { return -1 }
2   int l = height (root.left);
3   int r = height (root.right);
4   return 1 + max (l, r);
}
```

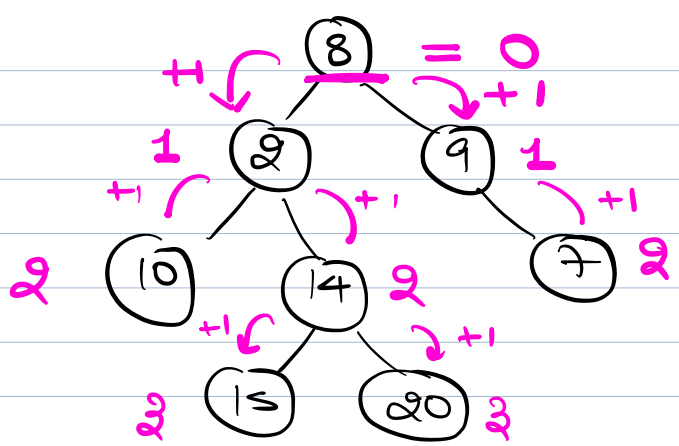
Q Given a binary tree

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

⇒ Info missing.

∀ nodes ⇒ fill the depth of the node.

N, L, R

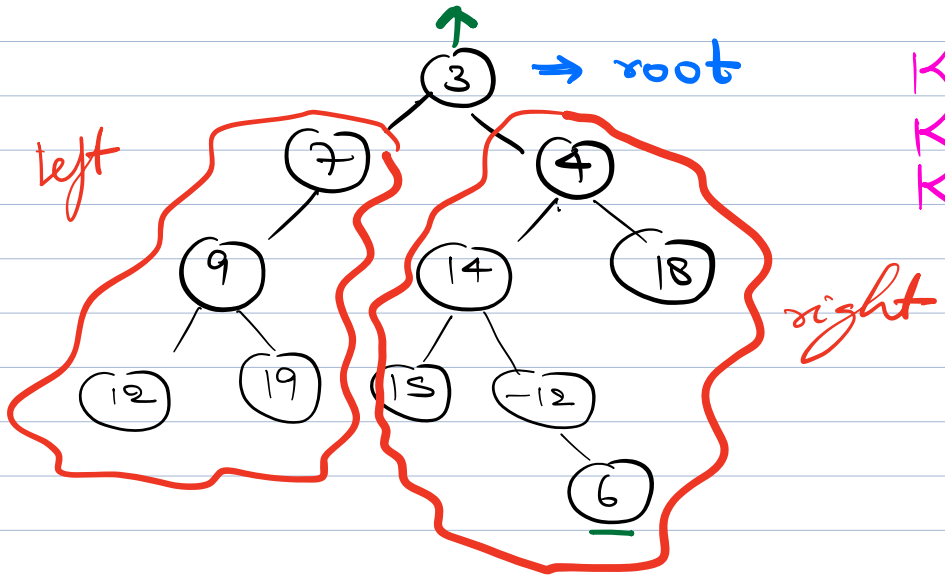



```
void fillDepth (Node root, int d) {
    if (root == NULL) { return; }
    root.depth = d;
    fillDepth (root.left, d+1);
    fillDepth (root.right, d+1);
}
```

T.C. = O(N)

S.C. = O(H)

Q Given root node of a BT which contains all unique values. Return true if a given (K) exists in the BT.



K = 6 ✓
K = 19 ✓
K = 25 ✗

```
bool check (Node root, int K){
    if (root == NULL) { return false; }
    if (root.data == K) { return True; }
    if ( check (root.left, K)) {
        return True;
    }
```

```
if (check(root.right, k)) {
    return True;
}
return false;
}
```

T.C. = O(N)

Given a BT & a node K present in Tree

H.W.

Print the path from root to K