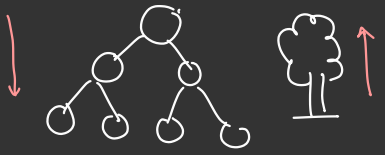
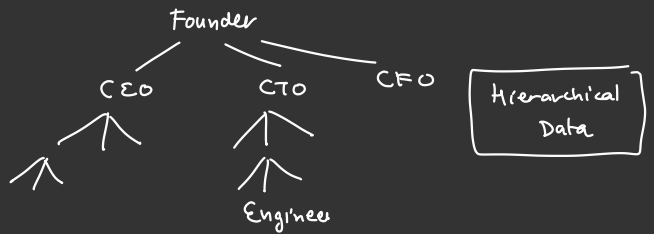
Trees

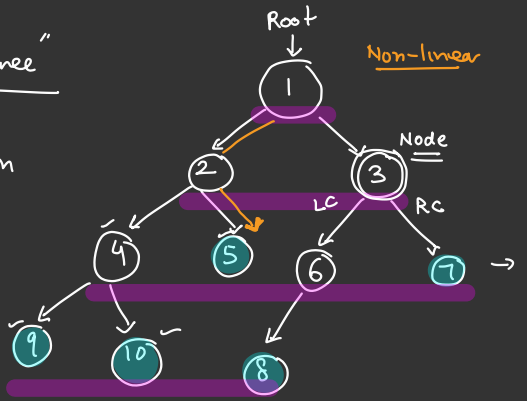


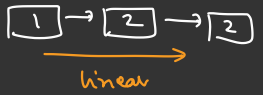
- Recap
- Problems.



"Binary Tree"

atmax
2 children





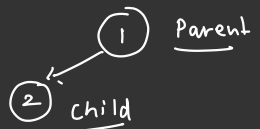
Leaf Nodes

4 Levels

→ Height = 4 (Tree)

→ Depth (5) = 3 (Node)

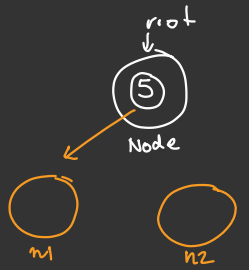
→ Sibling => Node with same parent



Ancestor (10) ↑

= 4, 2, 1

Descendant (2) = 4, 5, 9, 10

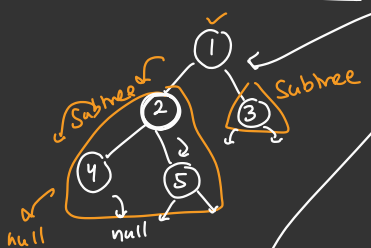



```
class Node {
    int data;
    Node left;
    Node right;
    Node(d) { data = d, left = right = null }
}
```

```
Node root = new Node(5);
```

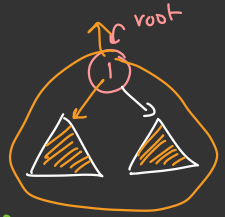
Algorithm To Build Tree.

+ve values



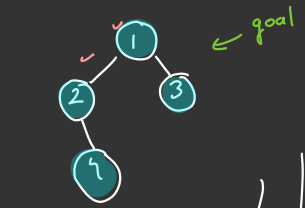

```
buildTree(){
    Read data;
    if(data == -1){
        return null;
    }
    Node root = new Node(data);
    root.left = buildTree();
    root.right = buildTree();
    return root;
}
```

Preorder

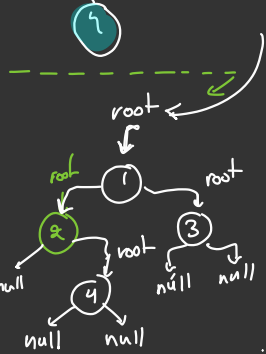


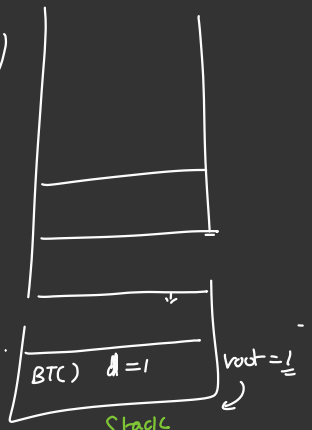
Input: 1 2 4 -1 -1 5 -1 -1 3 -1 -1

root



Root
Left
Right
} Preorder Traversal





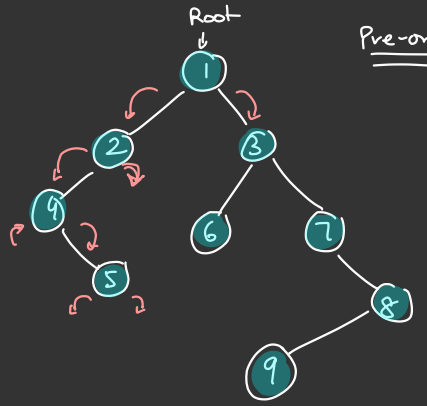

```
buildTree(){

    ⇒ Read data;
1   if(data == -1){
        return null;
    }
2→  Node root = new Node(data);
3→  root.left = buildTree();
4→  root.right = buildTree();
5   return root;
}
```

root = buildTree();

Input 1 2 -1 4 -1 -1 3 -1 -1

Recap of tree traversals

Printing The Tree



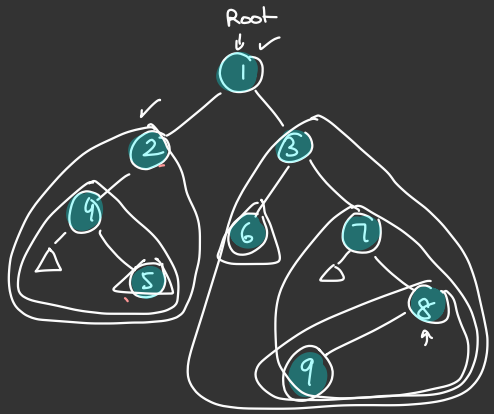
Pre-order

1, 2, 4, 5

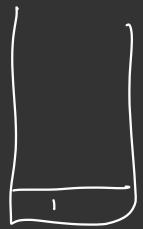
1, 2, 4, 5, 3, 6, 7, 8, 9 →

```
void preorderPrint(Node root){
    if(root==null){
        return;
    }
    System.out.print(root.data +",");
    preorderPrint(root.left); ←
    preorderPrint(root.right);←
}
```

Inorder

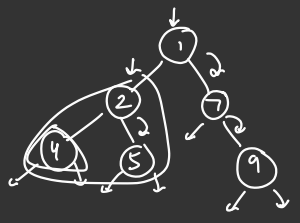


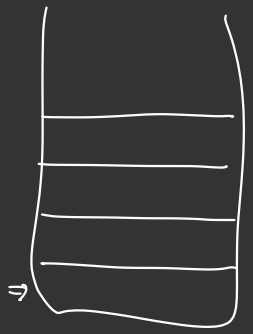
Left
Root
Right



4, 5, 2, 1, 6, 3, 7, 9, 8,

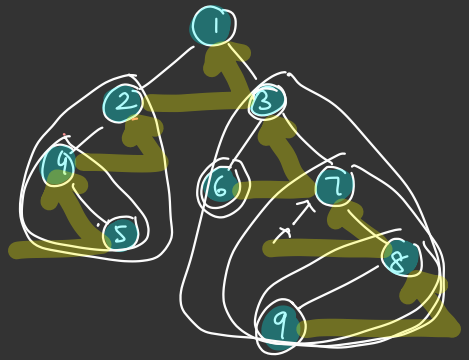
```
void inOrderPrint(Node root){
    if(root==null){
        return;
    }
    inOrderPrint(root.left);
    System.out.print(root.data +",");
    inOrderPrint(root.right);
}
```





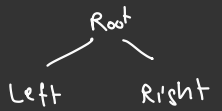
4, 2, 5, 1, 7, 9

Left Tree
print Root
Right Tree



Post order

5, 4, 2, 6, 9, 8, 7, 3, 1

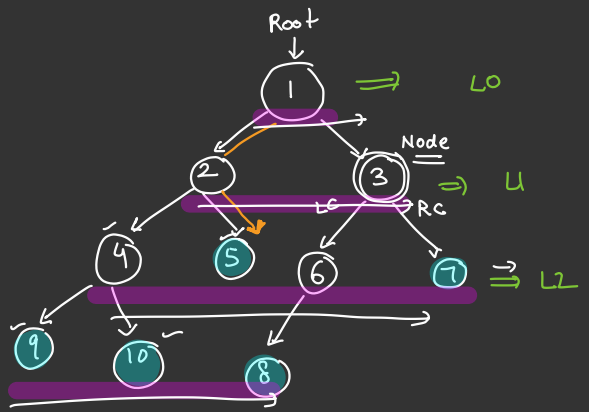



```
void postOrderPrint(Node root){
    if(root==null){
        return;
    }
    postOrderPrint(root.left);
    postOrderPrint(root.right);
    System.out.print(root.data +",");
}
```

Level order Traversal

↳ Rec. way

↳ Iterative way ⇒ BFS Breadth first Search (Level order Trav)



output

1

2 3

4 5 6 7

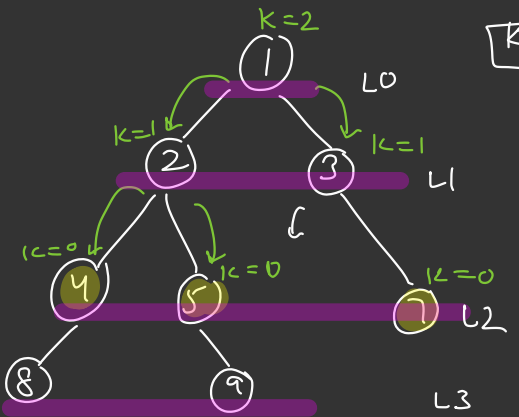
9 10 8

Depth

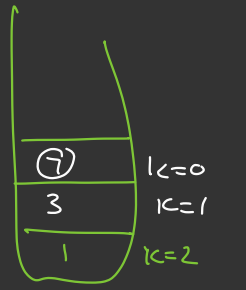
K = 2

4, 5, 6, 7

```
print Kth Level ( Node Root , int K) {
    → if (root == null)
          return
    → if (K == 0)
          ⇒ print (Root data)
          return ;
    // Rec case
```



K=2 ⇒ (4,5,7)



⇒ print K^{th} Level (Root Left, K-1)

⇒ print K^{th} Level (Root Right, K-1)

}

4, 5, 7

print All Levels (Node Root) {

N

for (; K=0 ; K < height(tree); K++) {

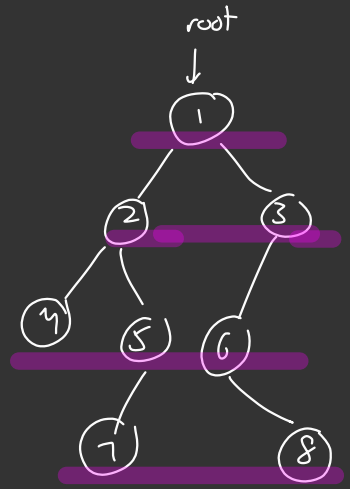
print K^{th} Level (root, K);

N

}

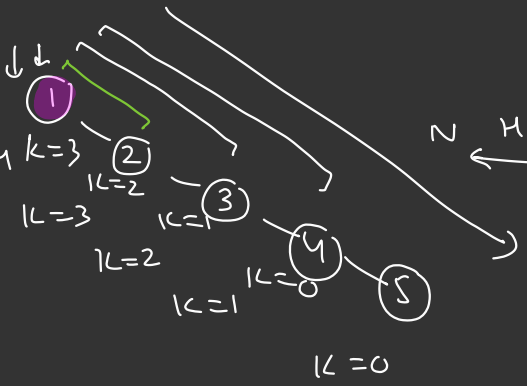
}

Time Complexity : $O(N^2)$



N Nodes

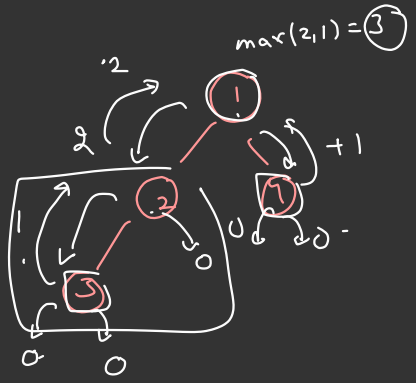
Skewed Tree
(worst case)
N = 5

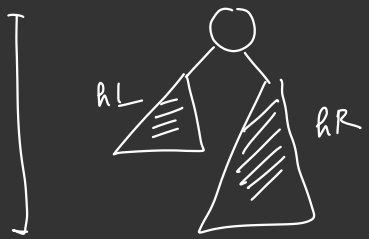


	units
K=0	1
K=1	2
K=2	3
K=3	4
K=4	5

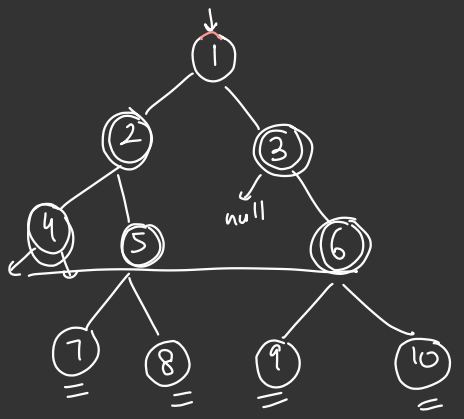
$1+2+3+\cdots+n$

$= O(N^2)$



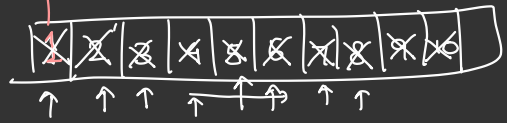


$H = \max(hL, hR) + 1$



Output 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Queue is empty ⇒ stop



Remove → front

Pushig → Rear

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

O(N) time

O(N) space

1

2 3

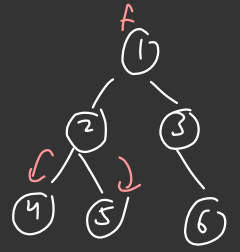
4 5 6

7 8 9 10

BFS

```
levelOrderTraversal(Node root){
    Queue<Node> q = new LinkedList<Node>();
    q.add(root);
    while(!q.empty()){
        Node f = q.poll(); // Retrieve + Remove.
        print(f.data)
        if(f.left != null){
            q.add(f.left);
        }
        if(f.right != null){
            q.add(f.right);
        }
    }
}
```

iterative also



Single line

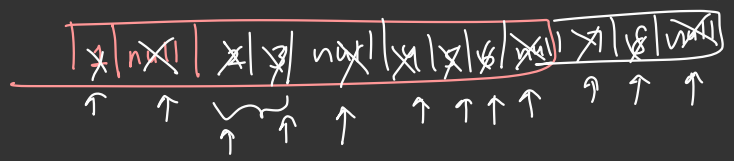
1, 2, 3, 4, 5, 6

Note down the end of each level.

1, 2, 3, 4, 5, 6

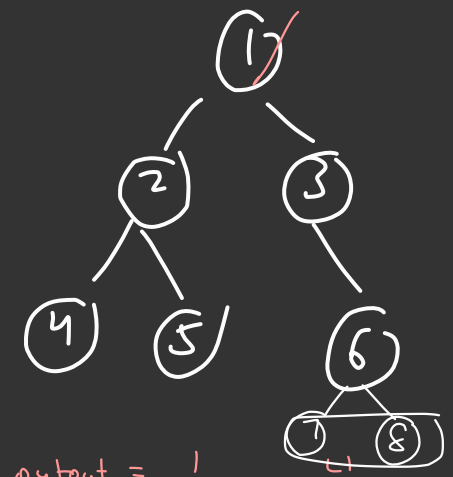
Queue <Node>

q.empty().



q.add(root)
q.add(null)

→ 1
→ 2 3
→ 4 5 6
→ 7 8
→ Stop → after popping last null



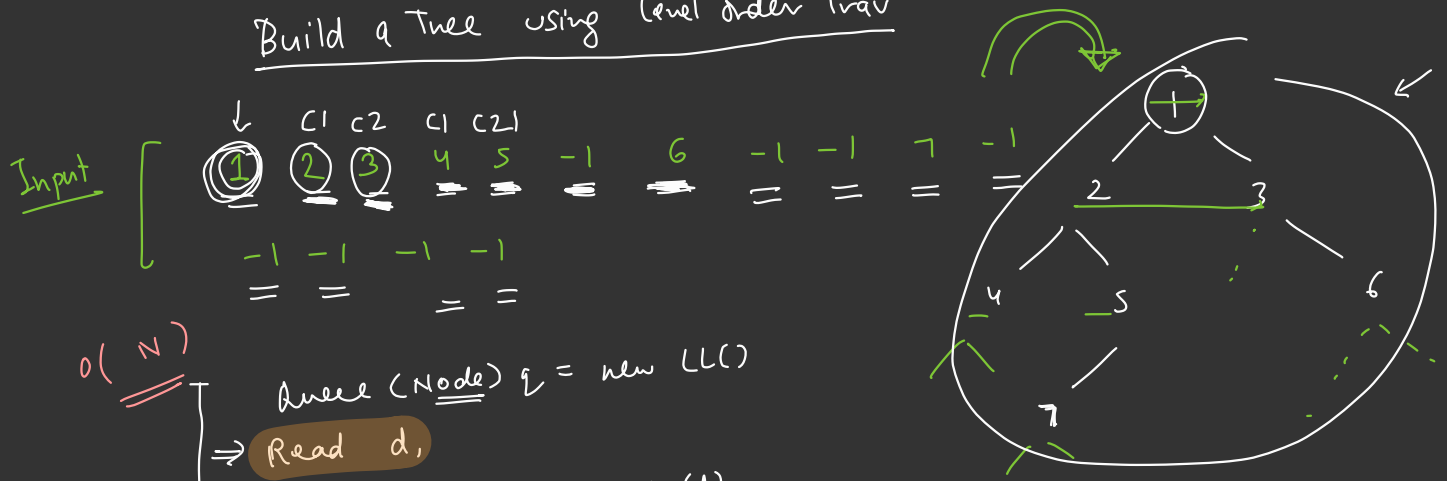
output = 1 L1
2 3 L2
4 5 6 L3

```
while(!q.empty()) {
    f = q.poll()
    if(f==null){
        ==
    }
    else {
        ==
```

}

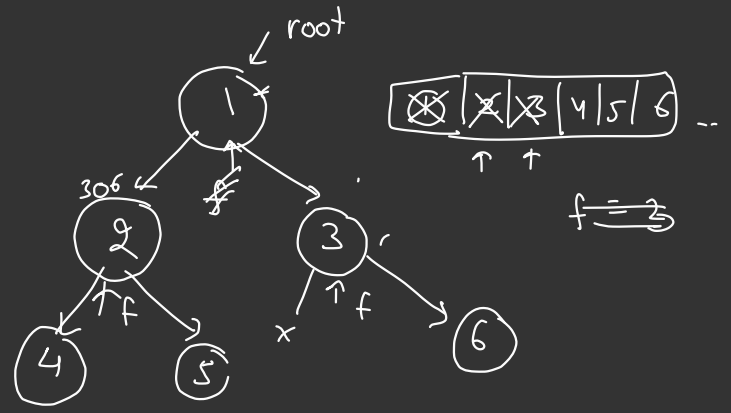
sc.nextInt()

Build a Tree using level order Trav



O(N)

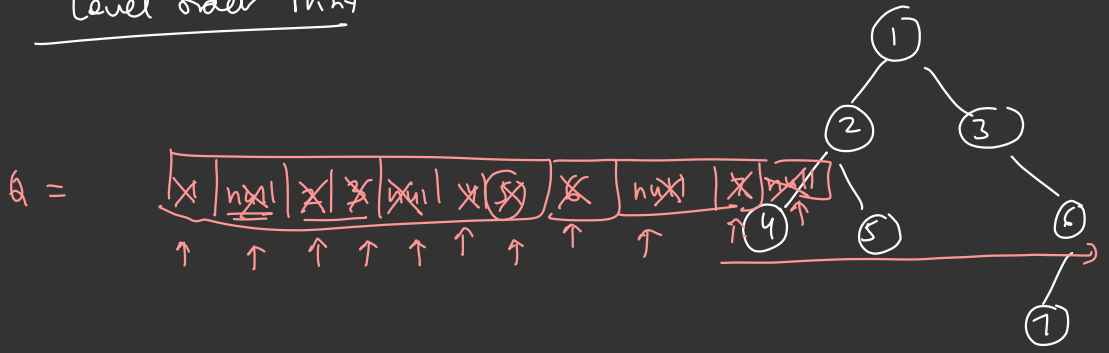
```
Queue (Node) q = new LL()
⇒ Read d,
⇒ Node root = new Node(d)
  q.add(root),
  while(!q.empty()){
    ⇒ f = q.poll();
      Read c1, c2;
      if(c1 != -1){
        ⇒ f.left = new Node(c1),
          q.add(f.left)
```



```
        }
        if(c2 != -1){
            f.right = new Node(c2);
            q.add(f.right)
        }
    }
}
```

```
int height(){
    return heightHelper(root) - 1;
}

private int heightHelper(Node root){
    if(root==null){
        return 0;
    }
    int h1 = heightHelper(root.left);
    int h2 = heightHelper(root.right);
    return Math.max(h1,h2) + 1;
   // return Math.max(heightHelper(root.left),heightHelper(root.right)) + 1;
}

int countNodes(Node root){
    if(root==null){
        return 0;
    }
    return 1 + countNodes(root.left) + countNodes(root.right);
}
int sumNodes(Node root){
    if(root==null){
        return 0;
    }
    return root.data + sumNodes(root.left) + sumNodes(root.right);
}
```

Level order Print



- 1
- → 2 3
  - → 4 5 6
- → 7
- →