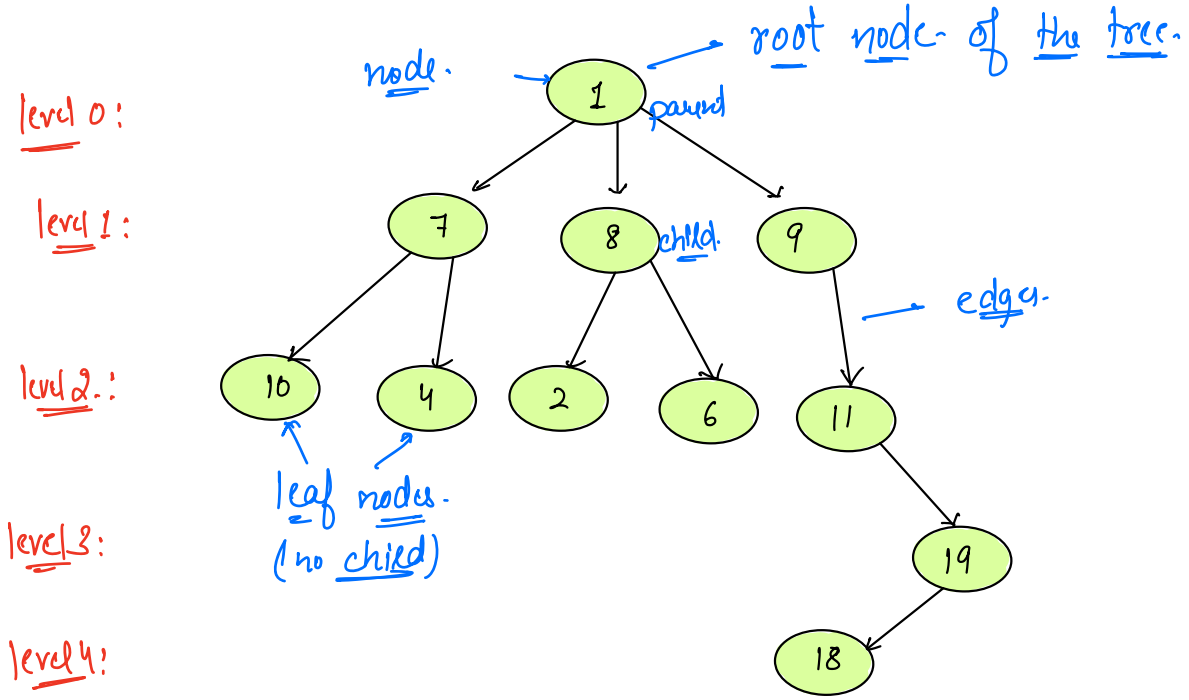
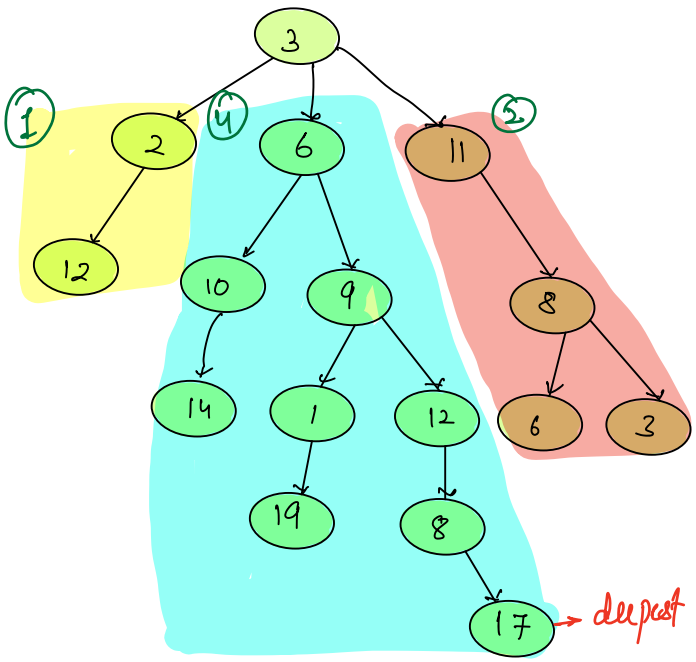


→ Every Node will have only one parent node except root node.

- N - nodes are there in the tree.
  No. of edges? ⇒ Exactly N-1 edges.

Height Of a Tree.

Height = no. of edges b/w the root node & deepest leaf node.



deepest leaf.

ans = max(1,4,2) + 1.

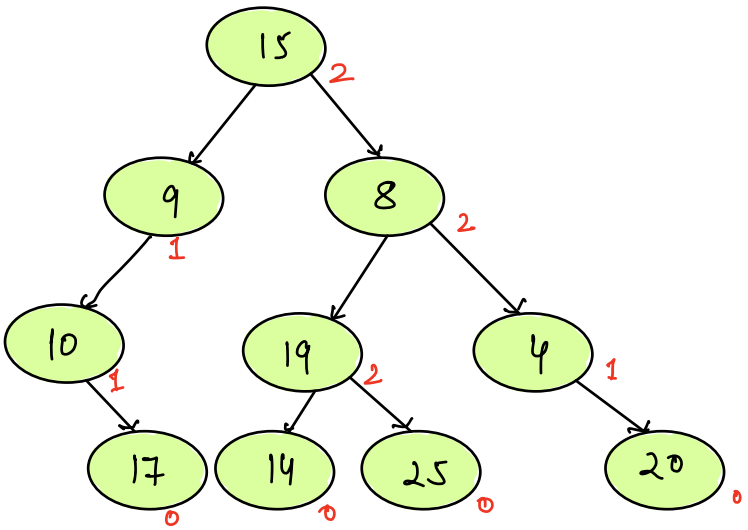
[ans = 5]

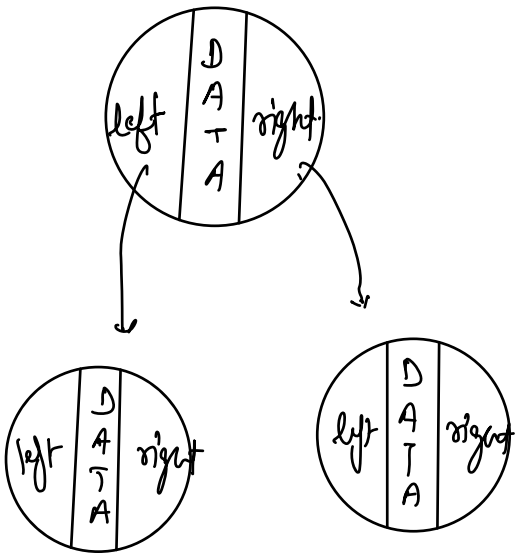
```
if (node == NULL){
    return -1
}
```

[if ht is defined as →
no of nodes from root node to deepest leaf node.
ans → 6.

if (node == NULL) {return 0}

Binary Tree. → Every node can have maximum 2 children.
0, 1, 2.



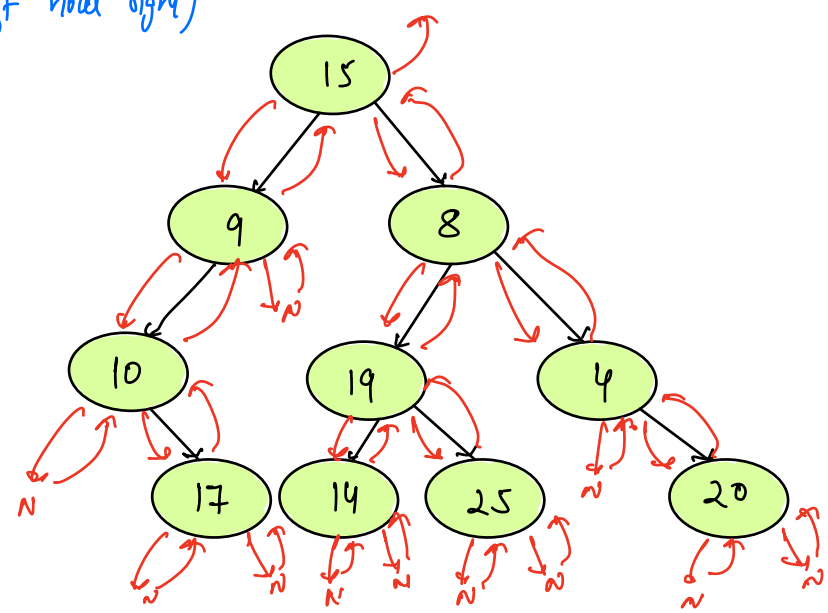



```
class Node {
    int data;
    Node left;
    Node right;
}
```

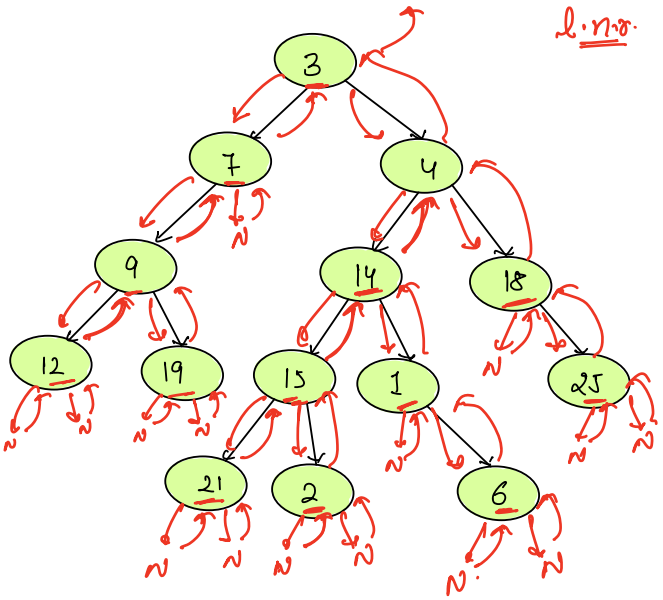
Traversals.

- Recursive.
  - → Inorder — left node right.
  - → Pre-order — node. left. right.
  - → Post-order — left. right. node.
- Iterative.
  - → level. order

In-order. (left node right)



o/p. → 10, 17, 9, 15, 14, 19, 25, 8, 4, 20.



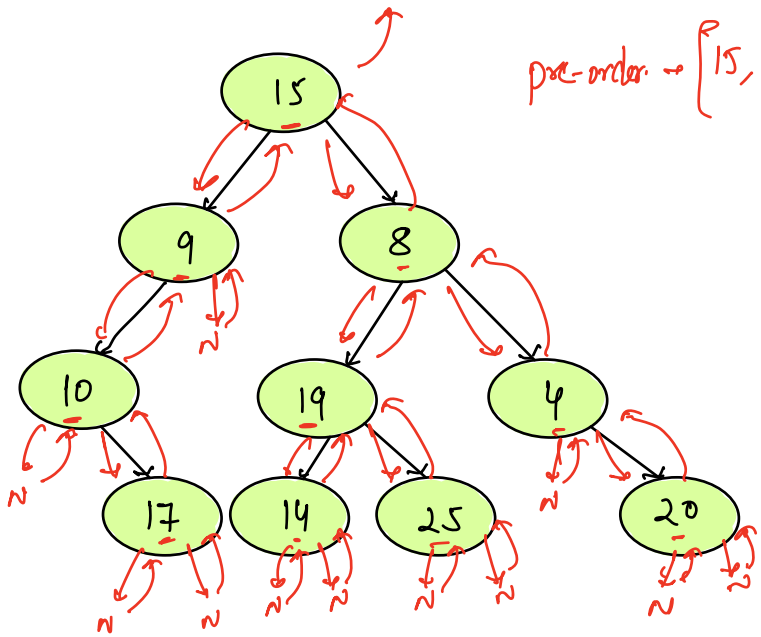
in-order →

[12, 9, 19, 7, 3, 21, 15, 2, 14, 1, 6, 4, 18, 25]

Ans: Given root node, traverse & print all the elements in in-order

```
void in-order ( Node root ) {
    if (root == NULL) { return }
    inorder ( root.left );
    print ( root.data );
    inorder ( root.right );
}
```

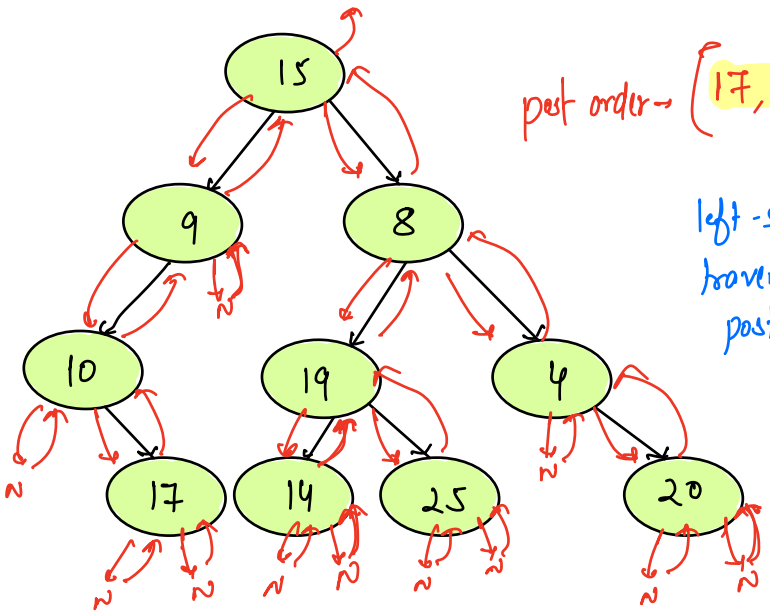
Pre-order node left right



pre-order → [15, 9, 10, 17, 8, 19, 14, 25, 4, 20]

```
void pre-order(Node root){
    if (root == NULL) { return }
    print(root.data);
    pre-order(root.left);
    pre-order(root.right);
}
```

Post-order. left right node.



post order → [17, 10, 9, 14, 25, 19, 20, 4, 8, 15]

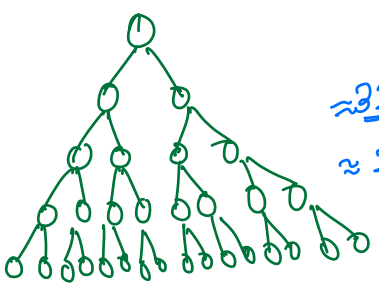
- 17, 10, 9 → left-subtree traversal in post-order
- 14, 25, 19, 20, 4, 8 → right-subtree traversal in post-order
- 15 → node.

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

T.C → O(N)

S.C → O(H. of tree)

log N to N.

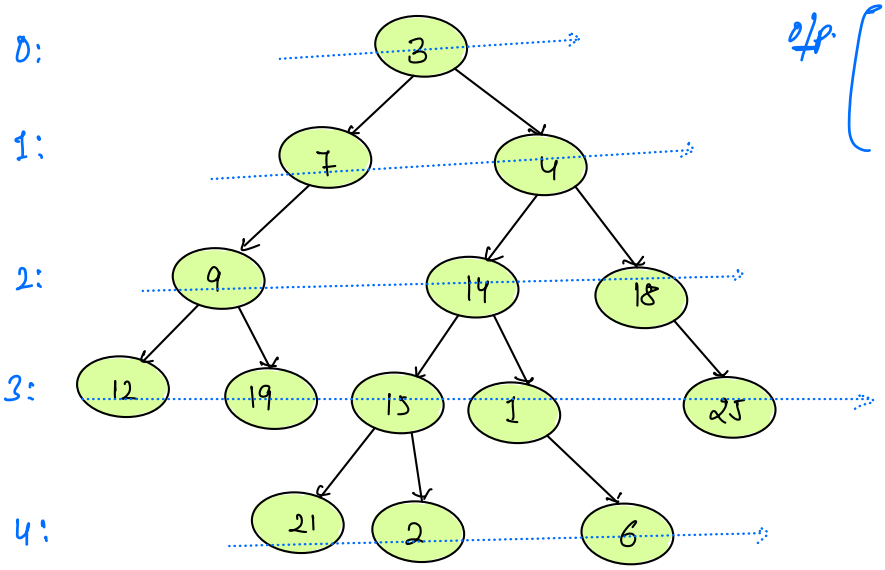


≈ 32 nodes → ht = 5

≈ 2^5 nodes → ht = 5

N nodes → ht = $\log_2 N$

Level Order Traversal (left to right)



o/p [3, 7, 4, 9, 14, 18, 12, 19, 15, 1, 25, 21, 2, 6.]

magic-box (Queue)

o/p → [3, 7, 4, 9, 14, 18, 12, 19, 15, 1, 25, 21, 2, 6]

Syntax in Java.

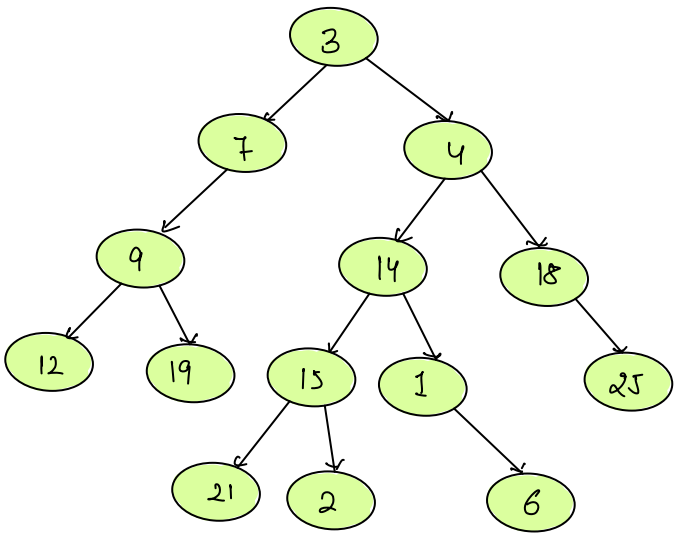
```
Queue<Node> q = new ArrayDeque<>();
q.addLast()      , q.size()
q.removeFirst()  , q.getFirst()
```

pseudo-code.

```
Queue<Node> q;
q.enqueue(root);
while( q.size() > 0){
    Node r = q.dequeue();
    print(r.data);
    if(r.left != NULL){ q.enqueue(r.left) };
    if(r.right != NULL){ q.enqueue(r.right) };
}
```

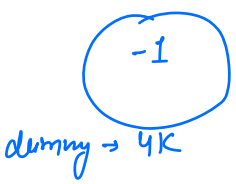
T.C → O(N)
S.C → O(N)

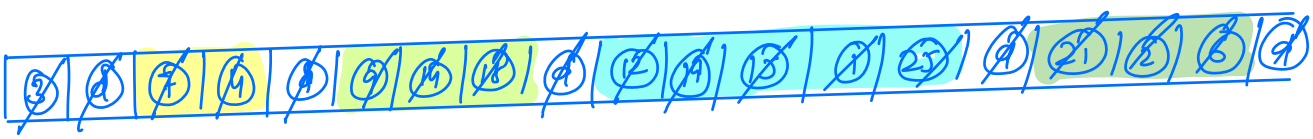
Level - Order LineWise -



o/p.

[
3
7 4
9 14 18
12 19 15 1 25
21 2 6
]





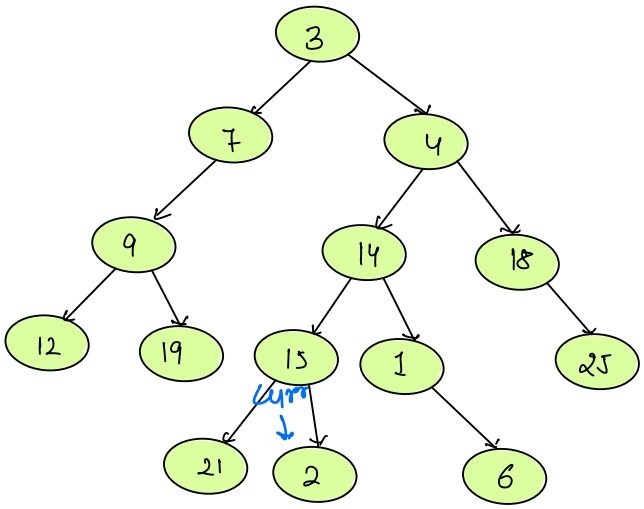
3
7 4
9 14 18
12 19 15 1 25
21 2 6

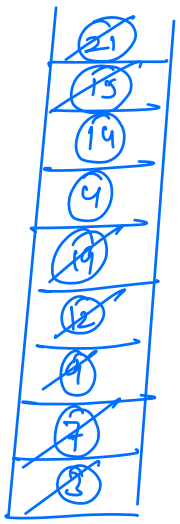
```
Queue<Node> q;                Node dummy = new Node(-1);
q.enqueue(root);              q.enqueue(dummy)
while( q.size() > 1){
        Node r = q.dequeue();
        if (r == dummy) {
                print("\n");
                q.enqueue(dummy);
        }
        else {
                print(r.data);
        if(r.left != NULL) { q.enqueue(r.left) };
        if(r.right != NULL) { q.enqueue(r.right) };
        }
}
```

T.C → O(N)
S.C → O(N)

In-order (iteratively)

left. node. right.





```
Stack st;
Node curr = root

while (st.size() == 0 || curr != NULL) {
    while (curr != NULL) {
        st.push(curr);
        curr = curr.left
    }
    curr = st.pop();
    print(curr.data);
    curr = curr.right
}
```

12, 9, 19, 7, 3, 21, 15,

for pre-order / post-order [todo]

[T.C → ?
S.C → O(N)]