Todays Content:

- → Binary Tree
- → Traversal
- → Specs
- → Iterative traversals
- → level order
- → Invert BT

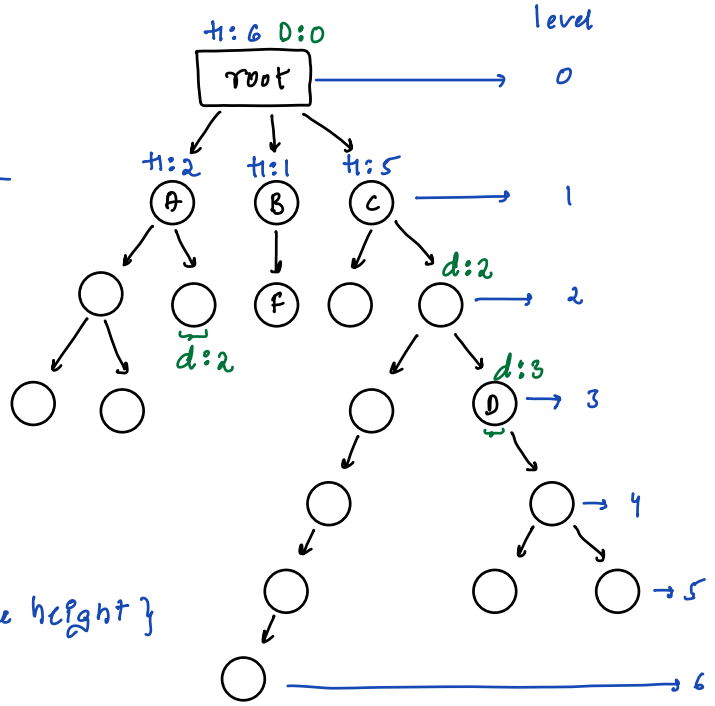
Tree Basics:

Height(Node):

length of longest path from node to any leaf of its

leaf node

no child



Height(Node) =

1 + max{child node height}

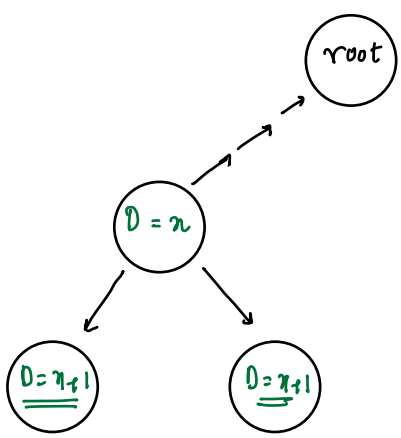
Height(leaf) = 0

Depth(Node) = length of path from root to given node

Depth(root) = 0

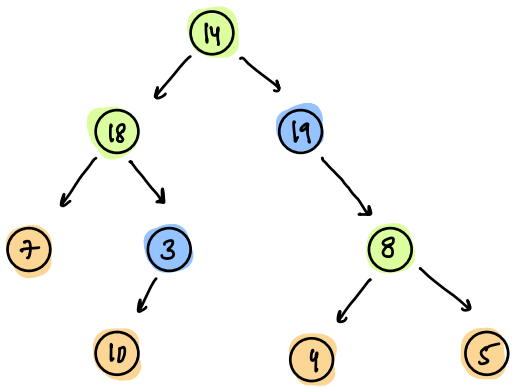
Depth(node) = n

Depth of its child node = n+1



Binary Tree: Any node can at max have 2 child nodes

0, 1, 2 child



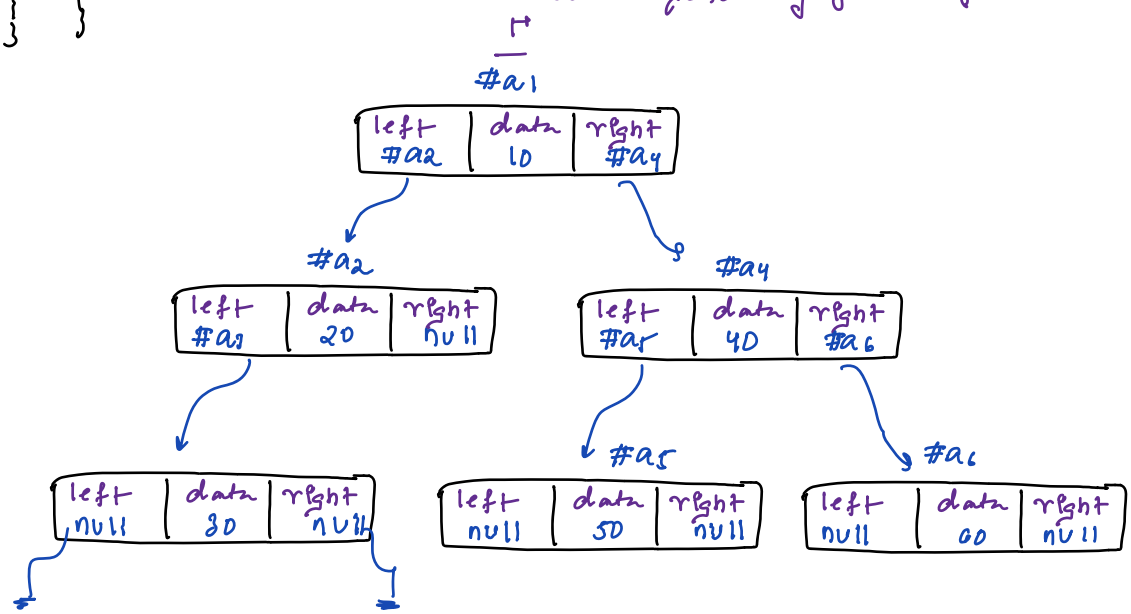
→ node, no children

→ nodes, 1 children

→ node 2 children

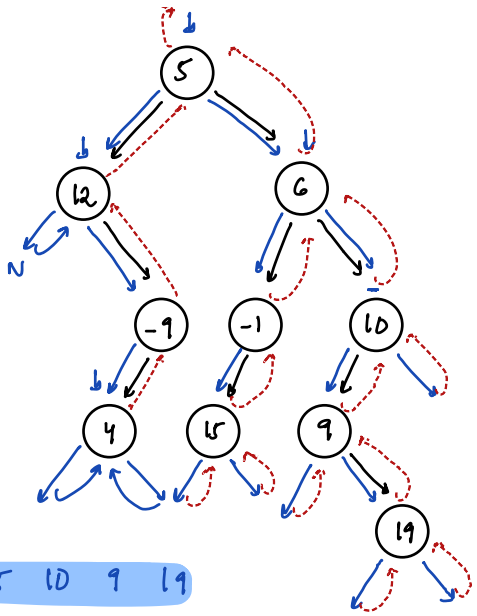
```
Class Node {
    int data
    Node left   } obj reference can hold address of node object
    Node right
    Node(x) { // constructor: To initialize data members of class
        data = x
        left = null
        right = null
    }
}
```

We are generally given only root node

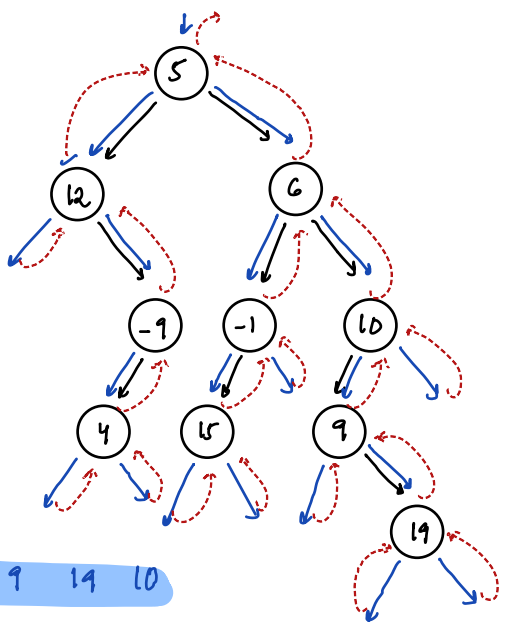


Tree Traversals :

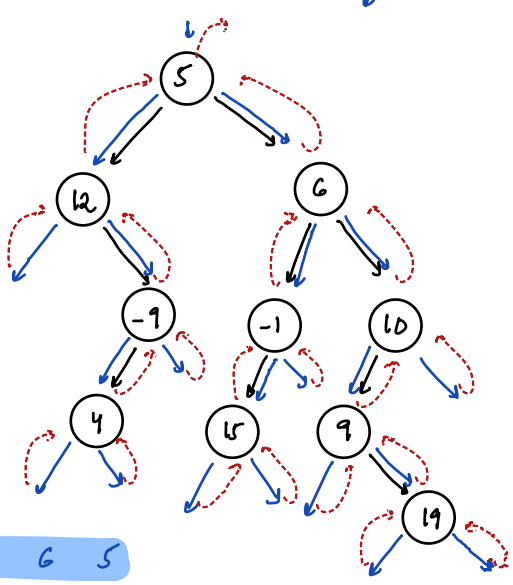
- → preorder : D L R
- → Inorder : L D R
- → postorder : L R D



preorder: 5 12 -9 4 6 -1 15 10 9 19



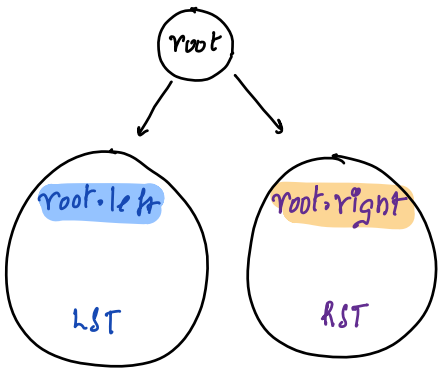
Inorder: 12 4 -9 5 15 -1 6 9 19 10



Postorder: 4 -9 12 15 -1 19 9 10 6 5

Ass: Given root node of tree, it will print entire tree in preorder

```
void preorder(Node root){ TC: O(N) SC: O(H)
                                  └ Height of Tree
    1. if(root==null){ return;}
    2. print(root.data)
    3. preorder(root.left)
    4. preorder(root.right)
}
```



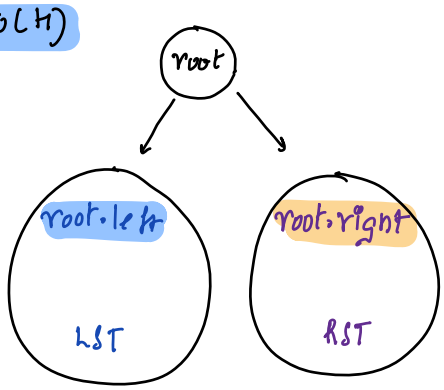
preorder: 1 2 3 4

inorder: 1 3 2 4

postorder: 1 3 4 2

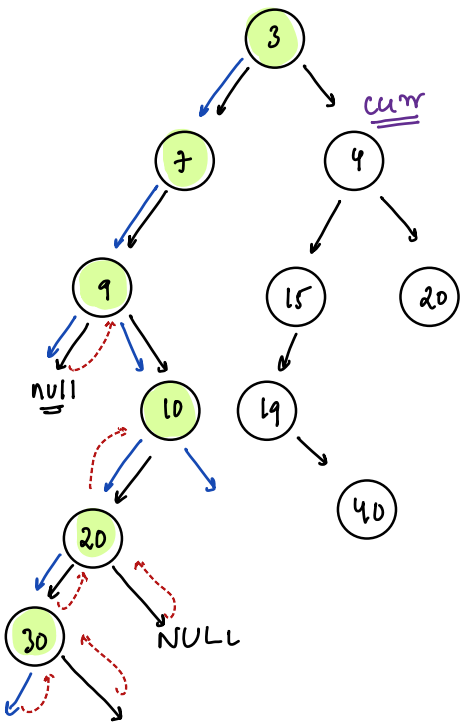
Ass: Given root node, calulate & return size of Tree

```
int size(Node root){ TC: O(N) SC: O(H)
    if(root==null){ return 0;
    l = size(root.left)
    r = size(root.right)
    return l+r+1
}
```



8:05 ——→ 8:15 Am

Inorder (Iteratively)



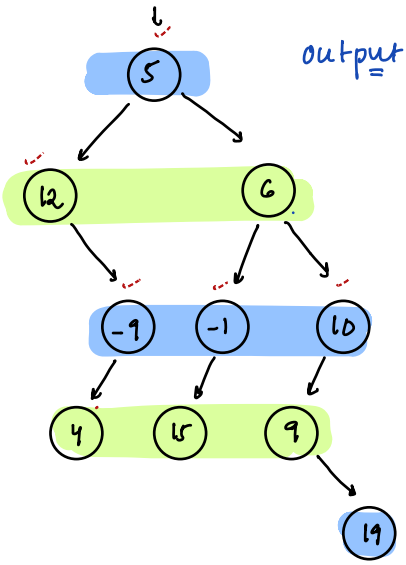
void inorder(Node root){ TC: O(N) SC: O(H)

```
Node curr = root
stack<Node> st
while(st.size()>0 || curr!=null){
    while(curr!=null){
        st.push(curr)
        curr = curr.left
    }
    curr = st.top()
    st.pop()
    print(curr.data)
    curr = curr.right
}
}
```

Inorder: 9 30 20 10 7 3 . . . . .

TODO: Iterative preorder & postorder

level order traversal :



output: 5 12 6 -9 -1 10 4 15 9 19

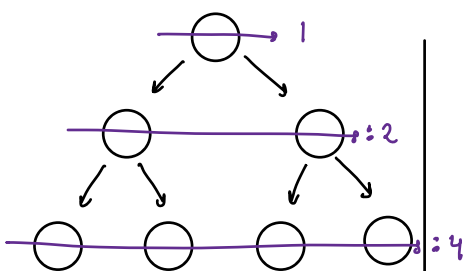
idea:

0	1	2	3	4
5	12 6	-9 -1 10	4 15 9	19

5 12 6 -9 -1 10 4 15 9 19

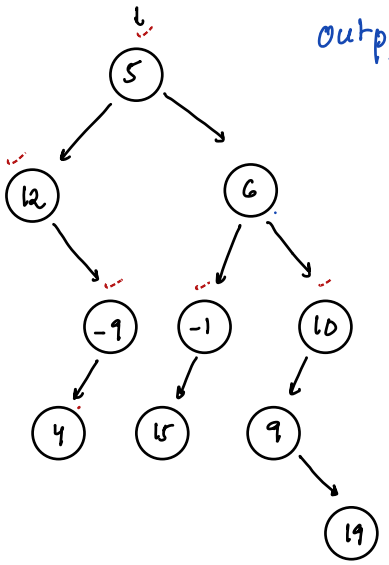
void levelorder(Node root){ TC: O(N) SC: (max nodes in a level) ≈ N/2 = O(N)

```
Queue<Node> q
q.enque(root)
while(q.size()>0){
    Node tmp = q.front()
    q.deque()
    print(tmp.data)
    if(tmp.left != null){
        q.enque(tmp.left)
    }
    if(tmp.right != null){
        q.enque(tmp.right)
    }
}
}
```

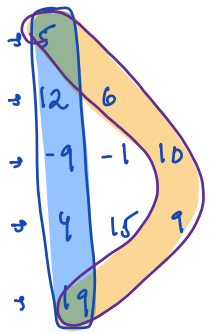


N=7, max nodes in level = $\frac{N+1}{2}$

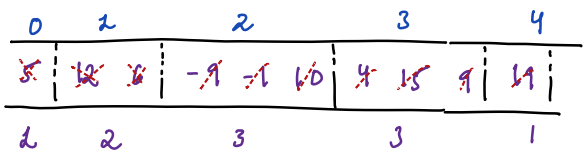
level order traversal : 2



output:

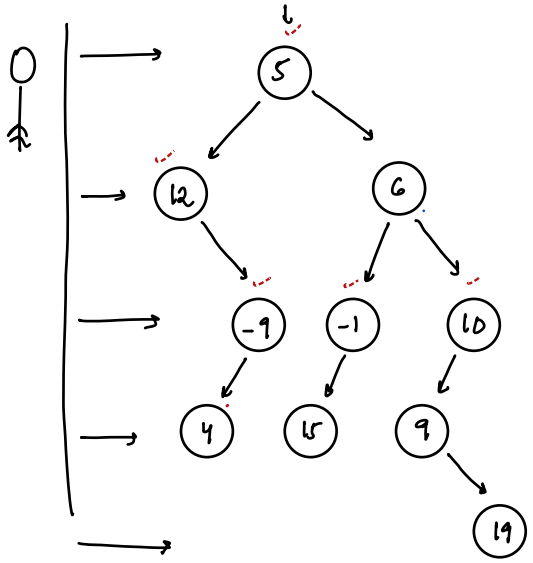


idea: Get no: of nodes in level, iterate those many times & print & push its children




```
void levelorder(Node root){ TC: O(N)   SC: (max nodes in a level) ≈ N/2 = O(N)
    if(root==null) {return}
    Queue<Node> q
    q.enque(root)
    while(q.size()>0){
        int n = q.size()
        i=1; i<=n; i++){ // It is helping us in printing a level
            Node tmp = q.front()
            q.deque()
            print(tmp.data)
            if(tmp.left!=null){
                q.enque(tmp.left)
            }
            if(tmp.right!=null){
                q.enque(tmp.right)
            }
        }
        print('\n') // new line
    }
}
```

left view:

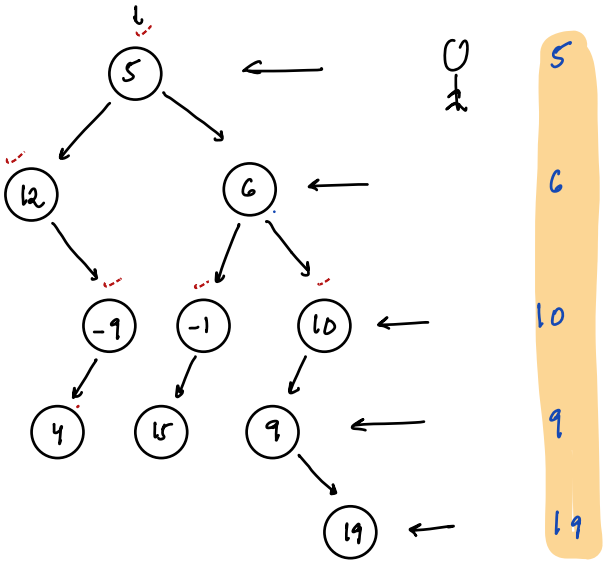


leftview of Tree
↳ 1st node in every level

5
12
-9
4
19

```
void leftview(Node root){ TC: O(N)   SC: (max nodes in a level) ≈ N/2 = O(N)
    if(root==null){return}
    Queue<Node> q
    q.enque(root)
    while(q.size()>0){
        int n = q.size()
        for(i=1; i<=n; i++){ // It is helping us in printing a level
            Node tmp = q.front()
            q.deque()
            if(i==1){print(tmp.data)}
            if(tmp.left!=null){
                q.enque(tmp.left)
            }
            if(tmp.right!=null){
                q.enque(tmp.right)
            }
        }
        print('\n') // new line
    }
}
```

right view:

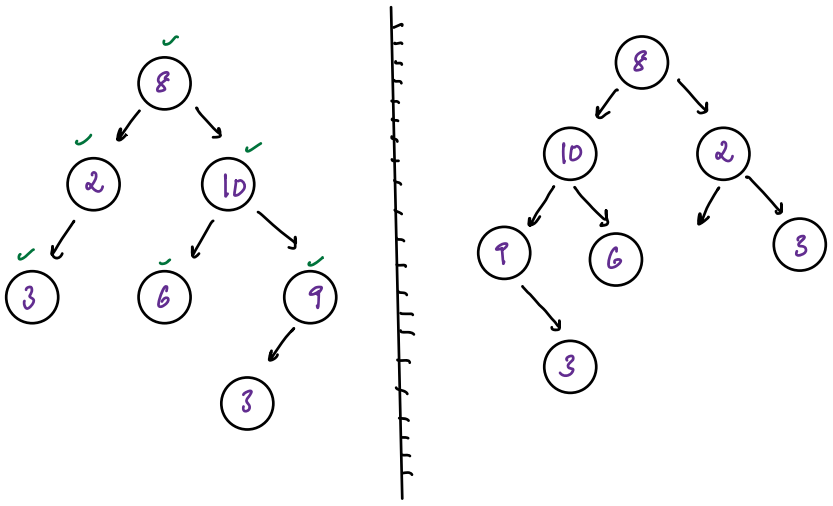


5 6 10 9 19

rightview: last element in every level

```
void rightview (Node root){ TC: O(N)   SC: (max nodes in a level) ≈ N/2 = O(N)
    if(root==null) {return}
    Queue<Node> q
    q.enque(root)
    while(q.size()>0){
        int n = q.size()
        i=1; i<=n; i++){ // It is helping us in printing a level
            Node tmp = q.front()
            q.deque()
            if(i==n) { print(tmp.data) }
            if(tmp.left!=null){
                q.enque(tmp.left)
            }
            if(tmp.right!=null){
                q.enque(tmp.right)
            }
        }
        print("\n") // new line
    }
```

Invert BT / Mirror view of BT



Ass: Given root, invert given binary tree

```
void mirror(Node root){
    if(root==null){return;}
    mirror(root.left)
    mirr(root.right)
    //swap root.left & root.right
     Node tmp = root.left
     root.left = root.right
     root.right = tmp
}
```