Agenda

1) Iterative Inorder **

2) Construct Binary tree from preorder and Inorder. **

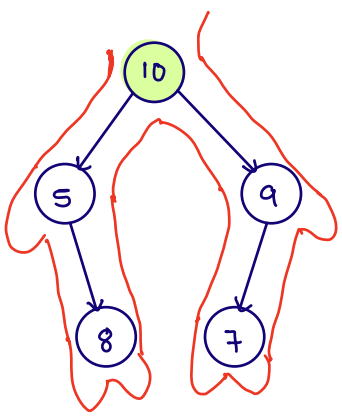
3) Level order traversal **
   → left view
   → right view

Q.1 Given root of a binary tree, return its inorder.

Note: Recursion is not allowed.

Inorder: 5 8 10 7 9

idea: copy the working of recursion

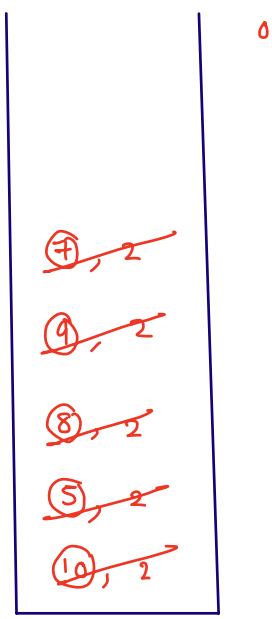


State Observe

0 → pre, add left child, inc.state

1 → In, print, add right child, inc.state

2 → post, st.pop()



Stack < Pair >

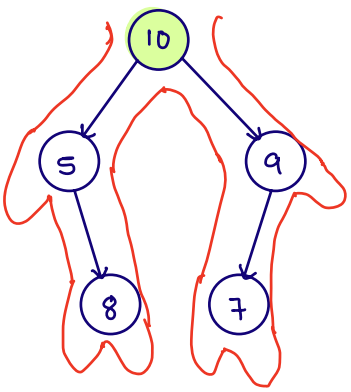
O/P: 5 8 10 7 9

```
class Pair {
    Node node;
    int state;
}
```

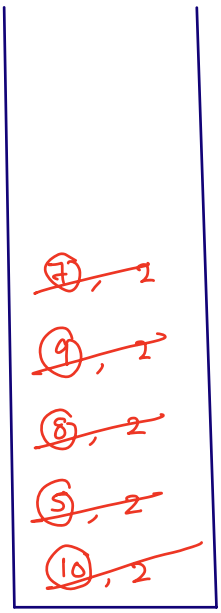
0 → pre , inc.state , add Left child

1 → in, print, inc.state, add right child

2 → post, st.pop()



O/P: 5 8 10 7 9



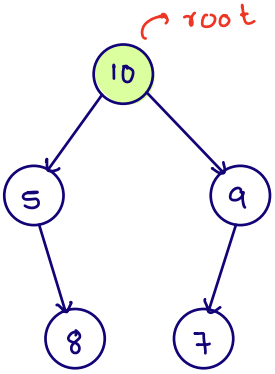
```
void IterativeTraversal(Node root) {
    Stack<Pair> st = new Stack<>();
    st.push(new Pair(root, 0));
    while (st.size() > 0) {
        Pair top = st.peek();
        if(top.state == 0) {
            // pre
            -> add left child and update top' state
        }
        else if (top.state == 1) {
            // in
            -> print, add right and update top's state
        }
        else {
            // post
            st.pop();
        }
    }
}
```

```java
public class Solution {
    class Pair {
        TreeNode node;
        int state;

        Pair(TreeNode node,int state) {
            this.node = node;
            this.state = state;
        }
    }

    public ArrayList<Integer> inorderTraversal(TreeNode root) {
        Stack<Pair>st = new Stack<>();
        ArrayList<Integer>ans = new ArrayList<>();

        st.push(new Pair(root,0));

        while(st.size() > 0) {
            Pair top = st.peek();

            if(top.state == 0) {
                //pre

                //inc state
                top.state++;

                //add left child pair
                TreeNode lc = top.node.left;
                if(lc != null) {
                    st.push(new Pair(lc,0));
                }
            }
            else if(top.state == 1) {
                //in
                ans.add(top.node.val);

                //inc state
                top.state++;

                //add right child pair
                TreeNode rc = top.node.right;
                if(rc != null) {
                    st.push(new Pair(rc,0));
                }
            }
            else if(top.state == 2){
                //post
                st.pop();
            }
        }

        return ans;
    }
}
```

Q.2 Construct a binary tree with given preorder and inorder and return root of binary tree.

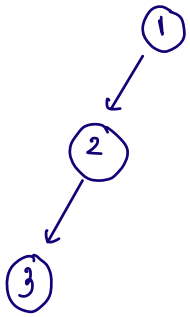
Pre: N L R

In: L N R



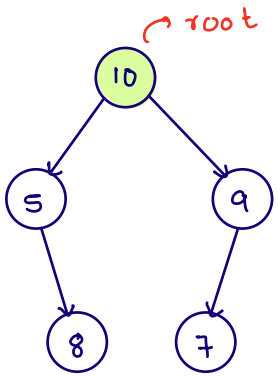
	0	1	2	3	4
pre =	10	5	8	9	7

	0	1	2	3	4
In =	5	8	10	7	9



	0	1	2
pre =	1	2	3

	0	1	2
in =	3	2	1



pre = 10 5 8 9 7 (indices 0 1 2 3 4; ps = 0, pe = 4)

in = 5 8 10 7 9 (indices 0 1 2 3 4; is = 0, idx = 2, ie = 4)

ps → pre start

pe → pre end

is → in start

ie → in end

dc = idx - is = 2

pre[ps] : 10

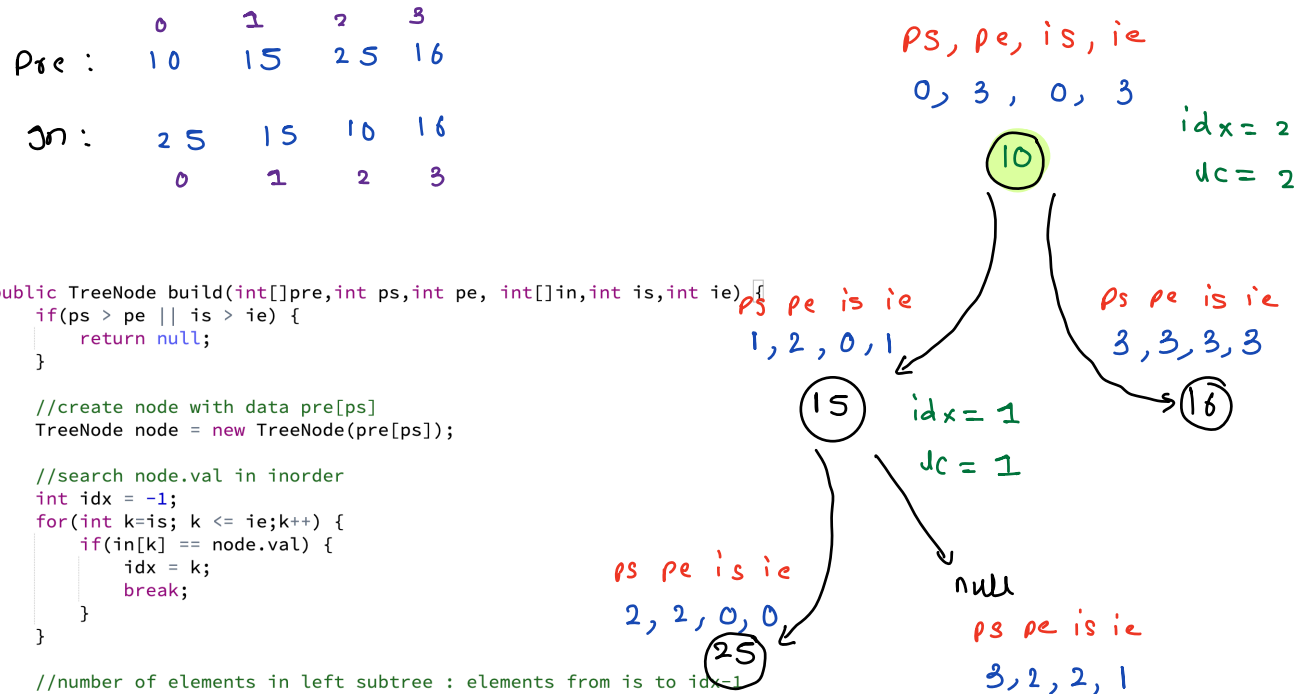
build left subtree

build right subtree

	left	right
pre	ps+1, ps+dc	ps+dc+1, pe
in	is, idx-1	idx+1, ie

no. of element that should go in left subtree

dc = idx - is

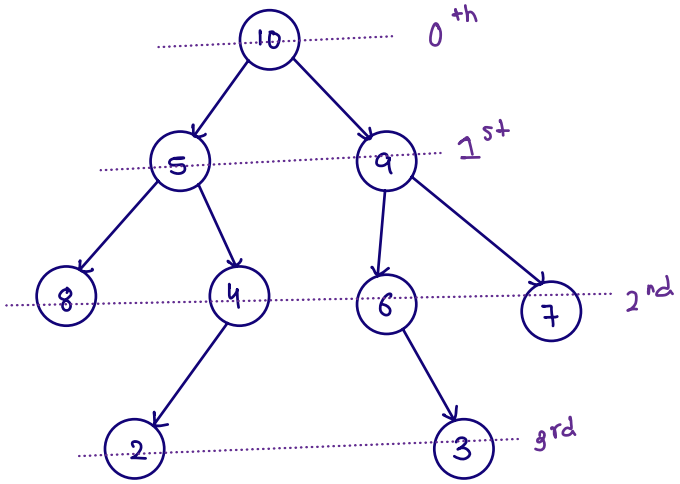



```java
public TreeNode build(int[]pre,int ps,int pe, int[]in,int is,int ie){
    if(ps > pe || is > ie) {
        return null;
    }

    //create node with data pre[ps]
    TreeNode node = new TreeNode(pre[ps]);

    //search node.val in inorder
    int idx = -1;
    for(int k=is; k <= ie;k++) {
        if(in[k] == node.val) {
            idx = k;
            break;
        }
    }

    //number of elements in left subtree : elements from is to idx-1
    int lc = idx - is;

    node.left = build(pre,ps+1,ps+lc,in,is,idx-1);
    node.right = build(pre,ps+lc+1,pe,in,idx+1,ie);

    return node;
}
```

complete code on next page

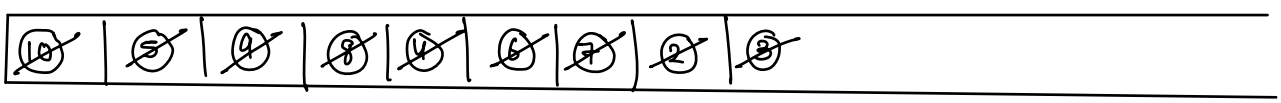
```java
public class Solution {
    public TreeNode build(int[]pre,int ps,int pe, int[]in,int is,int ie) {
        if(ps > pe || is > ie) {
            return null;
        }

        //create node with data pre[ps]
        TreeNode node = new TreeNode(pre[ps]);

        //search node.val in inorder
        int idx = -1;
        for(int k=is; k <= ie;k++) {
            if(in[k] == node.val) {
                idx = k;
                break;
            }
        }

        //number of elements in left subtree : elements from is to idx-1
        int lc = idx - is;

        node.left = build(pre,ps+1,ps+lc,in,is,idx-1);
        node.right = build(pre,ps+lc+1,pe,in,idx+1,ie);

        return node;
    }

    public TreeNode buildTree(int[] pre, int[] in) {
        int n = pre.length;
        return build(pre,0,n-1,in,0,n-1);
    }
}
```

Q.3 Given a binary tree, print its levelorder.

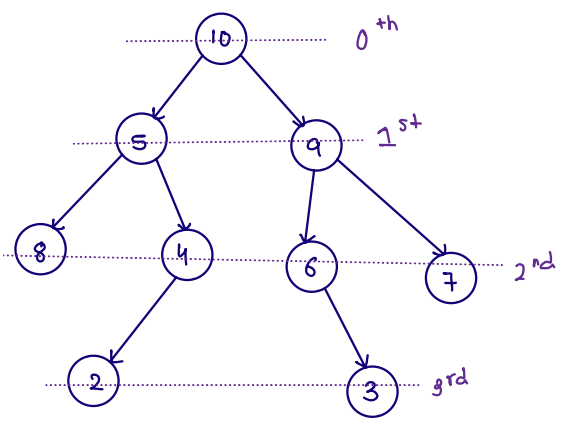


o/p :
10
5 9
8 4 6 7
2 3

queue <Node>



cs = 2

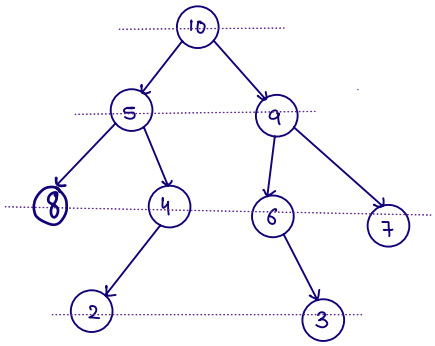


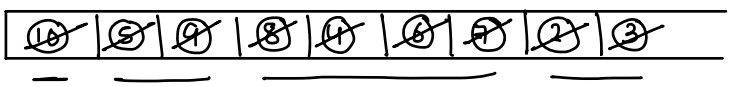

```
int cs = q.size();
q.size()>0
    cs times
        temp = q.remove();
        sop(temp.val);
        add child
    sopln();
```

o/p : 10 ↴
5 9 ↴
8 4 6 7 ↴
2 3 ↴

```
void printLevelOrder(Node root) {
    Queue<Node> q = new ArrayDeque<>();
    q.add(root);
    while (q.size() > 0) {
        int rs = q.size();
        for (int i = 1; i <= rs; i++) {
            Node temp = q.remove();
            sop(temp.val + " ");
            // add child
            if (temp.left != null) {
                q.add(temp.left);
            }
            if (temp.right != null) {
                q.add(temp.right);
            }
        }
        sopln();
    }
}
```

```
while (q.size() > 0) {
    int cs = q.size();
    for (int i=1; i<=cs; i++) {
        Node temp = q.remove();
        sop(temp.val + " ");
        // add child
        if (temp.left != null) {
            q.add(temp.left);
        }
        if (temp.right != null) {
            q.add(temp.right);
        }
    }
    sopln();
}
```





cs = 2

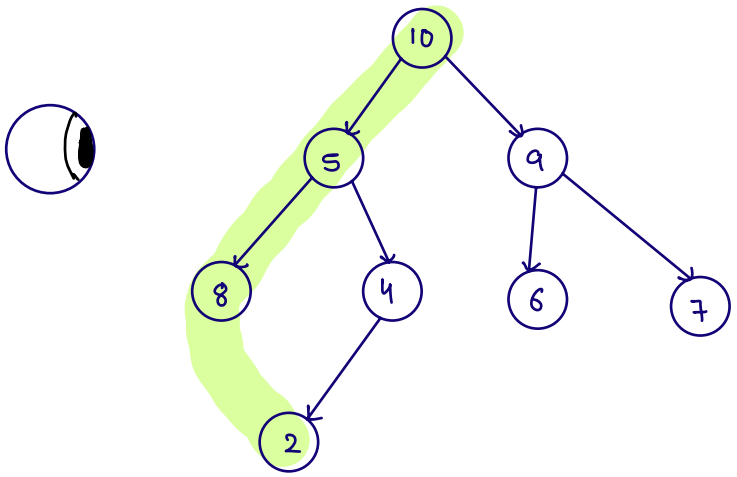
10 ↴

5 9 ↴

8 4 6 7 ↴

2 3 ↴

Left view of Binary tree

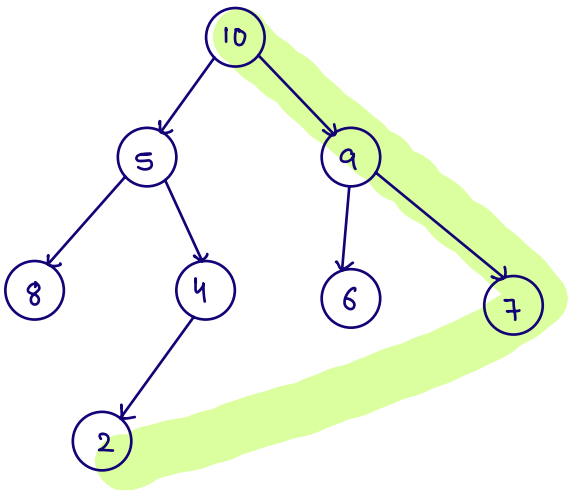


lv: 10 5 8 2

first node of each level

```
AL<Integer> leftview (Node root) {
    AL<Integer> ans = new AL<>();
    Queue<Node> q = new ArrayDeque<>();
    q.add(root);

    while (q.size() >0) {
        int cs = q.size();
        ans.add (q.peek().val);
        for (int i=1; i<=cs; i++) {
            Node temp = q.remove();
            // add child
            if (temp.left != null) {
                q.add (temp.left);
            }
            if (temp.right != null) {
                q.add (temp.right);
            }
        }
    }
    return ans;
}
```

Right view of Binary tree

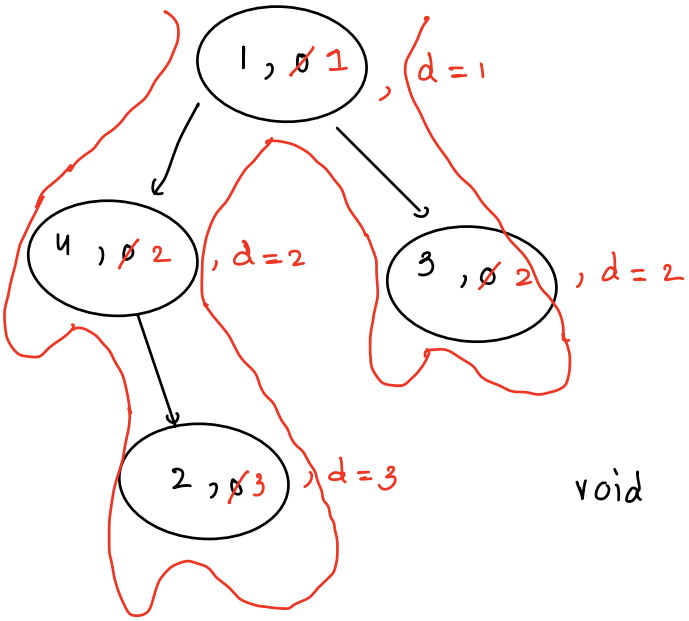


rv: 10 9 7 2

last node of each level

Code: todo

Doubts




```
TreeNode {
    int val;
    int depth;
    TreeNode left;
    TreeNode right;
}
```

```
void travel (TreeNode node, int d) {
    if(node == null) {
        return;
    }
    node.depth = d;
    travel(node.left, d+1);
    travel(node.right, d+1);
}

void solve (TreeNode A) {
    travel(A, 1);
}
```