Agenda

1) Iterative Inorder **

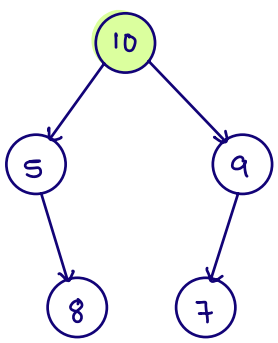
2) Level order traversal **
   → left view
   → right view

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

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

Note: Recursion is not allowed.

Idea: copy the working of Recursion in trees.

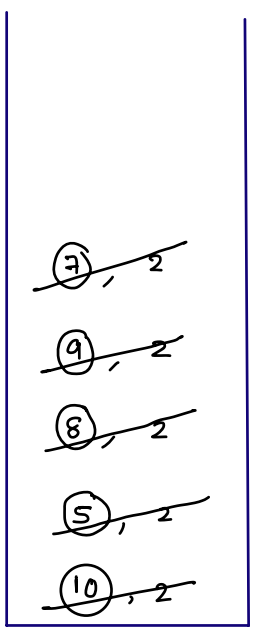


State — observation

0 : Pre, add left child, inc. state

1 : In, print val, add right child, inc. state

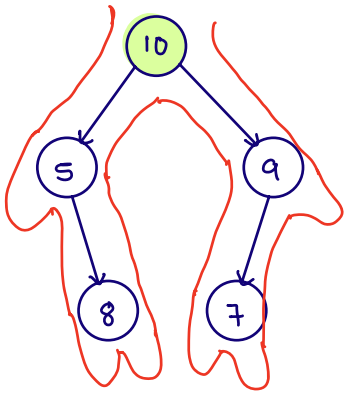
2 : Post, pop out



Stack < Pair >

5 8 10 7 9

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

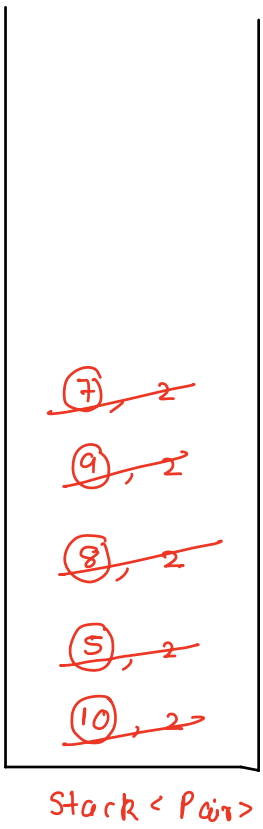


5 8 10 7 9

0 : Pre, add left child, inc. state

1 : In, print val, add right child, inc. state

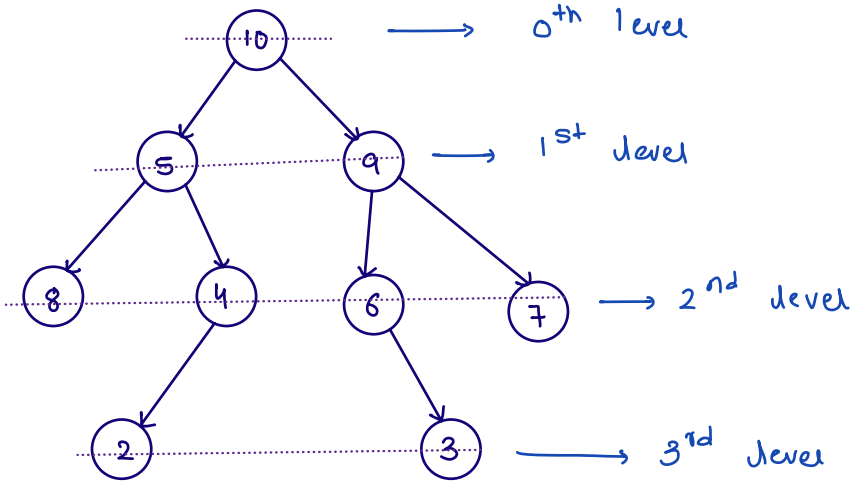
2 : Post, pop out




```
AL<Integer> Iterative_traversal (Node root) {
    Stack<Pair> st = new Stack<>();
    AL<Integer> ans = new AL<>();
    st.push(new Pair(root,0));
    while (st.size() >0) {
        Pair top = st.peek();
        if (top.state == 0) {
            // inc. state of top, add left child pair to stack
        }
        else if (top.state == 1) {
            // inc. state of top, print / ans, add right child pair to stack
        }
        else if (top.state == 2) {
            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 top's 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 top's 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 Given a binary tree, print its levelorder.

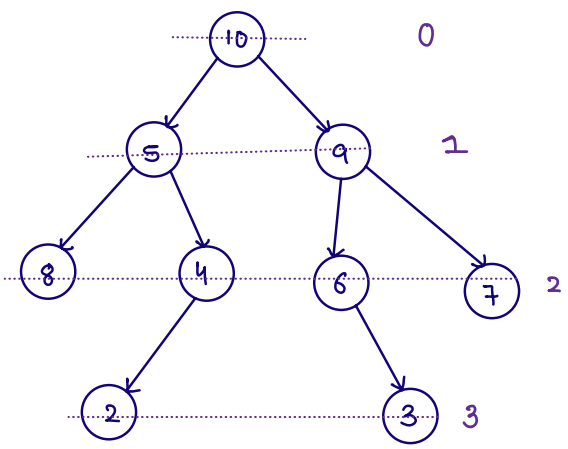


o/p

10
5 9
8 4 6 7
2 3

cs = 2

queue < Node >



int cs = q.size()

cs times
- → q.remove()
- → print it
- → add left & right child

Sopln()

10 ↴

5 9 ↴

8 4 6 7 ↴

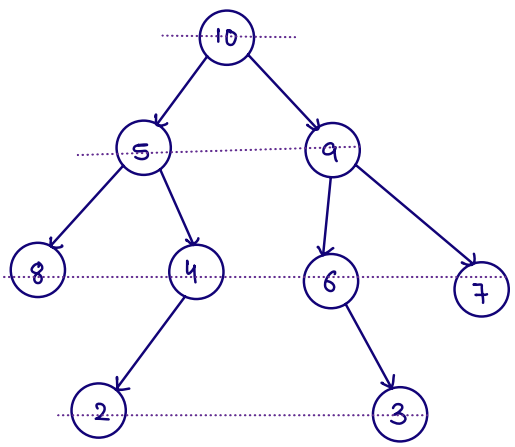
2 3 ↴

10	5	9	8	4	6	7	2	3

0 1 2 3

cs = 2

o/p : 10 ↴
5 9 ↴
8 4 6 7 ↴
2 3 ↴




```
int cs = q.size()
q.size()>0   cs times
                → q.remove()
                → print it
                → add left & right child
sopln()
```

```
void levelOrder(Node root) {
    Queue<Node> q = new ArrayDeque<>();
    q.add(root);
    while(q.size()>0) {
        int cs = q.size();
        for(int i=1; i<=cs; i++) {
            Node rem = q.remove();
            SOP(rem.val + " ");
            // add child
            if(rem.left != null) {
                q.add(rem.left);
            }
            if(rem.right != null) {
                q.add(rem.right);
            }
        }
        SOPln();
    }
}
```

todo: Return a 2D arrayList as answer

[[10], [5,9], [8,4,6,7], [2,3]]

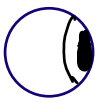
10 ↴

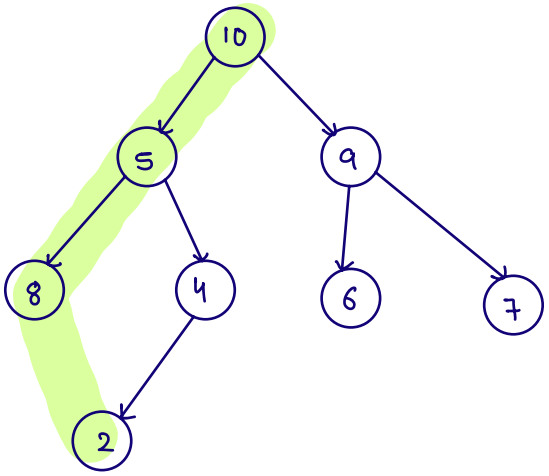
5 9 ↴

8 4 6 7 ↴

2 3 ↴

Left view of Binary tree





lv : 10 5 8 2

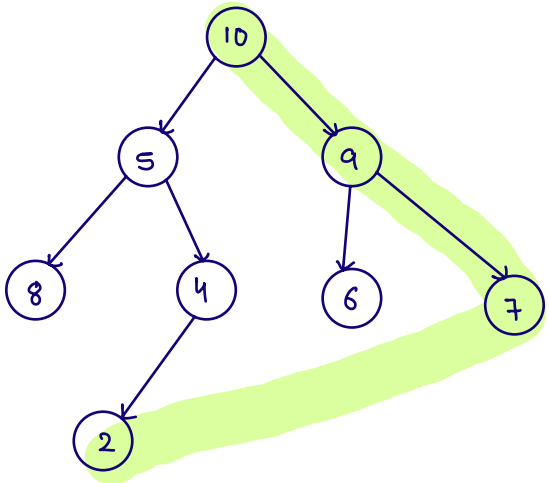
Left view:

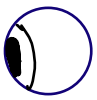
Each level's first node

```
ArrayList<Integer> leftView(Node root) {
    Queue<Node> q = new ArrayDeque<>();
    AL<Integer> ans = new AL<>();
    q.add(root);
    while(q.size()>0) {
        int cs = q.size();
        ans.add(q.peek().val);
        for(int i=1; i<=cs; i++) {
            Node rem = q.remove();
            // add child
            if(rem.left != null) {
                q.add(rem.left);
            }
            if(rem.right != null) {
                q.add(rem.right);
            }
        }
    }
    return ans;
}
```

3

Right view of Binary tree



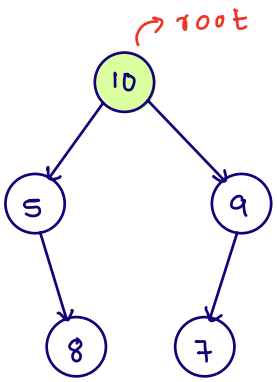


rv: 10 9 7 2

logic: add each level last value.

Code → todo

Q.3 Construct a binary tree with given preorder and inorder and return root of binary tree.



ps → pre start
pe → pre end
is → in start
ie → in end

ps pe

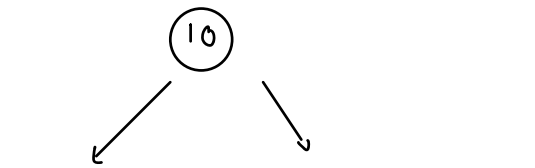
	0	1	2	3	4
pre =	10	5	8	9	7

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

ps, pe, is, ie

ps = 0, pe = 4, is = 0, ie = 4

pre[ps]



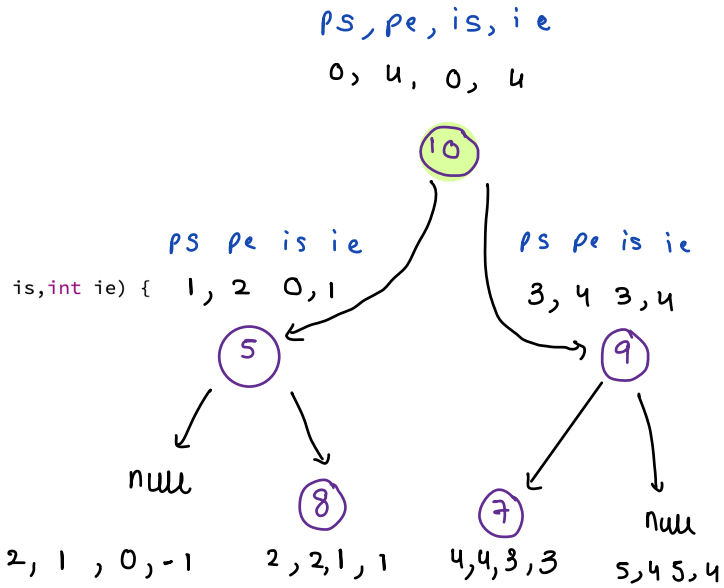
	left	right
In (LNR)	is, idx-1	idx+1, ie
pre (NLR)	ps+1, ps+lc	ps+lc+1, pe

Calculate count of left subtree elements : no. of ele from is to idx-1

lc = idx-is

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

in = 5 8 10 7 9 (0 1 2 3 4)

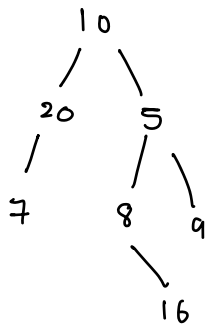



```java
static TreeNode build(int[]pre,int ps,int pe,int[]in,int is,int ie) {
    if(ps > pe || is > ie) {
        return null;
    }

    TreeNode node = new TreeNode(pre[ps]);

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

    //count of element 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 {
    static TreeNode build(int[]pre,int ps,int pe,int[]in,int is,int ie) {
        if(ps > pe || is > ie) {
            return null;
        }

        TreeNode node = new TreeNode(pre[ps]);

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

        //count of element 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);
    }
}
```

Doubts



10	20	7	5	8	16	9
ps						pe

7	20	10	8	16	5	9
is		idx				ie

(10) idc = 2

→ Pre & In (done)

↓

→ Post & In (very similar)