Today's Checklist

- What is a Tree data structure
- Representation
- Terminology
- Important properties of trees
- Types of Trees
- Applications of tree data structure
- What is a Binary Tree?
- Implementation
- Traversals
- Problems
- Types of Binary Trees

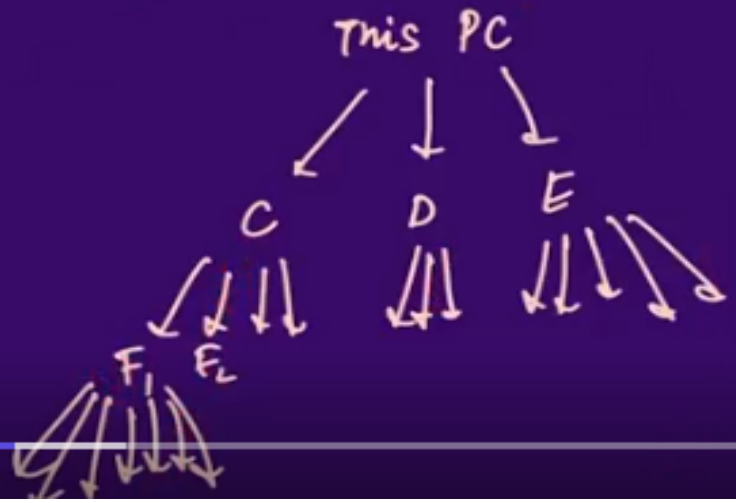
What is a Tree data structure

Array, LL, Stack, Queue → Linear data structures

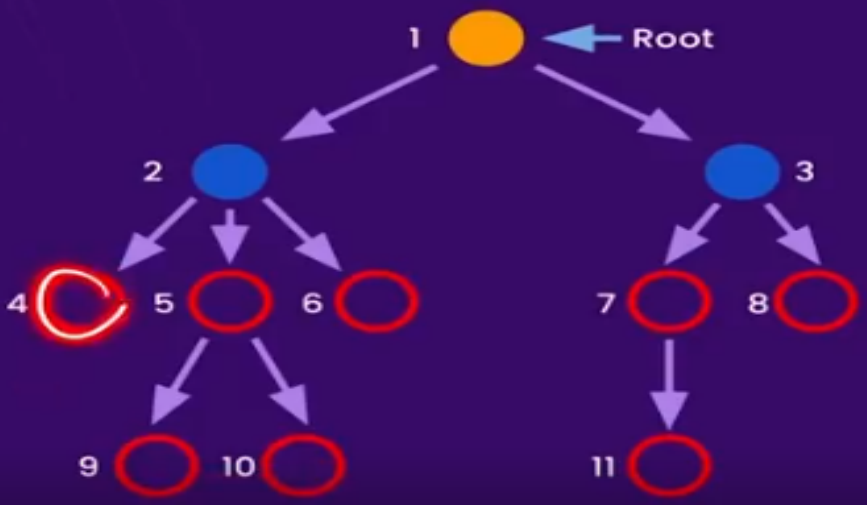
What is a Tree data structure

Array, LL, Stack, Queue → Linear data structures

Tree → hierarchical data structure



Representation

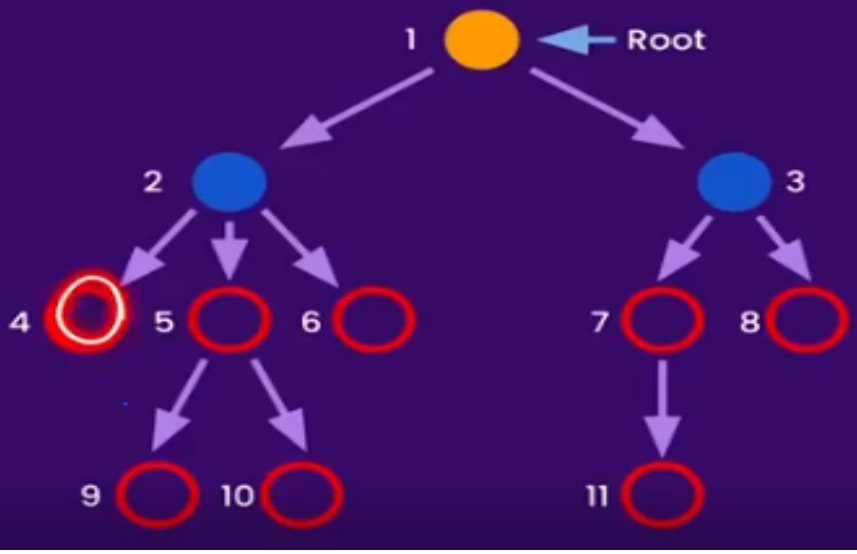


LL : head
Tree : Root

Terminology

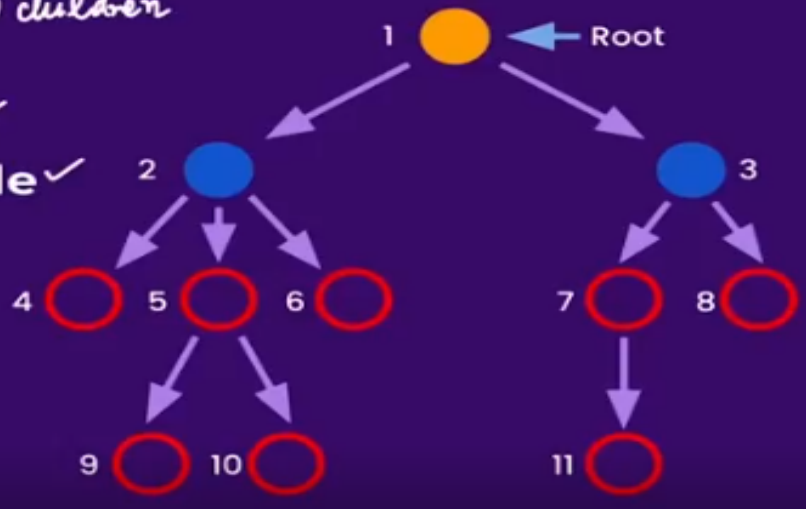
- Root ✓
- Child Node ✓
- Parent Node ✓
- Sibling Nodes ✓

↓
means nodes with a common parent



Terminology

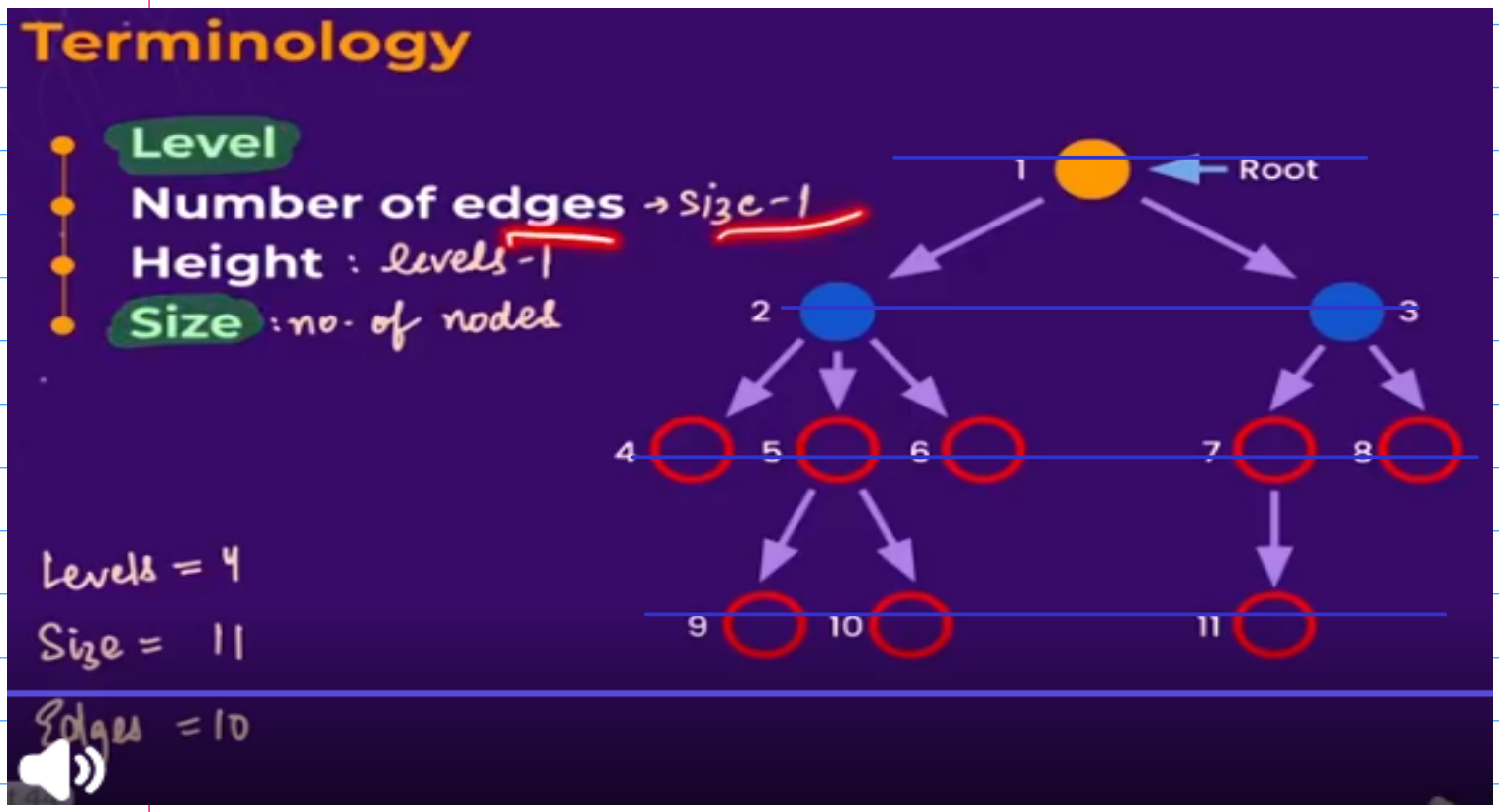
- Leaf Node → with 0 children
- Internal Node
- Ancestor Node ✓
- Descendant Node ✓



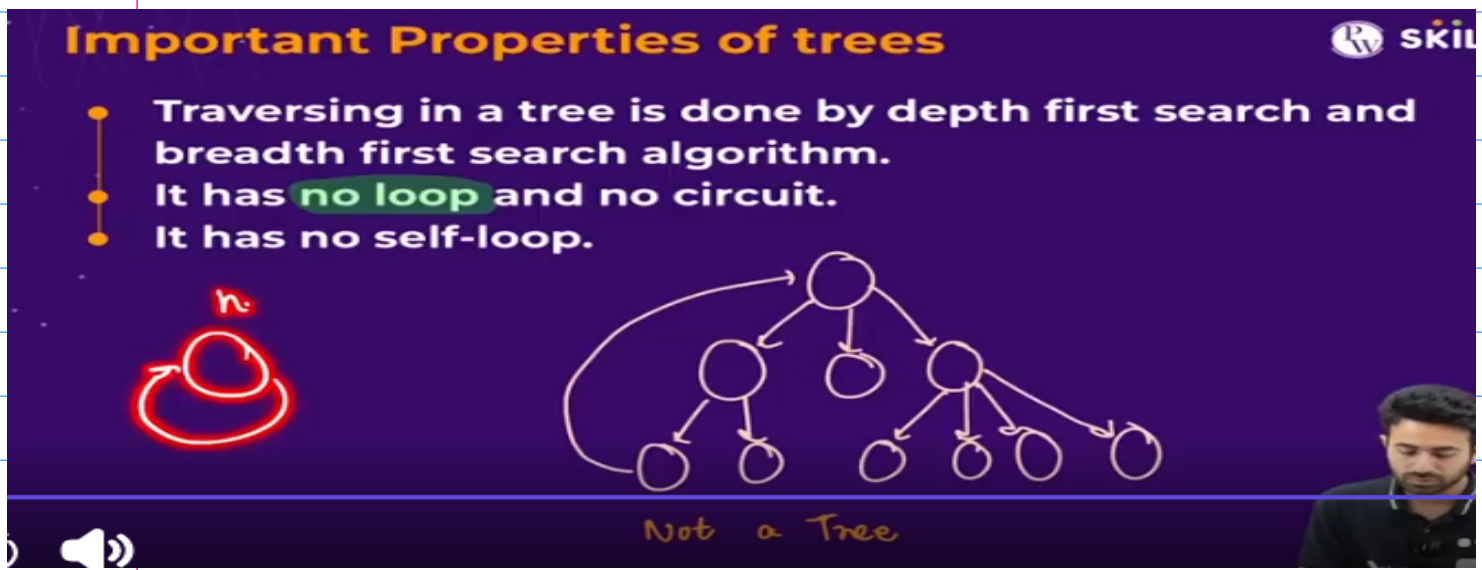
2 ke ancestor → 1, 2, 5

2 ke descendants : 4, 5, 6, 9, 10

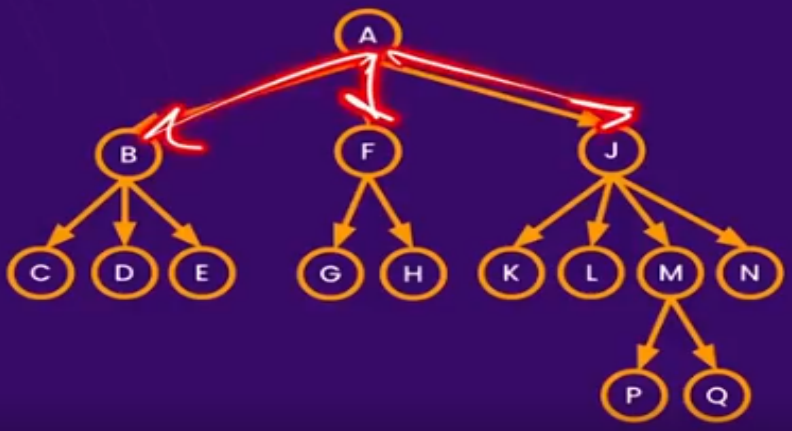
Internal node-->ye na to saroundig or na to leaf node hota hai .
Leaf node-->no child.
Root node-->ye voh node hai jiska koi parent node nhi hota hai iske keval child node hote hai.
Ancester node --> this node like purvaj.



Leval-->leval like a generation.
Number of edges=size-1.
height = levels-1.
size=number of nodes in a tree

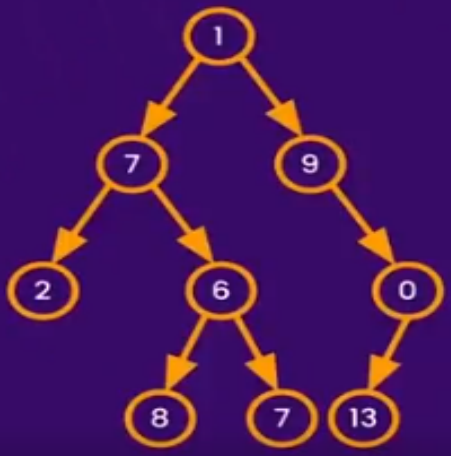


1. Generic Trees



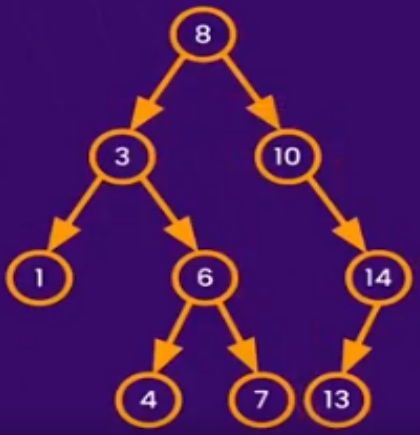
Any node in a generic Tree can have any no. of child nodes

2. Binary Trees



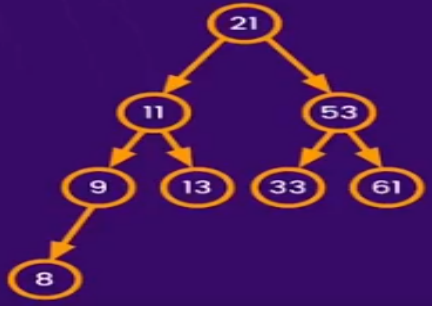
Any node can have 0, 1 or 2 children

3. Binary Search Trees



Iske baare me baad me acche se padhege

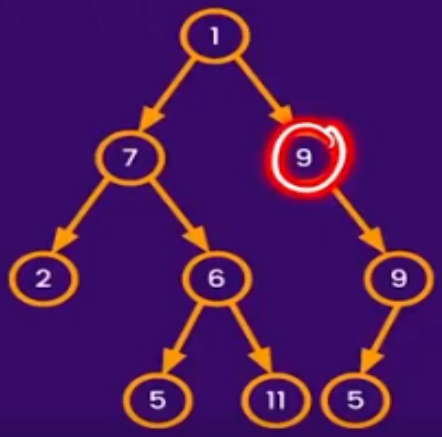
4. AVL Trees



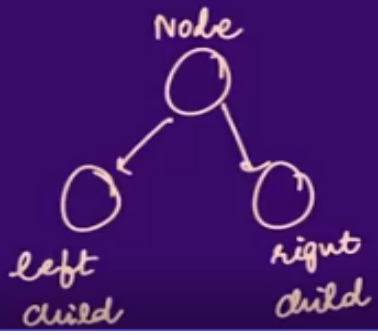
BST's which are self balancing

What are Binary Trees?

SKILLS



Trees whose each node has either 0, 1 or 2 children



Implementation

- Creating a Node class

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

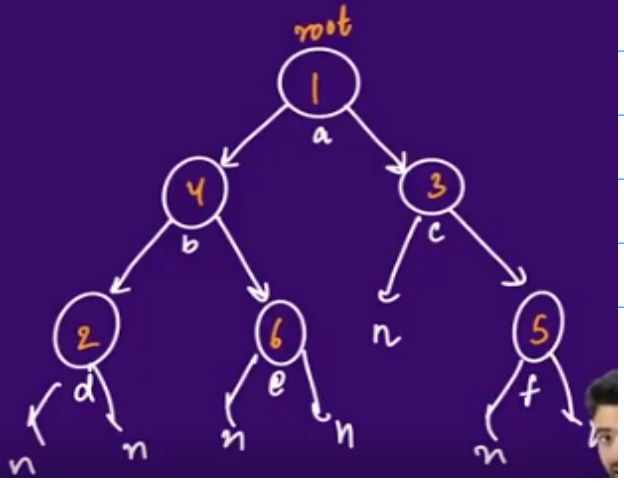
Creating a Node class

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

Implementation

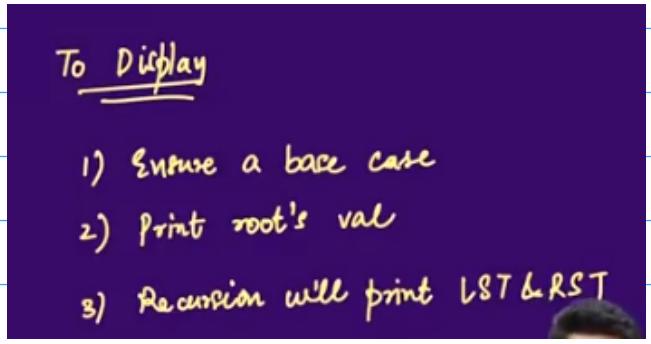
- Creating a Node class

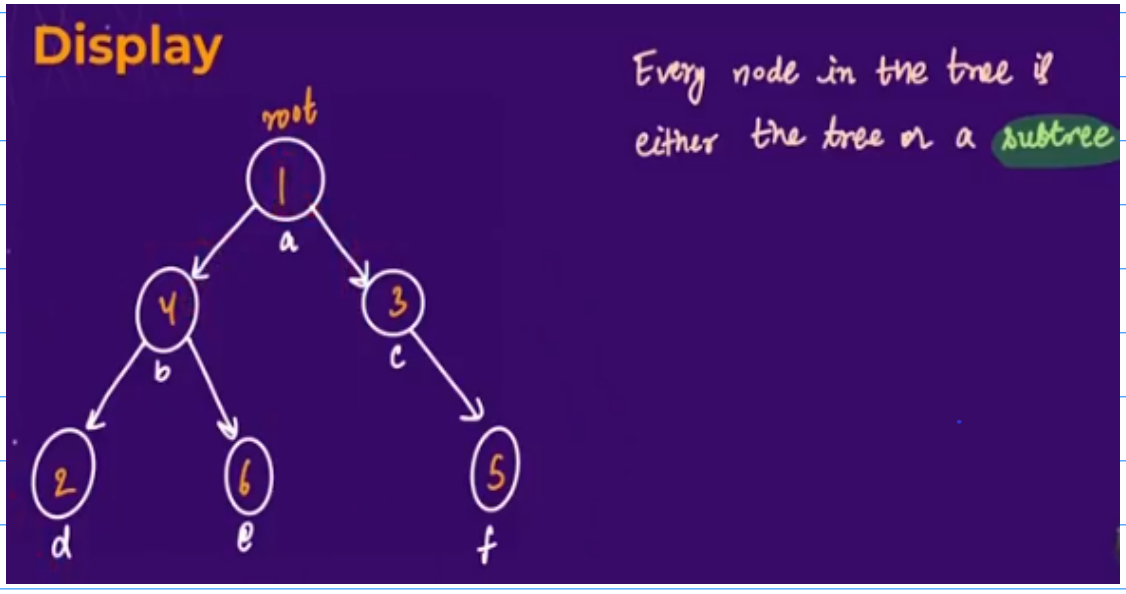
```
class Node {
    int val;
    Node left;
    Node right;
}
```



```
public static void main(String[] args) {
    Node a  = new Node(1);
    Node b = new Node(4);
    Node c = new Node(3);
    Node d = new Node(2);
    Node e = new Node(6);
    Node f = new Node(5);
    a.left = b;a.right = c;
    b.left = d;b.right = e;
    c.right =f;
    System.out.println(e.val);
    System.out.println(a.left.right.val);
    System.out.println(a.right.right.val);


}
```

subtree means part of the tree

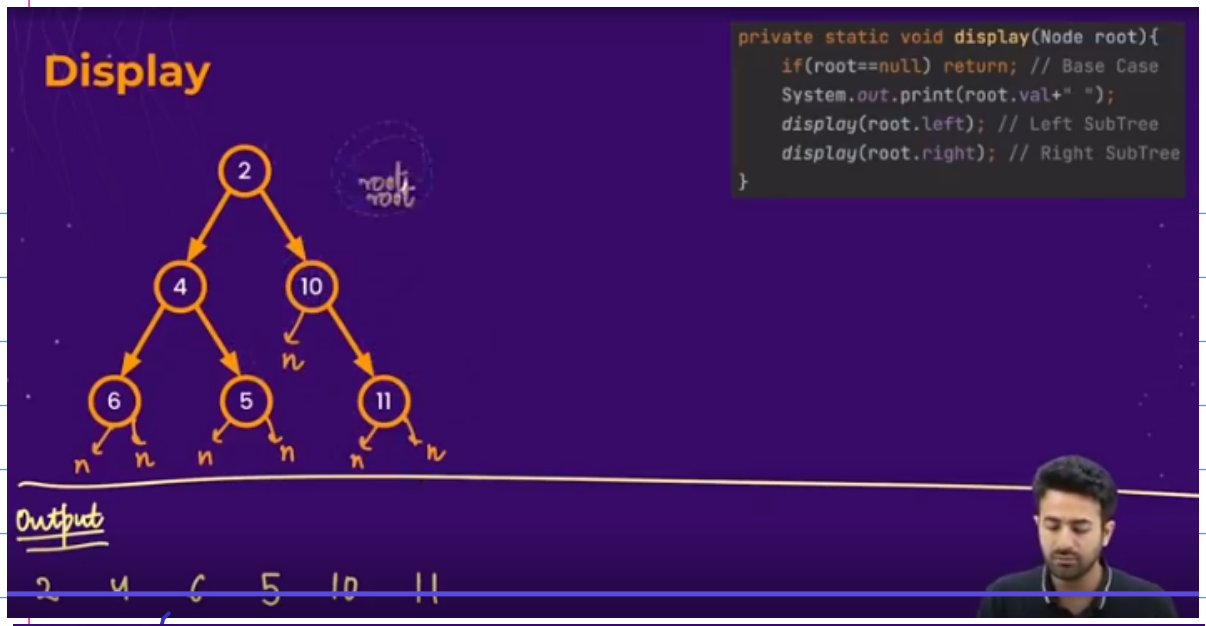


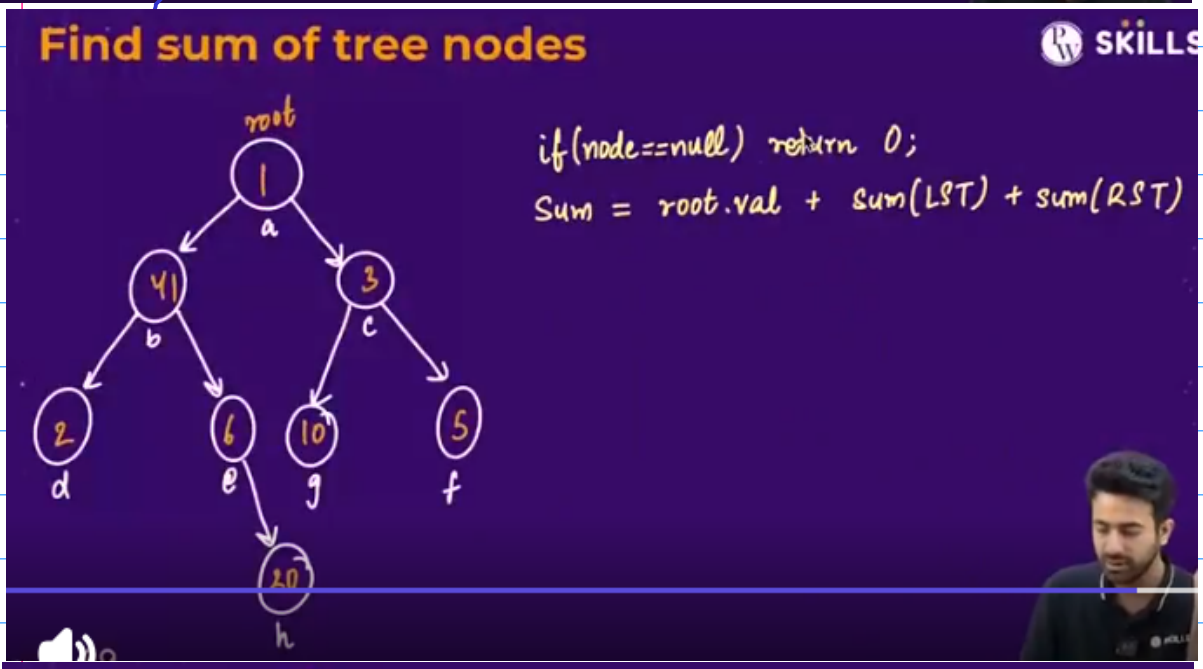


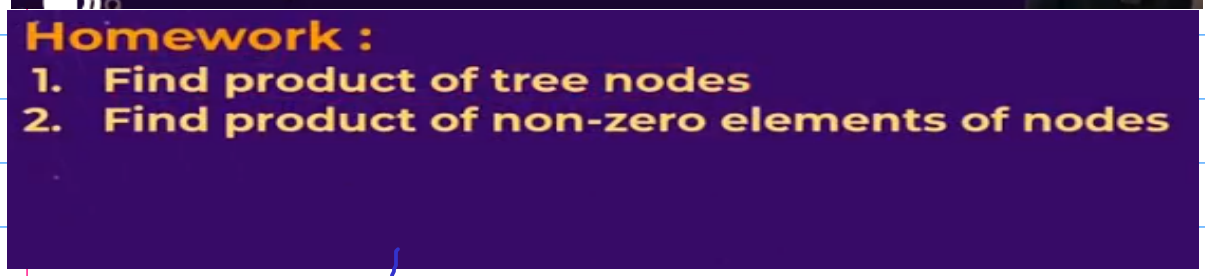

```
 static private void display(Node root){
        if(root==null) return;
        System.out.print(root.val+" ");
        display(root.left);
        display(root.right);
    }
    public static void main(String[] args) {
        Node a  = new Node(1);
        Node b = new Node(4);
        Node c = new Node(3);
        Node d = new Node(2);
        Node e = new Node(6);
        Node f = new Node(5);
        a.left = b;a.right = c;
        b.left = d;b.right = e;
        c.right =f;
        a.left.right.val = 56;
        display(a);
```

-->output

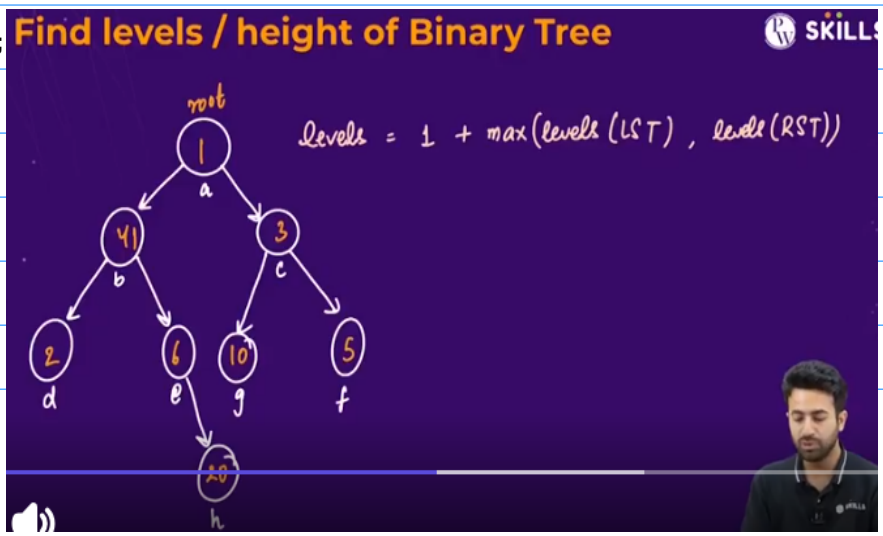
1 4 2 56 3 5
Process finished with exit code 0

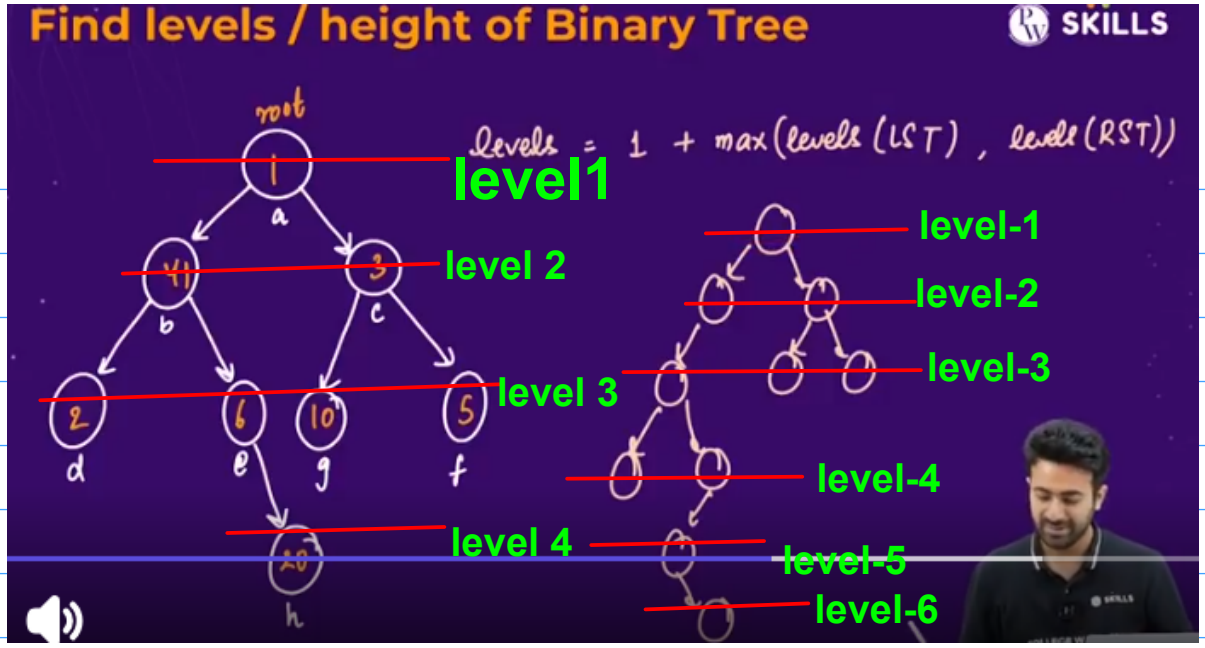







```
package Tree;

public class findProductOfTreeNode {
   static private int  product(Node root){
      if(root==null) return 1;
      return root.val*product(root.left)*product(root.right);


   }
   public static void main(String[] args) {
      Node a  = new Node(1);
      Node b = new Node(4);
      Node c = new Node(3);
      Node d = new Node(2);
      Node e = new Node(6);
      Node f = new Node(5);
      a.left = b;a.right = c;
      b.left = d;b.right = e;
      c.right =f;

      System.out.println( product(a));

   }
}
```





tree ki jo height hoti hai vo level se ek kam hota hai
ydi tree me height pucha hai to hm level nikalenge usme se 1 minus kar denge to tree ki height ajayegi

```java
private static int sum(Node root){
    if(root==null) return 0;
    return root.val + sum(root.left) + sum(root.right);
}

private static int size(Node root){
    if(root==null) return 0;
    return 1 + size(root.left) + size(root.right);
}

private static int levels(Node root){
    if(root==null) return 0;
    return 1 + Math.max(levels(root.left),levels(root.right));
}

private static int max(Node root){
    if(root==null) return Integer.MIN_VALUE;
    int a = root.val, b = max(root.left), c = max(root.right);
    return Math.max(a,Math.max(b,c));
}
```

treversal ka mtlb hai ki sabhi element per trevel karna hi traversal kahlata hai

```
    }
    private static void preorder(Node root){
        if(root==null) return;
        System.out.print(root.val+" ");
        preorder(root.left);
        preorder(root.right);
    }
    private static void inorder(Node root){
        if(root==null) return;
        inorder(root.left);
        System.out.print(root.val+" ");
        inorder(root.right);
    }
    private static void postorder(Node root){
        if(root==null) return;
        postorder(root.left);
        postorder(root.right);
        System.out.print(root.val+" ");
    }
```

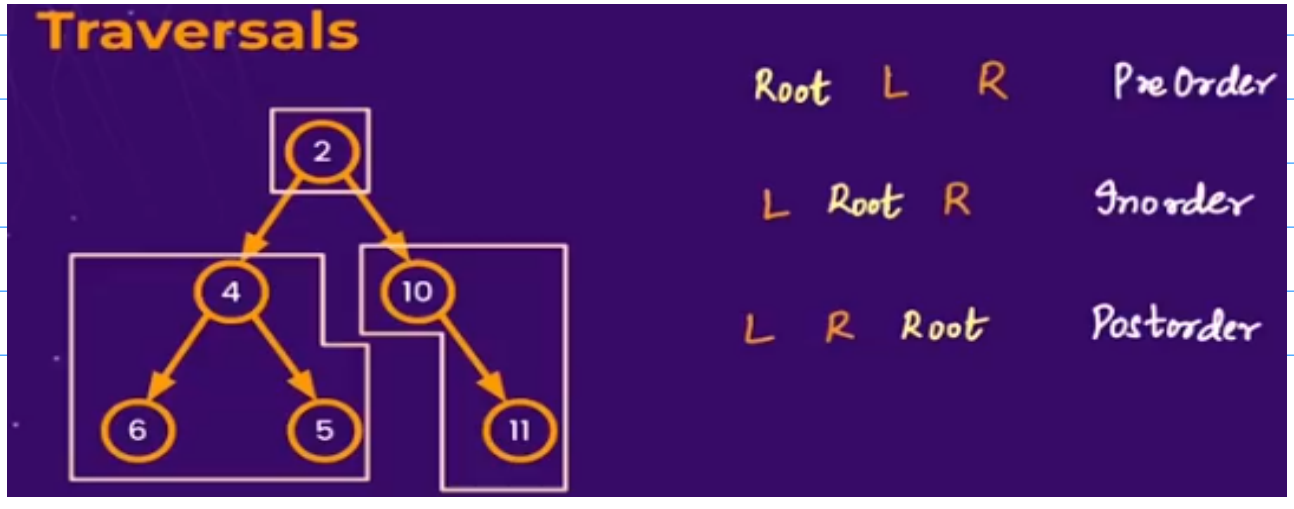
pre,post and in order traversal

```
public class treetraversal {
    public static void display(Node root){
        if(root==null) return;
        System.out.print(root.val+" ");
        display(root.left);
        display(root.right);
    }
    public static void pre(Node root){
        if(root==null) return;
        System.out.print(root.val+" ");
        display(root.left);
        display(root.right);
    }
    public static void in(Node root){
        if(root==null) return;
        display(root.left);
        System.out.print(root.val+" ");
        display(root.right);
    }
    public static void post(Node root){
        if(root==null) return;
        display(root.left);
        display(root.right);
        System.out.print(root.val+" ");
    }
```

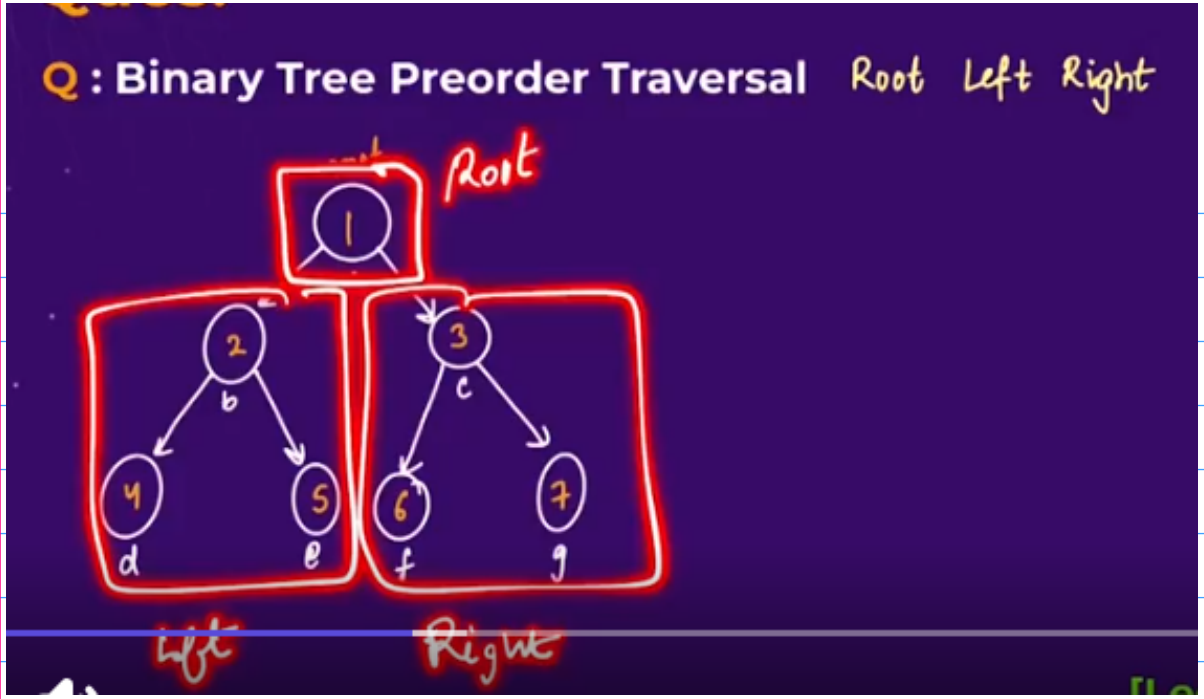
output

```
public static void main(String[] args) {
        Node a  = new Node(1);
        Node b = new Node(2);
        Node c = new Node(3);
        Node d = new Node(4);
        Node e = new Node(5);
        Node f = new Node(6);
        Node g = new Node(7);
        a.left = b;a.right = c;
        b.left = d;b.right = e;
        c.left = f;c.right =g;
    display(a);
        System.out.println();
        pre(a);
        System.out.println();
        in(a);
        System.out.println();
        post(a);
        System.out.println();

    }
}
```

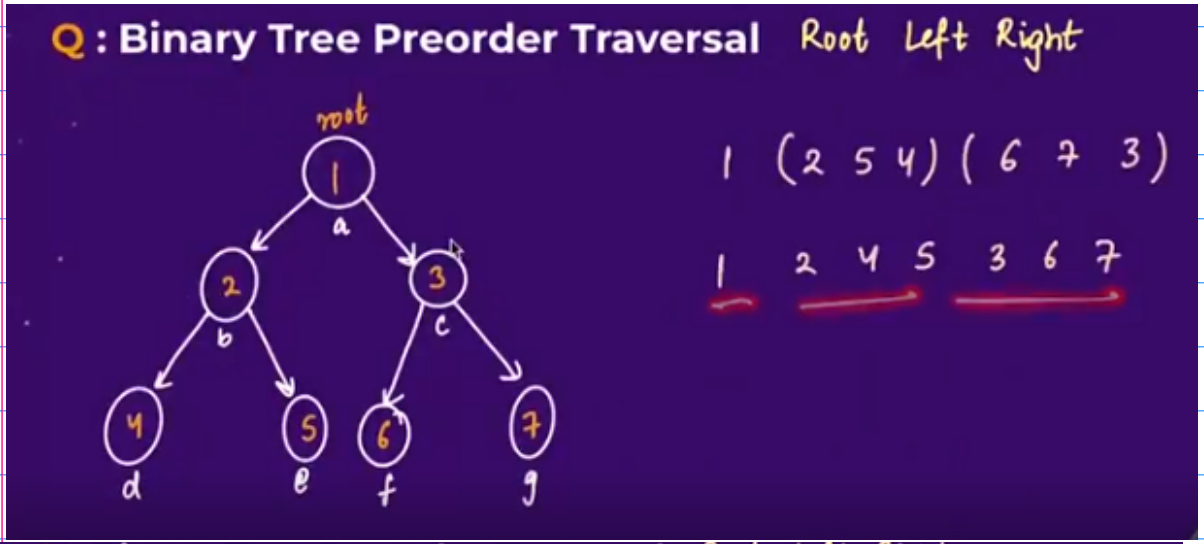
1 2 4 5 3 6 7 //display
1 2 4 5 3 6 7 //pre order
2 4 5 1 3 6 7 //in order
2 4 5 3 6 7 1 //post order



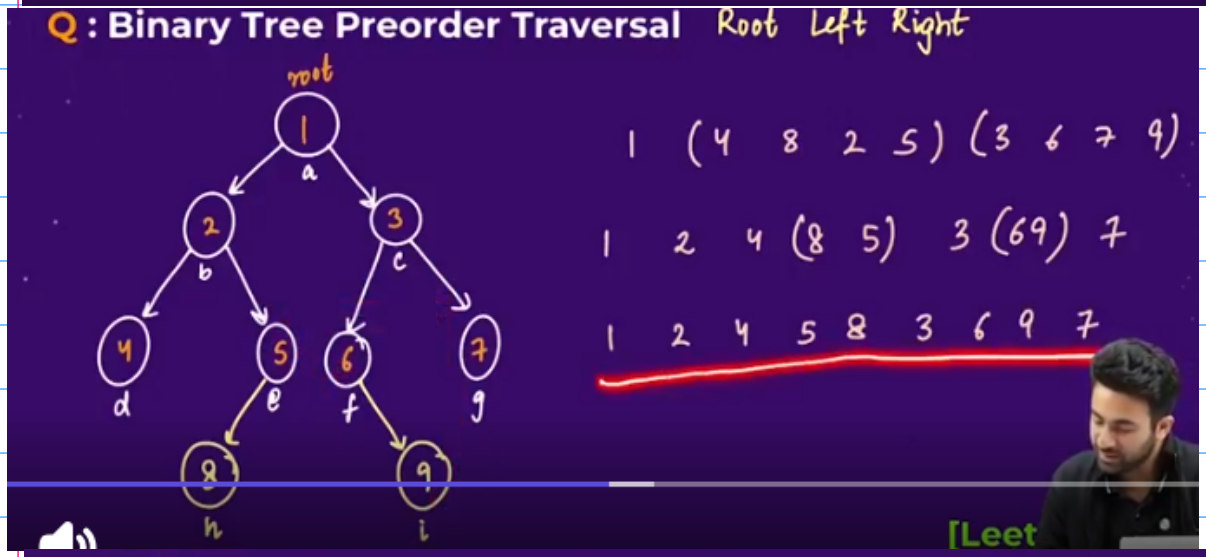
Q : Binary Tree Preorder Traversal Root Left Right



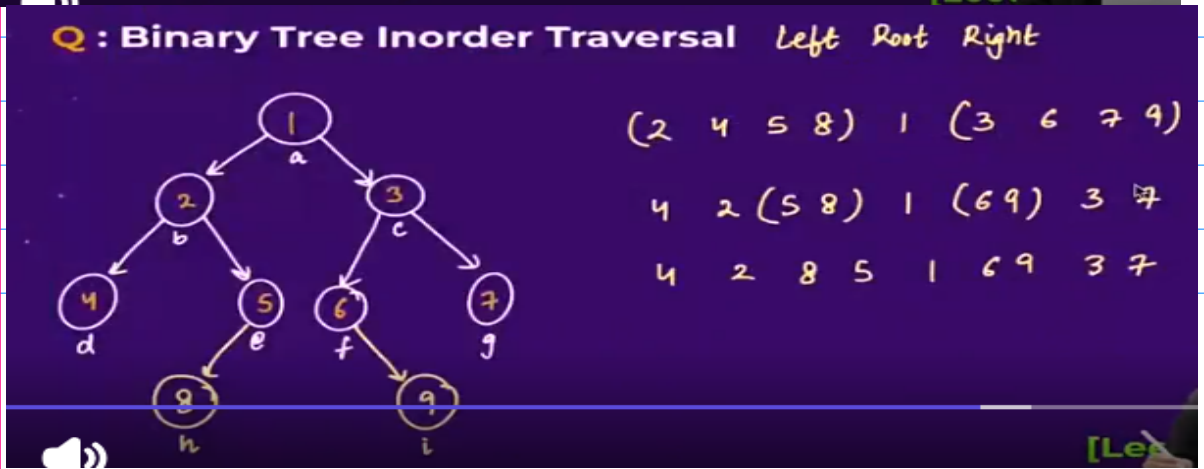
Q : Binary Tree Preorder Traversal Root Left Right

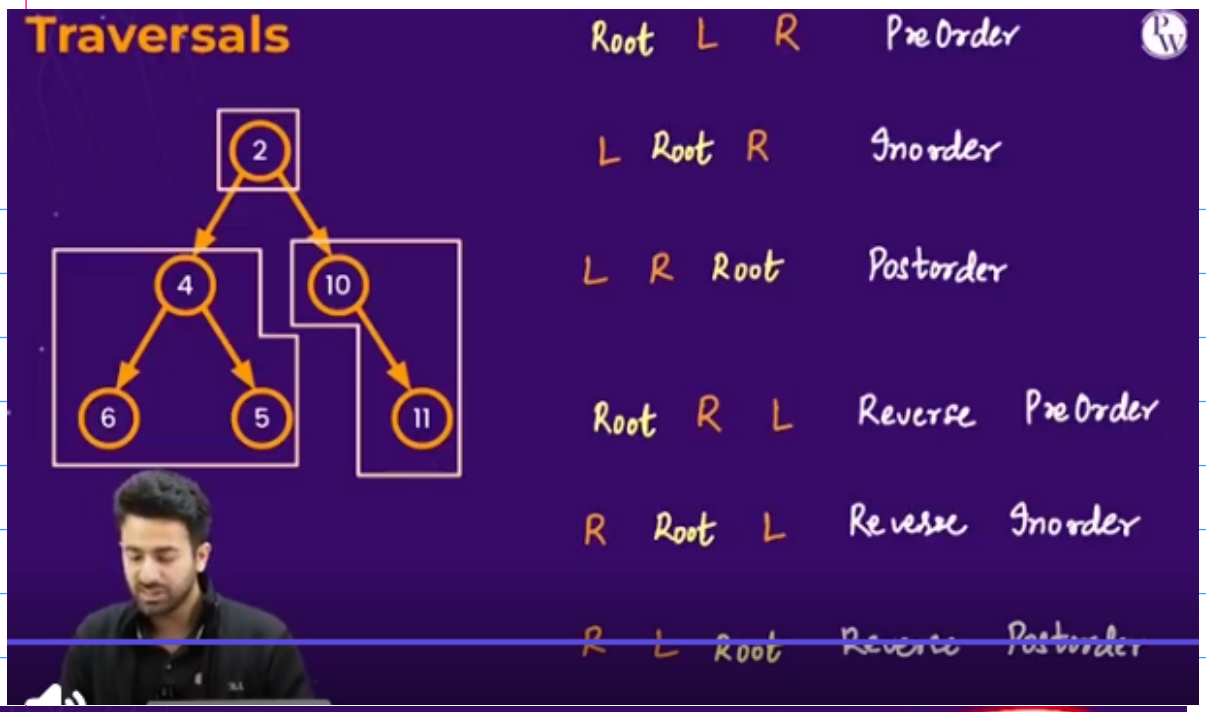


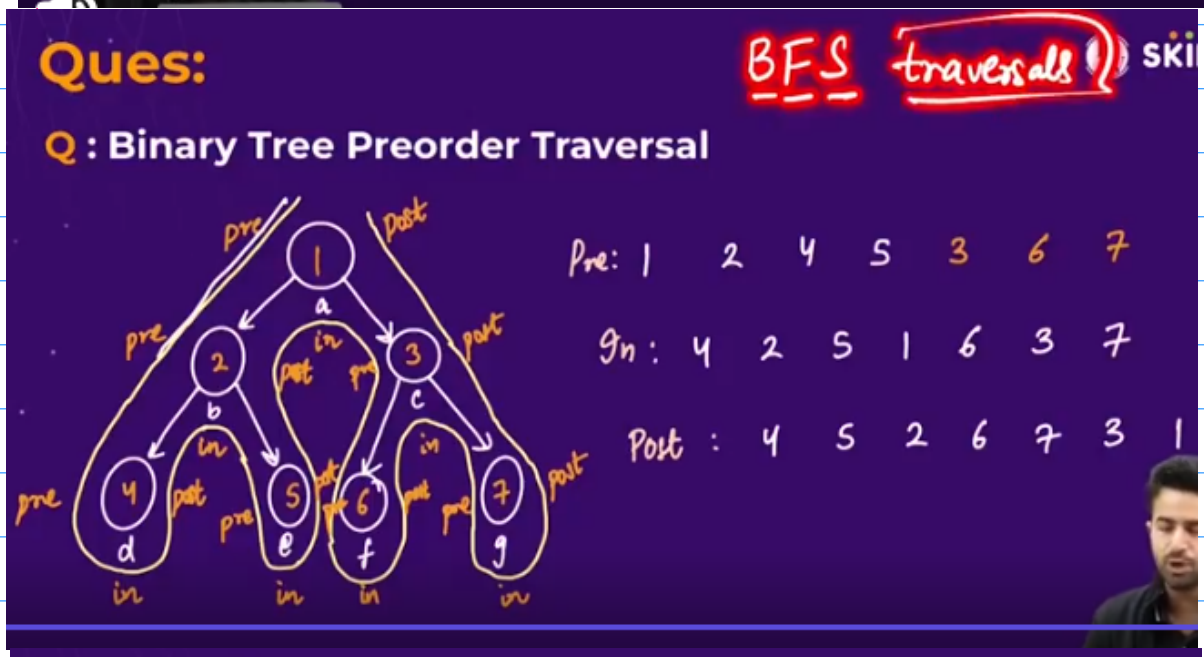
Q : Binary Tree Preorder Traversal Root Left Right

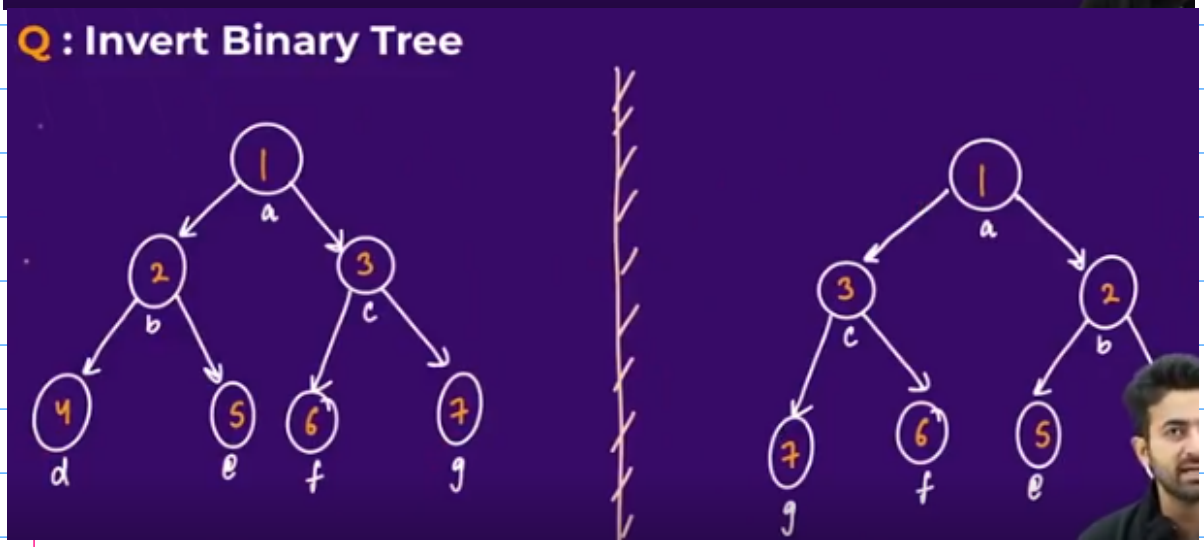


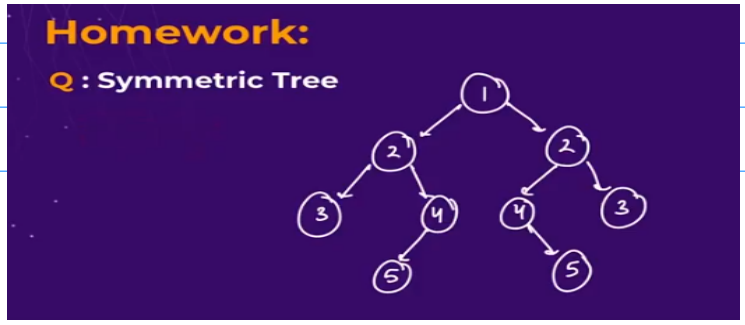
Q : Binary Tree Inorder Traversal Left Root Right



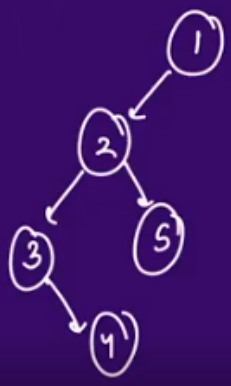

Traversals
2
4
10
6
5
11
Root L R Pre Order
L Root R Inorder
L R Root Postorder
Root R L Reverse Pre Order
R Root L Reverse Inorder
R L Root Reverse Postorder


Ques:
BFS traversals
Q : Binary Tree Preorder Traversal
pre
post
in
Pre: 1 2 4 5 3 6 7
In: 4 2 5 1 6 3 7
Post: 4 5 2 6 7 3 1


Q : Invert Binary Tree


Homework:
Q : Symmetric Tree

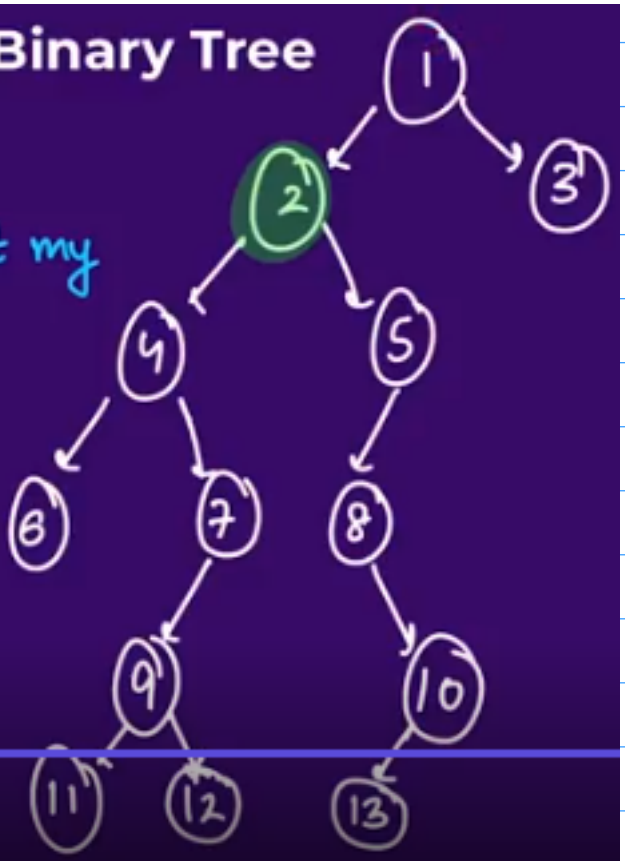
Q : Diameter of Binary Tree



diameter = [levels(LST) + levels(RST) + 1] - 1

Q : Diameter of Binary Tree

I was trying to get my diameter which is passing through the root.

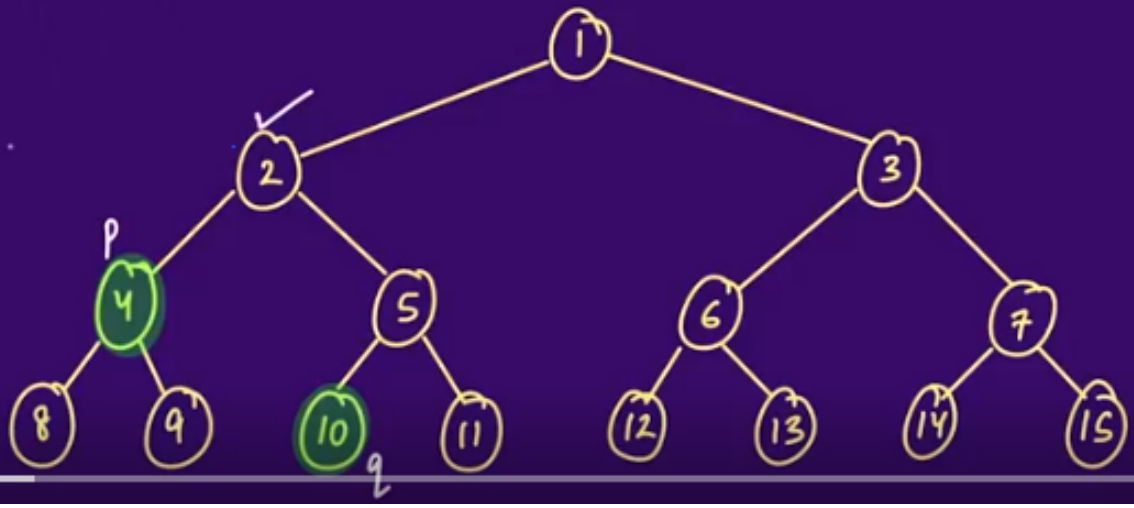


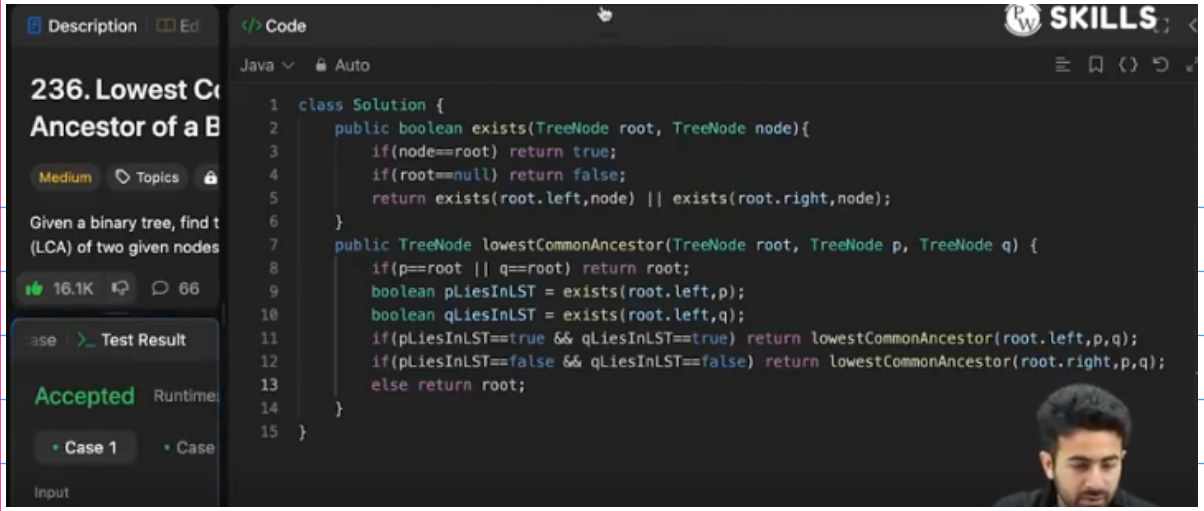
Very very very famous

Ques:

LCA is of 2 (or more) given nodes

Q : Lowest Common Ancestor of a Binary Tree





236. Lowest Common Ancestor of a Binary Tree

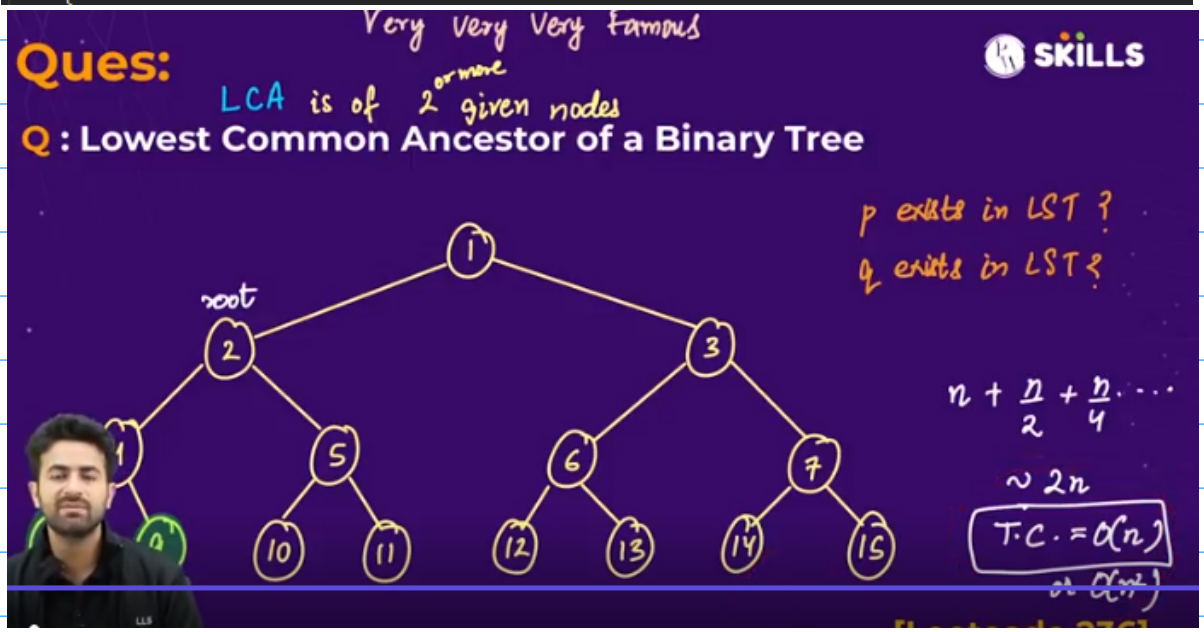
Solved

Medium | Topics | Companies

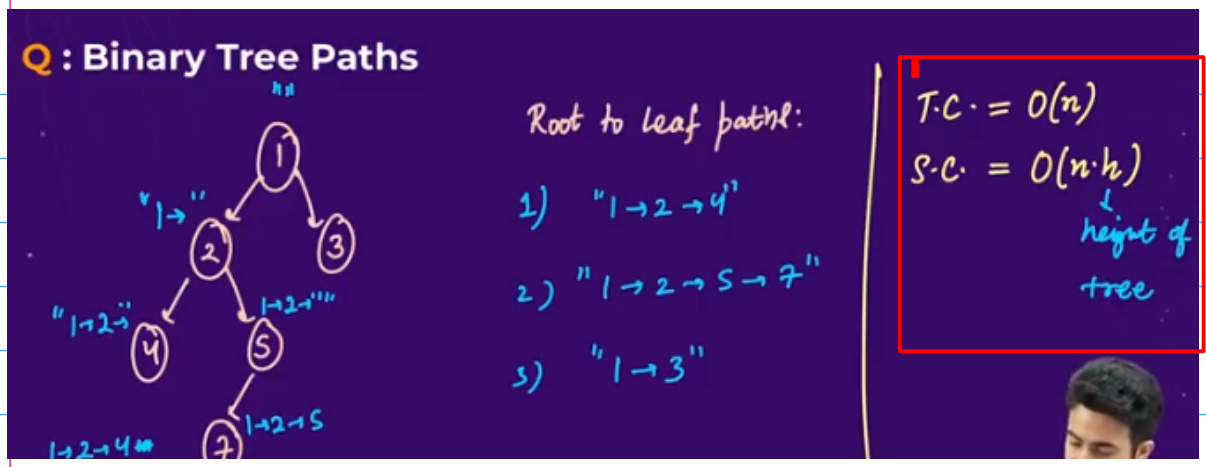
Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the definition of LCA on Wikipedia: "The lowest common ancestor is defined between two nodes `p` and `q` as the lowest node in `T` that has both `p` and `q` as descendants (where we allow **a node to be a descendant of itself**)."

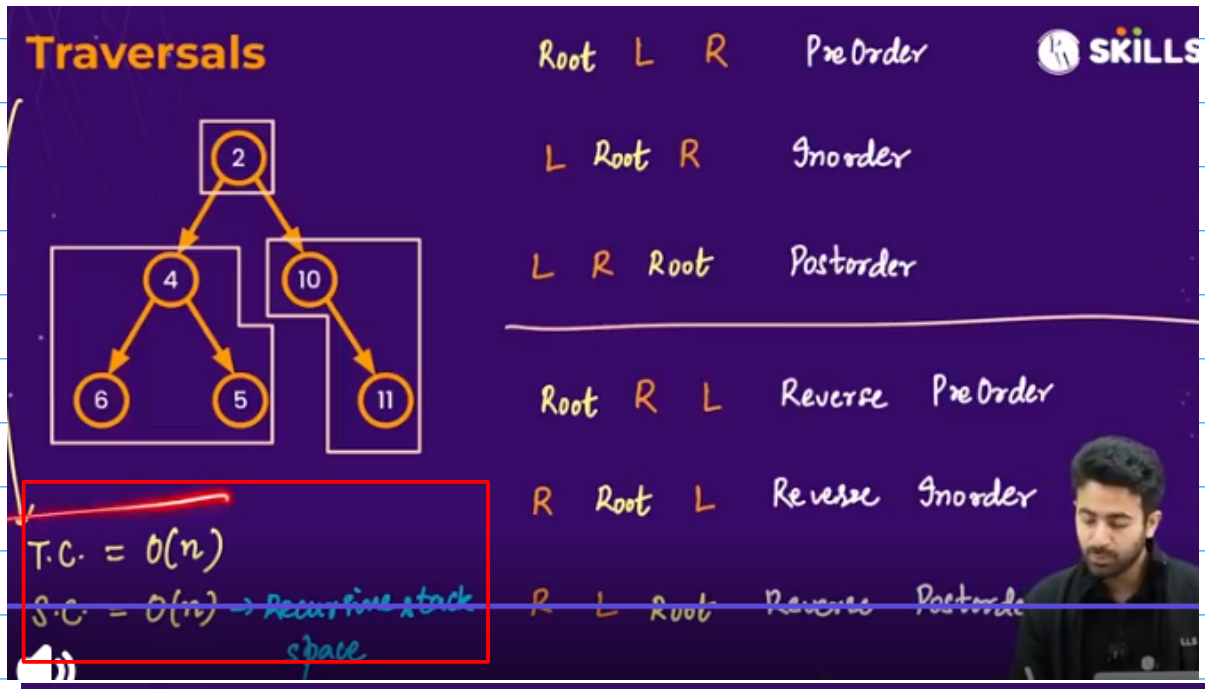
```java
class Solution {
    public boolean exist(TreeNode root,TreeNode node){
        if(node==root) return true;
        if(root==null) return false;
        return exist(root.left,node)||exist(root.right,node);
    }
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        if(p==root||q==root) return root;
        boolean pLieLST =exist(root.left,p);
        boolean qLieLST =exist(root.left,q);
        if(pLieLST==true&&qLieLST==true) return lowestCommonAncestor(root.left,p,q);
        if(pLieLST==false&&qLieLST==false) return lowestCommonAncestor(root.right,p,q);
        else return root;
    }
```

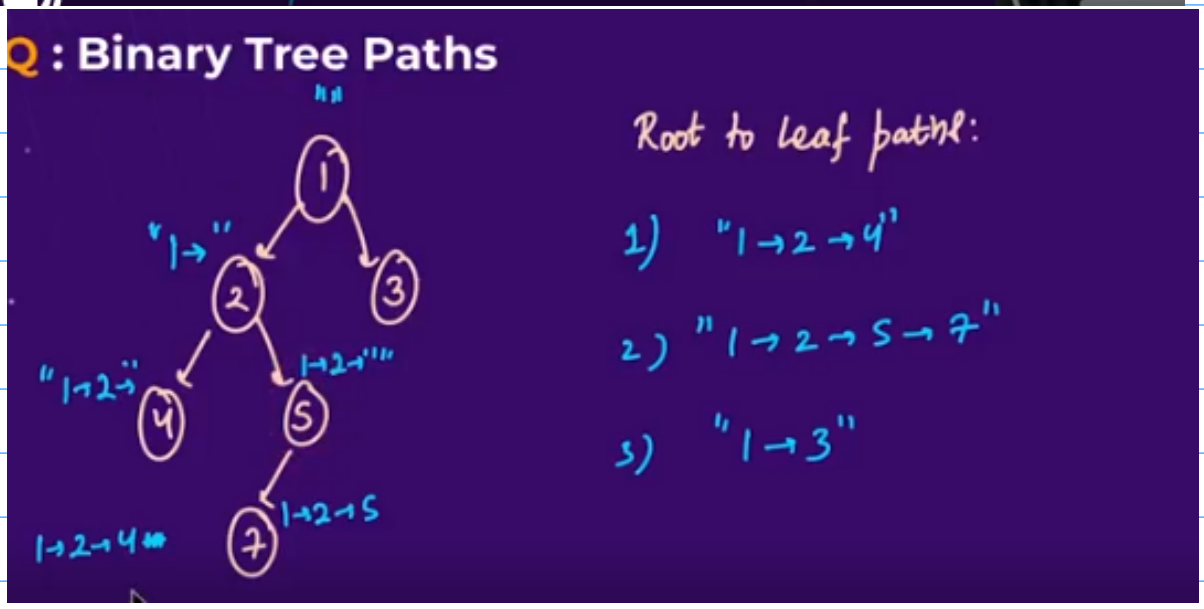


Time and space complexity



in all three case the time and space complexity





257. Binary Tree Paths
Solved
Easy
Topics
premium lock icon
Companies
Given the root of a binary tree, return all root-to-leaf paths in any order.

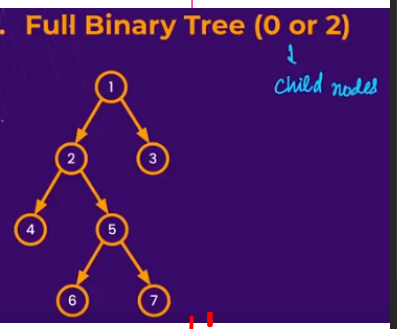
A leaf is a node with no children.

Example 1:

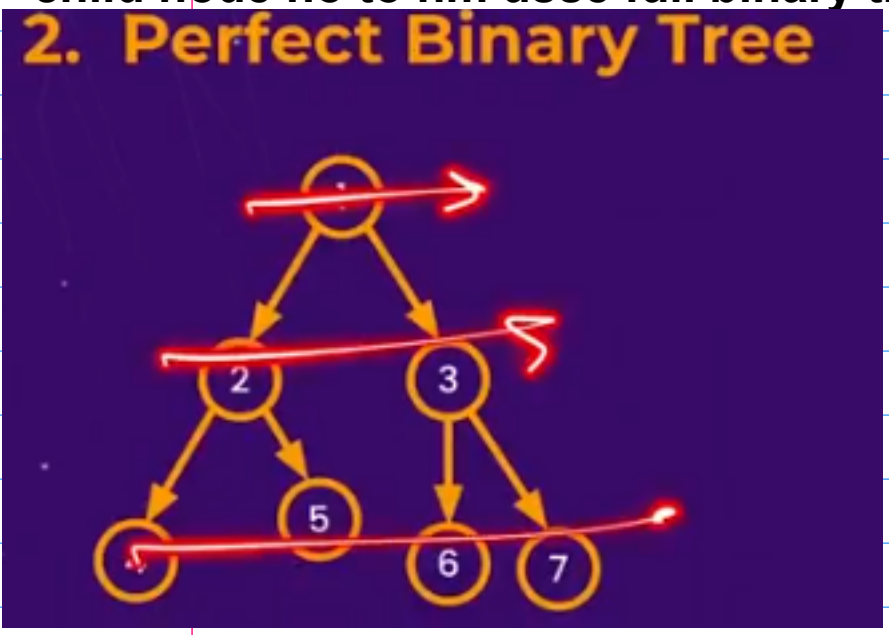
Input: root = [1,2,3,null,5]
Output: ["1->2->5","1->3"]
Example 2:

Input: root = [1]
Output: ["1"]

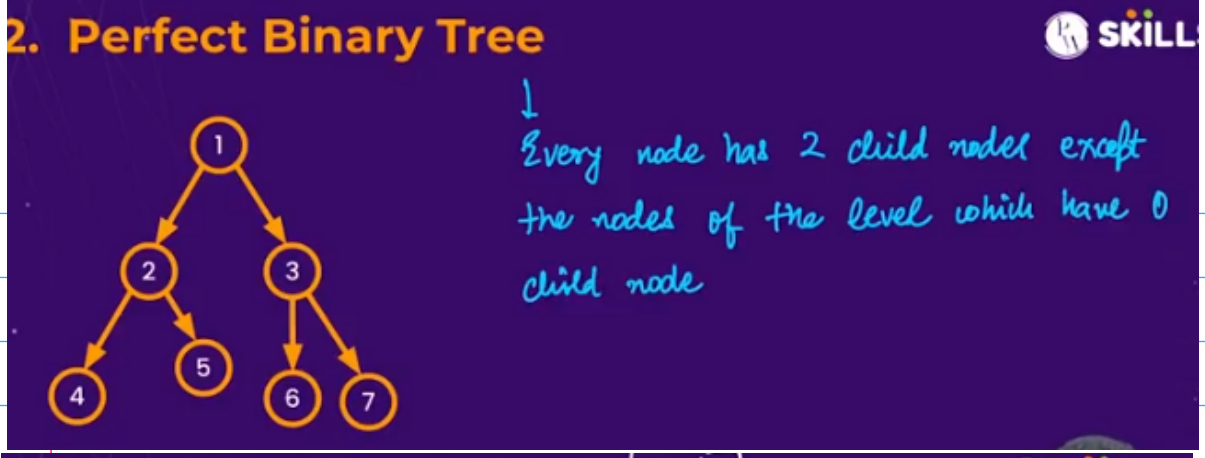
```java
class Solution {
    public void path(TreeNode root,String s,List<String> ans){
        if(root==null) return;
        if(root.left==null&&root.right==null){
            s+=root.val;
            ans.add(s);
            return;
        }
        path(root.left,s+root.val+"->",ans);
        path(root.right,s+root.val+"->",ans);
    }
    public List<String> binaryTreePaths(TreeNode root) {
      List<String> ans = new ArrayList<>();
      path(root,"",ans);
      return ans;
    }
}
```

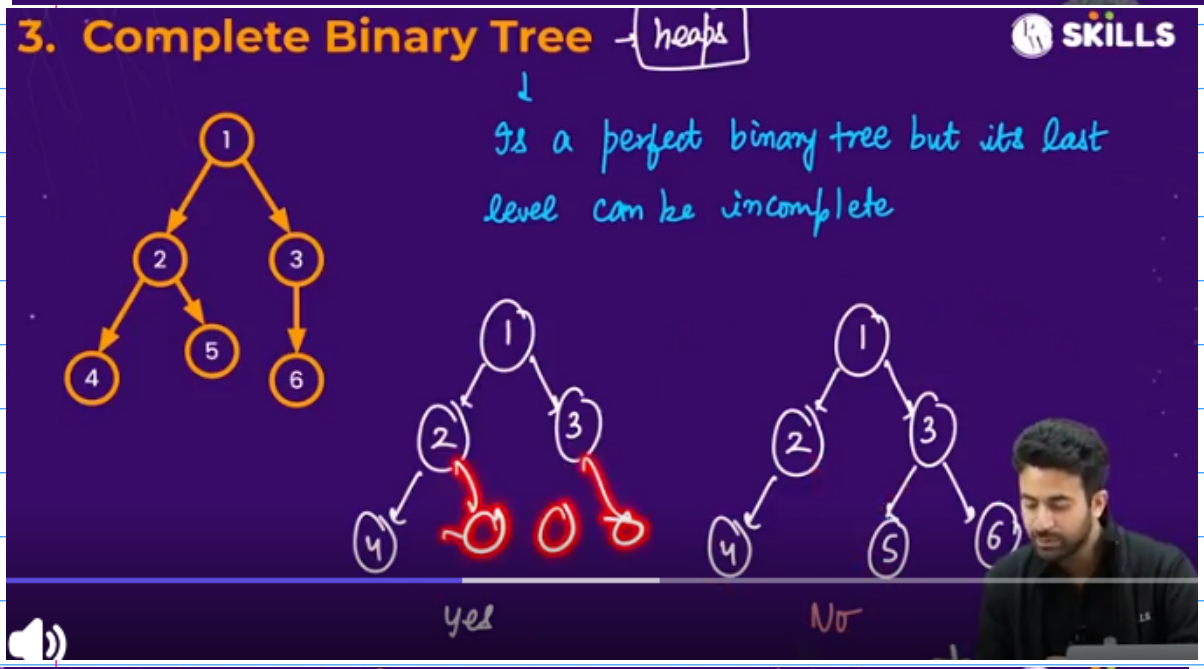


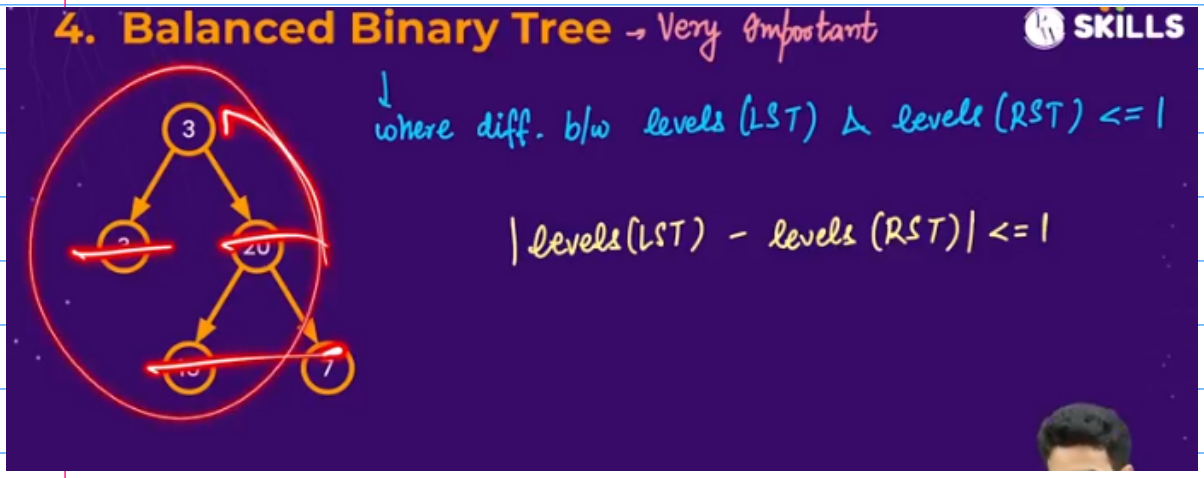
FULL BINARY TREE-->full binary tree vah tree hai jiske 0 ya to 2 child node ho to hm usse full binary tree kahte hai.



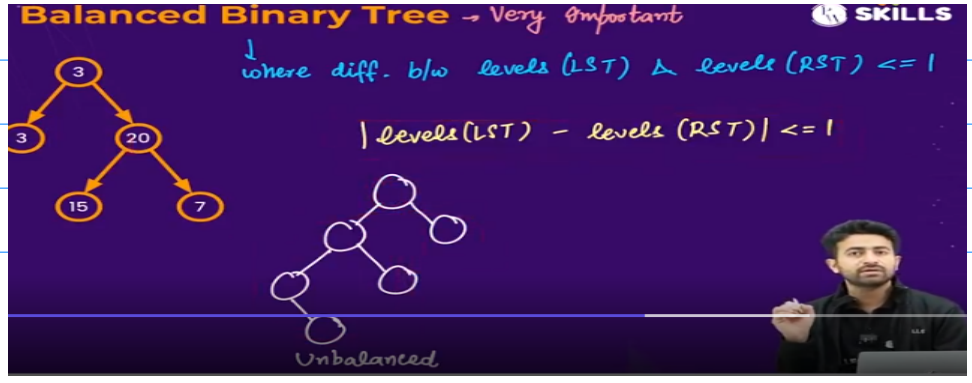
PERFECT BINARY TREE--> perfect binary tree vah tree hota hai jiske every node ke two child hote hai.but leaf node ke zero child hone chahiye.

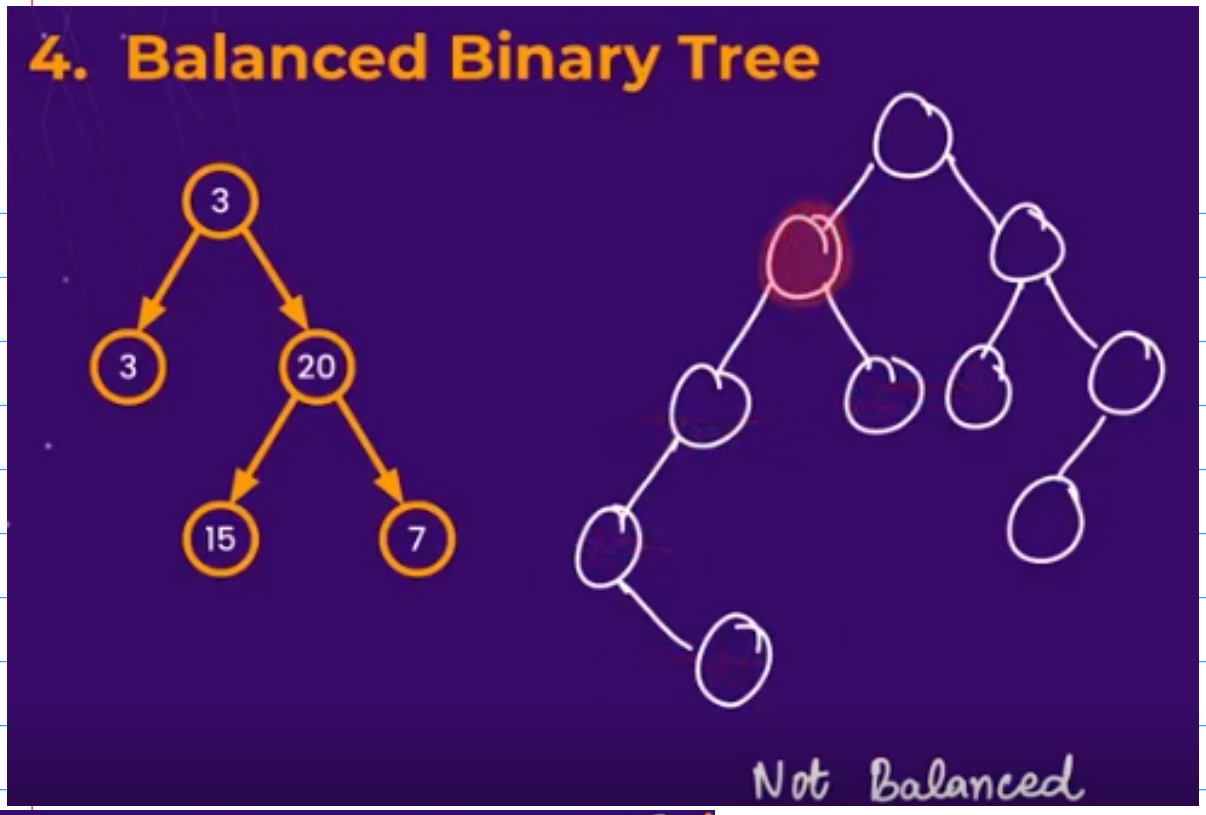


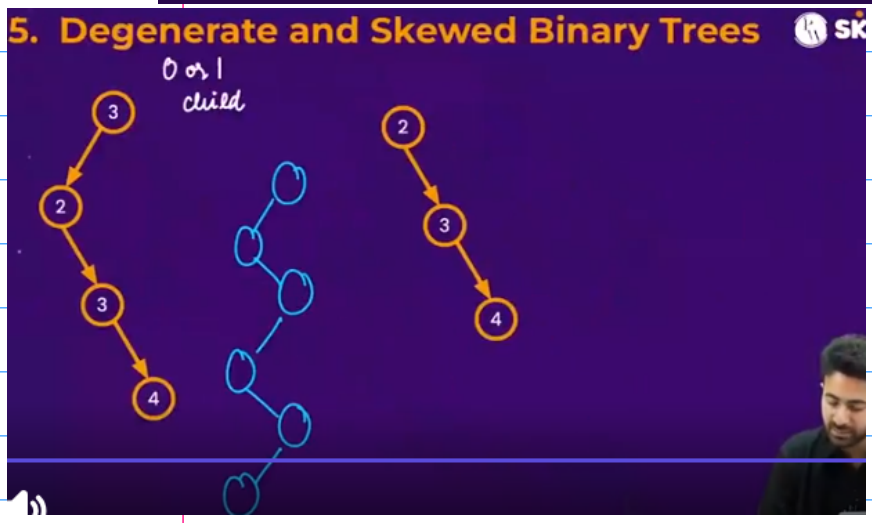




In balanced Binary tree the levels of left subtree and levels of right subtree ka diffrence hai <=1 hona chahiye tabhi vah balanced binary tree hoga. yah har ek root ke liye satisfy hona chahiye .

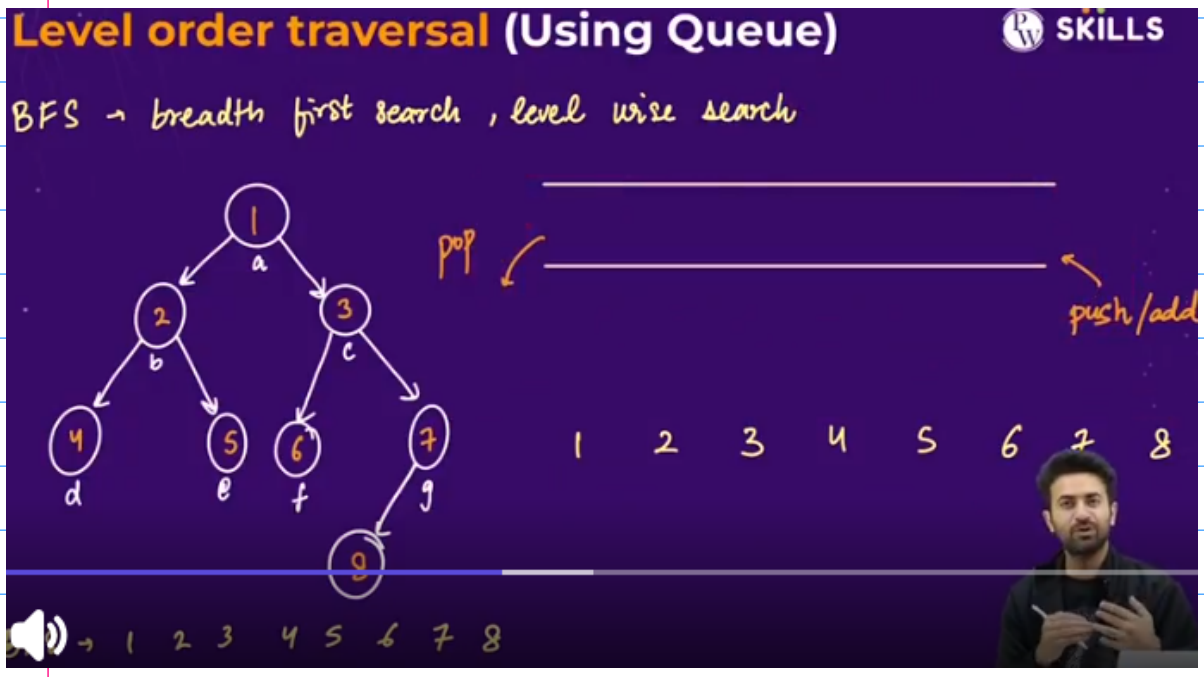


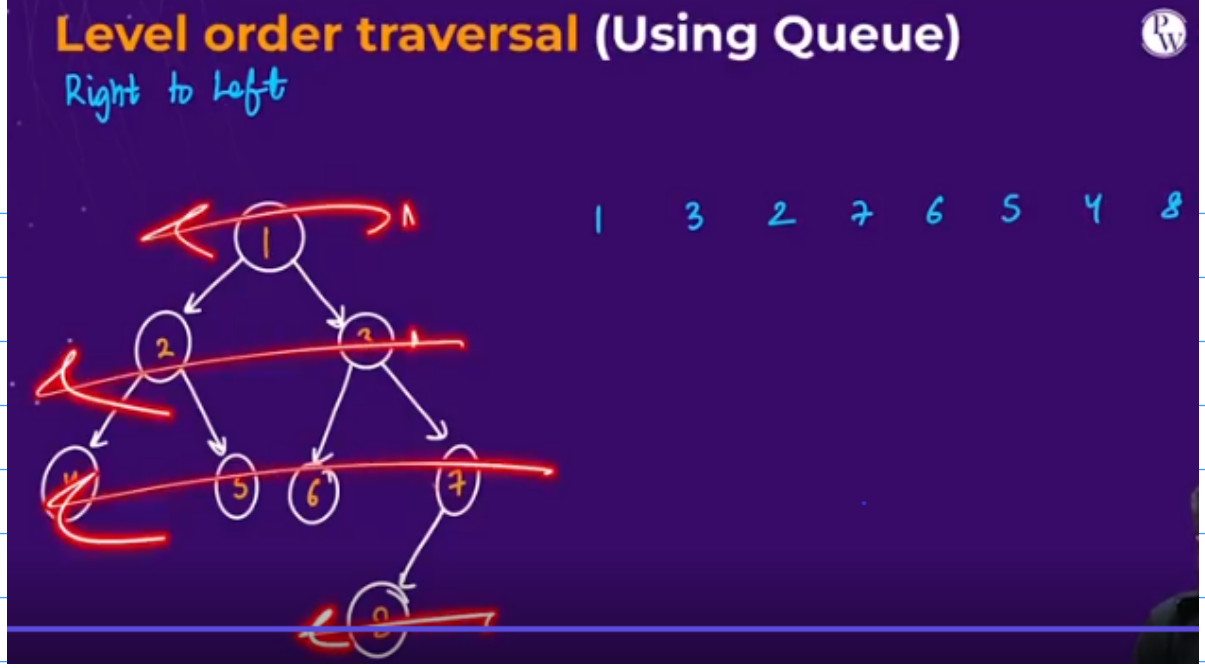




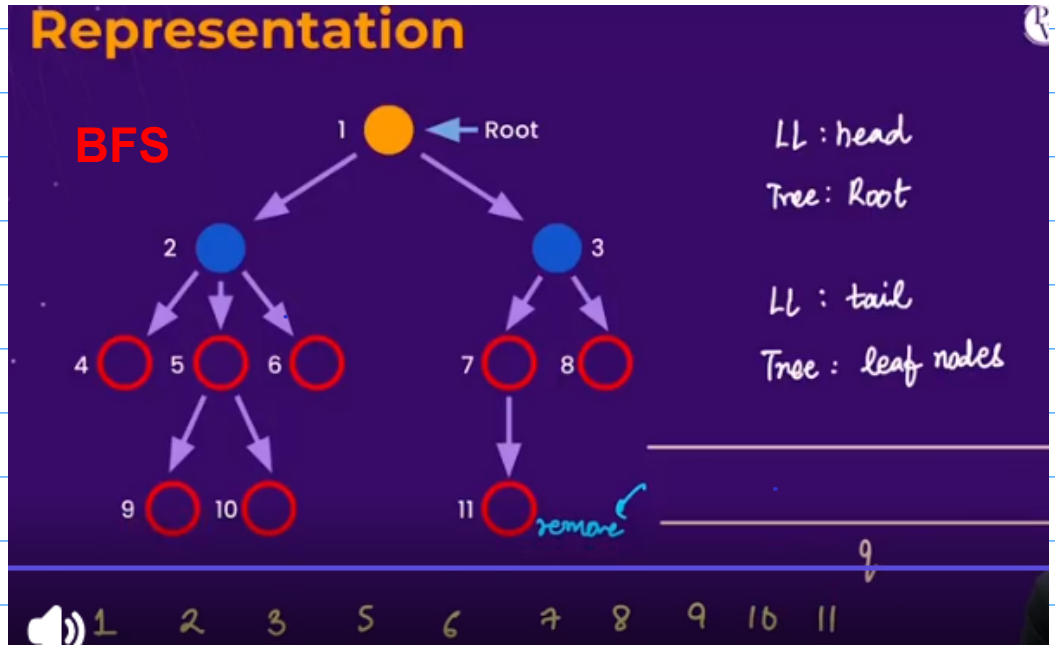
Degenerate binary tree-->
-->in this binary tree jiske 0 ya to 1 child hote hai for every node that is called Degenerate binary tree.

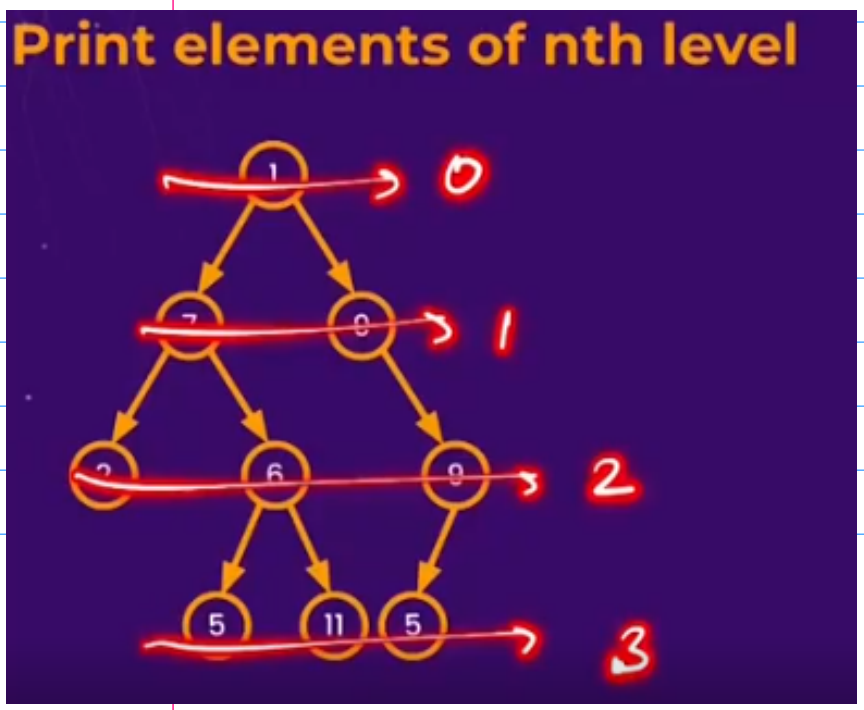
Skewed Binary trees-->
ye aisa binary tree hai jisme ye to ya left me increase hoga ya to keval right me hi progress kargega like a line

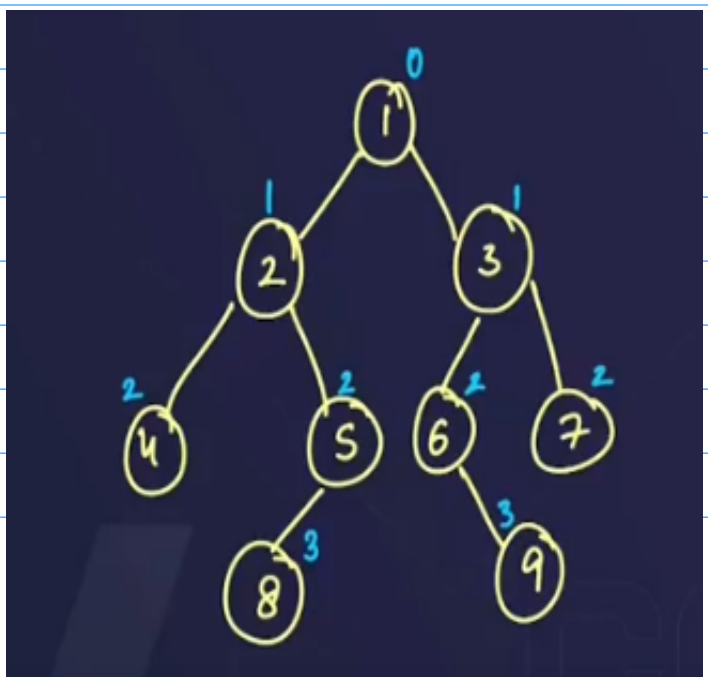




BFS--> means breadth first search jo hm queue se banate hai.







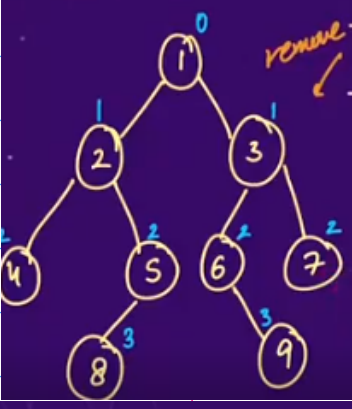
Ques: `Queue<Pair> q = new LinkedList<>();`

node, level

SKILLS

Q : Binary Tree Level Order Traversal (Queues)

leetcode 102

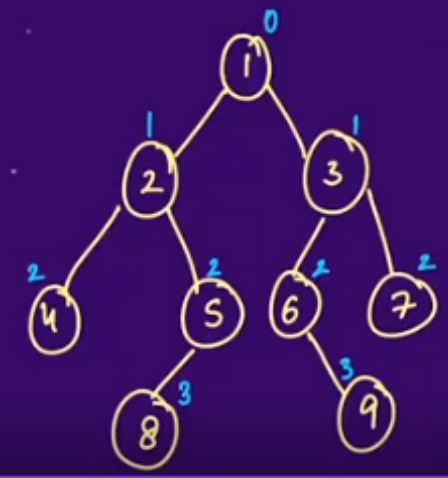


{1,0} (2,1) (3,1) (4,2) (5,2) (6,2) (7,2)
(8,3) (9,3)

1 2 3 4 5 6 7 8 9

Ques: T.C. = O(n) S.C. = O(n)

Q : Binary Tree Level Order Traversal



`List<List<Integer>> ans`

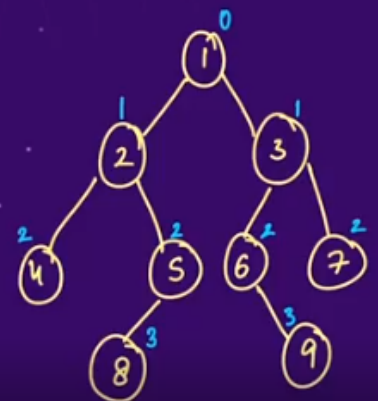
0 1 2 3

{ {3}, {3}, {3}, {3} }

For Ex:
you have to add 4

`ans.get(lvl).add(4)`

Q : Binary Tree Level Order Traversal (zigzag)



{ {1} {3 2} {4 5 6 7} {9 8} }

leetcode 103

Do it

1) BFS → queue<Pair>
2) DFS → preorder (nth level

1. Preorder Traversal (Iterative)

SKILLS

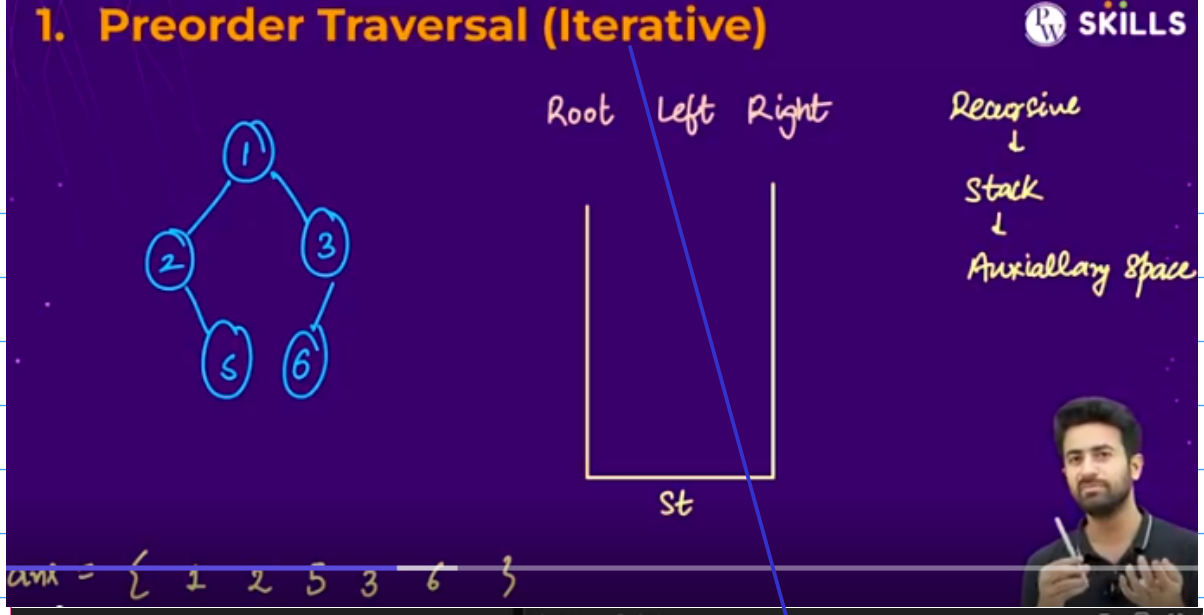
Root Left Right

St<Treenode>

Recursive
↓
Stack
↓
Auxiallary Space

ans = { 1 2 4 5 3 6 7 }

→ this is iterative method



144. Binary Tree Preorder Traversal

Solved

Easy | Topics | Companies

7.8K | 63

Testcase | Test Result

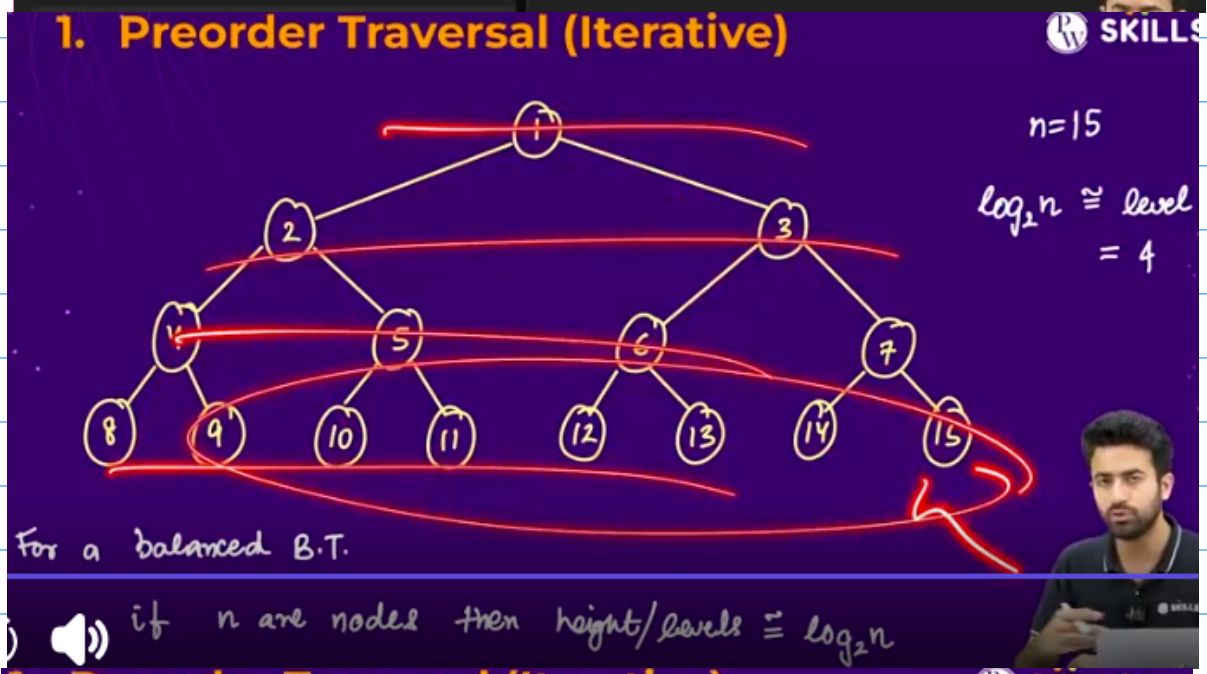
Accepted Runtime: 0 ms

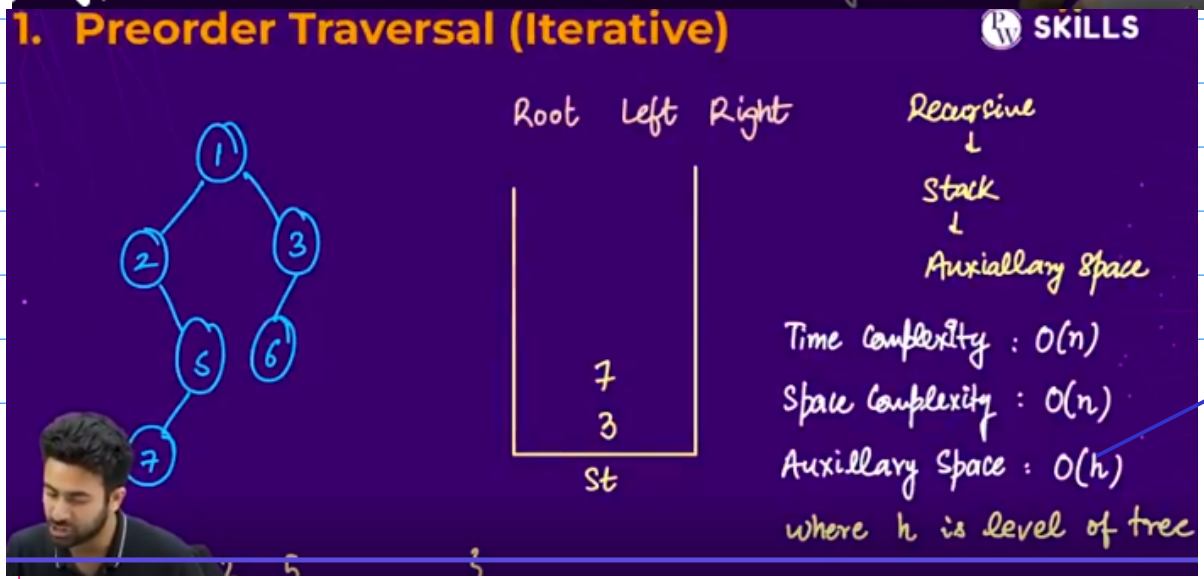
Case 1 | Case 2 | Case 3

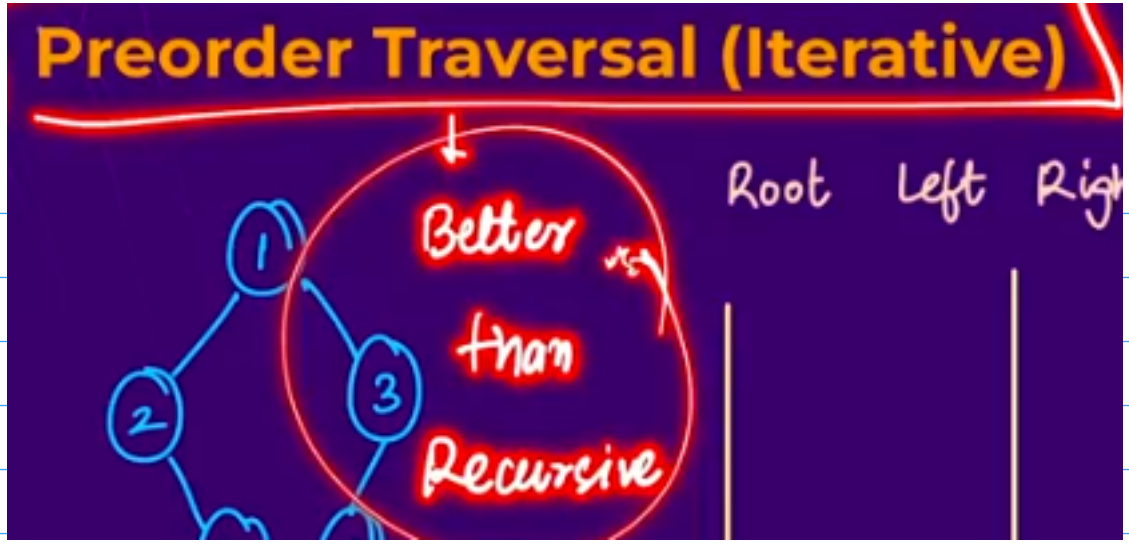
Input

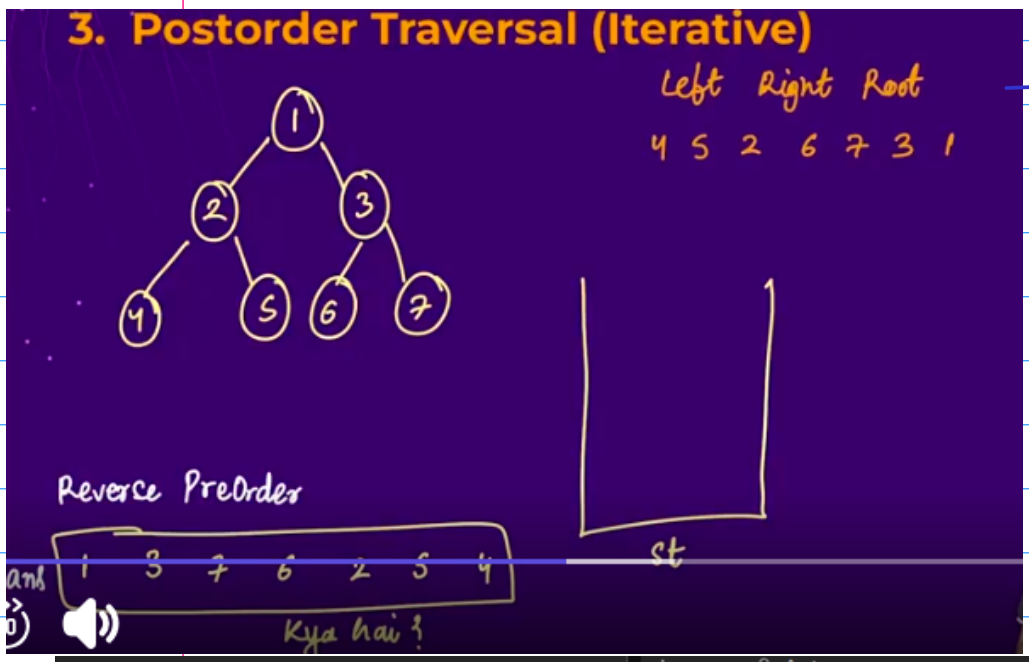
Java | Auto

```java
class Solution {
    public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> ans = new ArrayList<>();
        Stack<TreeNode> st = new Stack<>();
        if(root!=null) st.push(root);
        while(st.size()>0){
            TreeNode top = st.pop();
            ans.add(top.val);
            if(top.right!=null) st.push(top.right);
            if(top.left!=null) st.push(top.left);
        }
        return ans;
    }
}
```









→ **postorder nothing but it is a reverse preorder**

145. Binary Tree Postorder Traversal

Solved

Easy | Topics | Companies

Given the `root` of a binary tree, return *the postorder traversal of its nodes' values.*

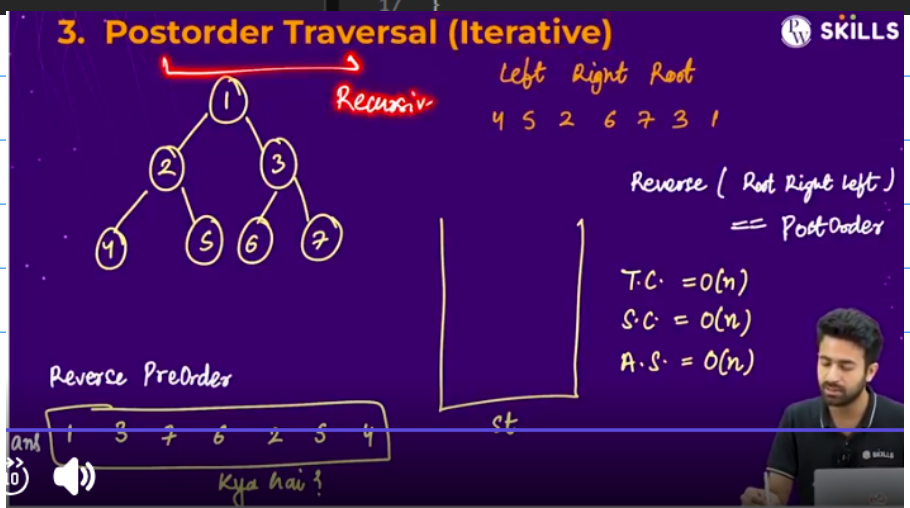
Example 1:

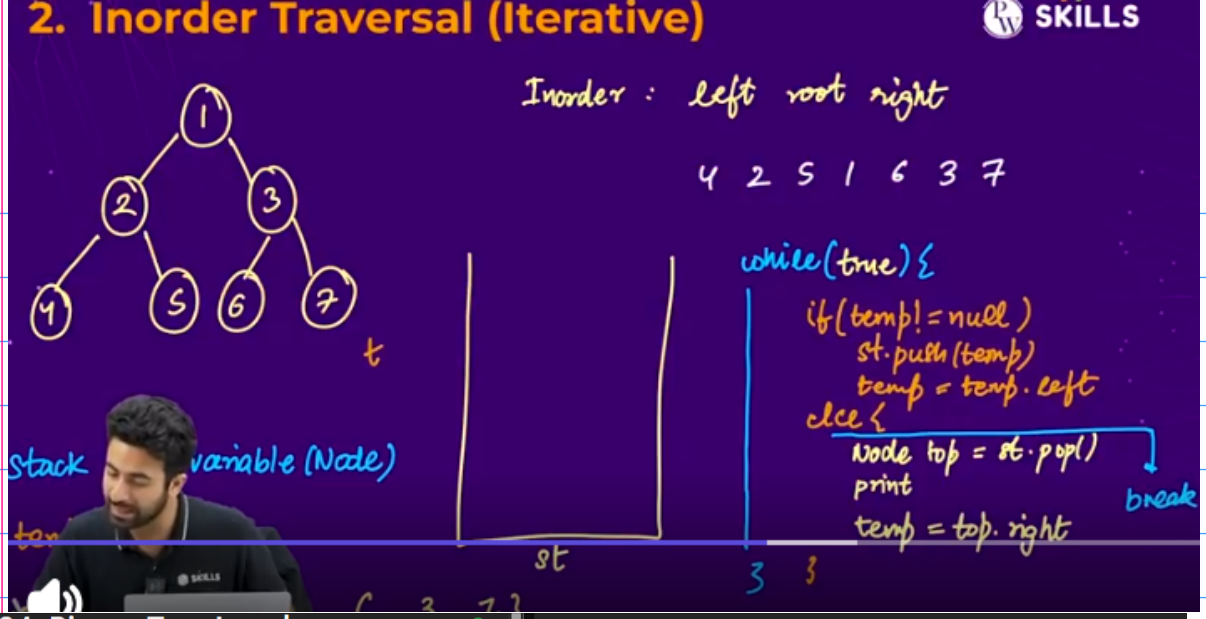
Input: `root = [1,null,2,3]`

Output: `[3,2,1]`

Java | Auto

```java
class Solution {
    public List<Integer> postorderTraversal(TreeNode root) {
        List<Integer> ans  = new ArrayList<>();//iterative method
        Stack<TreeNode> st = new Stack<>();
        if(root!=null) st.push(root);
        while(st.size()>0){
            TreeNode top = st.pop();
            ans.add(top.val);
            if(top.left!=null) st.push(top.left);
            if(top.right!=null) st.push(top.right);

        }
        Collections.reverse(ans);
        return ans;
    }
}
```





94. Binary Tree Inorder Traversal

Solved

Easy | Topics | Companies

Given the `root` of a binary tree, return *the inorder traversal of its nodes' values.*

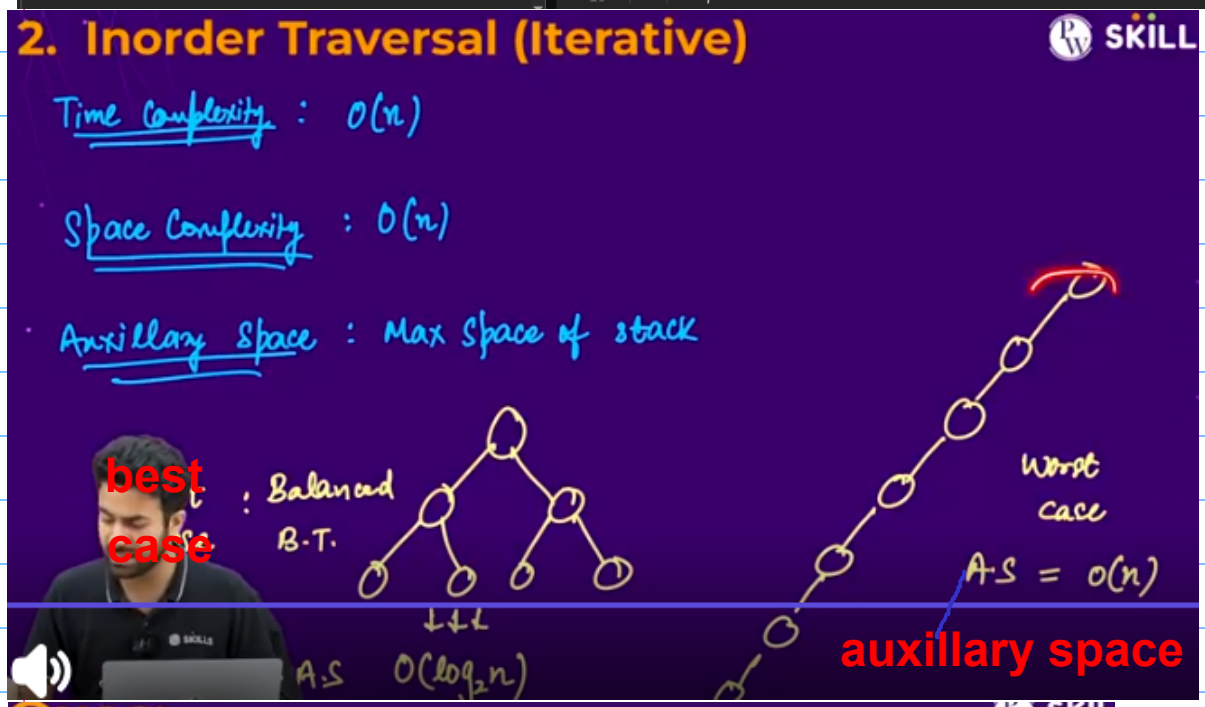
Example 1:

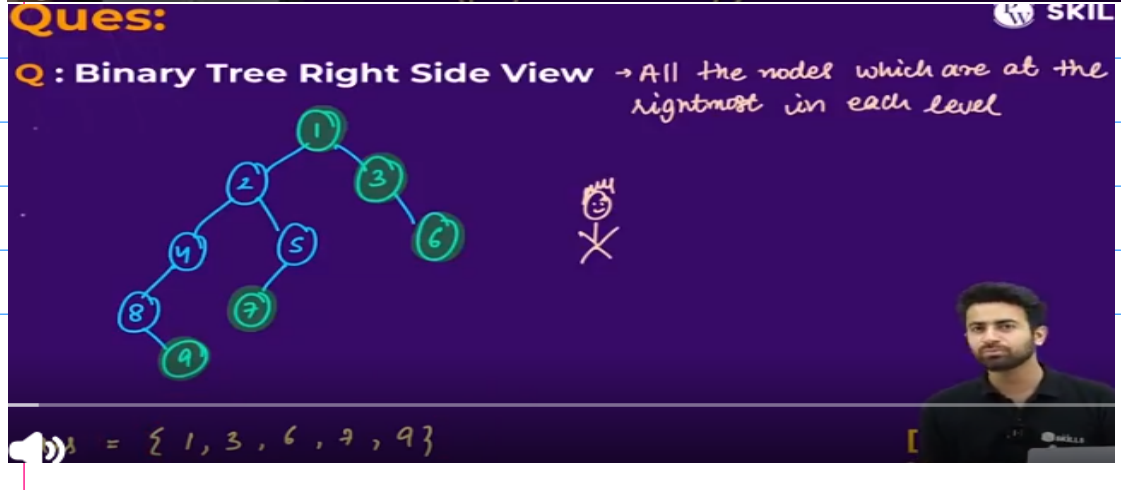
Input: `root = [1,null,2,3]`

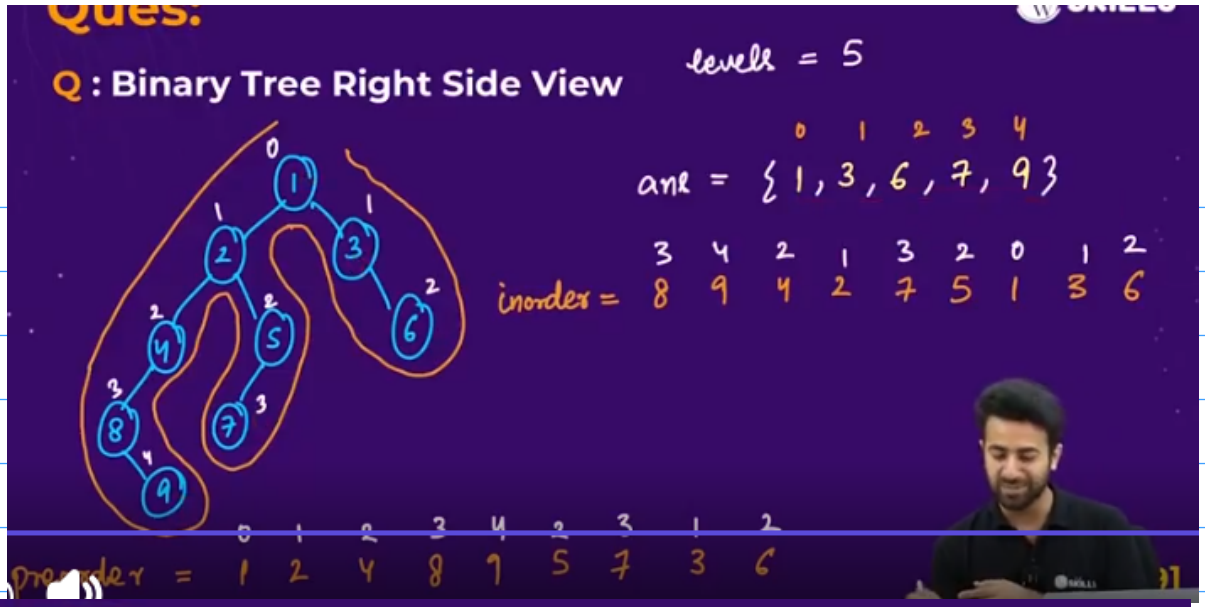
Output: `[1,3,2]`

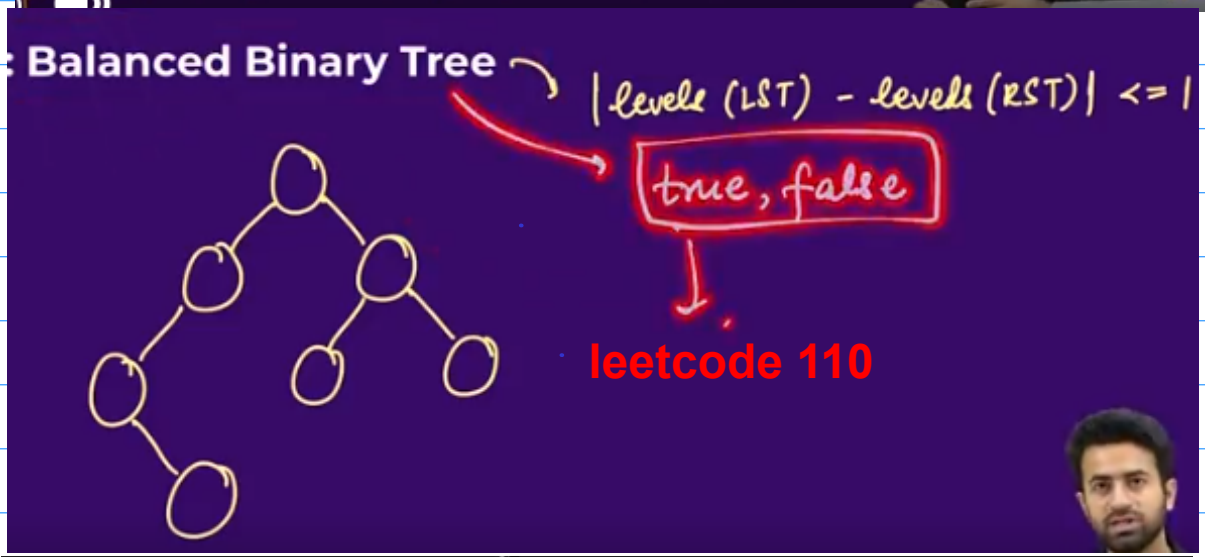
Explanation:

```java
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> ans = new ArrayList<>();
        Stack<TreeNode> st = new Stack<>();
        TreeNode temp = root;
        while(true){
            if(temp!=null){
                st.push(temp);
                temp = temp.left;
            }
            else{
            if(st.size()==0) break;
            TreeNode top = st.pop();
            ans.add(top.val);
            temp = top.right;


            }
        }
```









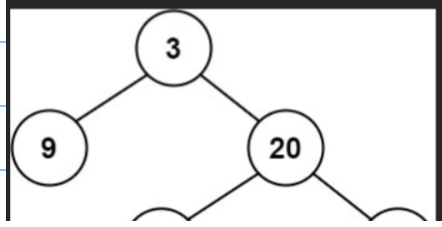
110. Balanced Binary Tree

Solved

Easy | Topics | Companies

Given a binary tree, determine if it is height-balanced.

Example 1:



Java | Auto

```java
class Solution {
    public int levels(TreeNode root){
        if(root==null) return 0;
        return 1+Math.max(levels(root.left),levels(root.right));
    }
    public boolean isBalanced(TreeNode root) {
        if(root==null) return true;
        int diff = Math.abs(levels(root.left)-levels(root.right));
        if(diff>1) return false;
        boolean lst =isBalanced(root.left);
        if(lst==false) return false;
        boolean rst =isBalanced(root.right);
        if(rst==false) return false;
        return true;
    }
}
```

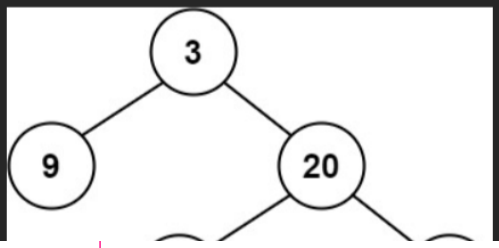
110. Balanced Binary Tree

Solved

Easy | Topics | Companies

Given a binary tree, determine if it is height-balanced.

Example 1:



Java | Auto

```java
        if (root == null)
            return 0;
        return 1 + Math.max(levels(root.left), levels(root.right));
    }

    public boolean isBalanced(TreeNode root) {
        if (root == null)
            return true;
        int diff = Math.abs(levels(root.left) - levels(root.right));
        if (diff > 1)
            return false;
        if (!isBalanced(root.left))
            return false;
        if (!isBalanced(root.right))
            return false;S
        return true;
    }
}
```

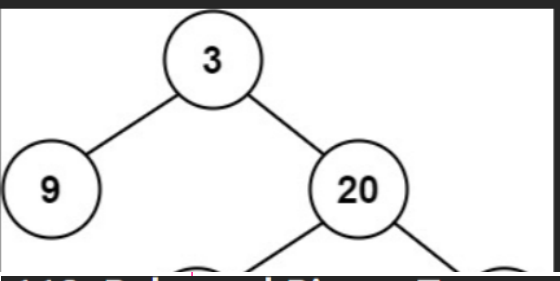
110. Balanced Binary Tree

Solved

Easy | Topics | Companies

Given a binary tree, determine if it is height-balanced.

Example 1:



Java | Auto

```java

class Solution {
    public int levels(TreeNode root) {
        if (root == null)
            return 0;
        return 1 + Math.max(levels(root.left), levels(root.right));
    }

    public boolean isBalanced(TreeNode root) {
        if (root == null)
            return true;
        int diff = Math.abs(levels(root.left) - levels(root.right));
        if (diff > 1) return false;
        return isBalanced(root.left)&&isBalanced(root.right) ;
    }
}
```

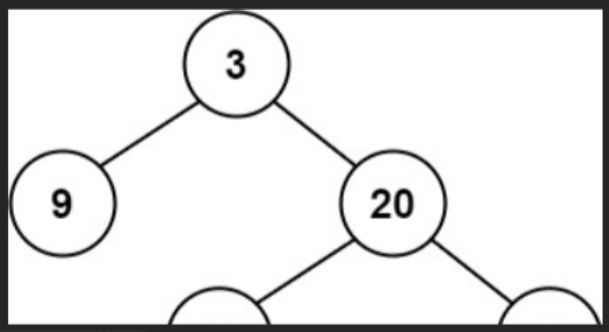
110. Balanced Binary Tree

Solved

Easy | Topics | Companies

Given a binary tree, determine if it is height-balanced.

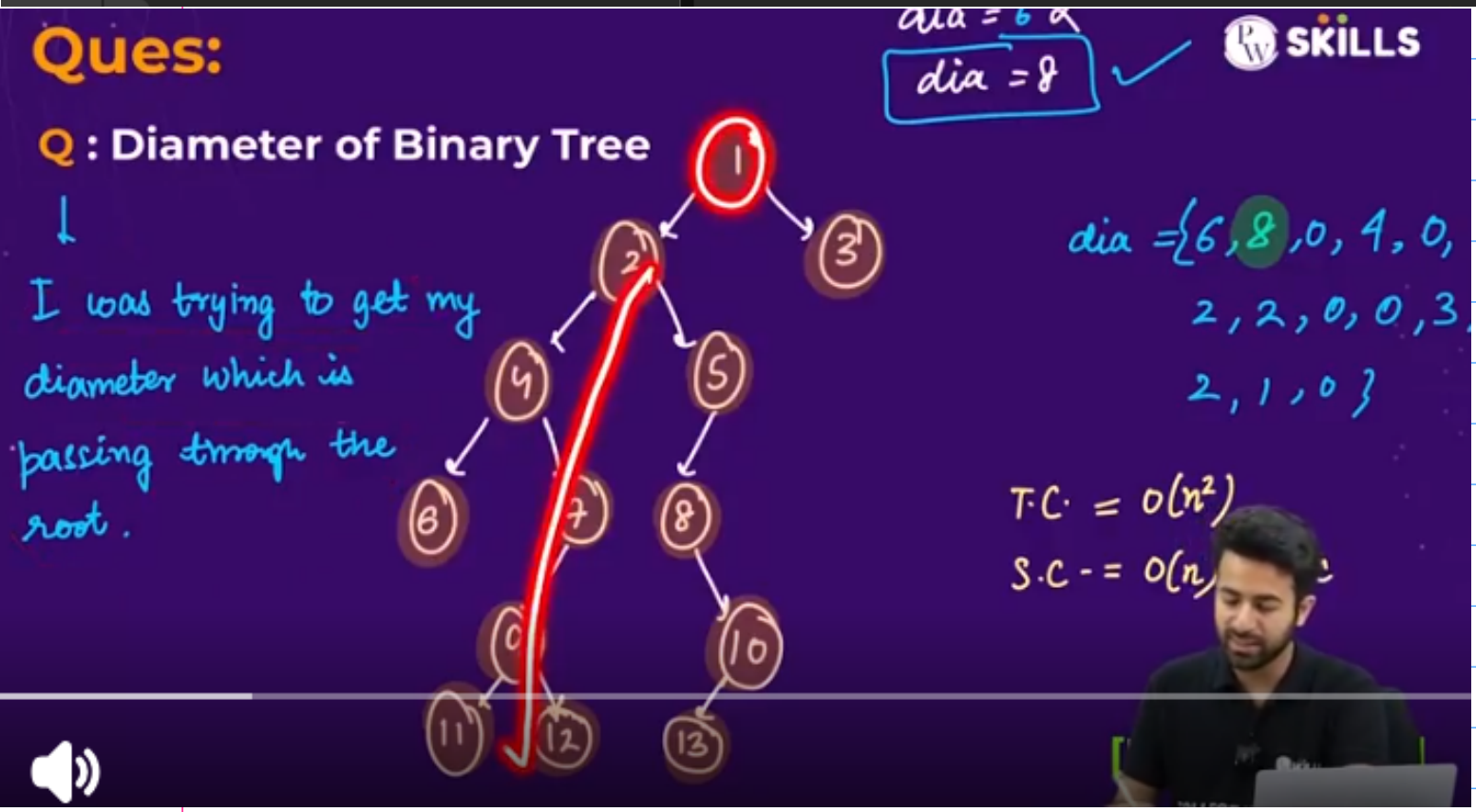
Example 1:



Java | Auto

```java
class Solution {
    static boolean ans;
    public int levels(TreeNode root) {
        if (root == null) return 0;
        int leftlevel = levels(root.left);
        int rightlevel  = levels(root.right);
    int diff = Math.abs(leftlevel-rightlevel);
    if(diff>1) ans= false;
        return 1 + Math.max(leftlevel,rightlevel );
    }

    public boolean isBalanced(TreeNode root) {
       ans=true;
       levels(root);
       return ans;
    }
}
```

optimise code



543. Diameter of Binary Tree

Solved

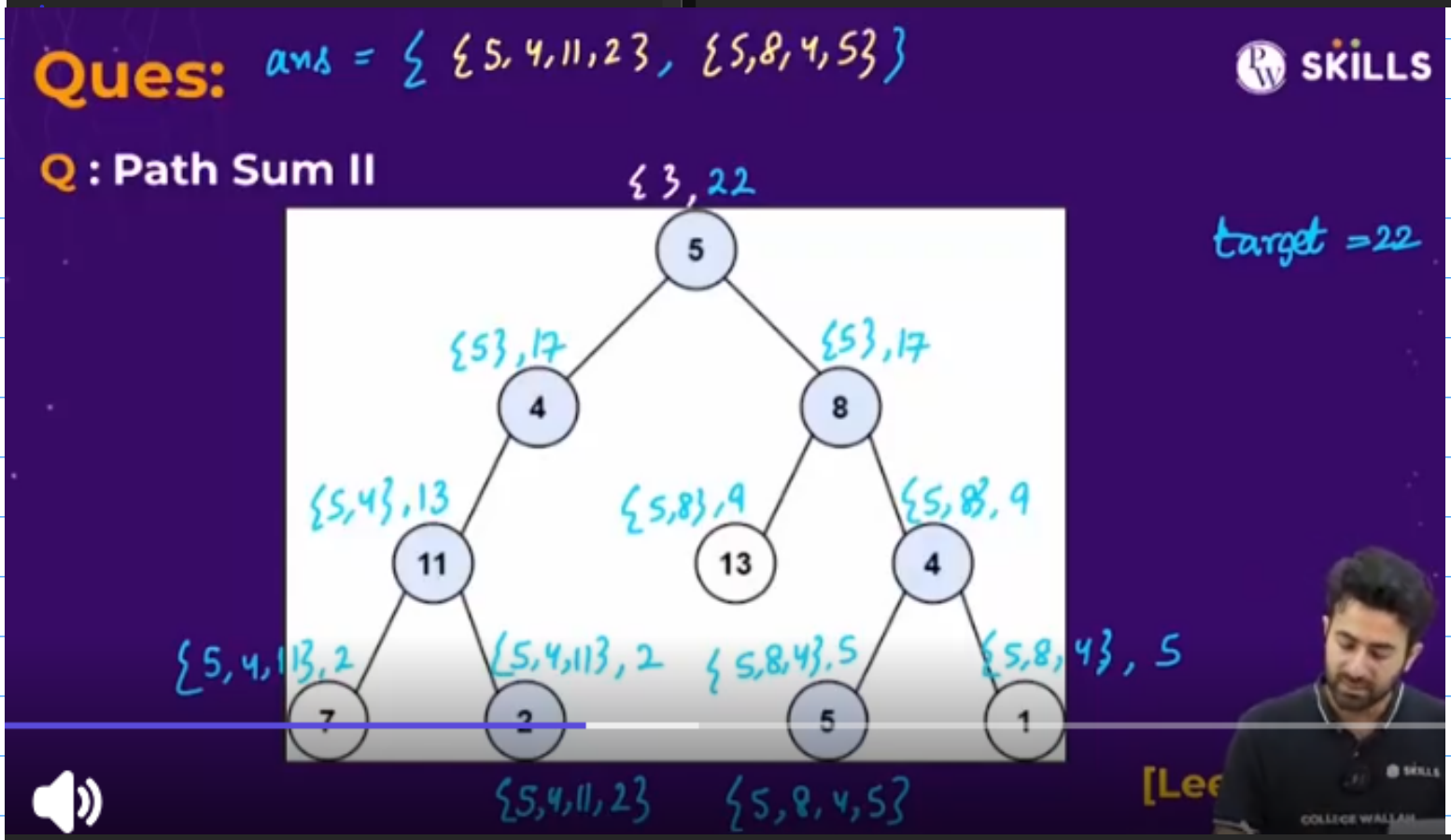
Easy | Topics | Companies

Given the `root` of a binary tree, return *the length of the **diameter** of the tree.*

The **diameter** of a binary tree is the **length** of the longest path between any two nodes in a tree. This path may or may not pass through the `root`.

The **length** of a path between two nodes is represented by the number of edges between them.

```java
class Solution {
    static int maxDia;
    public int levels(TreeNode root){
        if(root==null) return 0;
        int leftlevel = levels(root.left);
        int rightlevel = levels(root.right);
        int mydia = leftlevel+rightlevel;
        maxDia = Math.max(mydia,maxDia);
        return 1+Math.max(leftlevel,rightlevel);
    }
    public int diameterOfBinaryTree(TreeNode root) {
        maxDia= 0;
        levels(root);
        return maxDia;

    }
}
```



113. Path Sum II

Solved

Medium | Topics | Companies

Given the `root` of a binary tree and an integer `targetSum`, return *all **root-to-leaf** paths where the sum of the node values in the path equals* `targetSum`*. Each path should be returned as a list of the node **values**, not node references.*

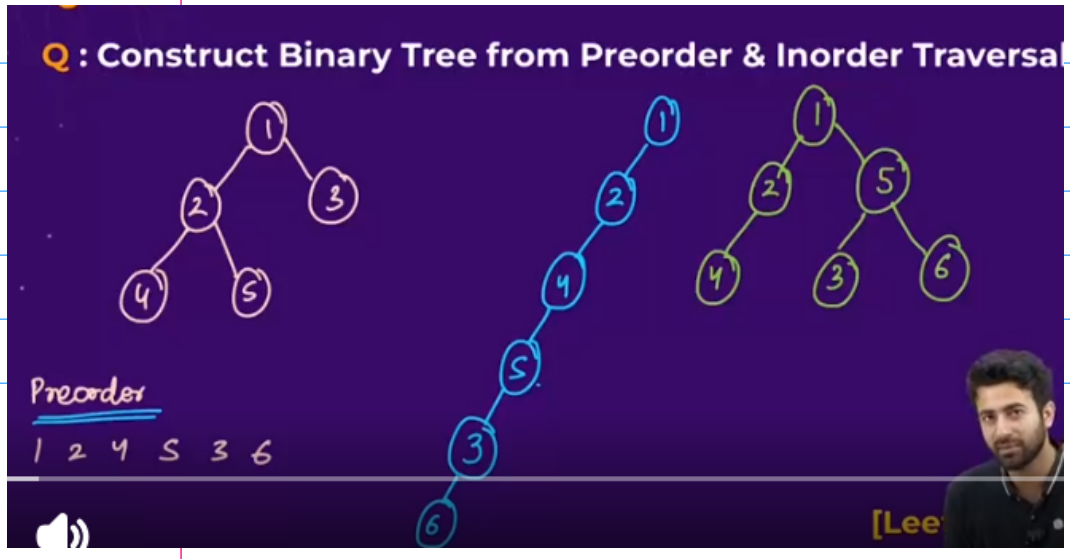
A **root-to-leaf** path is a path starting from the root and ending at any leaf node. A **leaf** is a node with no children.

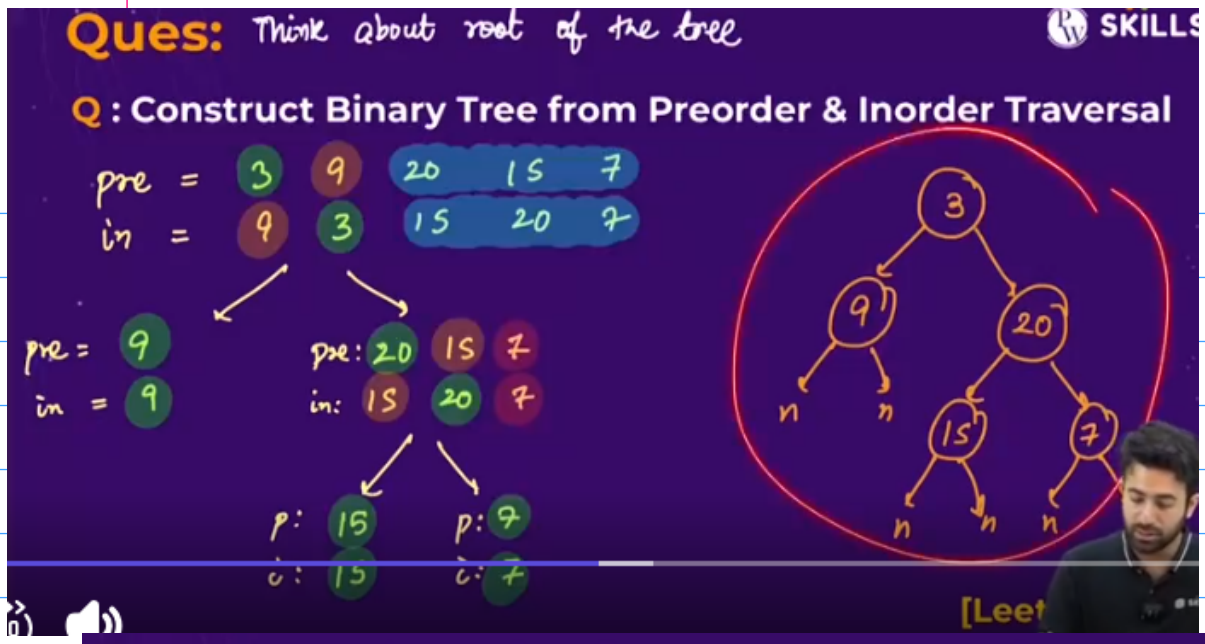
```
class Solution {
    public List<Integer> copy(List<Integer> arr) {
        List<Integer> list = new ArrayList<>();
        for (int ele : arr) {
            list.add(ele);
            }
        return list;
         }
 public void helper(TreeNode root, int target, List<Integer> arr, List<List<Integer>> ans) {
        if (root == null)
            return;
        if (root.left == null && root.right == null) {
            if (root.val == target) {
                arr.add(root.val);
                ans.add(arr);
            }
            return;
        }
        arr.add(root.val);
        List<Integer> arr1 = copy(arr);
        List<Integer> arr2 = copy(arr);
        helper(root.left, target - root.val, arr1, ans);
        helper(root.right, target - root.val, arr2, ans);

    }

    public List<List<Integer>> pathSum(TreeNode root, int targetSum) {
        List<List<Integer>> ans = new ArrayList<>();
        List<Integer> arr = new ArrayList<>();
        helper(root, targetSum, arr, ans);
        return ans;

    }
}
```

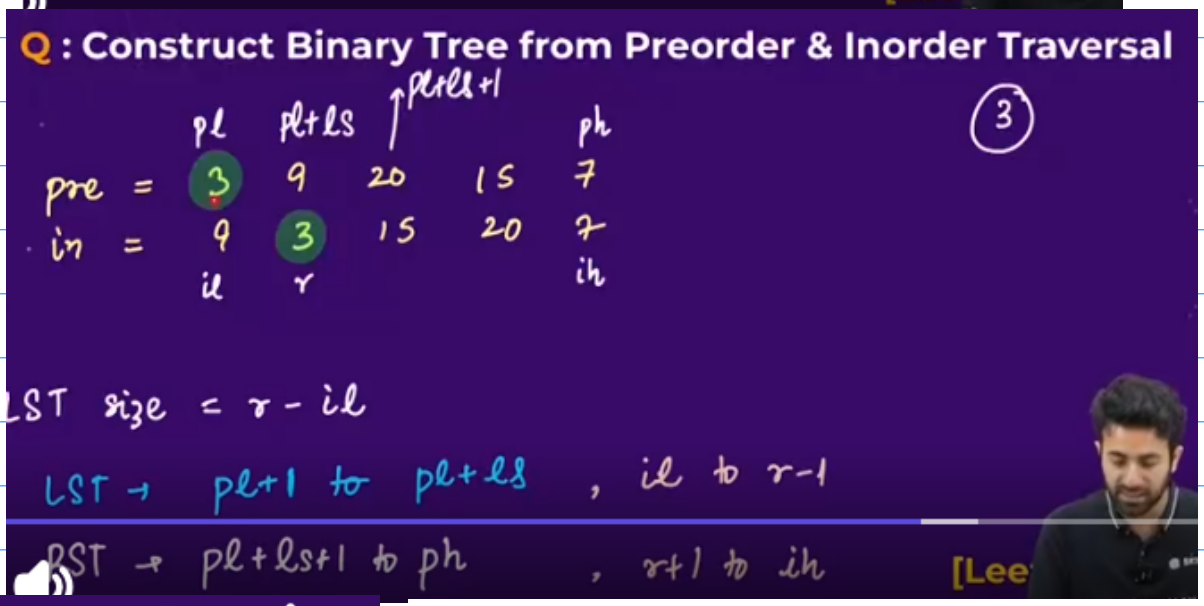
Q : Path Sum III

```
public int helper(TreeNode root, long sum){
    if(root==null) return 0;
    int count = 0;
    if(root.val==sum) count++;
    count += helper(root.left,sum-root.val) + helper(root.right,sum-root.val);
    return count;
}
public int pathSum(TreeNode root, int sum) {
    if(root==null) return 0;
    return helper(root,sum) + pathSum(root.left,sum) + pathSum(root.right,sum);
}
```

Q : Construct Binary Tree from Preorder & Inorder Traversal







Assignment Questions

Q. Find the product of all nodes in a Binary Tree.

Q. Find the minimum value in a Binary tree.

Q. Balanced Binary Tree (Leetcode 110).

Q. Symmetric Tree (Leetcode 101)

Q. Level Order Traversal(Right to Left).

Q. ZigZag level order traversal(Leetcode 103).

Q. Construct Binary Tree from Inorder & Postorder Traversal(Leetcode 106).

Q. Construct Binary Tree from Preorder & Postorder Traversal (Leetcode 889).

Q. **Left View of Binary Tree**
Given a Binary Tree, print the Left view of it.
The left view of a binary tree refers to the set of nodes that are visible when the tree is viewed from the left side.

Q. Path Sum I [LeetCode 112]