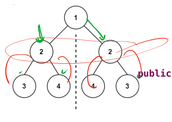
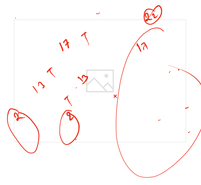
Lec-32

06/07/24 1:31 PM



```
public boolean Symmetric(TreeNode root1, TreeNode root2) {
    boolean left = Symmetric(root1.left, root2.right);
    boolean right = Symmetric(root1.right, root2.left);
}
```

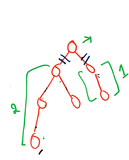


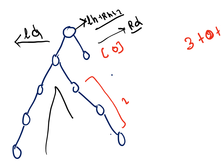
```
public int sumofnumber(TreeNode root, int sum) {
    if (root == null) {
        return 0;
    }
    if (root.left == null && root.right == null) {
        return sum * 10 + root.val;
    }
    int left = sumofnumber(root.left, sum * 10 + root.val);
    int right = sumofnumber(root.right, sum * 10 + root.val);
    return left + right;
}
```

```
public void RightView(TreeNode root, int curr, List<Integer> ll) {
    if (root == null) {
        return;
    }
    if (curr > max_Depth) {
        ll.add(root.val);
        max_Depth = curr;
    }
    RightView(root.right, curr + 1, ll);
    RightView(root.left, curr + 1, ll);
}
```

```
public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
    if (root == null) {
        return null;
    }
    if (root == p || root == q) {
        return root;
    }
    TreeNode left = lowestCommonAncestor(root.left, p, q);
    TreeNode right = lowestCommonAncestor(root.right, p, q);
}
```

```
public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
    if (root == null) {
        return null;
    }
    if (root == p || root == q) {
        return root;
    }
    TreeNode left = lowestCommonAncestor(root.left, p, q);
    TreeNode right = lowestCommonAncestor(root.right, p, q);
    if(left!=null && right!=null) {
        return root;
    }
    else if(left==null) {
        return right;
    }
    else {
        return left;
    }
}
```





```
public int diameter(TreeNode root) {
    if (root == null) {
        return 0;
    }
    int ld = diameter(root.left);
    int rd = diameter(root.right);
    int sd = ht(root.left) + ht(root.right) + 2;
    return Math.max(ld, Math.max(rd, sd));
}

public int ht(TreeNode root) {
    if (root == null) {
        return -1;
    }
    int lh = ht(root.left);
    int rh = ht(root.right);
    return Math.max(lh, rh) + 1;
}
```