1. **Pre-Order**

public static void preOrder(Node root) {

    if (root != null) {

        System.out.print(root.data + " ");

        preOrder(root.left);

        preOrder(root.right);

    }

  }

1. **Leve -Order**

**void levelOrder(Node root) {**

if (root == null) return;

**Queue<Node> queue = new LinkedList<>();**

queue.add(root);

while (!queue.isEmpty()) {

Node current = queue.poll();

System.out.print(current.data + " ");

***if (current.left != null) queue.add(current.left);***

***if (current.right != null) queue.add(current.right);***

}

**}**

1. **Height of Binary Tree**

public static int height(Node root) {

        // Write your code here.

        if(root == null) return -1;

**int l\_h = height(root.left);**

**int r\_h = height(root.right);**

        return Math.max (l\_h, r\_h) + 1;

    }

**4.Diameter of Binary Tree**

***public static int dia\_meter(TreeNode root) {***

        int max[] = new int[1];

***height(root, max);***

        return max[0];

    }

***private static int height(TreeNode root, int max[]) {***

        if (root == null) {

            return 0;

        }

***int l = height(root.left, max);***

***int r = height(root.right, max);***

        max[0] = Math.max(max[0], l + r);

        return 1 + Math.max(l, r);

***}***

**5. Max\_Path\_Sum**

**public static int max\_path\_sum(TreeNode root) {**

        int max\_val[] = new int[1];

        max\_val[0] = Integer.MIN\_VALUE;

**path\_sum(root, max\_val);**

        return max\_val[0];

**}**

**private static int path\_sum(TreeNode root, int max\_val[]) {**

*if (root == null) {*

*return 0;*

*}*

**int l = Math.max(0, path\_sum(root.left, max\_val));**

**int r = Math.max(0, path\_sum(root.right, max\_val));**

max\_val[0] = Math.max(max\_val[0], l + r + root.val);

        return Math.max(l, r) + root.val;

**}**

**6. 2 Tree are Same**

**public boolean isSameTree(TreeNode p, TreeNode q) {**

*if(p==null || q == null){*

*return (p==q);*

*}*

**return (p.val == q.val)**

*&&* **isSameTree(p.left, q.left)**

*&&* **isSameTree(p.right, q.right);**

**}**

**7. Lowest Common Ancestor**

**public static Node lca(Node root, int v1, int v2) {**

        if (root == null)

            return null;

        if (root.data == v1 || root.data == v2)

            return root;

**Node l = lca(root.left, v1, v2);**

**Node r = lca(root.right, v1, v2);**

*if(l != null && r != null){*

*return root;*

**}**

        return l != null ? l: r;

**}**

**8.** **latten a Binary Tree**

**public void flatten(TreeNode root) {**

        TreeNode curr = root;

***while(curr != null){***

***if(curr.left != null){***

***TreeNode pre = curr.left;***

*while(pre.right != null){*

*pre = pre.right;*

*}*

***pre.right = curr.right;***

***curr.right = curr.left;***

***curr.left = null;***

***}***

***curr = curr.right;***

        }    }