
Medium Problems
Bottom View of Binary Tree.java
Boundary Traversal of Binary Tree.java
Check if the Binary tree is height-balanced or not.java
Check if two trees are identical or not.java
Diameter of Binary Tree.java
Height of a Binary Tree.java
Maximum path sum.java
Right or Left View of Binary Tree.java
Symmetric Binary Tree.java
Top View of Binary Tree.java
Vertical Order Traversal of Binary Tree.java
Zig Zag Traversal of Binary Tree.java

```
/*
    Bottom View of Binary Tree
    Given a binary tree, print the bottom view from left to right.
    A node is included in bottom view if it can be seen when we look at the tree from bottom.

                        20
                      /    \
                    8       22
                  /   \        \
                5      3       25
                     /   \
                   10    14

    For the above tree, the bottom view is 5 10 3 14 25.
    If there are multiple bottom-most nodes for a horizontal distance from root, then
    print the later one in level traversal. For example, in the below diagram, 3 and
    4 are both the bottommost nodes at horizontal distance 0, we need to print 4.

                        20
                      /    \
                    8       22
                  /   \     /   \
                5      3 4     25
                     /    \
                   10       14

    For the above tree the output should be 5 10 4 14 25.


    Example 1:

    Input:
          1
        /   \
       3     2
    Output: 3 1 2
    Explanation:
    First case represents a tree with 3 nodes
    and 2 edges where root is 1, left child of
    1 is 3 and right child of 1 is 2.

    Thus nodes of the binary tree will be
    printed as such 3 1 2.
    Example 2:

    Input:
           10
         /    \
        20    30
       /  \
      40   60
    Output: 40 20 60 30
*/
```

Bottom_View_of_Binary_Tree

```
Codeium: Refactor | Explain
class Node{
    int data;
    Node left;
    Node right;
    Node(int key){
        this.data = key;
    }
}

Codeium: Refactor | Explain
class Pair{
    Node node;
    int hd;
    Pair(Node node, int hd){
        this.node = node;
        this.hd = hd;
    }
}

```

```java

Codeium: Refactor | Explain
public class Bottom_View_of_Binary_Tree {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static ArrayList <Integer> bottomView(Node root){
        ArrayList<Integer> ans = new ArrayList<>();
        TreeMap<Integer, Integer> map = new TreeMap<>();
        if(root == null) return ans;
        Queue<Pair> q = new LinkedList<>();
        q.add(new Pair(root, 0));
        while(!q.isEmpty()){
            Pair it = q.remove();
            int hd = it.hd;
            Node temp = it.node;
            map.put(hd, temp.data);
            if(temp.left != null) q.add(new Pair(temp.left, hd - 1));
            if(temp.right != null) q.add(new Pair(temp.right, hd + 1));
        }
        for(Integer num : map.values()){
            ans.add(num);
        }
        return ans;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        /*
        Real World Representation the below tree into the code.
                    1
                  /  \
                 2     3
                / \   / \
               4   5 6   7
                 /     / \
                8     9    10
        */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.left.right.left = new Node(8);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        root.right.right.left = new Node(9);
        root.right.right.right = new Node(10);
        System.out.println(bottomView(root));
    }
}

```

```
> Task :Bottom_View_of_Binary_Tree.main()
[4, 8, 6, 9, 7, 10]
```

Boundary_Traversal_of_Binary_Tree

```java
public class Boundary_Traversal_of_Binary_Tree {
    public static boolean isLeaf(Node node){
        return node.left == null && node.right == null;
    }
    public static void addLeftBoundry(Node root, ArrayList<Integer> res){
        Node curr = root.left;
        while(curr != null){
            if(isLeaf(curr) == false) res.add(curr.data);
            if(curr.left != null) curr = curr.left;
            else curr = curr.right;
        }
    }
    public static void addRightBoundry(Node root, ArrayList<Integer> res){
        ArrayList<Integer> temp = new ArrayList<>();
        Node curr = root.right;
        while(curr != null){
            if(isLeaf(curr) == false) temp.add(curr.data);
            if(curr.right != null) curr = curr.right;
            else curr = curr.left;
        }
        // Adding Right boundary in reverse order to Achieve Anti-ClockWise
        for(int i = temp.size() - 1; i >= 0; i--){
            res.add(temp.get(i));
        }
    }
    public static void addLeaves(Node root, ArrayList<Integer> res){
        if(isLeaf(root)){
            res.add(root.data);
        }
        if(root.left != null) addLeaves(root.left, res);
        if(root.right != null) addLeaves(root.right, res);
    }
    public static ArrayList<Integer> printBoundry(Node root){
        ArrayList<Integer> ans = new ArrayList<>();
        if(isLeaf(root) == false) ans.add(root.data);
        addLeftBoundry(root, ans);
        addLeaves(root, ans);
        addRightBoundry(root, ans);
        return ans;
    }
```

```
    public static void main(String[] args) {
        /*
            Real World Representation the below tree into the code.
                      1
                    /  \
                   2     3
                  / \   / \
                 4   5 6   7
                  /     / \
                 8     9    10
        */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.left.right.left = new Node(8);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        root.right.right.left = new Node(9);
        root.right.right.right = new Node(10);
        System.out.println(printBoundry(root));
    }
}
```

```
> Task :Boundary_Traversal_of_Binary_Tree.main()
[1, 2, 4, 8, 6, 9, 10, 7, 3]
```

Check_if_the_Binary_tree_is_height_balanced_or_not

```
/*
    110. Balanced Binary Tree
    Given a binary tree, determine if it is
    height-balanced

    Example 1:
    Input: root = [3,9,20,null,null,15,7]
    Output: true

    Example 2:
    Input: root = [1,2,2,3,3,null,null,4,4]
    Output: false
    Example 3:

    Input: root = []
    Output: true
*/
```

```java
public class Check_if_the_Binary_tree_is_height_balanced_or_not {
    public static int maxDepth(Node root) {
        if(root == null) return 0;
        int lh = maxDepth(root.left);
        if(lh == -1) return -1;
        int rh = maxDepth(root.right);
        if(rh == -1) return -1;
        if(Math.abs(lh - rh) > 1) return -1;
        return 1 + Math.max(lh, rh);
    }
    public static boolean isBalanced(Node root) {
        return maxDepth(root) != -1;
    }
    public static void main(String[] args) {
        /*
            Real World Representation the below tree into the code.
                    1
                  /  \
                 2     3
                / \   / \
               4   5 6   7
                /     / \
               8     9    10
        */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.left.right.left = new Node(8);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        root.right.right.left = new Node(9);
        root.right.right.right = new Node(10);
        System.out.println(isBalanced(root));
    }
}
```

```
> Task :Check_if_the_Binary_tree_is_height_balanced_or_not.main()
true
```

Check if two trees are identical or not

```
/*
   100. Same Tree
   Given the roots of two binary trees p and q, write a function to check if they are the same or not.

   Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

   Example 1:
   Input: p = [1,2,3], q = [1,2,3]
   Output: true
   Example 2:


   Input: p = [1,2], q = [1,null,2]
   Output: false

   Example 3:
   Input: p = [1,2,1], q = [1,1,2]
   Output: false
*/

   public class Check_if_two_trees_are_identical_or_not {
       public static boolean isSameTree(Node p, Node q) {
           if(p == null || q == null){
               return p == q;
           }
           return p.data == q.data && isSameTree(p.left, q.left) && isSameTree(p.right, q.right);
       }
       public static void main(String[] args) {
           /*
               Real World Representation the below tree into the code.
                        1
                      /  \
                    2     3
                   / \   / \
                  4   5 6   7
                    /     / \
                   8     9    10
           */

           // First Tree
           Node root = new Node(1);
           root.left = new Node(2);
           root.right = new Node(3);
           root.left.left = new Node(4);
           root.left.right = new Node(5);
           root.left.right.left = new Node(8);
           root.right.left = new Node(6);
           root.right.right = new Node(7);
           root.right.right.left = new Node(9);
           root.right.right.right = new Node(10);

           // Second Tree
           Node root2 = new Node(1);
           root2.left = new Node(2);
           root2.right = new Node(3);
           root2.left.left = new Node(4);
           root2.left.right = new Node(5);
           root2.left.right.left = new Node(8);
           root2.right.left = new Node(6);
           root2.right.right = new Node(7);
           root2.right.right.left = new Node(9);
           root2.right.right.right = new Node(10);

           System.out.println(isSameTree(root, root2));
       }
   }

```

```
> Task :Check_if_two_trees_are_identical_or_not.main()
true
```

Diameter_of_Binary_Tree

```
/*
  543. Diameter of Binary Tree
  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.

  Example 1:
  Input: root = [1,2,3,4,5]
  Output: 3
  Explanation: 3 is the length of the path [4,2,1,3] or [5,2,1,3].

  Example 2:
  Input: root = [1,2]
  Output: 1
*/
```

```
27
28  public class Diameter_of_Binary_Tree {
29      public static int height(Node root, int[] diameter){
30          if(root ==  null) return 0;
31
32          int lh = height(root.left, diameter);
33          int rh = height(root.right, diameter);
34          diameter[0] = Math.max(diameter[0], lh + rh);
35          return 1 + Math.max(lh, rh);
36      }
37      public static int diameterOfBinaryTree(Node root) {
38          int[] diameter = new int[1];
39          height(root, diameter);
40          return diameter[0];
41      }
42      public static void main(String[] args) {
43          /*
44              Real World Representation the below tree into the code.
45                        1
46                      /  \
47                     2     3
48                    / \   / \
49                   4   5 6   7
50                     /     / \
51                    8     9    10
52          */
53          Node root = new Node(1);
54          root.left = new Node(2);
55          root.right = new Node(3);
56          root.left.left = new Node(4);
57          root.left.right = new Node(5);
58          root.left.right.left = new Node(8);
59          root.right.left = new Node(6);
60          root.right.right = new Node(7);
61          root.right.right.left = new Node(9);
62          root.right.right.right = new Node(10);
63          System.out.println(diameterOfBinaryTree(root));
64      }
65  }
66
```

```
> Task :Diameter_of_Binary_Tree.main()
6
```

Height_of_a_Binary_Tree

```
/*
   104. Maximum Depth of Binary Tree
   Given the root of a binary tree, return its maximum depth.

   A binary tree's maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

   Example 1:


   Input: root = [3,9,20,null,null,15,7]
   Output: 3
   Example 2:

   Input: root = [1,null,2]
   Output: 2
*/
```

```java
public class Height_of_a_Binary_Tree {
    // Iterative
    /*
        public int maxDepth(TreeNode root) {
            int height = 0;
            Queue<TreeNode> queue = new LinkedList<>();
            if(root == null) return 0;
            queue.add(root);
            while(!queue.isEmpty()){
                int k = queue.size();
                for(int i = 0; i < k; i++){
                    if(queue.peek().left != null) queue.add(queue.peek().left);
                    if(queue.peek().right != null) queue.add(queue.peek().right);
                    queue.remove();
                }
                height++;
            }
            return height;
        }
    */

    // Recurive
    public static int maxDepth(Node root) {
        if(root == null) return 0;
        int lh = maxDepth(root.left);
        int rh = maxDepth(root.right);
        return 1 + Math.max(lh, rh);
    }
    public static void main(String[] args) {
        /*
            Real World Representation the below tree into the code.
                    1
                  /  \
                 2     3
                / \   / \
               4   5 6   7
                /     / \
               8     9    10
        */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.left.right.left = new Node(8);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        root.right.right.left = new Node(9);
        root.right.right.right = new Node(10);
        System.out.println(maxDepth(root));
    }
}
```

```
> Task :Height_of_a_Binary_Tree.main()
4
```

Maximum_path_sum

```
/*
  124. Binary Tree Maximum Path Sum
  A path in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them.
  A node can only appear in the sequence at most once.
   Note that the path does not need to pass through the root.

  The path sum of a path is the sum of the node's values in the path.

  Given the root of a binary tree, return the maximum path sum of any non-empty path.

  Example 1:
  Input: root = [1,2,3]
  Output: 6
  Explanation: The optimal path is 2 -> 1 -> 3 with a path sum of 2 + 1 + 3 = 6.

  Example 2:
  Input: root = [-10,9,20,null,null,15,7]
  Output: 42
  Explanation: The optimal path is 15 -> 20 -> 7 with a path sum of 15 + 20 + 7 = 42.
*/

--
public class Maximum_path_sum {
    public static int maximumPath(Node root, int[] maxSum){
        if(root == null) return 0;
        // Not Consider Negative Value Take "0" Instead
        int lSum = Math.max(0, maximumPath(root.left, maxSum));
        // Not Consider Negative Value Take "0" Instead
        int rSum = Math.max(0, maximumPath(root.right, maxSum));
        maxSum[0] = Math.max(maxSum[0], root.data + (lSum + rSum));
        return root.data + Math.max(lSum, rSum);
    }
    public static int maxPathSum(Node root) {
        int[] maxSum = new int[1];
        maxSum[0] = Integer.MIN_VALUE;
        maximumPath(root, maxSum);
        return maxSum[0];
    }
    public static void main(String[] args) {
        /*
            Real World Representation the below tree into the code.
                    1
                  /  \
                2     3
               / \   / \
              4   5 6   7
                /     / \
               8     9    10
        */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.left.right.left = new Node(8);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        root.right.right.left = new Node(9);
        root.right.right.right = new Node(10);
        System.out.println(maxPathSum(root));
    }
}

```

```
> Task :Maximum_path_sum.main()
36
```

Binary_Tree_Right_Side_View

```java
/*
  199. Binary Tree Right Side View
  Given the root of a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you
  can see ordered from top to bottom.

  Example 1:
  Input: root = [1,2,3,null,5,null,4]
  Output: [1,3,4]

  Example 2:
  Input: root = [1,null,3]
  Output: [1,3]

  Example 3:
  Input: root = []
  Output: []
*/

public class Binary_Tree_Right_Side_View {
    /*
        // Iteretive
        public static List<Integer> rightSideView(Node root) {
            List<Integer> ans = new ArrayList<>();
            Queue<Node> q = new LinkedList<>();
            if(root == null) return ans;
            q.add(root);
            while(!q.isEmpty()){
                int k = q.size();
                int RightMost = 0;
                for(int i = 0; i < k; i++){
                    if(q.peek().left != null) q.add(q.peek().left);
                    if(q.peek().right != null) q.add(q.peek().right);
                    RightMost = q.remove().data;
                }
                ans.add(RightMost);
            }
            return ans;
        }
    */

    // Recursive
    public static List<Integer> rightSideView(Node root) {
        List<Integer> ans = new ArrayList<>();
        rightView(root, ans, 0);
        return ans;
    }
    public static void rightView(Node root, List<Integer> ans, int currDepth){
        if(root == null) return;
        // Storing ans in our list
        if(currDepth == ans.size()) ans.add(root.data);
        // Right
        if(root.right != null) rightView(root.right, ans, currDepth + 1);
        // Left
        if(root.left != null) rightView(root.left, ans, currDepth + 1);
    }
```

```
70      public static void main(String[] args) {
71          /*
72              Visualisation of the tree in real world
73                        1
74                      /  \
75                    2     3
76                   / \   / \
77                  4   5 6   7
78                    /     / \
79                   8     9    10
80          */
81          Node root = new Node(1);
82          root.left = new Node(2);
83          root.right = new Node(3);
84          root.left.left = new Node(4);
85          root.left.right = new Node(5);
86          root.left.right.left = new Node(8);
87          root.right.left = new Node(6);
88          root.right.right = new Node(7);
89          root.right.right.left = new Node(9);
90          root.right.right.right = new Node(10);
91          System.out.println(rightSideView(root));
92      }
93  }
```

```
> Task :Binary_Tree_Right_Side_View.main()
[1, 3, 7, 10]
```

Symmetric_Binary_Tree

```java
/*
   101. Symmetric Tree
   Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center).

   Example 1:
   Input: root = [1,2,2,3,4,4,3]
   Output: true

   Example 2:
   Input: root = [1,2,2,null,3,null,3]
   Output: false
*/

public class Symmetric_Binary_Tree {
    public static boolean isSymmetric(Node root) {
        return root == null || isSymmetricHelp(root.left, root.right);
    }
    public static boolean isSymmetricHelp(Node left, Node right){
        if(left == null || right == null){
            return left == right;
        }
        if(left.data != right.data) return false;
        return isSymmetricHelp(left.left, right.right) && isSymmetricHelp(left.right, right.left);
    }
    public static void main(String[] args) {
        /*
            Visualisation of the tree in real world
                    1
                  /  \
                 2    3
                / \   / \
               4  5 6   7
                /     / \
               8     9   10
        */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(2);
        root.left.left = new Node(3);
        root.left.right = new Node(4);
        root.right.left = new Node(4);
        root.right.right = new Node(3);
        System.out.println(isSymmetric(root));
    }
}
```

```
> Task :Symmetric_Binary_Tree.main()
true
```

Top_View_of_Binary_Tree

```
/*
   Q. Top View of Binary Tree
   Given below is a binary tree. The task is to print the top view of binary tree. Top view of a binary tree is the set of nodes
    visible when the tree is viewed from the top. For the given below tree

        1
      /    \
     2      3
    /  \    /   \
   4    5  6   7

   Top view will be: 4 2 1 3 7
   Note: Return nodes from leftmost node to rightmost node. Also if 2 nodes are outside the shadow of the tree and are at same position
   then consider the extreme ones only(i.e. leftmost and rightmost).
   For ex - 1 2 3 N 4 5 N 6 N 7 N 8 N 9 N N N N N will give 8 2 1 3 as answer.
    Here 8 and 9 are on the same position but 9 will get shadowed.

   Example 1:

   Input:
      1
    /    \
   2      3
   Output: 2 1 3
   Example 2:

   Input:
        10
      /      \
    20        30
   /   \    /    \
   40   60  90    100
   Output: 40 20 10 30 100
*/
```

Codeium: Refactor | Explain

```
class Pair{
    int hd;
    Node node;
    Pair(int hd, Node node){
        this.hd = hd;
        this.node = node;
    }
}
```

Codeium: Refactor | Explain

```
42  class Node{
43      int data;
44      Node left;
45      Node right;
46      Node(int key){
47          this.data = key;
48      }
49  }
50
```

```java
Codeium: Refactor | Explain
public class Top_View_of_Binary_Tree {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static ArrayList<Integer> topView(Node root)    {
        ArrayList<Integer> ans = new ArrayList<>();
        if(root == null) return ans;
        Map<Integer, Integer> map = new TreeMap<>();
        Queue<Pair> q = new LinkedList<>();
        q.add(new Pair(0, root));
        while(!q.isEmpty()){
            Pair it = q.remove();
            int hd = it.hd;
            Node temp = it.node;
            if(map.get(hd) == null){
                map.put(hd, temp.data);
            }
            if(temp.left != null){
                q.add(new Pair(hd - 1, temp.left));
            }
            if(temp.right != null){
                q.add(new Pair(hd + 1, temp.right));
            }
        }
        for(Map.Entry<Integer, Integer> entry: map.entrySet()){
            ans.add(entry.getValue());
        }
        return ans;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
```

```
Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        /*
            Real World Representation the below tree into the code.
                    1
                  /  \
                2     3
               / \   / \
              4   5 6   7
                /     / \
               8     9    10
        */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.left.right.left = new Node(8);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        root.right.right.left = new Node(9);
        root.right.right.right = new Node(10);
        System.out.println(topView(root));
    }
}

```

```
> Task :Top_View_of_Binary_Tree.main()
[4, 2, 1, 3, 7, 10]
```

Vertical Order Traversal of a Binary Tree

```
/*
987. Vertical Order Traversal of a Binary Tree
Given the root of a binary tree, calculate the vertical order traversal of the binary tree.

For each node at position (row, col), its left and right children will be at positions (row + 1, col - 1) and
(row + 1, col + 1) respectively. The root of the tree is at (0, 0).

The vertical order traversal of a binary tree is a list of top-to-bottom orderings for each
column index starting from the leftmost column and ending on the rightmost column. There may be multiple nodes in the
same row and same column.
 In such a case, sort these nodes by their values.

Return the vertical order traversal of the binary tree.

Example 1:
Input: root = [3,9,20,null,null,15,7]
Output: [[9],[3,15],[20],[7]]
Explanation:
Column -1: Only node 9 is in this column.
Column 0: Nodes 3 and 15 are in this column in that order from top to bottom.
Column 1: Only node 20 is in this column.
Column 2: Only node 7 is in this column.

Example 2:
Input: root = [1,2,3,4,5,6,7]
Output: [[4],[2],[1,5,6],[3],[7]]
Explanation:
Column -2: Only node 4 is in this column.
Column -1: Only node 2 is in this column.
Column 0: Nodes 1, 5, and 6 are in this column.
        1 is at the top, so it comes first.
        5 and 6 are at the same position (2, 0), so we order them by their value, 5 before 6.
Column 1: Only node 3 is in this column.
Column 2: Only node 7 is in this column.

Example 3:
Input: root = [1,2,3,4,6,5,7]
Output: [[4],[2],[1,5,6],[3],[7]]
Explanation:
This case is the exact same as example 2, but with nodes 5 and 6 swapped.
Note that the solution remains the same since 5 and 6 are in the same location and should be ordered by their values.
*/
```

Codeium: Refactor | Explain

```
class Node{
    int data;
    Node left;
    Node right;
    Node(int key){
        this.data = key;
    }
}
```

Codeium: Refactor | Explain

```
class Tuple{
    Node node;
    int row;
    int col;
    public Tuple(Node node, int row, int col){
        this.node = node;
        this.row = row;
        this.col = col;
    }
}
```

```
Codeium: Refactor | Explain
71  public class Vertical_Order_Traversal_of_Binary_Tree {
        Codeium: Refactor | Explain | Generate Javadoc | ×
72      public static List<List<Integer>> verticalTraversal(Node root) {
73          TreeMap<Integer, TreeMap<Integer, PriorityQueue<Integer>>> map = new TreeMap<>();
74          Queue<Tuple> q = new LinkedList<>();
75          q.add(new Tuple(root, 0, 0));
76          while(!q.isEmpty()){
77              Tuple tuple = q.remove();
78              Node node = tuple.node;
79              int x = tuple.row;
80              int y = tuple.col;
81
82              if(!map.containsKey(x)){
83                  map.put(x, new TreeMap<>());
84              }
85              if(!map.get(x).containsKey(y)){
86                  map.get(x).put(y, new PriorityQueue<>());
87              }
88              map.get(x).get(y).add(node.data);
89
90              if(node.left != null){
91                  q.add(new Tuple(node.left, x - 1, y + 1));
92              }
93              if(node.right != null){
94                  q.add(new Tuple(node.right, x + 1, y + 1));
95              }
96          }
97          List<List<Integer>> list = new ArrayList<>();
98          for(TreeMap<Integer, PriorityQueue<Integer>> ys : map.values()){
99              list.add(new ArrayList<>());
100             for(PriorityQueue<Integer> nodes : ys.values()){
101                 while(!nodes.isEmpty()){
102                     list.get(list.size() - 1).add(nodes.remove());
103                 }
104             }
105         }
106         return list;
107     }
        Codeium: Refactor | Explain | Generate Javadoc | ×
```

```java
    public static void main(String[] args) {
        /*
            Real World Representation the below tree into the code.
                    1
                  /  \
                 2     3
                / \   / \
               4   5 6   7
                  /     / \
                 8     9   10
        */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.left.right.left = new Node(8);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        root.right.right.left = new Node(9);
        root.right.right.right = new Node(10);
        System.out.println(verticalTraversal(root));
    }
}
```

```
> Task :Vertical_Order_Traversal_of_Binary_Tree.main()
[[4], [2, 8], [1, 5, 6], [3, 9], [7], [10]]
```

Binary Tree Zigzag Level Order Traversal

```java
/*
    103. Binary Tree Zigzag Level Order Traversal
    Given the root of a binary tree, return the zigzag level order traversal of its nodes' values.
    (i.e., from left to right, then right to left for the next level and alternate between).

    Example 1:
    Input: root = [3,9,20,null,null,15,7]
    Output: [[3],[20,9],[15,7]]

    Example 2:
    Input: root = [1]
    Output: [[1]]

    Example 3:
    Input: root = []
    Output: []
*/
```

```java
Codeium: Refactor | Explain
public class Zig_Zag_Traversal_of_Binary_Tree {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static List<List<Integer>> zigzagLevelOrder(Node root) {
        List<List<Integer>> res = new ArrayList<>();
        Queue<Node> q = new LinkedList<>();
        if(root == null) return res;
        q.add(root);
        boolean flag = false;
        while(!q.isEmpty()){
            int k = q.size();
            List<Integer> sublist = new ArrayList<>();
            for(int i = 0; i < k; i++){
                if(q.peek().left != null) q.add(q.peek().left);
                if(q.peek().right != null) q.add(q.peek().right);
                if(flag){
                    // Adding in reverse order by Adding at starting
                    sublist.add(0, q.remove().data);
                }
                else{
                    sublist.add(q.remove().data);
                }
            }
            flag = !flag;
            res.add(sublist);
        }
        return res;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        /*
            Real World Representation the below tree into the code
                    1
                  /  \
                 2     3
                / \   / \
               4   5 6   7
                 /     / \
                8     9    10
        */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.left.right.left = new Node(8);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        root.right.right.left = new Node(9);
        root.right.right.right = new Node(10);
        System.out.println(zigzagLevelOrder(root));
    }
}

```

```
> Task :Zig_Zag_Traversal_of_Binary_Tree.main()
[[1], [3, 2], [4, 5, 6, 7], [10, 9, 8]]
```