Binary Tree

Ex.1

```
package A6BinaryTree;
import java.util.*;
  //Build tree. Data is given using  PreOrder sequence.
public class A1BinaryTreesYT {
        static class Node{  //this is node class which represents single node of each 
tree
             int data;
             Node left;
             Node right;
             Node(int data){   //constructor of Node class
                    this.data=data;
                    this.left=null;  //at starting left and right child of node is 
null
                    this.right=null;
             }
        }
        
        static class BinaryTree{
             static int idx =-1;  //this is for traversing on each point of array
             public static Node buildTree(int nodes[]) {  //this function returns 
root of a tree
                    idx++;
                    if(nodes[idx]==-1) {
                          return null;
                    }
                    
                    Node newNode = new Node(nodes[idx]);
                    newNode.left = buildTree(nodes);  //we create left side of tree
                    newNode.right = buildTree(nodes);  // we create right side of 
tree
                    return newNode;  //this is a root of a tree
             }
        }
        
        //preorder
        //time complexity: O(n) because we traverse on each node
        public static void preorder(Node root) {
             if(root == null) {
                    return;
             }
             System.out.print(root.data+" ");   //Ans: 1 2 4 5 3 6   //this is called 
preorder because root comes at a start(pre)
             preorder(root.left);
             preorder(root.right);
        }
        
        //inorder
        //time complexity: O(n) because we traverse on each node
        public static void inorder(Node root) {
             if(root == null) {
                    return;
             }
             inorder(root.left);
```

```
       System.out.print(root.data+" "); //Ans: 4 2 5 1 3 6    //this is called
inorder because root comes at middle
       inorder(root.right);
    }

    //postorder
    //time complexity: O(n) because we traverse on each node
    public static void postorder(Node root) {
       if(root==null) {
          return;
       }

       postorder(root.left);
       postorder(root.right);
       System.out.print(root.data+" ");  //Ans: 4 5 2 6 3 1  //this is called
postorder because root comes at last(post)

    }

    //levelOrder
    //here we not use recursion
    public static void levelOrder(Node root) {
        if(root==null) {
           return;
        }
       Queue<Node> q = new LinkedList<>();
       q.add(root);
       q.add(null);

       while(!q.isEmpty()) {
          Node currNode = q.remove();
          if(currNode==null) {
             System.out.println();
             if(q.isEmpty()) {
                break;
             }else {
                q.add(null);
             }
          }else {
             System.out.print(currNode.data+" ");
             if(currNode.left!=null) {
                q.add(currNode.left);
             }
             if(currNode.right!=null) {
                q.add(currNode.right);
             }
          }
       }
    }

    //My ans:
   //time complexity: O(n)
    public static int countOfNodes(Node root,int totalNodes) {
        if(root==null) {
           return 0;
```

```
               }
               totalNodes = countOfNodes(root.left,totalNodes+1) + 
countOfNodes(root.right,totalNodes+1) + 1;
             
             return totalNodes;
        }
        
        //Apna clg ans:
        //time complexity: O(n)
        public static int countNode(Node root) {
             if(root==null) { //base case
                    return 0;
             }
             int leftNodes = countNode(root.left);
             int rightNodes = countNode(root.right);
             
             return leftNodes + rightNodes + 1;
        }
        
        //time complexity: O(n)
        public static int sumOfNodesValue(Node root) {
             if(root==null) {
                    return 0;
             }
             
             int leftNodeSum = sumOfNodesValue(root.left);
             int rightNodeSum = sumOfNodesValue(root.right);
             
             return leftNodeSum + rightNodeSum + root.data;
        }
        
        //Height of tree
        //time complexity: O(n)
        public static int height(Node root) {
             if(root==null) {
                    return 0;
             }
             
             int leftSubTreeHeight = height(root.left);
             int rightsubTreeHeight = height(root.right);
             //My ans:
        /*   if(leftSubTreeHeight > rightsubTreeHeight) {
                    return leftSubTreeHeight + 1;
             }else {
                    return rightsubTreeHeight+1;
             } */
             //or Apna clg ans
             int myHeight = Math.max(leftSubTreeHeight, rightsubTreeHeight) + 1;
             return myHeight;
        }
        
        //Diameter of a tree
        //time complexity: O(n^2)
        public static int diameter(Node root) {
             if(root==null) {
                    return 0;
             }
             int leftsubtreeDiameter = diameter(root.left);
             int rightsubtreeDiameter = diameter(root.right);  //  ....here O(n)

             int DiameterwithRoot = height(root.left) + height(root.right) +1;  //
... here O(n) so total complexity O(n^2)

             return Math.max(Math.max(leftsubtreeDiameter, rightsubtreeDiameter),
DiameterwithRoot);
        }

        //Diameter with time complexity O(n)
          static class TreeInfo {
               int ht;
               int diam;

               TreeInfo(int ht, int diam){
                      this.ht = ht;
                      this.diam = diam;
               }
          }


          //Diameter with time complexity O(n)  here we not call function separately
for calculating height therefore O(n)
        public static TreeInfo diameterWithGoodComplexity(Node root) {

             if(root==null) {
                    return new TreeInfo(0,0);
             }

             TreeInfo left = diameterWithGoodComplexity(root.left);
             TreeInfo right = diameterWithGoodComplexity(root.right);

             int myHeight = Math.max(left.ht, right.ht) + 1;
             int diam1 = left.diam;
             int diam2 = right.diam;
             int diam3 = left.ht + right.ht + 1;

             int myDiam = Math.max(diam1, Math.max(diam2, diam3));

              TreeInfo myInfo = new TreeInfo(myHeight,myDiam);
             return myInfo;
        }

       public static void main(String[] args) {
             int nodes[] = {1,2,4,-1,-1,5,-1,-1,3,-1,6,-1,-1};
        BinaryTree tree = new BinaryTree();
       Node root =  tree.buildTree(nodes);
       System.out.println("Root: "+root.data);  //Ans: 1
       preorder(root);   //Ans: 1 2 4 5 3 6   //this is called preorder because root
comes at a start(pre)
       System.out.println();
       inorder(root);  //Ans: 4 2 5 1 3 6    //this is called inorder because root
comes at middle
       System.out.println();
       postorder(root);  //Ans: 4 5 2 6 3 1  //this is called postorder because root
comes at last(post)

       System.out.println();
       levelOrder(root);     //Ans
                       /*1
                         2 3
                         4 5 6  */
       System.out.println();
       System.out.println("Total Nodes: "+countOfNodes(root,0));//Ans : 6

       System.out.println("Total Nodes Apna clg ans :"+countNode(root)); //Ans: 6
       System.out.println("Sum of Nodes Value :"+sumOfNodesValue(root)); //Ans: 21
       System.out.println("Height of tree: "+height(root));//Ans: 3
       System.out.println("Diameter of Tree with complexity O(n^2):
"+diameter(root));  //Ans: 5
       System.out.println("Diameter of tree with complexity O(n):
"+diameterWithGoodComplexity(root).diam);
       }

}
```

Ex.2

```
package A6BinaryTree;
  //LeetCode Question: Subtree of another tree
  //Que. check given subtree is  present in another tree
public class A2SubTreeOfAnotherTree {
      /**
       * Definition for a binary tree node.
       * public class TreeNode {
       *     int val;
       *     TreeNode left;
       *     TreeNode right;
       *     TreeNode() {}
       *     TreeNode(int val) { this.val = val; }
       *     TreeNode(int val, TreeNode left, TreeNode right) {
       *         this.val = val;
       *         this.left = left;
       *         this.right = right;
       *     }
       * }
       */

      /* class Solution {

                public boolean isIdentical(TreeNode root, TreeNode subRoot){
                       if(root==null && subRoot==null){
                         return true;
                       }
                       if(root==null || subRoot==null){
                         return false;
                       }
                       if(root.val==subRoot.val){
                     return isIdentical(root.left, subRoot.left) &&
isIdentical(root.right, subRoot.right);
                       }
                        return false;


                }

                 public boolean isSubtree(TreeNode root, TreeNode subRoot) {
                       if(subRoot==null){
                         return true;
                     }

                     if(root == null){
                         return false;
                     }

                     if(root.val == subRoot.val){
                         if(isIdentical(root,subRoot)){
                             return true;
                         }

                     }
                       return  isSubtree(root.left,subRoot) ||
isSubtree(root.right,subRoot);


                 }
             } */
}
```

Ex.3

```
package A6BinaryTree;
import java.util.*;
//Apna clg Homework question
//Que. Find the some of nodes at  kth level of binary tree
public class A3SumOfNodesAtKthLevel {
      static class Node{
            int data;
            Node left;
            Node right;
            Node(int data){
                  this.data=data;
                  this.left=null;
                  this.right=null;
            }
      }
       static class BinaryTree{
```

```
  Node root;
  static int idx = -1;
  public static Node buildTree(int nodes[]) {
         idx++;
         if(nodes[idx]==-1) {
                   return null;
             }
         Node newNode = new Node(nodes[idx]);
           newNode.left = buildTree(nodes);
           newNode.right = buildTree(nodes);

         return newNode;  //here we return root of Node
  }

}

public static int sumofNodesatKthlevel(Node root, int level) {
  int sum=0;
  int count=1;
   Queue<Node> q = new LinkedList<>();
   q.add(root);
   q.add(null);

   while(!q.isEmpty()) {
      Node currNode = q.remove();
       if(count==level ) {
             if(currNode!=null) {
                     sum = sum + currNode.data;
              }

       }
      if(currNode == null) {
            if(q.isEmpty()) {
                   break;
            }else {
                   q.add(null);
                   count++;
            }

      }else {
            if(root.left!=null) {
                   q.add(root.left);
            }
            if(root.right!=null) {
                   q.add(root.right);
            }

      }
   }
  return sum;
}

public static void main(String[] args) {
      int nodes[] = {1,2,4,-1,-1,5,-1,-1,3,-1,6,-1,-1};
  BinaryTree tree = new BinaryTree();
```

```
         Node root = tree.buildTree(nodes);
         System.out.println("Root: "+root.data); //Ans:1
          System.out.println("Sum of nodes at kth level 
:"+sumofNodesatKthLevel(root,2));
       }

}
```