CS162 ASSIGNMENT 11

NAME:

ARCHIT AGRAWAL

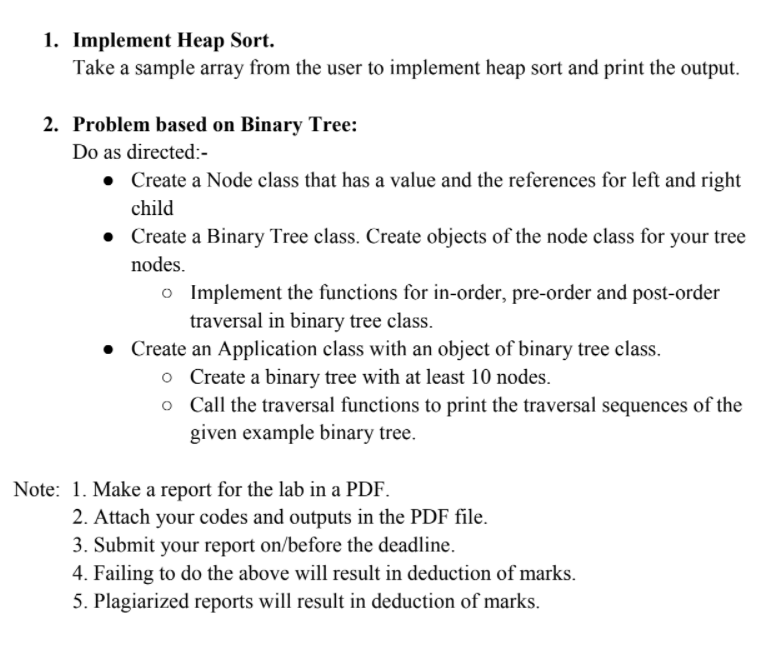
ROLL NO. :

202052307

SECTION:

A

**Question**



***CODE***

//Archit Agrawal

//202052307

//This code contains the code for Heap sort

//and implementation of Binary tree with pre-order, in-order, post-order

//traversal methods

import java.util.\*;

class HeapSort{

    void heapify(Integer []arr, int n, int index){

        int largest = index;

        int l = 2 \* index + 1;

        int r = 2 \* index + 2;

        if(l < n && arr[l] > arr[largest]){

            largest = l;

        }

        if(r < n && arr[r] > arr[largest]){

            largest = r;

        }

        if(largest != index){

            int temp = arr[largest];

            arr[largest] = arr[index];

            arr[index] = temp;

            heapify(arr, n, largest);

        }

    }

    void sort(Integer []arr){

        int n = arr.length;

        for(int i = n/2 - 1; i >= 0; i--){

            heapify(arr, n, i);

        }

        for(int i = n - 1; i >= 0; i--){

            int temp = arr[0];

            arr[0] = arr[i];

            arr[i] = temp;

            heapify(arr, i, 0);

        }

    }

}

class Node<T>{

    T data;

    Node<T>left;

    Node<T> right;

    public Node(T key){

        this.data = key;

        this.left = null;

        this.right = null;

    }

}

class BinaryTree<T>{

    Node<T> root;

    public BinaryTree(){

        root = null;

    }

    public BinaryTree(T key){

        root = new Node(key);

    }

    public Node<T> constructLevelOrder(T []arr, Node root, int index){

        if(index < arr.length){

            Node<T> temp =  new Node<T>(arr[index]);

            root = temp;

            root.left = constructLevelOrder(arr, root.left, 2 \* index + 1);

            root.right = constructLevelOrder(arr, root.right, 2 \* index + 2);

        }

        return root;

    }

    public void preOrderTraversal(Node node){

        if(node == null) return;

        //pre-order is node-left-right(NLR) traversal

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

        preOrderTraversal(node.left);

        preOrderTraversal(node.right);

    }

    public void inOrderTraversal(Node node){

        if(node == null) return;

        //in-order is left-node-right(LNR) traversal

        inOrderTraversal(node.left);

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

        inOrderTraversal(node.right);

    }

    public void postOrderTraversal(Node node){

        if(node == null) return;

        //post-order is left-right-node(LRN) traversal

        postOrderTraversal(node.left);

        postOrderTraversal(node.right);

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

    }

    public void preOrderTraversal(){

        preOrderTraversal(root);

    }

    public void inOrderTraversal(){

        inOrderTraversal(root);

    }

    public void postOrderTraversal(){

        postOrderTraversal(root);

    }

}

public class Application{

    public static void main(String []args){

        //IMPLEMENTING HEAP SORT

        Scanner sc = new Scanner(System.in);

        System.out.println("Implementation of Heap Sort");

        System.out.print("Enter the size of array to be Heap Sorted : ");

        int n = sc.nextInt();

        System.out.println();

        System.out.println("The original array :");

        Integer[] a = new Integer[n];

        Random rand = new Random();

        for (int i = 0; i < n; i++) {

            a[i] = rand.nextInt(9000) + 1000;

        }

        for(Integer i: a){

            System.out.print(i + " ");

        }

        HeapSort hs = new HeapSort();

        hs.sort(a);

        System.out.println();

        System.out.println("The Sorted Array is :");

        for(Integer i : a){

            System.out.print(i + " ");

        }

        System.out.println();

        System.out.println();

        System.out.println("Implementing binary tree and pre-order, in-order, post-order traversal methods on it.");

        //IMPLEMENTION BINARY TREE AND PRE-ORDER, IN-ORDER, POST-ORDER TRAVERSALS

        BinaryTree<Integer> tree = new BinaryTree<Integer>();

        System.out.println();

        System.out.println("A tree is made using level ordering insertion from the array {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}");

        //this is a BinaryTree object made using the constructLevelOrder method

        Integer [] arr = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};

        tree.root = tree.constructLevelOrder(arr, tree.root, 0);

        System.out.println();

        System.out.println();

        System.out.println("Pre-Order Traversal of the Tree is:");

        tree.preOrderTraversal();

        System.out.println();

        System.out.println();

        System.out.println("In-Order Traversal of the Tree is:");

        tree.inOrderTraversal();

        System.out.println();

        System.out.println();

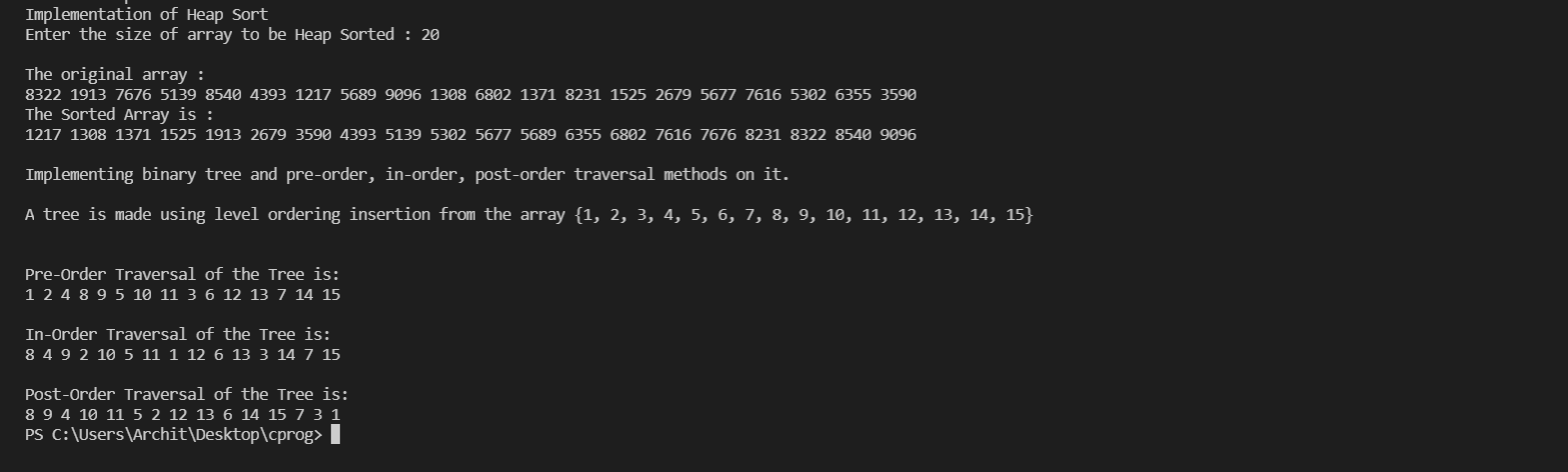
        System.out.println("Post-Order Traversal of the Tree is:");

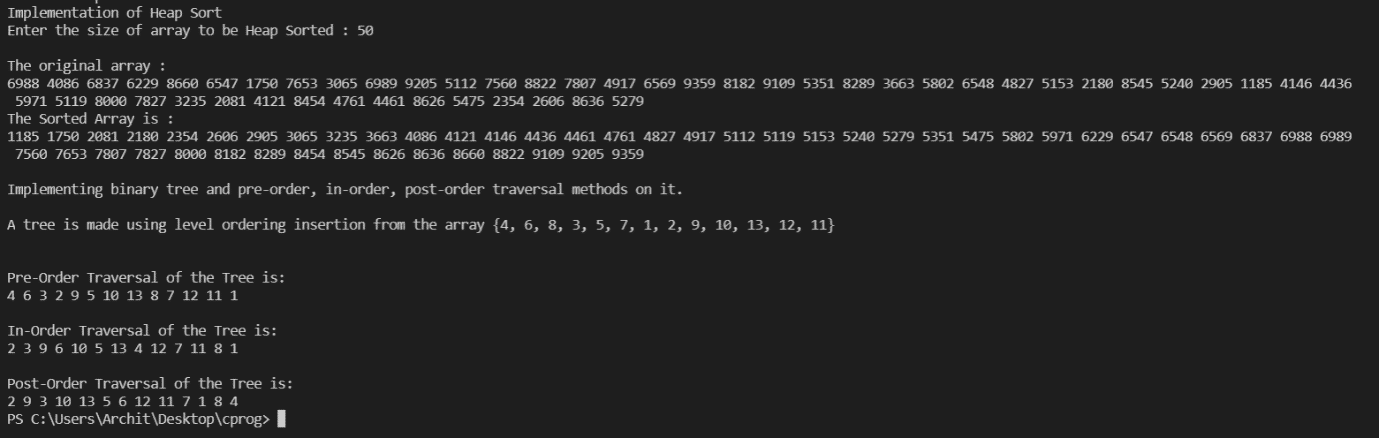
        tree.postOrderTraversal();

    }

}

***OUTPUT***

***Output 1:***

***Output 2:***