1. Binary Tree operations

import java.util.\*;

class Node{

    public int data;

    Node left;

    Node right;

    public Node(int val){

        this.data=val;

        this.left=null;

        this.right=null;

    }

}

class Pair {

        int data;

        int depth;

        Pair(int data, int hd) {

            this.data = data;

            this.depth = depth;

        }

    }

class BinaryTree{

    Node root;

    BinaryTree(){

        root=null;

    }

    void construct(ArrayList<Integer>arr){

        if (arr.isEmpty()) return;

        root=new Node(arr.get(0));

        Queue<Node> q=new LinkedList<>();

        int i=1;

        q.add(root);

        while(!q.isEmpty() && i < arr.size()){

            Node temp=q.poll();

            if(i<arr.size() && arr.get(i) != -1){

            temp.left=new Node(arr.get(i));

            q.add(temp.left);

            }

            i++;

            if(arr.size()>i && arr.get(i) != -1){

            temp.right=new Node(arr.get(i));

            q.add(temp.right);

            }

            i++;

        }

    }

    void inorder(Node root){

        if(root==null)return;

        inorder(root.left);

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

        inorder(root.right);

    }

    void preorder(Node root){

        if(root== null)return;

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

        preorder(root.left);

        preorder(root.right);

    }

    void leftview(BinaryTree tree){

        Queue<Node> q=new LinkedList<>();

        q.add(tree.root);

        while(! q.isEmpty()){

            int n=q.size();

            for(int i=0;i<n;i++){

                Node temp=q.poll();

                if(i==0)System.out.print(temp.data+" ");

                if(temp.left!=null) q.add(temp.left);

                if(temp.right!=null) q.add(temp.right);

            }

        }

    }

    void rightview(BinaryTree tree){

        if(tree==null) return;

        Queue<Node> q=new LinkedList<>();

        q.add(tree.root);

        while(! q.isEmpty()){

            int n=q.size();

            for(int i=0;i<n;i++){

                Node temp=q.poll();

                if(i==n-1) System.out.print(temp.data+" ");

                if(temp.left!=null) q.add(temp.left);

                if(temp.right != null) q.add(temp.right);

            }

        }

    }

        void topview(BinaryTree tree){

            if(tree==null)return;

            ArrayList<Integer> left=new ArrayList<>();

            ArrayList<Integer> right= new ArrayList<>();

            Queue<Node> q1=new LinkedList<>();

            q1.add(tree.root);

            left.add(tree.root.data);

            while(!q1.isEmpty()){

             int n=q1.size();

             for(int i=0;i<n;i++){

                   Node temp=q1.poll();

                   Boolean l=true;

                   Boolean r=true;

                   if(i==0 && temp.left!=null){

                    left.add(temp.left.data);

                    }

                   if(i==n-1 && temp.right!= null){

                    right.add(temp.right.data);

                    r=false;

                   }

                   if(temp.left != null) q1.add(temp.left);

                   if(temp.right !=null) q1.add(temp.right);

             }

            }

            for(int i=left.size()-1;i>=0;i--){

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

            }

            for(int i=0;i<right.size();i++) System.out.print(right.get(i)+" ");

        }

        void dfs(Node root,int hd,int depth, Map<Integer,Pair>hdMap){

            if(root==null) return;

            if (!hdMap.containsKey(hd) || depth>=hdMap.get(hd).depth){

                 hdMap.put(hd,new Pair(root.data,depth));

            }

            dfs(root.left,hd-1,depth+1,hdMap);

            dfs(root.right,hd+1,depth+1,hdMap);

        }

        void bottomView(Node root){

            Map<Integer,Pair> hdMap= new TreeMap<>();

            dfs(root,0,0,hdMap);

            for(Pair val: hdMap.values()){

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

            }

        }

    public static void main(String args[]){

        ArrayList<Integer> arr=new ArrayList<>();

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the number of elements in the array");

        int n=sc.nextInt();

        System.out.println("Enter the elements of the array");

        for(int i=0;i<n;i++){

            arr.add(sc.nextInt());

        }

        BinaryTree tree=new BinaryTree();

        tree.construct(arr);

        System.out.println("Inorder:");

        tree.inorder(tree.root);

        System.out.println("\nPreorder");

        tree.preorder(tree.root);

        System.out.println("\nLeft view");

        tree.leftview(tree);

        System.out.println("\nRight View");

        tree.rightview(tree);

        System.out.println("\nTop View");

        tree.topview(tree);

        System.out.println("\nBotom View");

        tree.bottomView(tree.root);

    }

}

1. Binary Tree to Binary Search Tree

import java.util.\*;

class BST extends BinaryTree{

    void inordertraversal(Node root,ArrayList<Integer> nodes){

        if(root==null)return;

        inordertraversal(root.left,nodes);

        nodes.add(root.data);

        inordertraversal(root.right,nodes);

    }

    void printInorder(Node root){

        if(root==null) return;

        printInorder(root.left);

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

        printInorder(root.right);

    }

    void constructBST(Node root, ArrayList<Integer> nodes,int[] index){

        if(root==null)return;

        constructBST(root.left, nodes, index);

        root.data=nodes.get(index[0]);

        index[0]++;

        constructBST(root.right, nodes, index);

    }

    Node binaryToBST(Node root){

        ArrayList<Integer> nodes= new ArrayList<>();

        inordertraversal(root, nodes);

        Collections.sort(nodes);

        int[] index={0};

        constructBST(root, nodes, index);

        return root;

    }

    public static void main(String args[]){

        BST tree=new BST();

        ArrayList<Integer> arr=tree.inputcollection();

        tree.construct(arr);

        tree.printInorder(tree.root);

        System.out.println("\n|");

        Node root= tree.binaryToBST(tree.root);

        tree.printInorder(root);

    }

}