# SKILL-7(2100032102)[sec-13]

### Recover Binary Search Tree Code:

class Solution {

private TreeNode first; private TreeNode prev; private TreeNode middle; private TreeNode last;

public void recoverTree(TreeNode root) { first = null ;

middle = null; last = null;

prev = new TreeNode(Integer.MIN\_VALUE); inorder(root);

if(last != null){ int t = first.val;

first.val = last.val; last.val = t;

}else{

int t = first.val; first.val = middle.val; middle.val = t;

}

}

public void inorder(TreeNode root){ if(root == null) return; inorder(root.left);

if(root.val < prev.val){ if(first == null){

first = prev; middle = root;

}else{

last = root;

}

}

prev = root; inorder(root.right);

}

}

# OUTPUT



### Code:

1. All Elements in Binary Search Trees

class Solution {

public List < Integer > getAllElements(TreeNode root1, TreeNode root2) { Stack < TreeNode > s1 = new Stack();

while (root1 != null) { s1.push(root1); root1 = root1.left;

}

Stack < TreeNode > s2 = new Stack(); while (root2 != null) {

s2.push(root2); root2 = root2.left;

}

List < Integer > result = new ArrayList();

find(result, s1, s2); return result;

}

public void popStack(List < Integer > result, Stack < TreeNode > s) { TreeNode n = s.pop();

result.add(n.val); n = n.right;

while (n != null) { s.push(n);

n = n.left;

}

}

public void find(List < Integer > result, Stack < TreeNode > s1, Stack < TreeNode > s2) { while (!s1.isEmpty() || !s2.isEmpty()) {

if (!s1.isEmpty() && !s2.isEmpty()) {

if (s1.peek().val <= s2.peek().val) {

Else

{

popStack(result, s1);

}

popStack(result, s2);

}

} else if (!s1.isEmpty()) {

while (!s1.isEmpty()) { popStack(result, s1);

}

break;

} else {

while (!s2.isEmpty()) { popStack(result, s2);

}

break;

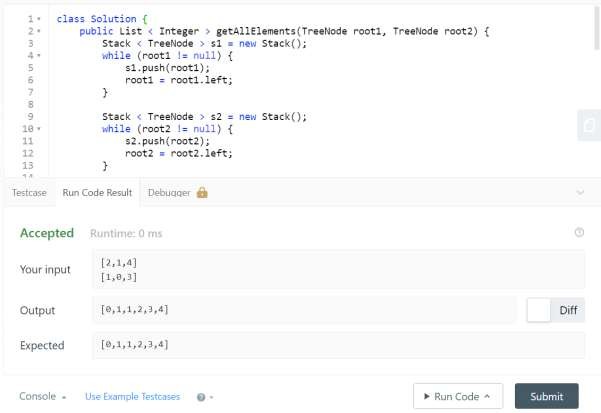
}

}

}

}

OUTPUT



1. **Find Leftmost and Rightmost nodes for a given node**

Code:

package pack1;

public class GFG {

static void LeftRightNode(int preorder[], int n)

{

int min = Integer.***MAX\_VALUE***, max = Integer.***MIN\_VALUE***;

for (int i = 0; i < n; i++)

{

if (min > preorder[i]) min = preorder[i];

if (max < preorder[i]) max = preorder[i];

}

System.***out***.println("Leftmost node is " + min); System.***out***.println("Rightmost node is " + max);

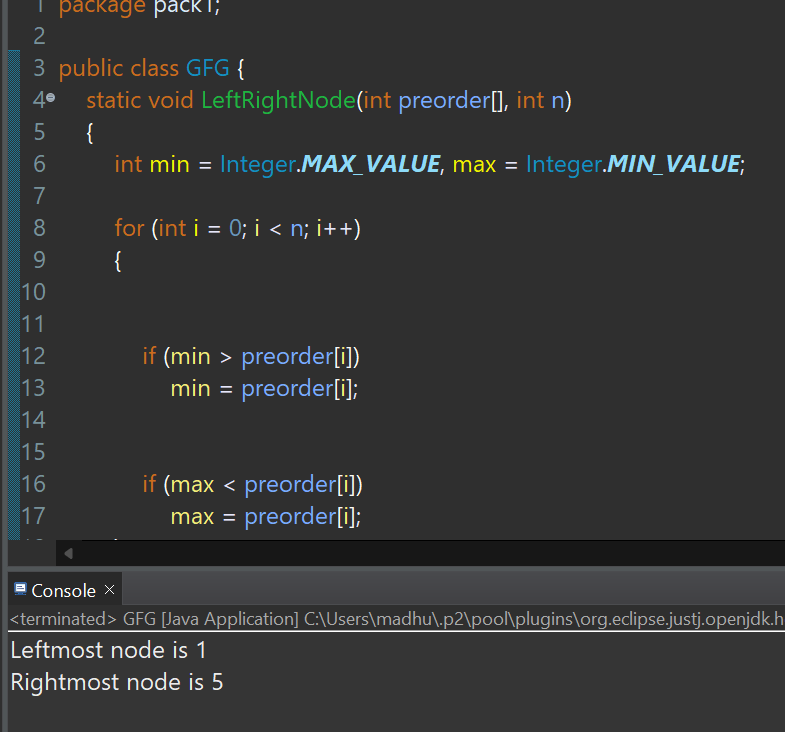
}

public static void main(String[] args)

{

int preorder[] = { 3, 2, 1, 5, 4 }; int n = 5; *LeftRightNode*(preorder, n);

## OUTPUT



1. Convert BST into Skewed tree

### Code:

package pack2; import java.io.\*; public class Node {

int val; Node left, right; Node(int item)

{

val = item;

left = right = null;

}

}

class GFG

{

public static Node *node*; static Node *prevNode* = null; static Node *headNode* = null;

static void flattenBTToSkewed(Node root, int order)

{

if(root == null)

{

return;

}

if(order > 0)

{

*flattenBTToSkewed*(root.right, order);

}

else

{

*flattenBTToSkewed*(root.left, order);

}

Node rightNode = root.right; Node leftNode = root.left;

if(*headNode* == null)

{

*headNode* = root; root.left = null; *prevNode* = root;

}

else

{

*prevNode*.right = root; root.left = null; *prevNode* = root;

}

### 2121

if (order > 0)

{

*flattenBTToSkewed*(leftNode, order);

}

else

{

*flattenBTToSkewed*(rightNode, order);

}

}

static void traverseRightSkewed(Node root)

{

if(root == null)

{

return;

}

System.***out***.print(root.val + " ");

*traverseRightSkewed*(root.right);

}

public static void main (String[] args)

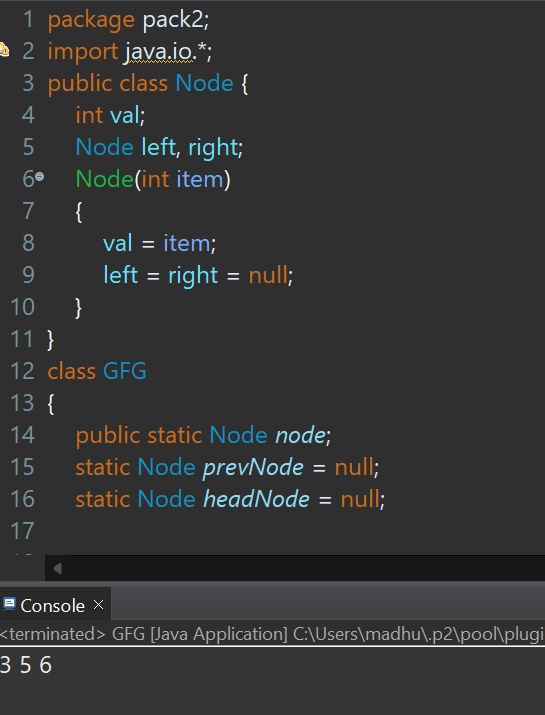
{

GFG tree = new GFG(); tree.*node* = new Node(5); tree.*node*.left = new Node(3); tree.*node*.right = new Node(6); int order = 0;

*flattenBTToSkewed*(*node*, order);

*traverseRightSkewed*(*headNode*);

## OUTPUT



}

}

}

}