2100032209

B.Sravani

Binary Search Tree problems

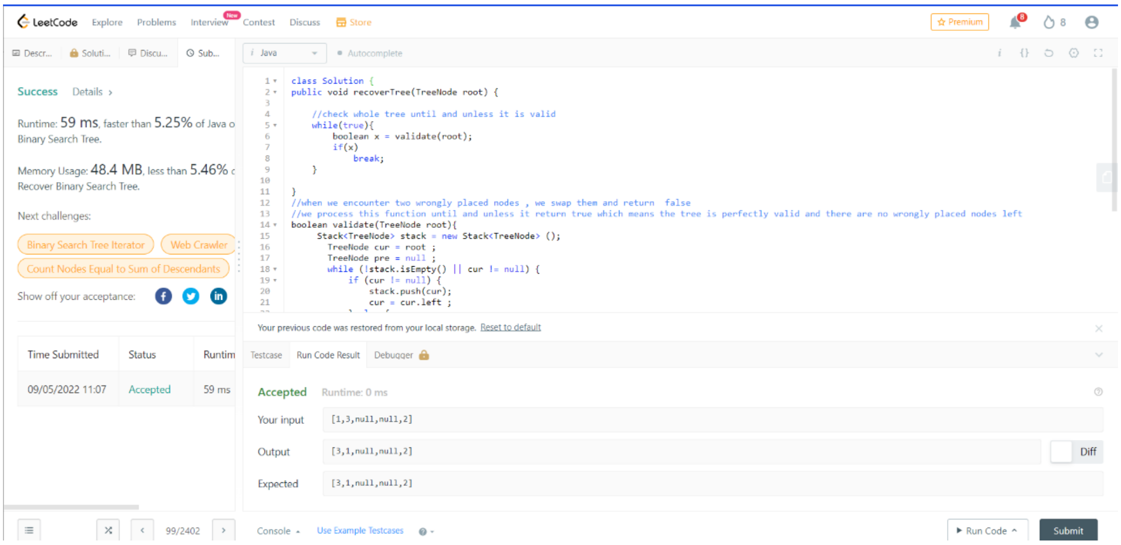
1.Recover Binary Search Tree

```
public class BinarySearchTree {
 class Node
 {
  int key;
  Node left,right;
  public Node(int item)
  {
   key=item;
   left=right=null;
  }
 }
 Node root;
 BinarySearchTree()
 {
  root=null;
 }
 BinarySearchTree(int value)
 {
  root=new Node(value);
 }
  void insert(int key)
  {
   root = insertRec(root, key);
```

```
}
  Node insertRec(Node root, int key)
  {
      if (root == null)
      {
          root = new Node(key);
          return root;
      }
      else if (key < root.key)
          root.left = insertRec(root.left, key);
      else if (key > root.key)
          root.right = insertRec(root.right, key);
      return root;
  }
  void inorder()

{
 inorderRec(root);


}
  void inorderRec(Node root)
  {
      if (root != null)
     {
          inorderRec(root.left);
          System.out.println(root.key);
          inorderRec(root.right);
```

```
        }
      }



  public static void main(String[] args) {


    BinarySearchTree tree = new BinarySearchTree();
        tree.insert(30);
        tree.insert(50);
        tree.insert(20);
        tree.insert(40);
        tree.insert(70);
        tree.inorder();



  }
}
```

OUTPUT::



Success Details ›

Runtime: 59 ms, faster than 5.25% of Java o
Binary Search Tree.

Memory Usage: 48.4 MB, less than 5.46% c
Recover Binary Search Tree.

Next challenges:

Binary Search Tree Iterator   Web Crawler
Count Nodes Equal to Sum of Descendants

Show off your acceptance:

Time Submitted	Status	Runtim
09/05/2022 11:07	Accepted	59 ms

```
class Solution {
public void recoverTree(TreeNode root) {

    //check whole tree until and unless it is valid
    while(true){
        boolean x = validate(root);
        if(x)
            break;
    }

}
//when we encounter two wrongly placed nodes , we swap them and return  false
//we process this function until and unless it return true which means the tree is perfectly valid and there are no wrongly placed nodes left
boolean validate(TreeNode root){
     Stack<TreeNode> stack = new Stack<TreeNode> ();
       TreeNode cur = root ;
       TreeNode pre = null ;
       while (!stack.isEmpty() || cur != null) {
           if (cur != null) {
               stack.push(cur);
               cur = cur.left ;
```

Your previous code was restored from your local storage. Reset to default

Accepted Runtime: 0 ms

Your input [1,3,null,null,2]

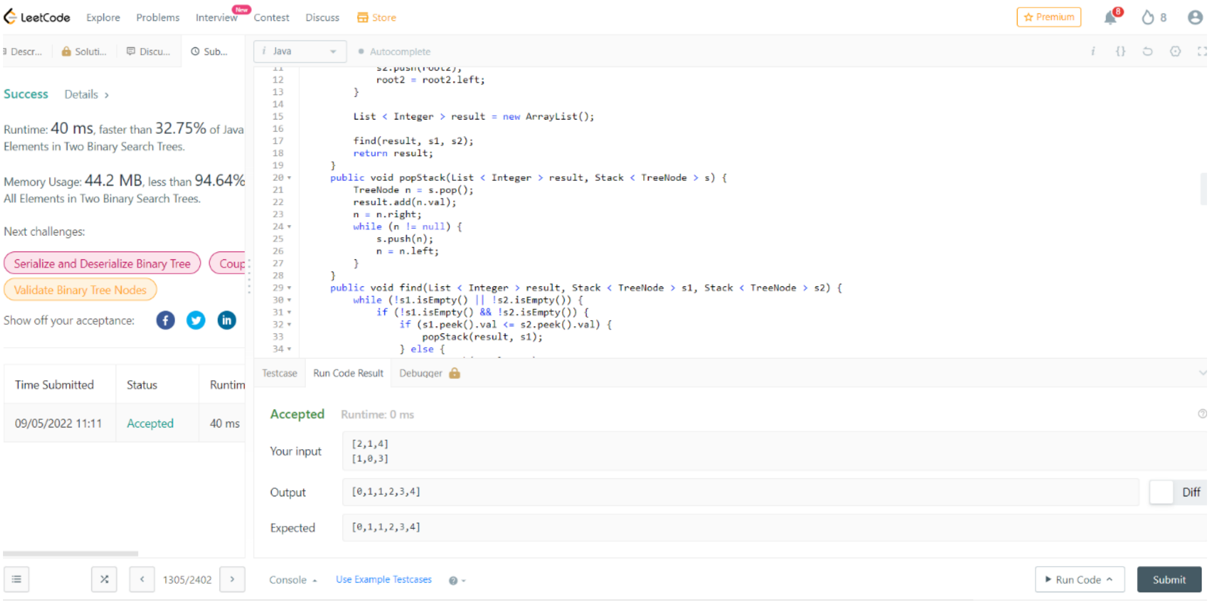
Output [3,1,null,null,2]

Expected [3,1,null,null,2]

2 . All Elements in Two 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) {
                    popStack(result, s1);
                } else {
                    popStack(result, s2);
                }
            } else if (!s1.isEmpty()) {
                while (!s1.isEmpty()) {
                    popStack(result, s1);
                }
                break;
            } else {
                while (!s2.isEmpty()) {
                    popStack(result, s2);
                }
                break;
            }
        }
    }
}
```

OUTPUT::



Success Details ›

Runtime: 40 ms, faster than 32.75% of Java
Elements in Two Binary Search Trees.

Memory Usage: 44.2 MB, less than 94.64%
All Elements in Two Binary Search Trees.

Next challenges:

Serialize and Deserialize Binary Tree
Coup
Validate Binary Tree Nodes

Show off your acceptance:

Time Submitted	Status	Runtim
09/05/2022 11:11	Accepted	40 ms

```
            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) {
                    popStack(result, s1);
                } else {
```

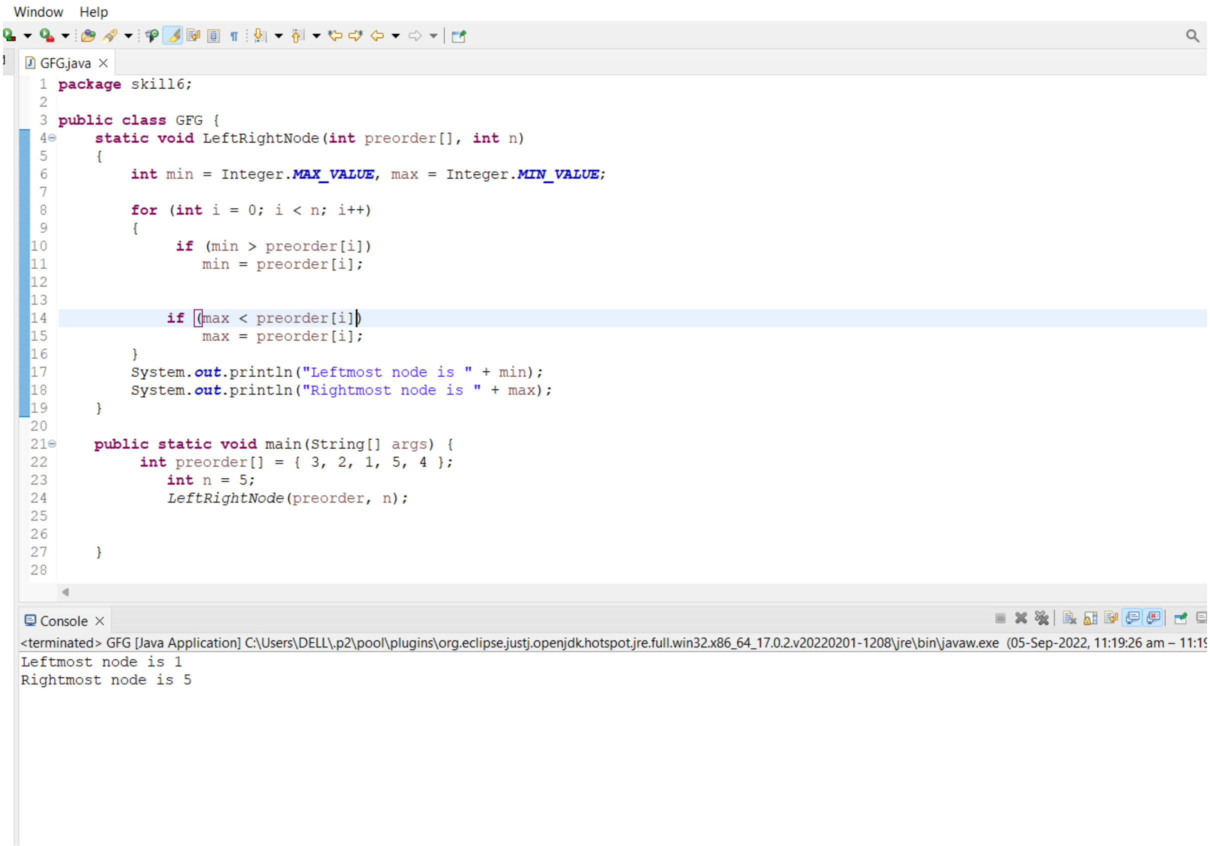
Accepted Runtime: 0 ms

```
Your input   [2,1,4]
             [1,0,3]
Output       [0,1,1,2,3,4]
Expected     [0,1,1,2,3,4]
```

3.Find Leftmost and Rightmost nodes for a given node:

```
package skill6;

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:



```
package skill6;

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);


    }

```

```
<terminated> GFG [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe  (05-Sep-2022, 11:19:26 am – 11:1
Leftmost node is 1
Rightmost node is 5
```

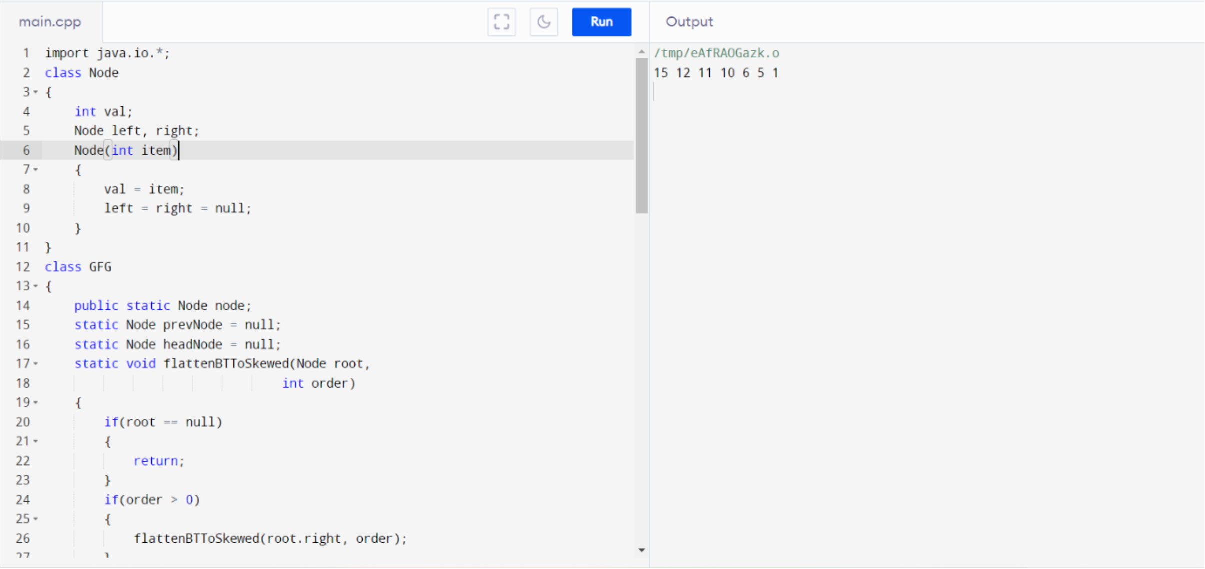
4. Convert BST into Skewed Tree:

```
import java.io.*;

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;

        }



        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::



```
import java.io.*;
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);
        }
```

```
/tmp/eAfRAOGazk.o
15 12 11 10 6 5 1
```