

Skilling 7

Name: P V Brahma

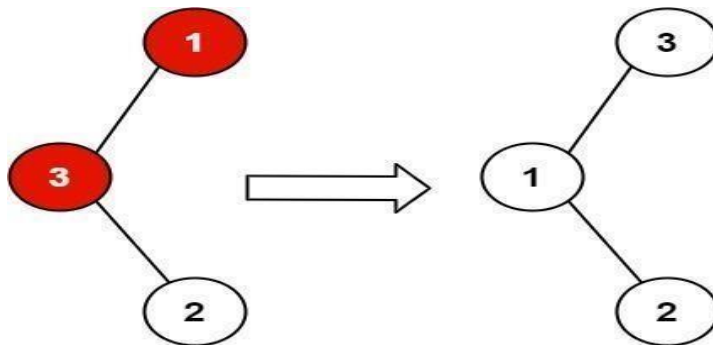
Id No : 2100030430

Binary Search Tree problems

1. Recover Binary Search Tree

You are given the **root** of a binary search tree (BST), where the values of **exactly** two nodes of the tree were swapped by mistake. *Recover the tree without changing its structure.*

Example 1:

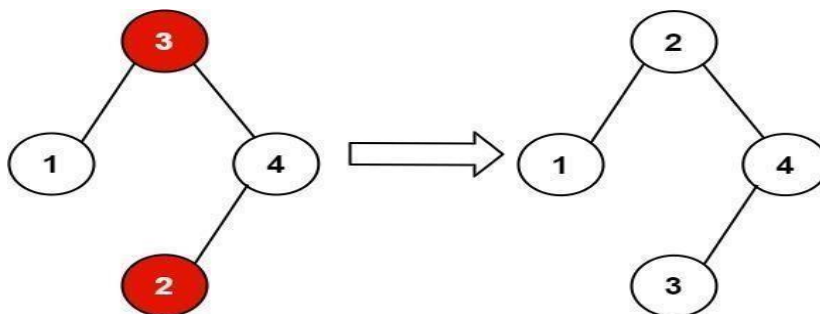


Input: root = [1,3,null,null,2]

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

Explanation: 3 cannot be a left child of 1 because $3 > 1$. Swapping 1 and 3 makes the BST valid.

Example 2:



Input: root = [3,1,4,null,null,2]

Output: [2,1,4,null,null,3]

Explanation: 2 cannot be in the right subtree of 3 because $2 < 3$. Swapping 2 and 3 makes the BST valid.

Constraints:

- The number of nodes in the tree is in the range $[2, 1000]$.
- $-2^{31} \leq \text{Node.val} \leq 2^{31} - 1$

SOLUTION

```
class Solution {  
  
    private TreeNode first;  
  
    private TreeNode second;  
  
    private TreeNode prev;  
  
    public void recoverTree(TreeNode root) {  
        if(root==null) return;  
  
        first = null;  
        second = null;  
        prev = null;  
  
        inorder(root);  
  
        int temp = first.val;  
        first.val = second.val;
```

```
second.val = temp;
```

```
}
```

```
private void inorder(TreeNode root){
```

```
if(root==null) return;
```

```
inorder(root.left);
```

```
if(first==null && (prev != null && prev.val>=root.val)){
```

```
first = prev;
```

```
}
```

```
if(first!=null && prev.val>=root.val){
```

```
second = root;
```

```
}
```

```
prev = root;
```

```
inorder(root.right);
```

```
}
```

```
}
```

LeetCode

99. Recover Binary Search Tree

Medium 6031 196 Add to List Share

You are given the `root` of a binary search tree (BST), where the values of **exactly** two nodes of the tree were swapped by mistake. Recover the tree without changing its structure.

Example 1:

```
graph TD
    subgraph "Initial Tree"
        1((1)) --> 3((3))
        1 --> 2((2))
    end
    subgraph "Recovered Tree"
        1r((1)) --> 2r((2))
        1r --> 3r((3))
    end
```

Input: `root = [1,3,null,null,2]`
Output: `[3,1,null,null,2]`
Explanation: 3 cannot be a left child of 1 because 3 > 1. Swapping 1 and 3 makes the BST valid.

```
1 * class Solution {
2
3     private TreeNode first;
4     private TreeNode second;
5     private TreeNode prev;
6
7 *     public void recoverTree(TreeNode root) {
8         if(root==null) return;
9
10        first = null;
11        second = null;
12        prev = null;
13
14        inorder(root);
15
16        int temp = first.val;
17        first.val = second.val;
18        second.val = temp;
19    }
20
21 *     private void inorder(TreeNode root){
22         if(root==null) return;
23         inorder(root.left);
24     }
25 }
```

Accepted Runtime: 0 ms

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

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

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

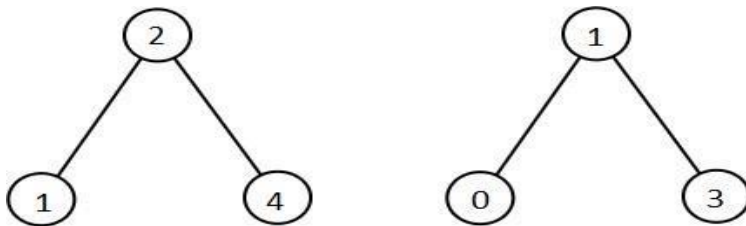
Run Code Result Debugger

Run Code Submit

2. All Elements in Two Binary Search Trees

Given two binary search trees `root1` and `root2`, return a list containing all the integers from both trees sorted in **ascending** order.

Example 1:



Input: `root1 = [2,1,4]`, `root2 = [1,0,3]`

Output: `[0,1,1,2,3,4]`

Example 2:



Input: `root1 = [1,null,8]`, `root2 = [8,1]`

Output: `[1,1,8,8]`

Constraints:

- The number of nodes in each tree is in the range `[0, 5000]`.
- $-10^5 \leq \text{Node.val} \leq 10^5$

SOLUTION

```
class Solution {
    public List<Integer> getAllElements(TreeNode root1, TreeNode root2) {

        List<Integer> list1 = new
ArrayList<Integer>();
        List<Integer> list2 = new ArrayList<Integer>();

        List<Integer> ans = new ArrayList<Integer>();
        inorder(root1,list1);
        inorder(root2
,list2);
        int i=0,j=0;
```

```

while(i<list1.size() && j<list2.size()){
    if(list1.get(i) <
list2.get(j)){

        ans.add(list1.get(i));

        i++;

    }

    else{

        ans.add(list2.get(j));

        j++;

    }

}
while(i<list1.size()){
    ans.add(list1.get(i));

    i++;

}

while(j<list2.size()){
    ans.add(list2.get(j));

    j++;

}

return ans;

public void inorder(TreeNode root, List<Integer> list){
if(root == null)return; inorder(root.left,list);
list.add(root.val);
    inorder(root.right,list);

```

}

1305. All Elements in Two Binary Search Trees

Medium 2461 71 Add to List Share

Given two binary search trees *root1* and *root2*, return a list containing all the integers from both trees sorted in ascending order.

Example 1:

Input: *root1* = [2,1,4], *root2* = [1,0,3]

Output: [0,1,1,2,3,4]

Example 2:

Input: *root1* = [1], *root2* = [8]

Output: [1,8]

```
1 * class Solution {
2 *     public List<Integer> getAllElements(TreeNode root1, TreeNode root2) {
3 *         List<Integer> list1 = new ArrayList<Integer>();
4 *         List<Integer> list2 = new ArrayList<Integer>();
5 *         List<Integer> ans = new ArrayList<Integer>();
6 *
7 *         inorder(root1, list1);
8 *         inorder(root2, list2);
9 *
10 *         int i=0, j=0;
11 *
12 *         while(i<list1.size() && j<list2.size()){
13 *             if(list1.get(i) < list2.get(j)){
14 *                 ans.add(list1.get(i));
15 *                 i++;
16 *             }
17 *             else{
18 *                 ans.add(list2.get(j));
19 *                 j++;
20 *             }
21 *         }
22 *         while(i<list1.size()){
23 *             ans.add(list1.get(i));
24 *             i++;
25 *         }
26 *         while(j<list2.size()){
27 *             ans.add(list2.get(j));
28 *             j++;
29 *         }
30 *         return ans;
31 *     }
32 * }
```

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:

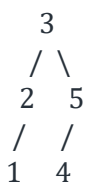
Given a preorder sequence of the binary search tree of **N** nodes. The task is to find its leftmost and rightmost nodes.

Examples:

Input : N = 5, preorder[] = { 3, 2, 1, 5, 4 }

Output : Leftmost = 1, Rightmost = 5

The BST constructed from this preorder sequence would be:



Leftmost Node of this tree is equal to 1

Rightmost Node of this tree is equal to 5

Input : N = 3 preorder[] = { 2, 1, 3 }

Output : Leftmost = 1, Rightmost = 3

SOLUTION

```
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];
    }

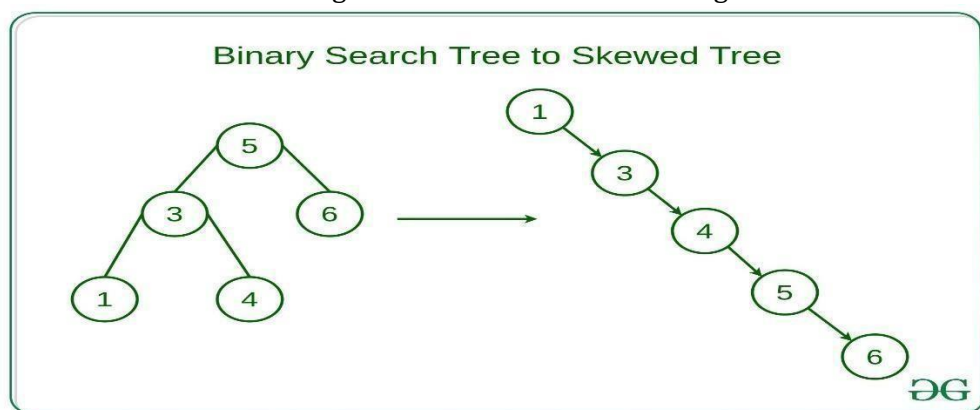
    System.out.println("Leftmost node is " + min);
    System.out.println("Rightmost node is " + max);
} public static void main(String[]
args)
{
int preorder[] = { 3, 2, 1, 5, 4 }; int n =
5;
    LeftRightNode(preorder, n);

}
}

```

4. Convert BST into Skewed Tree:

Given a Binary Search Tree and a binary integer **K**, the task is to convert Binary search tree into a Skewed Tree in increasing order if **K = 0** or in decreasing order if **K = 1**.



Examples:

Input: K = 0,

```

5
 /\
3 6
Output:
3
 \
 5
  \
   6

```

Input: K = 1,

```

2

```

```

    /\
    13 Output:
    3
    \
    2
    \
    1

```

SOLUTION

```

package p10;
import java.io.*;
public class Binarysearchtree {
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);
        }
    }
}

```

Reference links:

1. <https://leetcode.com/problems/recover-binary-search-tree/>
2. <https://leetcode.com/problems/all-elements-in-two-binary-search-trees/>
3. <https://www.geeksforgeeks.org/find-leftmost-and-rightmost-node-of-bst-from-its-given-preorder-traversal/>
4. <https://www.geeksforgeeks.org/convert-a-binary-search-tree-into-a-skewed-tree-in-increasing-or-decreasing-order/>