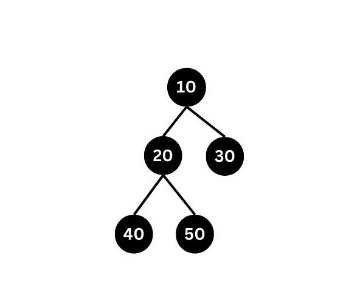
**Problem statement**

Send feedback

For a given Binary Tree of integers, find and return the sum of all the nodes data.

**Example:**



public class Solution {

public static int getSum(BinaryTreeNode<Integer> root) {

        //Your code goes here.

         if (root==null)

        {

            return 0;

        }

        int smallSum=getSum(root.left)+getSum(root.right);

        return root.data+smallSum;

    }

}

**Problem statement**

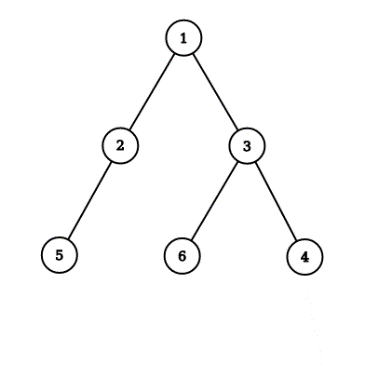
Send feedback

You are given a ***‘Binary Tree’***.

Return the preorder traversal of the Binary Tree.

**Example:**

Input: Consider the following Binary Tree:



Output:

Following is the preorder traversal of the given Binary Tree: [1, 2, 5, 3, 6, 4]

**Detailed explanation**( Input/output format, Notes, Images )

**Sample Input 1:**

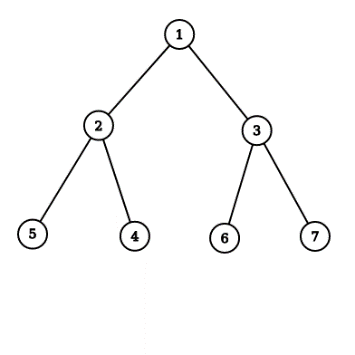
1 2 3 5 4 6 7 -1 -1 -1 -1 -1 -1 -1 -1

**Sample Output 1:**

1 2 5 4 3 6 7

**Explanation of Sample Input 1:**

The Binary Tree given in the input is as follows:



**Sample Input 2:**

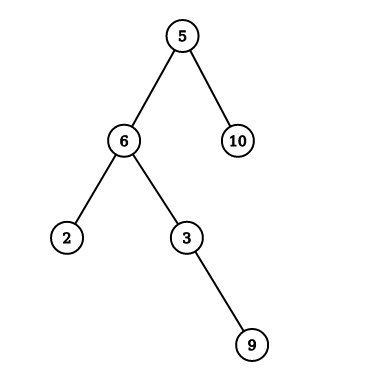
5 6 10 2 3 -1 -1 -1 -1 -1 9 -1 -1

**Sample Output 2:**

5 6 2 3 9 10

**Explanation of Sample Input 2:**

The Binary Tree given in the input is as follows:



public class Solution {

    public static void preOrder(BinaryTreeNode<Integer> root) {

        //Your code goes here

         if (root==null)

        {

            return;

        }

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

        preOrder(root.left);

        preOrder(root.right);

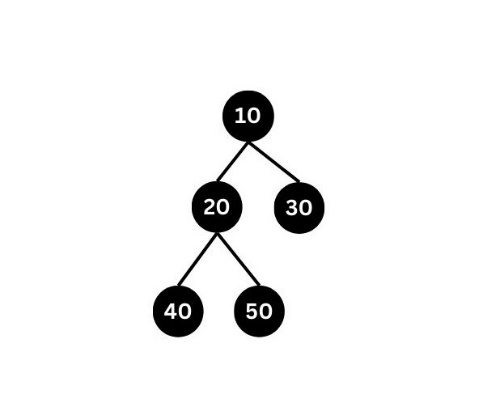
    }

}

**Problem statement**

Send feedback

For a given Binary Tree of integers, print the post-order traversal.



The postorder traversal for the given binary tree will be 40 50 20 30 10

public class Solution {

    public static void postOrder(BinaryTreeNode<Integer> root) {

        //Your code goes here

          if (root==null)

        {

            return;

        }

        postOrder(root.left);

        postOrder(root.right);

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

    }

}

**Problem statement**

Send feedback

A node in Binary trees can have how many children ?

Ans = 0 or 1 or 2

**Problem statement**

Send feedback

Consider the below code

takeInput()

print("Enter root data")

rootData = int(input())

if (rootData == -1)

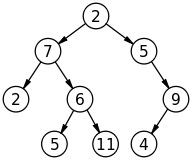
return None

root = BinaryTreeNode(rootData)

root.left = takeInput()

root.right = takeInput()

return root



What will be the input(excluding -1) to above code to construct this tree ?

1) 2 7 2 6 5 11 5 9 4

2) 2 7 5 6 11 2 5 4 9

3) 2 7 5 2 6 9 5 11 4

**Options:**Pick one correct answer from below

Attempts left:

1/2

**1**

**2**

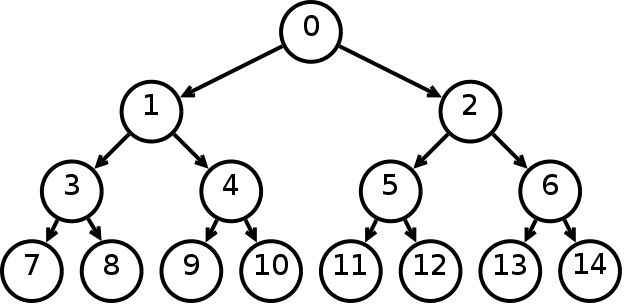
**3**

Ans = 1

**Problem statement**

Send feedback

Print given tree level wise



1) 0 1 3 7 8 4 9 10 2 5 11 12 6 13 14

2) 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

3) 0 2 1 6 5 4 3 14 13 12 11 10 9 8 7

Ans = 2

**Problem statement**

Send feedback

The maximum and minimum number of nodes in a binary tree of height 5 are :

Note:

A tree with zero node has height 0.

A tree with one node has height 1.

Ans = 63 and 6

Share your experience with this lecture?

Submit

Ask a doubt

**Problem statement**

Send feedback

For a given a binary tree of integers and an integer X, find and return the total number of nodes of the given binary tree which are having data greater than X.

**Detailed explanation**( Input/output format, Notes, Images )

**Constraints:**

1 <= N <= 10^5

Where N is the total number of nodes in the binary tree.

Time Limit: 1 sec

**Sample Input 1:**

1 4 2 3 -1 -1 -1

2

**Sample Output 1:**

2

**Explanation for Sample Input 1:**

Out of the four nodes of the given binary tree, [3, 4] are the node data that are greater than X = 2.

**Sample Input 2:**

5 6 10 2 3 -1 -1 -1 -1 -1 9 -1 -1

5

**Sample Output 2:**

3

public class Solution {

    public static int countNodesGreaterThanX(BinaryTreeNode<Integer> root, int x) {

        //Your code goes here

        if (root==null)

        {

            return 0;

        }

        int smallOutput=countNodesGreaterThanX(root.left,x)+countNodesGreaterThanX(root.right,x);

        if (root.data>x)

        {

            return smallOutput+1;

        }

        else

        {

            return smallOutput;

        }

    }

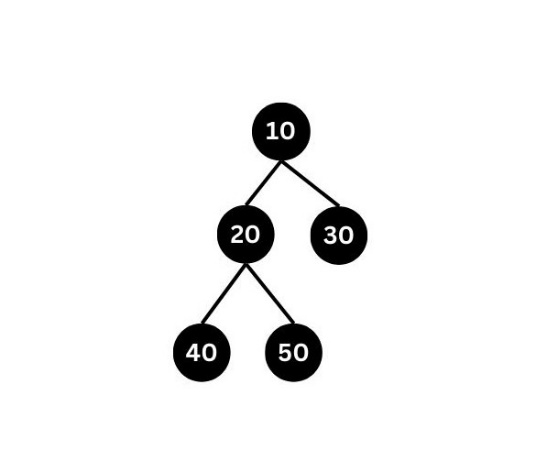
}

**Problem statement**

Send feedback

For a given Binary Tree of integers, find and return the height of the tree. Height is defined as the total number of nodes along the longest path from the root to any of the leaf node.

**Example:**



Height of the given tree is 3.

**Detailed explanation**( Input/output format, Notes, Images )

**Constraints:**

0 <= N <= 10^5

Where N is the total number of nodes in the binary tree.

public class Solution {

    public static int height(BinaryTreeNode<Integer> root) {

        //Your code goes here

        if (root==null)

        {

            return 0;

        }

        int smallLeftOutput=height(root.left);

        int smallRightOutput=height(root.right);

        if (smallLeftOutput>smallRightOutput)

        {

            return smallLeftOutput+1;

        }

        else

        {

            return smallRightOutput+1;

        }

    }

}

**Problem statement**

Send feedback

For a given Binary Tree of type integer and a number X, find whether a node exists in the tree with data X or not.

**Detailed explanation**( Input/output format, Notes, Images )

**Constraints:**

1 <= N <= 10^5

Where N is the total number of nodes in the binary tree.

Time Limit: 1 sec.

\*

    Following is the structure used to represent the Binary Tree Node

    class BinaryTreeNode<T> {

        T data;

        BinaryTreeNode<T> left;

        BinaryTreeNode<T> right;

        public BinaryTreeNode(T data) {

            this.data = data;

            this.left = null;

            this.right = null;

        }

    }

\*/

public class Solution {

    public static boolean isNodePresent(BinaryTreeNode<Integer> root, int x) {

        //Your code goes here

          if (root==null)

        {

            return false;

        }

        if (root.data==x)

        {

            return true;

        }

        else

        {

            return (isNodePresent(root.left,x)||isNodePresent(root.right,x));

        }

    }

}

**Problem statement**

Send feedback

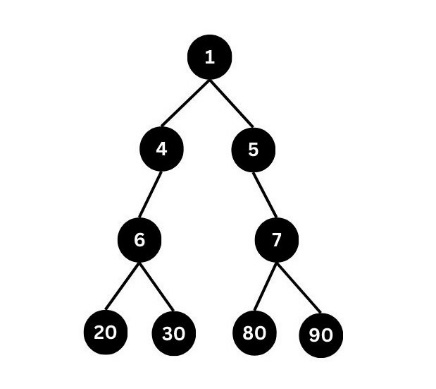
For a given Binary Tree of type integer, print all the nodes without any siblings.

**Example Input :**

1 4 5 6 -1 -1 7 20 30 80 90 -1 -1 -1 -1 -1 -1 -1 -1

**Explanation:**

The input tree when represented in a two-dimensional plane, it would look like this:



In respect to the root, node data in the left subtree that satisfy the condition of not having a sibling would be 6, taken in a top-down sequence. Similarly, for the right subtree, 7 is the node data without any sibling.

Since we print the siblings in the left-subtree first and then the siblings from the right subtree, taken in a top-down fashion, we print 6 7.

**Example Output:**

6 7

**Detailed explanation**( Input/output format, Notes, Images )

**Constraints:**

1 <= N <= 10^5

Where N is the total number of nodes in the binary tree.

Time Limit: 1 second

**Sample Input 1:**

5 6 10 2 3 -1 -1 -1 -1 -1 9 -1 -1

**Sample Output 1:**

9

/\*

    Following is the structure used to represent the Binary Tree Node

    class BinaryTreeNode<T> {

        T data;

        BinaryTreeNode<T> left;

        BinaryTreeNode<T> right;

        public BinaryTreeNode(T data) {

            this.data = data;

            this.left = null;

            this.right = null;

        }

    }

\*/

public class Solution {

    public static void printNodesWithoutSibling(BinaryTreeNode<Integer> root) {

        //Your code goes here

        if (root==null)

        {

            return;

        }

        if (root.left==null && root.right==null)

        {

            return;

        }

        if (root.left==null)

        {

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

            printNodesWithoutSibling(root.right);

        }

        else if (root.right==null)

        {

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

            printNodesWithoutSibling(root.left);

        }

        else

        {

            printNodesWithoutSibling(root.left);

            printNodesWithoutSibling(root.right);

        }

    }

}

**Problem statement**

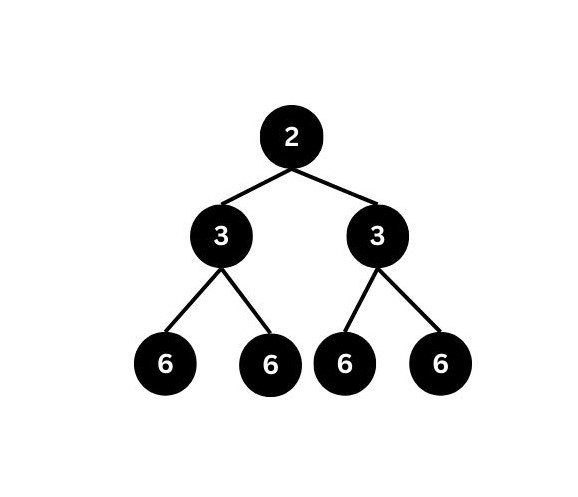
Send feedback

For a given a Binary Tree of integers, replace each of its data with the depth of the tree.

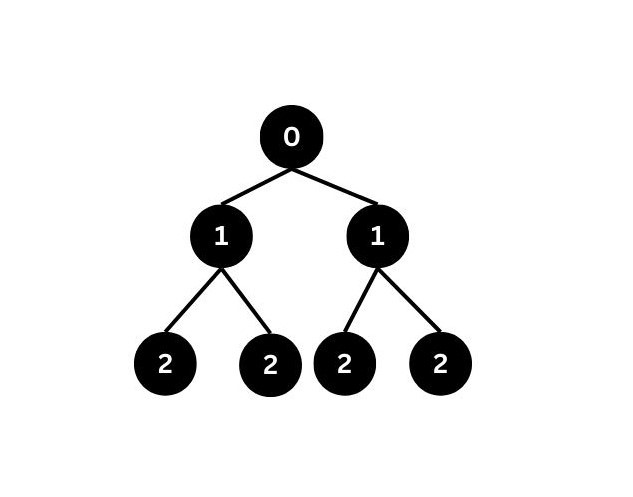
Root is at depth 0, hence the root data is updated with 0. Replicate the same further going down the in the depth of the given tree.

The modified tree will be printed in the in-order fashion.

**Example:**



The above tree after updating will look like this:



Output: 2 1 2 0 2 1 2 (printed in the in-order fashion)

public class Solution {

    public static void changeToDepthTree(BinaryTreeNode<Integer> root) {

        //Your code goes here

         changeToDepthTree(root,0);

    }

    public static void changeToDepthTree(BinaryTreeNode<Integer> root, int depth)

    {

        if (root==null)

        {

            return;

        }

        root.data=depth;

        changeToDepthTree(root.left,depth+1);

        changeToDepthTree(root.right,depth+1);

    }

}

**Problem statement**

Send feedback

For a given Binary Tree of type integer and a number X, find whether a node exists in the tree with data X or not.

**Detailed explanation**( Input/output format, Notes, Images )

**Constraints:**

1 <= N <= 10^5

Where N is the total number of nodes in the binary tree.

Time Limit: 1 sec.

**Sample Input 1:**

8 3 10 1 6 -1 14 -1 -1 4 7 13 -1 -1 -1 -1 -1 -1 -1

7

**Sample Output 1:**

true

**Explanation For Output 1:**

Clearly, we can see that 7 is present in the tree. So, the output will be true.

**Sample Input 2:**

2 3 4 -1 -1 -1 -1

10

**Sample Output 2:**

false

**p**ublic class Solution {

    public static boolean isNodePresent(BinaryTreeNode<Integer> root, int x) {

        //Your code goes here

          if (root==null)

        {

            return false;

        }

        if (root.data==x)

        {

            return true;

        }

        else

        {

            return (isNodePresent(root.left,x)||isNodePresent(root.right,x));

        }

    }

}

**Problem statement**

Send feedback

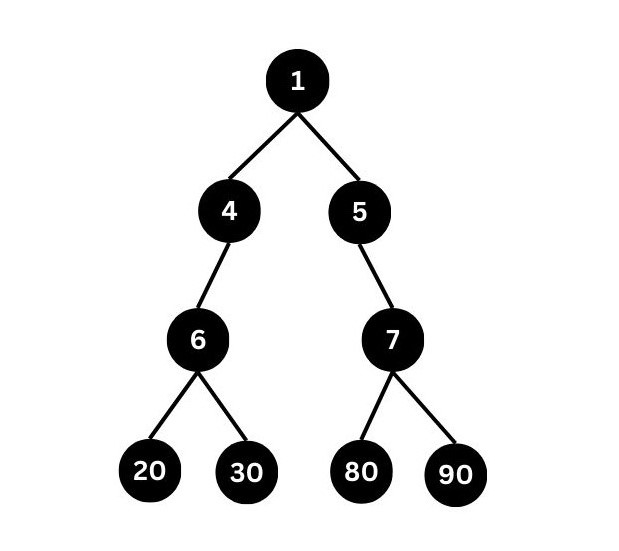
For a given Binary Tree of type integer, print all the nodes without any siblings.

**Example Input :**

1 4 5 6 -1 -1 7 20 30 80 90 -1 -1 -1 -1 -1 -1 -1 -1

**Explanation:**

The input tree when represented in a two-dimensional plane, it would look like this:



In respect to the root, node data in the left subtree that satisfy the condition of not having a sibling would be 6, taken in a top-down sequence. Similarly, for the right subtree, 7 is the node data without any sibling.

Since we print the siblings in the left-subtree first and then the siblings from the right subtree, taken in a top-down fashion, we print 6 7.

**Example Output:**

6 7

**Detailed explanation**( Input/output format, Notes, Images )

**Constraints:**

1 <= N <= 10^5

Where N is the total number of nodes in the binary tree.

Time Limit: 1 second

**Sample Input 1:**

5 6 10 2 3 -1 -1 -1 -1 -1 9 -1 -1

**Sample Output 1:**

9

public class Solution {

    public static void printNodesWithoutSibling(BinaryTreeNode<Integer> root) {

        //Your code goes here

        if (root==null)

        {

            return;

        }

        if (root.left==null && root.right==null)

        {

            return;

        }

        if (root.left==null)

        {

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

            printNodesWithoutSibling(root.right);

        }

        else if (root.right==null)

        {

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

            printNodesWithoutSibling(root.left);

        }

        else

        {

            printNodesWithoutSibling(root.left);

            printNodesWithoutSibling(root.right);

        }

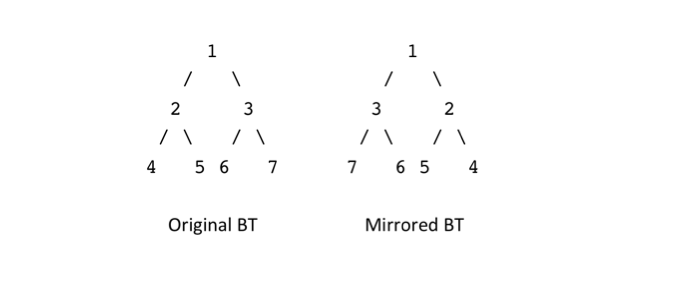
    }

}

**Problem statement**

Send feedback

For a given Binary Tree of type integer, update it with its corresponding mirror image.



**Problem statement**

Send feedback

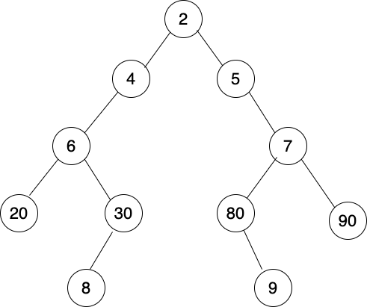
For a given Binary of type integer, find and return the ‘Diameter’.

**Diameter of a Tree**

The diameter of a tree can be defined as the maximum distance between two leaf nodes.

Here, the distance is measured in terms of the total number of nodes present along the path of the two leaf nodes, including both the leaves.

**Example:**



**Problem statement**

Send feedback

For a given a Binary Tree of type integer, print the complete information of every node, when traversed in a level-order fashion.

To print the information of a node with data D, you need to follow the exact format :

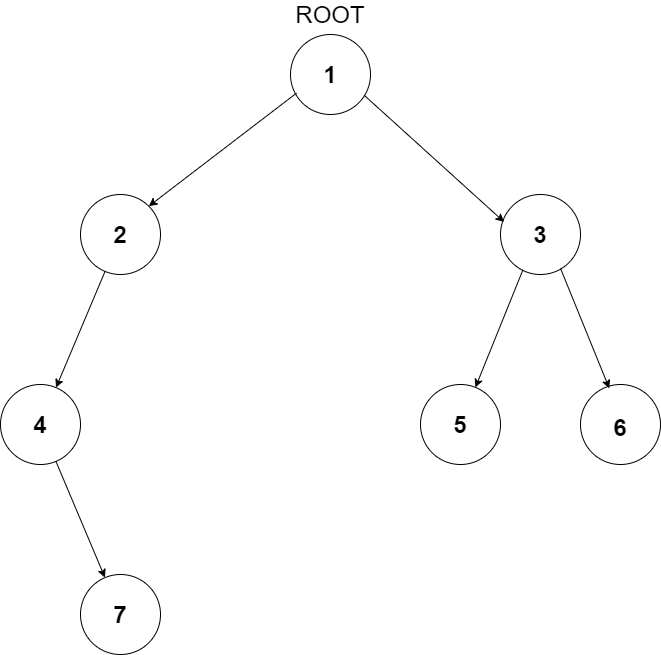
D:L:X,R:Y

Where D is the data of a node present in the binary tree.

X and Y are the values of the left(L) and right(R) child of the node.

Print -1 if the child doesn't exist.

**Example:**



public class Solution {

    public static void printLevelWise(BinaryTreeNode<Integer> root) {

        //Your code goes here

        if (root==null)

        {

            return;

        }

        Queue<BinaryTreeNode<Integer>> nodesToPrint=new LinkedList<BinaryTreeNode<Integer>>();

        nodesToPrint.add(root);

        while(!nodesToPrint.isEmpty())

        {

            BinaryTreeNode<Integer> front = nodesToPrint.poll();

            System.out.print(front.data+":");

            if (front.left!=null)

            {

                nodesToPrint.add(front.left);

                System.out.print("L:"+front.left.data);

            }

            else

            {

                System.out.print("L:-1");

            }

            if (front.right!=null)

            {

                nodesToPrint.add(front.right);

                System.out.print(",R:"+front.right.data);

            }

            else

            {

                System.out.print(",R:-1");

            }

            System.out.println();

        }

    }

}

Build tree using inorder and preorder

class BinaryTreeNode:

def \_\_init\_\_(self,data):

self.data=data;

self.left=None

self.right=None

def printTree(root):

if root==None:

return

print(root.data)

printTree(root.left)

printTree(root.right)

def printTreeDetailed(root):

if root==None:

return

print(root.data,end=":")

if root.left!=None:

print("L",root.left.data,end=",")

if root.right!=None:

print("R",root.right.data,end="")

print()

printTreeDetailed(root.left)

printTreeDetailed(root.right)

def buildTreeFromPreIn(pre,inorder):

if len(pre)==0:

return None

rootData=pre[0]

root=BinaryTreeNode(rootData)

rootIndexInInOrder=-1

for i in range(0,len(inorder)):

if inorder[i]==rootData:

rootIndexInInorder=i

break

if rootIndexInInorder==-1:

return None

leftInorder=inorder[0:rootIndexInInorder]

rightInorder=inorder[rootIndexInInorder+1:]

lenLeftSubtree=len(leftInorder)

leftPreorder=pre[1:lenLeftSubtree+1]

rightPreorder=pre[lenLeftSubtree+1:]

leftChild=buildTreeFromPreIn(leftPreorder,leftInorder)

rightChild=buildTreeFromPreIn(rightPreorder,rightInorder)

root.left=leftChild

root.right=rightChild

return root

pre=[1,2,4,5,3,6,7]

inorder=[4,2,5,1,6,3,7]

root=buildTreeFromPreIn(pre,inorder)

printTreeDetailed(root)

**Problem statement**

Send feedback

The inorder and preorder traversal of a binary tree are

Inorder: d b e a p q r

PreOrder: a b d e q p r

The postorder traversal of the binary tree is:

(A) d e b p r q a

(B) e d b r p q a

(C) e d b p r q a

(D) d e p r b q a

Ans = A

**Problem statement**

Send feedback

For a given preorder and inorder traversal of a Binary Tree of type integer stored in an array/list, create the binary tree using the given two arrays/lists. You just need to construct the tree and return the root.

**Note:**

Assume that the Binary Tree contains only unique elements.

**Detailed explanation**( Input/output format, Notes, Images )

**Constraints:**

1 <= N <= 10^3

Where N is the total number of nodes in the binary tree.

/\*

    Following is the structure used to represent the Binary Tree Node

    class BinaryTreeNode<T> {

        T data;

        BinaryTreeNode<T> left;

        BinaryTreeNode<T> right;

        public BinaryTreeNode(T data) {

            this.data = data;

            this.left = null;

            this.right = null;

        }

    }

\*/

public class Solution {

    public static BinaryTreeNode<Integer> buildTree(int[] preOrder, int[] inOrder) {

        //Your code goes here

          BinaryTreeNode<Integer> root = buildTree(preOrder, inOrder, 0 ,preOrder.length-1, 0, inOrder.length-1);

        return root;

    }

    public static BinaryTreeNode<Integer> buildTree(int[] preorder, int[] inorder,int siPre, int eiPre, int siIn, int eiIn)

    {

        //Base case - If number of elements in the pre-order is 0

        if (siPre>eiPre)

        {

            return null;

        }

        //Defining the root node for current recursion

        int rootData=preorder[siPre];

        BinaryTreeNode<Integer> root = new BinaryTreeNode<Integer>(rootData);

        //Finding root data's location in Inorder (Assuming root data exists in Inorder)

        int rootIndexInorder=-1;

        for (int i=siIn;i<=eiIn;i++)

        {

            if (rootData==inorder[i])

            {

                rootIndexInorder=i;

                break;

            }

        }

        //Defining index limits for Left Subtree Inorder

        int siInLeft=siIn;

        int eiInLeft=rootIndexInorder-1;

        //Defining the index limits for Left Subtree Preorder

        int siPreLeft=siPre+1;

        int leftSubTreeLength = eiInLeft - siInLeft + 1;

        int eiPreLeft=(siPreLeft)+(leftSubTreeLength-1);

        //Defining index limits for Right Subtree Inorder

        int siInRight=rootIndexInorder+1;

        int eiInRight=eiIn;

        //Defining index limits for Right Subtree Preorder

        int siPreRight=eiPreLeft+1;

        int eiPreRight=eiPre;

        BinaryTreeNode<Integer> leftChild = buildTree(preorder, inorder, siPreLeft, eiPreLeft, siInLeft, eiInLeft);

        BinaryTreeNode<Integer> rightChild = buildTree(preorder, inorder, siPreRight, eiPreRight, siInRight, eiInRight);

        root.left=leftChild;

        root.right=rightChild;

        return root;

    }

}

**Problem statement**

Send feedback

For a given postorder and inorder traversal of a Binary Tree of type integer stored in an array/list, create the binary tree using the given two arrays/lists. You just need to construct the tree and return the root.

**Note:**

Assume that the Binary Tree contains only unique elements.

**Detailed explanation**( Input/output format, Notes, Images )

**Constraints:**

1 <= N <= 10^4

Where N is the total number of nodes in the binary tree.