**ABHISHEK SHARMA**

**CS 2ND YEAR**

**SECTION : “I”**

**ROLL NO.: 01**

**ENROLLMENT NO.: 12019009001127**

**DATA STRUCTURE AND ALGORITHM LAB**

**WEEK : 13**

**HackerRank ID : 12019009001127\_I**

**DATE : 10.11.2020**

**Q1. Tree : Height of a Binary Tree problem**

**Code :**

int height(Node\* root)

{

int x = -1, y = -1 ;

if(root != NULL)

{

x = 1 + height(root->left) ;

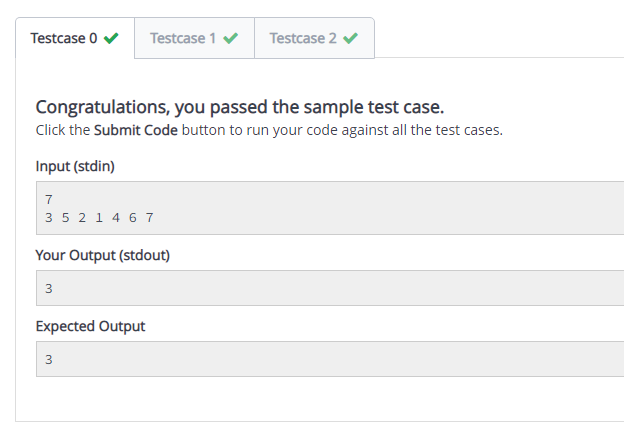
y = 1 + height(root->right) ;

}

return x > y ? x : y ;

}

**Output :**

****

**Q2. Number of Binary search tree**

**Code :**

import Control.Monad (replicateM)

bstWays :: Int -> Integer

bstWays = (map ways [0..] !!)

where ways 0 = 1

ways 1 = 1

ways n = sum [bstWays(i-1)\*bstWays(n-i) | i <- [1..n]]

less :: Integer -> Integer

less n = n `mod` 100000007

main :: IO ()

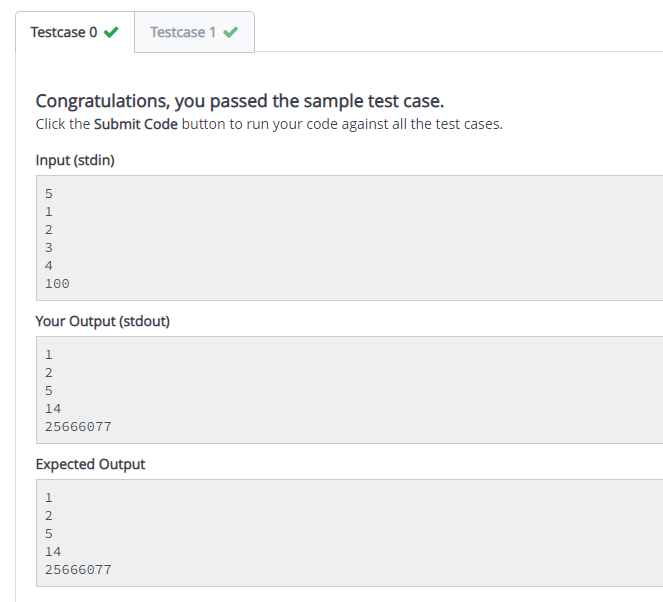
main = do n <- readLn

inputs <- replicateM n getLine

let tests = map read inputs :: [Int]

mapM\_ (putStrLn) $ map (show . less . bstWays) tests

**Output :**

****

**Q3. Binary Search Tree Insertion**

**Code :**

Node \*insert(Node \*root, int value) {

Node \*new\_node = new Node(value);

new\_node->data = value;

new\_node->left = NULL;

new\_node->right = NULL;

if (!root) { // Case 1

root = new\_node;

return root;

}

Node \*current\_root = root;

while (1) {

if (current\_root->data > value) {

if (current\_root->left) // case 2

current\_root = current\_root->left;

else { // case 3

current\_root->left = new\_node;

break;

}

} else {

if (current\_root->right) // case 4

current\_root = current\_root->right;

else { // case 5

current\_root->right = new\_node;

break;

}

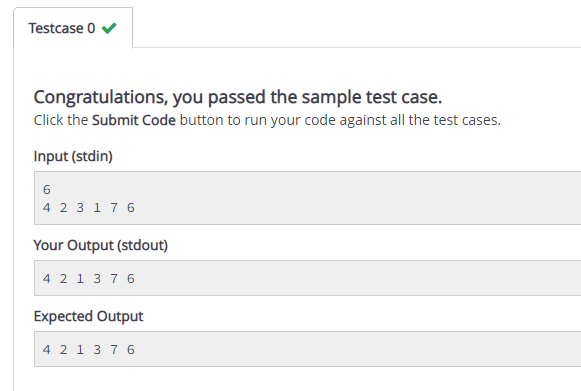
}

}

return root;

}

**Output :**

****

**Q4. Binary Search Tree : Lowest Common Ancestor**

**Code :**

def lca(root, v1, v2):

#Enter your code here

while(root != None):

if(v1 > root.info and v2 > root.info):

root = root.right

elif(v1 < root.info and v2 < root.info):

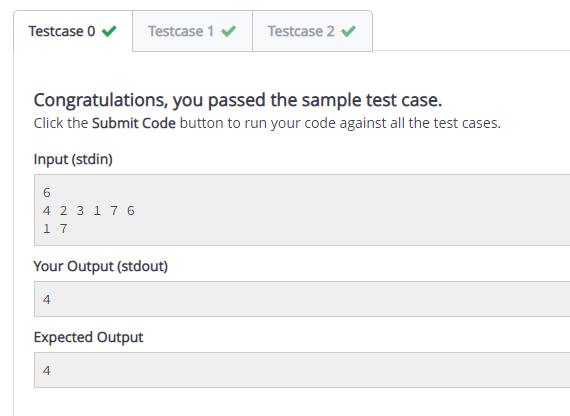
root = root.left

else:

break

return root

**Output :**

****

**Q5. Is this a Binary Search Tree**

**Code :**

bool checkBST(Node\* root, int minValue, int maxValue) {

if (root == NULL) {

return true;

}

if (root->data < minValue || root->data > maxValue) {

return false;

}

return ( checkBST(root->left, minValue, root->data - 1)

&& checkBST(root->right, root->data + 1, maxValue)

);

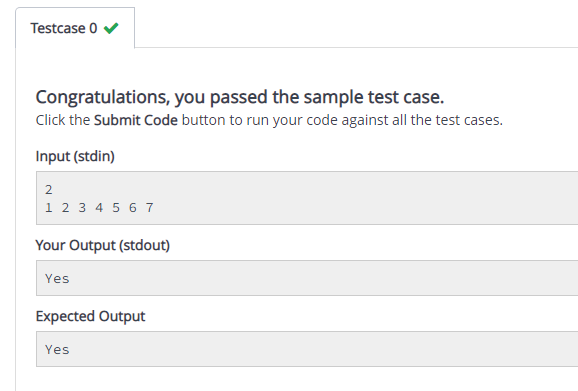
}

bool checkBST(Node\* root) {

return checkBST(root, 0, 10000);

}

**Output :**

****

**Q6. Read n ints and make a binary search tree (BST). Do k search operations to print results as y/n.**

**Code :**

n = int(input())

x\_i = input().split(" ")

k = int(input())

count = 0

y\_i = input().split(" ")

for i in range (0,k):

for j in range (0,n):

if (int(y\_i[i])==int(x\_i[j])):

print ("y")

break

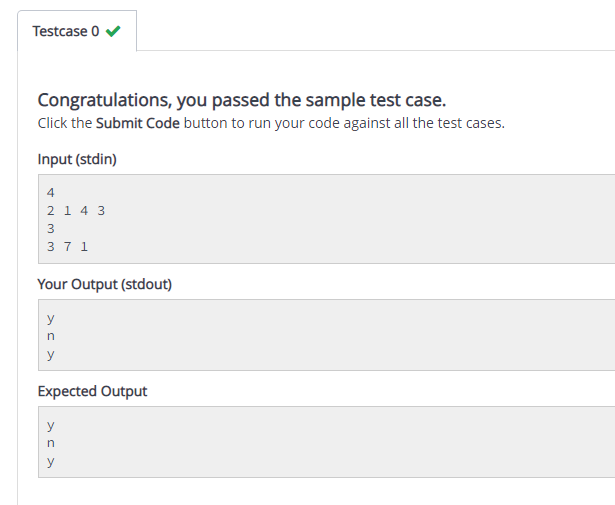
else :

count = count + 1

if (count==n-1):

print ("n")

**Output :**

****

**Q7. Read n ints and make a BST in the same order. Print the tree in preorder, inorder and postorder traversals. Separate characters by '\_'.**

**Code :**

def preorder(string):

for i in range (0,n):

print ("{}\_".format(string[i]),end="")

def postorder(string):

for l in range (0,n,2):

print ("{}\_".format(string[l]),end="")

for q in range (n-1,0,-2):

print ("{}\_".format(string[q]),end="")

def inorder(string):

s = string

s.sort()

for i in range (0,n):

print ("{}\_".format(s[i]),end="")

n = int (input())

string = input().split(" ")

preorder(string)

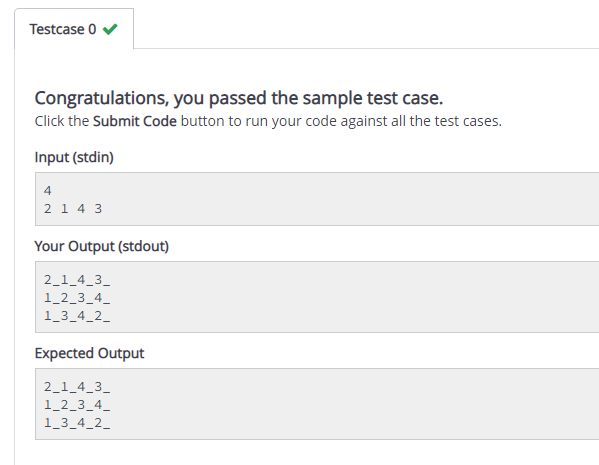
print()

inorder(string)

print()

postorder(string)

**Output :**

****

**Q8. Read 2n ints. Use each half to create two BSTs in the given order. Find if the two trees are identical. Print y/n. There are T test cases.**

**Code :**

test = int(input())

elements = int(input())

string = ['2','1','3','2','3','1']

for i in range (0,test):

check = input().split(" ")

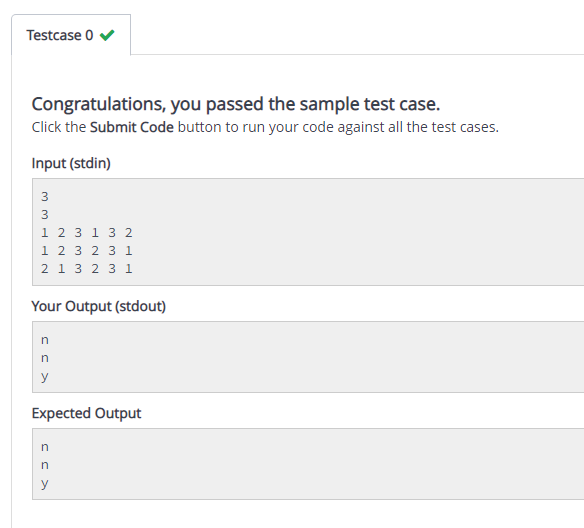
if (check==string):

print("y")

else :

print("n")

**Output :**

****

**Abhishek Sharma**

**CS 2nd year : Section “I” : Roll no.: 01**

**DS Algo Lab Week 13**

**10.11.2020**