# Python3 program to find largest subtree

# sum in a given binary tree.

# Function to create new tree node.

class newNode:

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

self.key = key

self.left = self.right = None

# Helper function to find largest

# subtree sum recursively.

def findLargestSubtreeSumUtil(root, ans):

# If current node is None then

# return 0 to parent node.

if (root == None):

return 0

# Subtree sum rooted at current node.

currSum = (root.key +

findLargestSubtreeSumUtil(root.left, ans) +

findLargestSubtreeSumUtil(root.right, ans))

# Update answer if current subtree

# sum is greater than answer so far.

ans[0] = max(ans[0], currSum)

# Return current subtree sum to

# its parent node.

return currSum

# Function to find largest subtree sum.

def findLargestSubtreeSum(root):

# If tree does not exist,

# then answer is 0.

if (root == None):

return 0

# Variable to store maximum subtree sum.

ans = [-999999999999]

# Call to recursive function to

# find maximum subtree sum.

findLargestSubtreeSumUtil(root, ans)

return ans[0]

# Driver Code

if \_\_name\_\_ == '\_\_main\_\_':

#

# 1

# / \

# / \

# -2 3

# / \ / \

# / \ / \

# 4 5 -6 2

root = newNode(1)

root.left = newNode(-2)

root.right = newNode(3)

root.left.left = newNode(4)

root.left.right = newNode(5)

root.right.left = newNode(-6)

root.right.right = newNode(2)

print(findLargestSubtreeSum(root))

# Python implementation to construct a BST

# from its level order traversal

import math

# Node of a BST

class Node:

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

self.data = data

self.left = None

self.right = None

# Function to get a new node

def getNode(data):

# Allocate memory

newNode = Node(data)

# put in the data

newNode.data = data

newNode.left = None

newNode.right = None

return newNode

# Function to construct a BST from

# its level order traversal

def LevelOrder(root, data):

if(root == None):

root = getNode(data)

return root

if(data <= root.data):

root.left = LevelOrder(root.left, data)

else:

root.right = LevelOrder(root.right, data)

return root

def constructBst(arr, n):

if(n == 0):

return None

root = None

for i in range(0, n):

root = LevelOrder(root, arr[i])

return root

# Function to print the inorder traversal

def inorderTraversal(root):

if (root == None):

return None

inorderTraversal(root.left)

print(root.data, end=" ")

inorderTraversal(root.right)

# Driver program

if \_\_name\_\_ == '\_\_main\_\_':

arr = [7, 4, 12, 3, 6, 8, 1, 5, 10]

n = len(arr)

root = constructBst(arr, n)

print("Inorder Traversal: ", end="")

root = inorderTraversal(root)

# Python3 implementation to check if the

# given array can represent Level Order

# Traversal of Binary Search Tree

INT\_MIN, INT\_MAX = float('-inf'), float('inf')

# To store details of a node like node's

# data, 'min' and 'max' to obtain the

# range of values where node's left

# and right child's should lie

class NodeDetails:

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

self.data = data

self.min = min

self.max = max

# function to check if the given array

# can represent Level Order Traversal

# of Binary Search Tree

def levelOrderIsOfBST(arr, n):

# if tree is empty

if n == 0:

return True

# queue to store NodeDetails

q = []

# index variable to access array elements

i = 0

# node details for the root of the BST

newNode = NodeDetails(arr[i], INT\_MIN, INT\_MAX)

i += 1

q.append(newNode)

# until there are no more elements

# in arr[] or queue is not empty

while i != n and len(q) != 0:

# extracting NodeDetails of a

# node from the queue

temp = q.pop(0)

# check whether there are more elements

# in the arr[] and arr[i] can be left

# child of 'temp.data' or not

if i < n and (arr[i] < temp.data and

arr[i] > temp.min):

# Create NodeDetails for newNode

#/ and add it to the queue

newNode = NodeDetails(arr[i], temp.min, temp.data)

i += 1

q.append(newNode)

# check whether there are more elements

# in the arr[] and arr[i] can be right

# child of 'temp.data' or not

if i < n and (arr[i] > temp.data and

arr[i] < temp.max):

# Create NodeDetails for newNode

#/ and add it to the queue

newNode = NodeDetails(arr[i], temp.data, temp.max)

i += 1

q.append(newNode)

# given array represents level

# order traversal of BST

if i == n:

return True

# given array do not represent

# level order traversal of BST

return False

# Driver code

if \_\_name\_\_ == "\_\_main\_\_":

arr = [7, 4, 12, 3, 6, 8, 1, 5, 10]

n = len(arr)

if levelOrderIsOfBST(arr, n):

print("Yes")

else:

print("No")