**Assignment 20 – Tree**

Question-1

Given a binary tree, your task is to find subtree with maximum sum in tree.

Examples:

Input1 :

1

/   \\

2      3

/ \    / \

4   5  6   7

Output1 : 28

As all the tree elements are positive, the largest subtree sum is equal to sum of all tree elements.

Input2 :

1

/    \\

-2      3

/ \    /  \

4   5  -6   2

Output2 : 7

Subtree with largest sum is :

-2

/ \

4   5

Also, entire tree sum is also 7.

Solve:-

# Node definition for a binary tree

class TreeNode:

def \_\_init\_\_(self, val=0, left=None, right=None):

self.val = val

self.left = left

self.right = right

def findMaxSubtreeSum(node):

global max\_sum

if node is None:

return 0

left\_sum = findMaxSubtreeSum(node.left)

right\_sum = findMaxSubtreeSum(node.right)

current\_sum = left\_sum + node.val + right\_sum

max\_sum = max(max\_sum, current\_sum)

return current\_sum

def maxSubtreeSum(root):

global max\_sum

max\_sum = float('-inf')

findMaxSubtreeSum(root)

return max\_sum

# Create the binary tree from the example

root = TreeNode(1)

root.left = TreeNode(2)

root.right = TreeNode(3)

root.left.left = TreeNode(4)

root.left.right = TreeNode(5)

root.right.left = TreeNode(6)

root.right.right = TreeNode(7)

result = maxSubtreeSum(root)

print(result)

**# Output: 28**

Question-2

Construct the BST (Binary Search Tree) from its given level order traversal.

Example:

Input: arr[] = {7, 4, 12, 3, 6, 8, 1, 5, 10}

Output: BST:

7

/    \\

4     12

/  \\     /

3   6  8

/   /   \

1  5   10

SOLVE:-

class Node:

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

self.val = val

self.left = None

self.right = None

def constructBST(level\_order, start, end):

if start > end:

return None

root = Node(level\_order[start])

i = start + 1

while i <= end:

if level\_order[i] > root.val:

break

i += 1

root.left = constructBST(level\_order, start + 1, i - 1)

root.right = constructBST(level\_order, i, end)

return root

def levelOrderTraversal(root):

if root is None:

return []

queue = [root]

level\_order = []

while queue:

node = queue.pop(0)

level\_order.append(node.val)

if node.left:

queue.append(node.left)

if node.right:

queue.append(node.right)

return level\_order

def constructBSTFromLevelOrder(level\_order):

root = constructBST(level\_order, 0, len(level\_order) - 1)

return root

level\_order = [7, 4, 12, 3, 6, 8, 1, 5, 10]

root = constructBSTFromLevelOrder(level\_order)

# Verify the constructed BST by performing a level order

result = levelOrderTraversal(root)

print(result)

# Output: [7, 4, 12, 3, 6, 8, 1, ,5, 10]

Question-3

Given an array of size n. The problem is to check whether the given array can represent the level order traversal of a Binary Search Tree or not.

Examples:

Input1 : arr[] = {7, 4, 12, 3, 6, 8, 1, 5, 10}

Output1 : Yes

For the given arr[], the Binary Search Tree is:

7

/    \\

4     12

/  \\     /

3   6  8

/   /   \

1  5   10

Input2 : arr[] = {11, 6, 13, 5, 12, 10}

Output2 : No

The given arr[] does not represent

Solve

def canRepresentBST(arr, n, index, min\_val, max\_val):

if index >= n:

return True

if arr[index] < min\_val or arr[index] > max\_val:

return False

left\_child\_index = 2 \* index + 1

right\_child\_index = 2 \* index + 2

return (

canRepresentBST(arr, n, left\_child\_index, min\_val, arr[index] - 1)

and canRepresentBST(arr, n, right\_child\_index, arr[index] + 1, max\_val)

)

def isLevelOrderBST(arr):

n = len(arr)

return canRepresentBST(arr, n, 0, float('-inf'), float('inf'))

# Example usage:

arr1 = [7, 4, 12, 3, 6, 8, 1, 5, 10]

result1 = isLevelOrderBST(arr1)

print(result1) # Output: True

arr2 = [11, 6, 13, 5, 12,