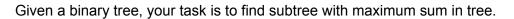
Question-1





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.
```

```
Soln:
class TreeNode:
  def __init__(self, val=0, left=None, right=None):
    self.val = val
    self.left = left
    self.right = right
def maxSubtreeSum(root):
  if root is None:
    return 0
  left_sum = maxSubtreeSum(root.left)
  right_sum = maxSubtreeSum(root.right)
  subtree_sum = root.val + left_sum + right_sum
  global maxSum
  maxSum = max(maxSum, subtree_sum)
  return subtree_sum
def maxSumSubtree(root):
  global maxSum
  maxSum = float('-inf')
  maxSubtreeSum(root)
  return maxSum
```

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
     / \
       12
  /\//
  3 6 8
 / / \
1 5
         10
Soln:
from collections import deque
class TreeNode:
  def __init__(self, val=0, left=None, right=None):
     self.val = val
     self.left = left
    self.right = right
def constructBST(level_order):
  if not level_order:
     return None
  queue = deque()
  root = TreeNode(level_order[0])
  queue.append(root)
  while queue and i < len(level_order):
    node = queue.popleft()
    left_val = level_order[i]
     i += 1
     if left_val != -1:
       left_child = TreeNode(left_val)
       node.left = left_child
```

```
queue.append(left_child)
     if i < len(level order):
        right_val = level_order[i]
        i += 1
        if right_val != -1:
          right_child = TreeNode(right_val)
          node.right = right_child
          queue.append(right_child)
  return root
def inorderTraversal(root):
  if root is None:
     return []
  result = []
  stack = []
  curr = root
  while curr or stack:
     while curr:
        stack.append(curr)
        curr = curr.left
     curr = stack.pop()
     result.append(curr.val)
     curr = curr.right
  return result
```

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:

The given arr[] does not represent the level order traversal of a BST.

Soln:

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def isLevelOrderBST(arr):
    if not arr:
        return True

    n = len(arr)
    stack = []
    root = TreeNode(arr[0])
    stack.append(root)

i = 1
    while i < n:</pre>
```

```
parent = stack[-1]

if arr[i] < parent.val:
    return False

while stack and arr[i] > stack[-1].val:
    parent = stack.pop()

new_node = TreeNode(arr[i])
if parent.left is None:
    parent.left = new_node
else:
    parent.right = new_node

stack.append(new_node)
i += 1
```