Assignment 20 Solutions

Q1.**Given a binary tree, your task is to find subtree with maximum sum in tree.**

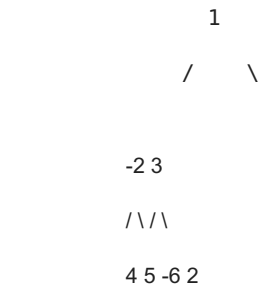
Examples:

Input1 :

Output1 : 28

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

Input2 :



Output2 : 7

Subtree with largest sum is :

Also, entire tree sum is also 7.

In [52]:
```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right


def findMaxSubtreeSum(node):
    if node is None:
        return 0

    left_sum = findMaxSubtreeSum(node.left)
    right_sum = findMaxSubtreeSum(node.right)

    subtree_sum = node.val + left_sum + right_sum

    global max_sum
    if subtree_sum > max_sum:
        max_sum = subtree_sum

    return subtree_sum


def findMaximumSubtreeSum(root):
    global max_sum
    max_sum = float('-inf')
    findMaxSubtreeSum(root)
    return max_sum
```

```
# Test case
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(2)

print(findMaximumSubtreeSum(root))
```

7

Q2.**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
```

In [53]:
```
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None


def constructBST(level_order):
    if not level_order:
        return None

    root = Node(level_order[0])
    queue = [(root, float('-inf'), float('inf'))]
    i = 1

    while i < len(level_order):
        parent, min_val, max_val = queue[0]
        queue.pop(0)

        if level_order[i] < parent.data and min_val < level_order[i] < parent.data:
            left_child = Node(level_order[i])
            parent.left = left_child
            queue.append((left_child, min_val, parent.data))
            i += 1

        if i < len(level_order) and level_order[i] > parent.data and parent.data < level_order[i] < max_val:
            right_child = Node(level_order[i])
            parent.right = right_child
            queue.append((right_child, parent.data, max_val))
            i += 1

    return root


def inorderTraversal(node):
    if node:
        inorderTraversal(node.left)
        print(node.data, end=" ")
        inorderTraversal(node.right)


# Example usage
level_order = [7, 4, 12, 3, 6, 8, 1, 5, 10]
bst_root = constructBST(level_order)
```

```
inorderTraversal(bst_root)
```

```
1 3 4 5 6 7 8 10 12
```

Q3.**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 the level order traversal of a BST.

In [54]:
```
INT_MIN, INT_MAX = float('-inf'), float('inf')

class NodeDetails:
        def __init__(self, data, min, max):
                self.data = data
                self.min = min
                self.max = max

def levelOrderIsOfBST(arr, n):
        if n == 0:
                return True

        q = []
        i = 0
        newNode = NodeDetails(arr[i], INT_MIN, INT_MAX)
        i += 1
        q.append(newNode)

        while i != n and len(q) != 0:
                temp = q.pop(0)
                if i < n and (arr[i] < temp.data and arr[i] > temp.min):
                        newNode = NodeDetails(arr[i], temp.min, temp.data)
                        i += 1
                        q.append(newNode)
                if i < n and (arr[i] > temp.data and arr[i] < temp.max):
                        newNode = NodeDetails(arr[i], temp.data, temp.max)
                        i += 1
                        q.append(newNode)

        if i == n:
                return True
        return False

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")
```

```
Yes
```

In [ ]:
```

```