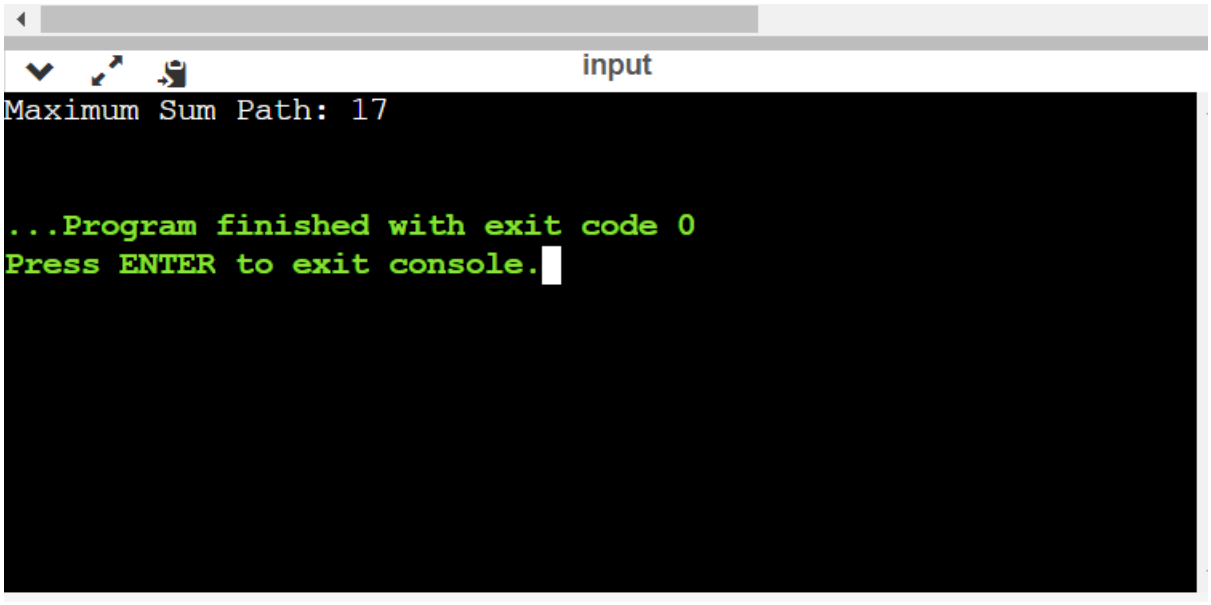
Name=Saransh Mittal
Rollno=2000290130145

Day-19

Problem 1: Write a program to find the maximum sum path in a binary tree. A path in a binary tree is a sequence of nodes where every adjacent pair of nodes are connected by an edge. A node can only appear in the sequence at most once. A path need not pass from the root. We need to find the path with the maximum sum in the binary tree.

```
class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None



def find_max_sum_path(root):
    if root is None:
        return 0

    max_sum = float('-inf')

    def dfs(node):
        nonlocal max_sum
        if node is None:
            return 0

        left_sum = max(dfs(node.left), 0)
        right_sum = max(dfs(node.right), 0)

        current_sum = node.value + left_sum + right_sum
        max_sum = max(max_sum, current_sum)

        return node.value + max(left_sum, right_sum)
```

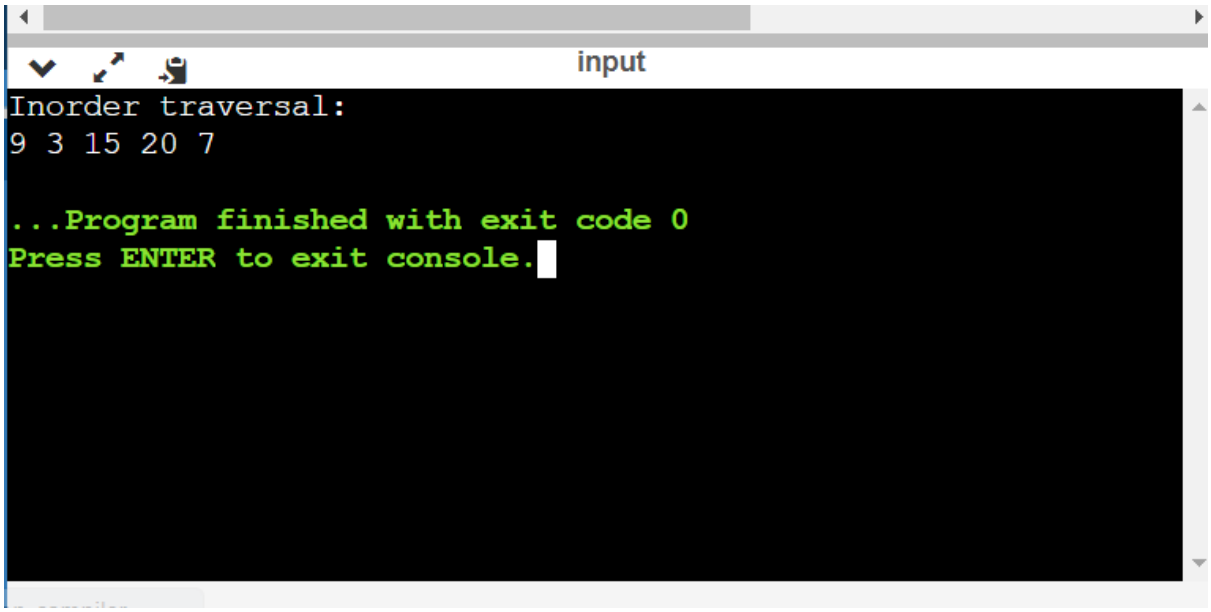
```
    dfs(root)

    return max_sum
root1 = Node(1)
root1.left = Node(2)
root1.right = Node(3)
root1.left.left = Node(4)
root1.left.right = Node(5)
root1.right.right = Node(6)
print("Maximum Sum Path:", find_max_sum_path(root1))
```



Problem 2: Construct A Binary Tree from Inorder and Preorder Traversal.

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

def buildTree(inorder, preorder):
    if not inorder or not preorder:
        return None

    root_val = preorder[0]
    root = TreeNode(root_val)

    root_index = inorder.index(root_val)

    root.left = buildTree(inorder[:root_index], preorder[1:root_index + 1])
    root.right = buildTree(inorder[root_index + 1:], preorder[root_index + 1:])

    return root

def inorderTraversal(root):
    if root:
        inorderTraversal(root.left)
        print(root.val, end=" ")
        inorderTraversal(root.right)

inorder = [9, 3, 15, 20, 7]
preorder = [3, 9, 20, 15, 7]
root = buildTree(inorder, preorder)
print("Inorder traversal:")
```

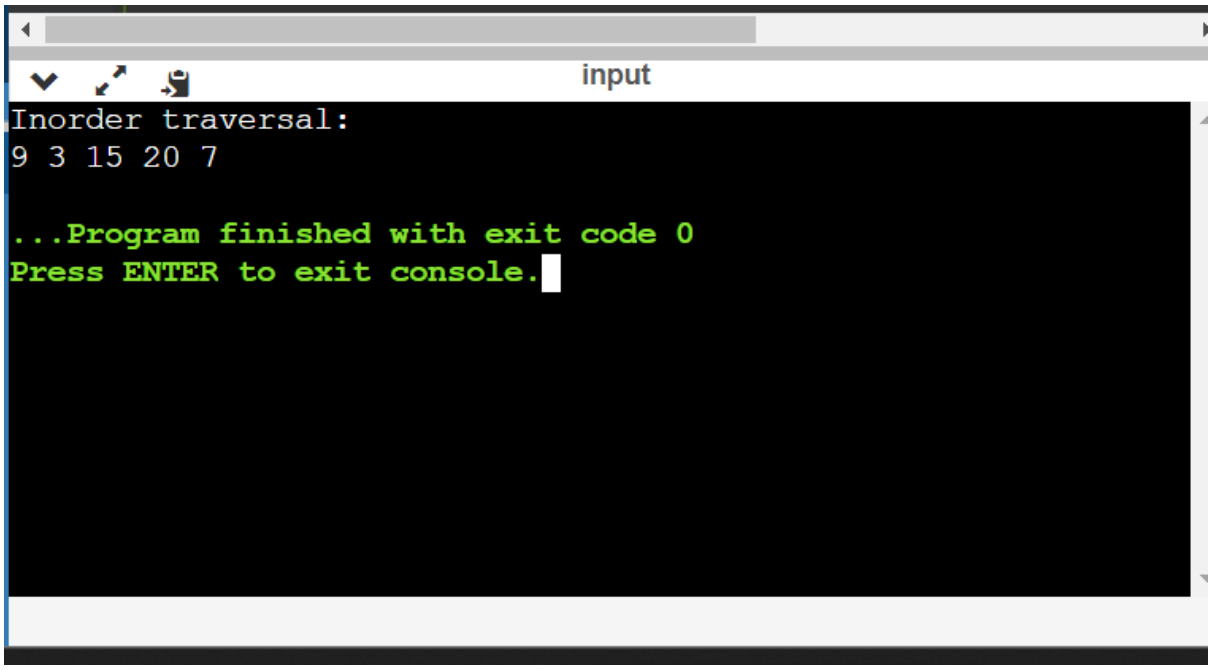
inorderTraversal(root)



Problem 3: Construct A Binary Tree from Inorder and Postorder Traversal.

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


def build_tree(inorder, postorder):
    if not inorder or not postorder:
        return None
    root_val = postorder[-1]
    root = TreeNode(root_val)

    root_index = inorder.index(root_val)

    root.left = build_tree(inorder[:root_index], postorder[:root_index])
    root.right = build_tree(inorder[root_index + 1:], postorder[root_index:-1])

    return root


def inorder_traversal(root):
    if root:
        inorder_traversal(root.left)
        print(root.val, end=' ')
        inorder_traversal(root.right)


def postorder_traversal(root):
    if root:
```

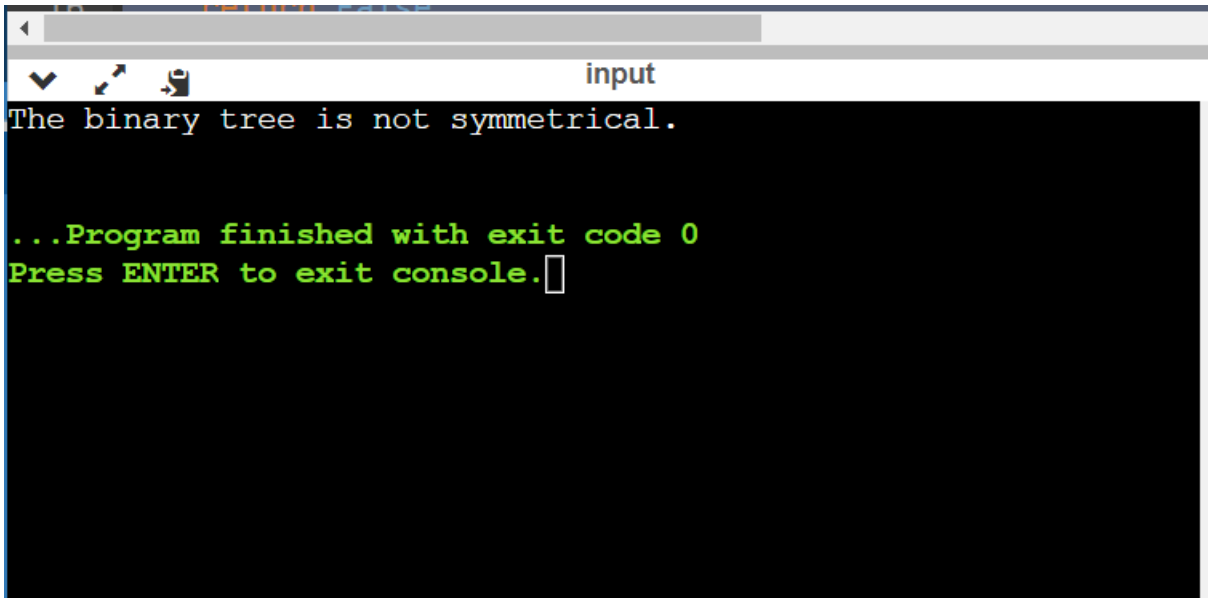
```
        postorder_traversal(root.left)
        postorder_traversal(root.right)
        print(root.val, end=' ')


inorder = [9, 3, 15, 20, 7]
postorder = [9, 15, 7, 20, 3]

tree_root = build_tree(inorder, postorder)

print("Inorder traversal:")
inorder_traversal(tree_root)
```



Problem 4: Write a program to check whether a binary tree is symmetrical or not.

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


def isMirror(root1, root2):
    if root1 is None and root2 is None:
        return True


    if root1 is not None and root2 is not None:
        if root1.data == root2.data:
            return (isMirror(root1.left, root2.right) and
                    isMirror(root1.right, root2.left))


    return False


def isSymmetric(root):
    if root is None:
        return True
    return isMirror(root, root)


root = Node(1)
root.left = Node(2)
root.right = Node(2)
root.left.left = Node(3)
root.left.right = Node(4)
root.right.left = Node(4)
root.right.right = Node(3)
```

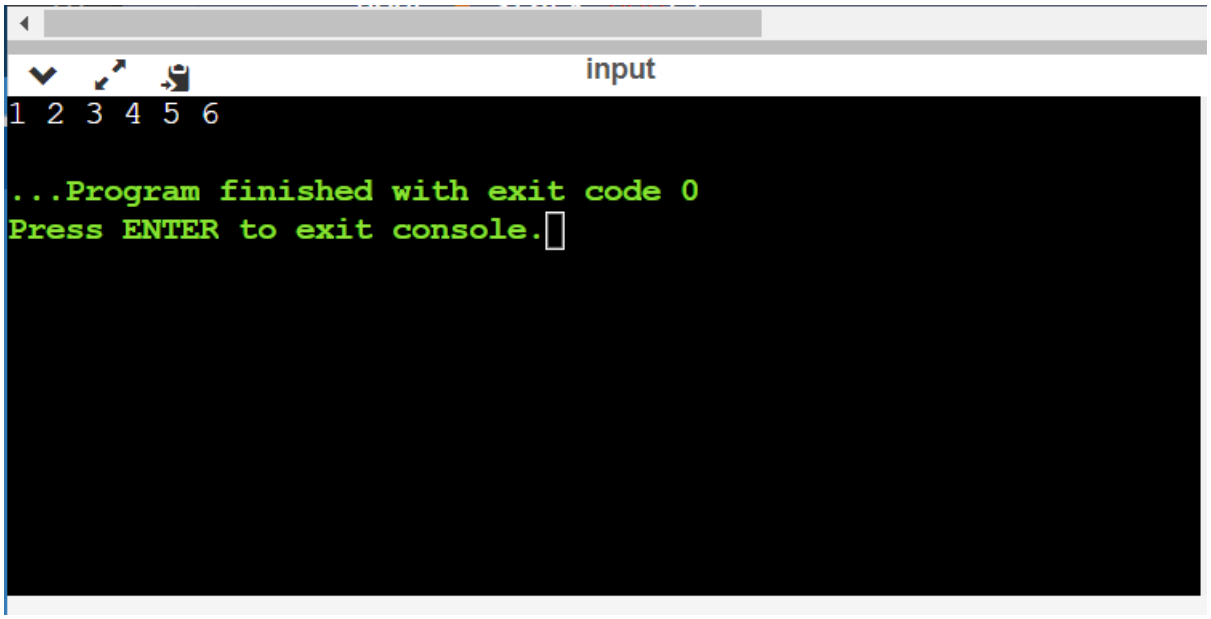
```
root.left.left.left = Node(5)

root.left.left.right = Node(6)

root.right.right.right = Node(5)


if isSymmetric(root):

    print("The binary tree is symmetrical.")

else:

    print("The binary tree is not symmetrical.")
```



Problem 5: Flatten Binary Tree To Linked List. Write a program that flattens a given binary tree to a linked list.

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


class Solution:
    def flatten(self, root):
        if not root:
            return None

        stack = []
        stack.append(root)

        while stack:
            node = stack.pop()
            if node.right:
                stack.append(node.right)
            if node.left:
                stack.append(node.left)
            if stack:
                node.right = stack[-1]
            node.left = None

        return root

root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(5)
```

```
root.left.left = TreeNode(3)
root.left.right = TreeNode(4)
root.right.right = TreeNode(6)
solution = Solution()
flattened = solution.flatten(root)
current = flattened
while current:
    print(current.val, end=" ")
    current = current.right
```



Problem 6: Write a program that converts any binary tree to one that follows the children sum property.

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

def children_sum_property(root):
    if root is None or (root.left is None and root.right is None):
        return

    children_sum_property(root.left)
    children_sum_property(root.right)

    deficit = 0
    if root.left:
        deficit += root.left.data
    if root.right:
        deficit += root.right.data - root.data

    if deficit > 0:
        if root.left:
```

```
            root.left.data += deficit
        else:
            root.left = Node(deficit)
    elif deficit < 0:
        root.data -= deficit

def inorder(root):
    if root:
        inorder(root.left)
        print(root.data, end=" ")
        inorder(root.right)

root = Node(10)
root.left = Node(4)
root.right = Node(6)
root.left.left = Node(3)
root.left.right = Node(1)

print("Original tree:")
inorder(root)
print()

children_sum_property(root)
```

```
print("Modified tree (following children sum property):")

inorder(root)

print()
```