Binary Search Tree - Applications

1. Write a Python program to create a Balanced Binary Search Tree (BST) using an array of elements.

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

def ArrayToBST(array1):
    if not array1:
        return None
    mid = len(array1) // 2
    rootNode = Tree(array1[mid])
    rootNode.left = ArrayToBST(array1[:mid])
    rootNode.right = ArrayToBST(array1[mid+1:])
    return rootNode

def preOrderTraversal(Node):
    if Node:
        print(Node.value, end=" ")
        preOrderTraversal(Node.left)
        preOrderTraversal(Node.right)

array = [1,2,3,4,5,6,7]
b1 = ArrayToBST(array)
preOrderTraversal(b1)
```

Output:

```
4 2 1 3 6 5 7
```

2. Write a Python program to create two binary trees and check whether they are identical.

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

def isIdentical(aRoot,bRoot):
    if aRoot == None and bRoot == None:
        return True
    if aRoot == None or bRoot == None:
        return False

    return (aRoot.value == bRoot.value and
            isIdentical(aRoot.left,bRoot.left) and
            isIdentical(aRoot.right,bRoot.right))

def Tree1():
    rootNode = Tree(1)
    rootNode.left = Tree(2)
    rootNode.right = Tree(3)
    rootNode.left.left = Tree(4)
    rootNode.left.right = Tree(5)
```

Submitted by Nanda Krishnan V , MSc Cybersecurity

```
    return rootNode

def Tree2():
    rootNode = Tree(1)
    rootNode.left = Tree(2)
    rootNode.right = Tree(3)
    rootNode.left.left = Tree(4)
    rootNode.left.right = Tree(5)
    return rootNode

def Tree3():
    rootNode = Tree(1)
    rootNode.left = Tree(24)
    rootNode.right = Tree(23)
    rootNode.left.left = Tree(24)
    rootNode.left.right = Tree(5)
    return rootNode

firstTree = Tree1()
secondTree = Tree2()
thirdTree = Tree3()

identity = isIdentical(firstTree,secondTree)
Random = isIdentical(firstTree,thirdTree)

print("Round One: ")
if identity:
    print("The Trees are Identical!!")
else:
    print("OOPS, They are not Identical")

print("Round Two: ")
if Random:
    print("The Trees are Identical!!")
else:
    print("OOPS, They are not Identical")
```

Output:

```
Round One:
The Trees are Identical!!
Round Two:
OOPS, They are not Identical
PS D:\DUK\Programs\S1>
```

3. Write a Python program to display the largest values in each binary tree level.

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

def largestElement(root):
    if not root:
        return []

    queue = [root]
    max_levels = []

    while queue:
```

```
        level_max = max(node.val for node in queue)
        max_levels.append(level_max)
        queue = [child for node in queue for child in (node.left, node.right) if child]

    return max_levels

def tree1():
    root = Tree(1)
    root.left = Tree(3)
    root.right = Tree(2)
    root.left.left = Tree(5)
    root.left.right = Tree(3)
    root.right.right = Tree(9)
    return root


tree = tree1()
max_values = largestElement(tree)
print("Maximum values at each level:")
for level, max_val in enumerate(max_values):
    print(f"Level {level}: {max_val}")
```

Output:

```
Maximum values at each level:
Level 0: 1
Level 1: 3
Level 2: 9
PS D:\DUK\Programs\S1>
```

4. Write a Python program to check whether a given binary tree is a valid binary search tree (BST) or not

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

def isBST(rootNode, lower=float('-inf'), upper=float('inf')):
    if not rootNode:
        return True

    if rootNode.value <= lower or rootNode.value >= upper:
        return False

    return (isBST(rootNode.left, lower, rootNode.value) and
            isBST(rootNode.right, rootNode.value, upper))

def tree1():
    root = Tree(2)
    root.left = Tree(1)
    root.right = Tree(3)
    return root

def tree2():
    root = Tree(2)
    root.left = Tree(10)
    root.right = Tree(1)
    return root

tree0 = tree1()
tree2 = tree1()
```

```
print("Tree 1:")
if isBST(tree0):
    print("The tree is a valid BST.")
else:
    print("The tree is not a valid BST.")

print("Tree 2:")
if isBST(tree2):
    print("The tree is a valid BST.")
else:
    print("The tree is not a valid BST.")
```

Output:

```
Tree 1:
The tree is a valid BST.
Tree 2:
The tree is a valid BST.
PS D:\DUK\Programs\S1>
```