Advanced Data Structures with Python Laboratory (MCAC294)

Assignment – 2

1. Write a Python program to create a binary tree using recursive function and display that level wise.

```
from collections import deque
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.value = key
def insert_recursive(root, key):
    if root is None:
        return Node(key)
    if key < root.value:
        root.left = insert_recursive(root.left, key)
    else:
        root.right = insert_recursive(root.right, key)
    return root
def level_order_traversal(root):
    if not root:
        return
    queue = deque([root])
    while queue:
        node = queue.popleft()
        print(node.value, end=" ")
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)
root = None
for key in [10, 5, 15, 3, 7, 12, 18]:
    root = insert_recursive(root, key)
print("Level-wise display:")
```

level_order_traversal(root)

Output:-

```
Level-wise display:
10 5 15 3 7 12 18
```

2. Write a Python program to create a binary tree using non-recursive function and display that level wise.

```
from collections import deque
class Node:
    def __init__(self, key):
        self.value = key
        self.left = None
        self.right = None
class BinaryTree:
    def __init__(self):
        self.root = None
    def insert_non_recursive(self, key):
        new_node = Node(key)
        if self.root is None:
            self.root = new_node
            return
        queue = deque([self.root])
        while queue:
            temp = queue.popleft()
            if not temp.left:
                temp.left = new_node
                break
            else:
                queue.append(temp.left)
            if not temp.right:
                temp.right = new_node
                break
            else:
                queue.append(temp.right)
def level_order_traversal(root):
```

```
    if not root:
        return
    queue = deque([root])
    while queue:
        node = queue.popleft()
        print(node.value, end=" ")
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)
bt = BinaryTree()
for key in [10, 5, 15, 3, 7, 12, 18]:
    bt.insert_non_recursive(key)
print("Level-wise display:")
level_order_traversal(bt.root)
```

Output:-

```
Level-wise display:
10 5 15 3 7 12 18
```

3. Write a Python program to create a binary tree using array only and display the tree level wise.

```
class ArrayBinaryTree:
    def __init__(self):
        self.tree = []
    def insert(self, key):
        self.tree.append(key)
    def level_order_traversal(self):
        print(" ".join(map(str, self.tree)))
abt = ArrayBinaryTree()
for key in [10, 5, 15, 3, 7, 12, 18]:
    abt.insert(key)
print("Level-wise display:")
abt.level_order_traversal()
```

Output:-

```
Level-wise display:
10 5 15 3 7 12 18
```

4. Write a Python program to identify the height of a binary tree.

```
class Node:
    def __init__(self, key):
        self.value = key
        self.left = None
        self.right = None
def insert_recursive(root, key):
    if root is None:
        return Node(key)
    if key < root.value:
        root.left = insert_recursive(root.left, key)
    else:
        root.right = insert_recursive(root.right, key)
    return root
def tree_height(root):
    if root is None:
        return -1
    return max(tree_height(root.left), tree_height(root.right)) + 1
root = None
for key in [10, 5, 15, 3, 7, 12, 18]:
    root = insert_recursive(root, key)
print("Height of the tree:", tree_height(root))
```

Output:-

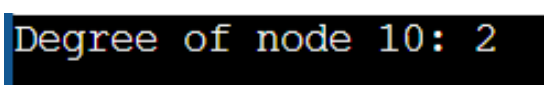
```
Height of the tree: 2
```

5. Write a Python program to identify degree of a given node.

```
class Node:
    def __init__(self, key):
        self.value = key
```

```
        self.left = None
        self.right = None
def insert_recursive(root, key):
    if root is None:
        return Node(key)
    if key < root.value:
        root.left = insert_recursive(root.left, key)
    else:
        root.right = insert_recursive(root.right, key)
    return root
def find_node(root, key):
    if root is None or root.value == key:
        return root
    if key < root.value:
        return find_node(root.left, key)
    return find_node(root.right, key)
def node_degree(node):
    if not node:
        return -1
    degree = 0
    if node.left:
        degree += 1
    if node.right:
        degree += 1
    return degree
root = None
for key in [10, 5, 15, 3, 7, 12, 18]:
    root = insert_recursive(root, key)
node_key = 10
node = find_node(root, node_key)
if node:
    print(f"Degree of node {node_key}: {node_degree(node)}")
else:
    print(f"Node {node_key} not found in the tree.")
```

Output:-



6. Write a Python program to count number of leaf node present in a binary tree.

```
class Node:
    def __init__(self, key):
        self.value = key
        self.left = None
        self.right = None
def insert_recursive(root, key):
    if root is None:
        return Node(key)
    if key < root.value:
        root.left = insert_recursive(root.left, key)
    else:
        root.right = insert_recursive(root.right, key)
    return root
def count_leaf_nodes(root):
    if root is None:
        return 0
    if root.left is None and root.right is None:
        return 1  # It's a leaf node
    return count_leaf_nodes(root.left) + count_leaf_nodes(root.right)
root = None
for key in [10, 5, 15, 3, 7, 12, 18]:
    root = insert_recursive(root, key)
print("Number of leaf nodes:", count_leaf_nodes(root))
```

Output:-

```
Number of leaf nodes: 4
```

7. Write a Python program to count number of internal node present in a binary tree.

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

```
        self.right = None
def insert_recursive(root, key):
    if root is None:
        return Node(key)
    if key < root.value:
        root.left = insert_recursive(root.left, key)
    else:
        root.right = insert_recursive(root.right, key)
    return root
def count_internal_nodes(root):
    if root is None or (root.left is None and root.right is None):
        return 0  # If it's None or a leaf node, return 0
    return 1 + count_internal_nodes(root.left) +
count_internal_nodes(root.right)
root = None
for key in [10, 5, 15, 3, 7, 12, 18]:
    root = insert_recursive(root, key)
print("Number of internal nodes:", count_internal_nodes(root))
```

Output:-

```
Number of internal nodes: 3
```

8. Write a Python program to count number of node present in a given binary tree using linked list.

```
class Node:
    def __init__(self, key):
        self.value = key
        self.left = None
        self.right = None
def insert_recursive(root, key):
    if root is None:
        return Node(key)
    if key < root.value:
        root.left = insert_recursive(root.left, key)
    else:
        root.right = insert_recursive(root.right, key)
```

```
    return root
def count_nodes_linkedlist(root):
    if root is None:
        return 0  # Base case: if tree is empty, return 0
    return 1 + count_nodes_linkedlist(root.left) +
count_nodes_linkedlist(root.right)
root = None
for key in [10, 5, 15, 3, 7, 12, 18]:
    root = insert_recursive(root, key)
print("Total number of nodes:", count_nodes_linkedlist(root))
```

Output:-

```
Total number of nodes: 7
```

9. Write a Python program to count number of node present in a given binary tree using array.

```
class ArrayBinaryTree:
    def __init__(self):
        self.tree = []
    def insert(self, key):
        self.tree.append(key)
    def count_nodes(self):
        return len(self.tree)
abt = ArrayBinaryTree()
for key in [10, 5, 15, 3, 7, 12, 18]:
    abt.insert(key)
print("Total number of nodes:", abt.count_nodes())
```

Output:-

```
Total number of nodes: 7
```

10. Write a Python program to count number of siblings present in a binary tree.

```
class Node:
    def __init__(self, key):
        self.value = key
        self.left = None
        self.right = None
def insert_recursive(root, key):
    if root is None:
        return Node(key)
    if key < root.value:
        root.left = insert_recursive(root.left, key)
    else:
        root.right = insert_recursive(root.right, key)
    return root
def count_sibling_pairs(root):
    if root is None:
        return 0
    count = 0
    if root.left and root.right:
        count += 1  # If both children exist, it's a sibling pair
    return count + count_sibling_pairs(root.left) +
count_sibling_pairs(root.right)
def count_total_siblings(root):
    return count_sibling_pairs(root) * 2
root = None
for key in [10, 5, 15, 3, 7, 12, 18]:
    root = insert_recursive(root, key)
print("Total number of sibling nodes:", count_total_siblings(root))
```

Output:-