Advanced Data Structures with Python Laboratory (MCAC294)

Assignment – 3

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

```
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 inorder_traversal(root):
    if root:
        inorder_traversal(root.left)
        print(root.value, end=" ")
        inorder_traversal(root.right)
root = None
for key in [50, 30, 70, 20, 40, 60, 80]:
    root = insert_recursive(root, key)
print("BST (Inorder Traversal):")
inorder_traversal(root)
```

Output:-

```
BST (Inorder Traversal):
20 30 40 50 60 70 80
```

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

```
class Node:
    def __init__(self, key):
        self.value = key
        self.left = None
        self.right = None
class BST:
    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
        current = self.root
        while True:
            if key < current.value:
                if current.left is None:
                    current.left = new_node
                    break
                current = current.left
            else:
                if current.right is None:
                    current.right = new_node
                    break
                current = current.right
def inorder_traversal(root):
    if root:
        inorder_traversal(root.left)
        print(root.value, end=" ")
        inorder_traversal(root.right)
bst = BST()
for key in [50, 30, 70, 20, 40, 60, 80]:
    bst.insert_non_recursive(key)
print("BST (Inorder Traversal):")
inorder_traversal(bst.root)
```

Output:-

```
BST (Inorder Traversal):
20 30 40 50 60 70 80
```

3. Write a Python program to insert (by using a function) a specific element into an existing binary search tree and then display that.

```
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 inorder_traversal(root):
    if root:
        inorder_traversal(root.left)
        print(root.value, end=" ")
        inorder_traversal(root.right)
root = None
for key in [50, 30, 70, 20, 40, 60, 80]:
    root = insert_recursive(root, key)
root = insert_recursive(root, 55)
print("BST after inserting 55 (Inorder Traversal):")
inorder_traversal(root)
```

Output:-

```
BST after inserting 55 (Inorder Traversal):
20 30 40 50 55 60 70 80
```

4. Write a Python program to search an element in a BST and show the result.

```
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 search_bst(root, key):
    if root is None:
        return None
    if root.value == key:
        return root
    if key < root.value:
        return search_bst(root.left, key)
    return search_bst(root.right, key)
root = None
for key in [50, 30, 70, 20, 40, 60, 80]:
    root = insert_recursive(root, key)
search_key = 40
result = search_bst(root, search_key)
if result:
    print(f"Element {search_key} found in BST")
else:
    print(f"Element {search_key} not found in BST")
```

Output:-

```
Element 40 found in BST
```

5. Write a Python program to take user name as input and display the sorted sequence of characters using BST.

```
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 inorder_traversal(root):
    if root:
        inorder_traversal(root.left)
        print(root.value, end=" ")
        inorder_traversal(root.right)
root = None
name = "binary"
for char in name:
    root = insert_recursive(root, char)
print("Sorted characters using BST:")
inorder_traversal(root)
print()
```

Output:-

```
Sorted characters using BST:
a b i n r y
```

6. Write a Python program to sort a given set of integers using BST.

```
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 inorder_collect(node, sorted_arr):
    if node:
        inorder_collect(node.left, sorted_arr)
        sorted_arr.append(node.value)
        inorder_collect(node.right, sorted_arr)
def bst_sort(arr):
    root = None
    for num in arr:
        root = insert_recursive(root, num)
    sorted_arr = []
    inorder_collect(root, sorted_arr)
    return sorted_arr
arr = [34, 12, 45, 2, 19]
sorted_arr = bst_sort(arr)
print("Sorted Array using BST:", sorted_arr)
```

Output:-

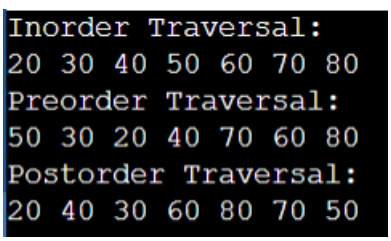
```
Sorted Array using BST: [2, 12, 19, 34, 45]
```

7. Write a Python program to display a BST using In-order, Pre-order, Post-order.

```
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 inorder_traversal(root):
    if root:
        inorder_traversal(root.left)
        print(root.value, end=" ")
        inorder_traversal(root.right)
def preorder_traversal(root):
    if root:
        print(root.value, end=" ")
        preorder_traversal(root.left)
        preorder_traversal(root.right)
def postorder_traversal(root):
    if root:
        postorder_traversal(root.left)
        postorder_traversal(root.right)
        print(root.value, end=" ")
root = None
for key in [50, 30, 70, 20, 40, 60, 80]:
    root = insert_recursive(root, key)
print("Inorder Traversal:")
inorder_traversal(root)
print("\nPreorder Traversal:")
preorder_traversal(root)
print("\nPostorder Traversal:")
postorder_traversal(root)
```

Output:-



8. Write a Python program to Count the number of nodes present in an existing BST and display the highest element present in the BST.

```
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(root):
    if root is None:
        return 0
    return 1 + count_nodes(root.left) + count_nodes(root.right)
def find_max(root):
    if root is None:
        return None
    while root.right:
        root = root.right
    return root.value
root = None
for key in [50, 30, 70, 20, 40, 60, 80]:
    root = insert_recursive(root, key)
print("\nTotal number of nodes in BST:", count_nodes(root))
print("Highest element in BST:", find_max(root))
```

Output:-

```
Total number of nodes in BST: 7
Highest element in BST: 80
```

9. Write a Python program to prove that binary search tree is better than binary tree.

```
import time
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 search_bst(root, key):
    if root is None or root.value == key:
        return root
    if key < root.value:
        return search_bst(root.left, key)
    return search_bst(root.right, key)
def search_unordered_tree(root, key):
    if root is None:
        return None
    if root.value == key:
        return root
    left_result = search_unordered_tree(root.left, key)
    if left_result:
        return left_result
    return search_unordered_tree(root.right, key)
root = None
for key in [50, 30, 70, 20, 40, 60, 80]:
    root = insert_recursive(root, key)
start = time.time()
search_bst(root, 40)
end = time.time()
print("\nBST Search Time:", end - start)
start = time.time()
```

```
search_unordered_tree(root, 40)
end = time.time()
print("Unordered Binary Tree Search Time:", end - start)
```

Output:-

```
BST Search Time: 1.1920928955078125e-06
Unordered Binary Tree Search Time: 2.6226043701171875e-06
```