- **Binary Tree:**

```
class TreeNode:
    def __init__(self, value=0, left=None, right=None):
        self.value = value
        self.left = left
        self.right = right
def inorder(node):
    if node:
        inorder(node.left)
        print(node.value, end=' ')
        inorder(node.right)
def preorder(node):
    if node:
        print(node.value, end=' ')
        preorder(node.left)
        preorder(node.right)
def postorder(node):
    if node:
        postorder(node.left)
        postorder(node.right)
        print(node.value, end=' ')
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.left = TreeNode(4)
root.left.right = TreeNode(5)
print("Inorder Traversal:")
inorder(root)
print("\nPreorder Traversal:")
preorder(root)
print("\nPostorder Traversal:")
postorder(root)
```

- **Output:**

```
In order Traversal:
4 2 5 1 3
Preorder Traversal:
1 2 4 5 3
Post order Traversal:
4 5 2 3 1
```

- **Binary search tree:**
  - ✓ **Insertion :**

```
class TreeNode:
    def __init__(self, value=0, left=None, right=None):
        self.value = value
        self.left = left
        self.right = right
def insert(root, value):
    if root is None:
        return TreeNode(value)
    if value < root.value:
        root.left = insert(root.left, value)
    else:
```

```
  root.right = insert(root.right, value)
    return root
def inorder(node):
    if node:
        inorder(node.left)
        print(node.value, end=' ')
        inorder(node.right)
root = TreeNode(10)
root = insert(root, 5)
root = insert(root, 15)
root = insert(root, 3)
root = insert(root, 7)
print("Inorder Traversal after insertions:")
inorder(root)
```

- ✓ **Output:**

```
3 5 7 10 15
```

- ✓ **Searching:**

```
def search(root, value):
    if root is None or root.value == value:
        return root
    if value < root.value:
        return search(root.left, value)
    else:
        return search(root.right, value)
search_value = 7
found_node = search(root, search_value)
print(f"Search for value {search_value}: {'Found' if found_node else 'Not Found'}")
search_value = 20
found_node = search(root, search_value)
print(f"Search for value {search_value}: {'Found' if found_node else 'Not Found'}")
```

- ✓ **Output:**

```
Search for value 7: Found
Search for value 20: Not Found
```

- ✓ **Deletion:**

```
def find_min(node):
    current = node
    while current.left is not None:
        current = current.left
    return current
def delete(root, value):
    if root is None:
        return root
    if value < root.value:
        root.left = delete(root.left, value)
    elif value > root.value:
        root.right = delete(root.right, value)
    else:
        if root.left is None:
            return root.right
        elif root.right is None:
            return root.left
        temp = find_min(root.right)
        root.value = temp.value
        root.right = delete(root.right, temp.value)
    return root
root = delete(root, 5)
```

```
print("Inorder Traversal after deletion of 5:")
inorder(root)
```

- ✓ **Output:**

```
3 7 10 15
```

- ➢ **Binary Tree traversal:**
  - ✓ **IN order:**

```
class TreeNode:
    def __init__(self, value=0, left=None, right=None):
        self.value = value
        self.left = left
        self.right = right
def inorder(node):
    if node:
        inorder(node.left)
        print(node.value, end=' ')
        inorder(node.right)
# Example tree:
#       4
#      / \
#     2   6
#    / \ / \
#   1  3 5  7
root = TreeNode(4)
root.left = TreeNode(2)
root.right = TreeNode(6)
root.left.left = TreeNode(1)
root.left.right = TreeNode(3)
root.right.left = TreeNode(5)
root.right.right = TreeNode(7)
print("Inorder Traversal:")
inorder(root)
```

  - ✓ **Output:**

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

  - ✓ **Preorder transversal**:

```
def preorder(node):
    if node:
        print(node.value, end=' ')
        preorder(node.left)
        preorder(node.right)
print("Preorder Traversal:")
preorder(root)
```

  - ✓ **Output:**

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

  - ✓ **Post order transversal:**

```
def postorder(node):
    if node:
        postorder(node.left)
        postorder(node.right)
        print(node.value, end=' ')

print("Postorder Traversal:")
postorder(root)
```

  - ✓ **Output:**

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