Data Structures & Algorithms – Lab #9

Aim: Getting Familiar with Traversing a Binary Search Tree using **DFS** and **BFS** Algorithms

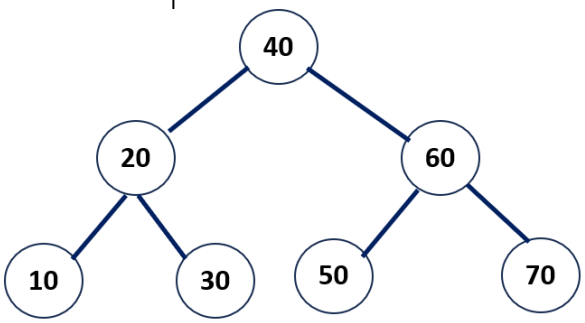
Topics:

1. **Depth-First Search (DFS)**
    a. In-Order Traversal
    b. Pre-Order Traversal
    c. Post-Order Traversal
2. **Breadth-First Search (BFS)**

Lab Questions

Q1 – Use **binarytree** library to create and visualize the following BST.

```
from binarytree import Node

###### Creating all Node objects in the BST ##
root = Node(40)
node20 = Node(20)
node10 = Node(10)
node30 = Node(30)
node60 = Node(60)
node50 = Node(50)
node70 = Node(70)

### Assigning left and right nodes to each node ###
root.left = node20
root.right = node60

node20.left = node10
node20.right = node30

node60.left = node50
node60.right = node70

################## BST Visualization ###################
print(root)
```



Q2 – Implement <u>all three ways of **DFS**</u> to traverse the BST. (**In-Order**, **Pre-Order**, **Post-Order**)

```
################ DFS (In-order Traversal) ############
def inorder(root):
    if root is None:
        return []

    result = []
    result.extend(inorder(root.left))
    result.append(root.value)
    result.extend(inorder(root.right))

    return result

################ Calling In-order Function ############
print("In-order --> ", inorder(root))
```

```
################ DFS (Pre-order Traversal) ############
def preorder(root):
    if root is None:
        return []

    result = []
    result.append(root.value)
    result.extend(preorder(root.left))
    result.extend(preorder(root.right))

    return result

################ Calling pre-order Function ############
print("Pre-order --> ", preorder(root))
```

```
################ DFS (Post-order Traversal) ############
def postorder(root):
    if root is None:
        return []

    result = []
    result.extend(postorder(root.left))
    result.extend(postorder(root.right))
    result.append(root.value)

    return result

################ Calling Post-order Function ############
print("Post-order --> ", postorder(root))
```

Q3 – Implement **BFS (Level-Order)** to traverse the BST.

```
from collections import deque
def BFS(root):
    if root is None:
        return []

    result = []
    queue = deque([root])

    while queue:
        level_size = len(queue)
        visited = []

        for i in range(level_size):
            node = queue.popleft()
            visited.append(node.value)

            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)

        result.append(visited)
    return result

############## BFS Traversal ###################
print("BFS --> ",BFS(root))
```