Assignment No. 05

Title: Convert Given Binary Tree into a Threaded Binary Tree and Analyze Time & Space Complexity

Objectives:

- Understand the concept of a Threaded Binary Tree.
- Learn the implementation of converting a binary tree into a one-way threaded binary tree.
- Analyze the time and space complexity of the algorithm.

Introduction: A binary tree is a hierarchical data structure where each node has at most two children: left and right. Traditional binary trees require additional storage and traversal techniques to efficiently navigate through the tree. A **Threaded Binary Tree (TBT)** helps in optimizing tree traversal by utilizing empty right child pointers to store in-order successors, thus reducing the need for stack or recursive traversal.

A **One-Way Threaded Binary Tree** is a special type of binary tree where the right NULL pointers are converted into threads pointing to the in-order successor of the node. This allows in-order traversal without using additional stack space or recursion.

Theory:

1. **Threading in Binary Trees:**
    - A NULL right child is replaced with a pointer to its in-order successor.
    - This allows an efficient in-order traversal without recursion.
2. **Properties of One-Way Threaded Binary Trees:**
    - Right threads replace NULL pointers and point to the in-order successor.
    - Left pointers remain unchanged.
    - Traversal is more efficient compared to a traditional binary tree.
3. **Advantages:**
    - Eliminates the need for stack or recursion during traversal.
    - Reduces space complexity.
    - Improves traversal efficiency.

Implementation Details: The given C++ implementation follows these steps:

1. **Node Structure:** Each node consists of:

- data: Stores the value.
- left: Pointer to the left child.
- right: Pointer to the right child or in-order successor.
- rightThread: Boolean flag indicating if right is a thread.

2. **Insertion:**
   - If the right pointer is NULL, it is replaced with a thread pointing to the in-order successor.
   - The tree is traversed using standard binary search tree (BST) insertion logic.
3. **Leftmost Function:**
   - Finds the leftmost node in a subtree.
4. **Inorder Traversal:**
   - Starts from the leftmost node and follows in-order traversal using threads.
   - If a node has a thread, it follows it instead of recursively traversing.

Algorithm Analysis:

1. **Time Complexity:**
   - **Insertion:** O(h), where h is the height of the tree (in a balanced tree, h = O(log n)).
   - **Traversal (In-order):** O(n), as each node is visited once.
2. **Space Complexity:**
   - **O(n)** for storing n nodes.
   - **O(1)** additional space for traversal since no recursion or stack is used.

Conclusion: A **One-Way Threaded Binary Tree** optimizes traversal by replacing NULL right pointers with in-order successor links, reducing space complexity while maintaining efficient traversal operations. The implemented algorithm efficiently inserts nodes and performs in-order traversal without using extra stack space.