Implementing Binary Tree

This C++ program demonstrates the creation and traversal of a binary tree. It provides functionality to insert nodes at specified positions and performs three types of tree traversals: level-order traversal, in-order traversal, pre-order traversal, and post-order traversal.

1. **Function: bool insert(int parentValue, int value, char childPosition)**
    - **Explanation:** This function inserts a new node with the given **value** as a child of the node with the specified **parentValue**. The **childPosition** specifies whether the new node should be the left ('L') or right ('R') child of the parent node.
        - Starts from the root and recursively searches for the parent node with a value matching **parentValue**.
        - Checks if the desired child position is available (left or right).
        - Inserts the new node if available; otherwise, returns **false**.
    - **Time Complexity: O(n) - In the worst case, the function may traverse all nodes in the tree.**
    - **Space Complexity: The insert function has a space complexity of O(h), where 'h' is the height of the binary tree.**
2. **Function: void levelOrderTraversal()**
    - **Explanation:** This function performs a level-order traversal of the binary tree, printing the values of nodes level by level.
        - Utilizes a queue to traverse the tree level by level.
        - Enqueues each node's left and right children, ensuring nodes at the same level are processed first.
    - **Time Complexity: O(n) - It visits all nodes once.**
    - **Space Complexity: O(w) - Determined by the maximum width (number of nodes in the level with the most nodes).**
3. **Functions: void inOrderTraversal(Node* root), void preOrderTraversal(Node* root), void postOrderTraversal(Node* root)**
    - **Explanation:** These functions are helper functions for in-order, pre-order, and post-order traversals, respectively. They print the values of nodes while traversing the tree in their respective orders.
        - Use recursive algorithms to traverse the tree.

        - In-order traversal visits left subtree, current node, and right subtree.
        - Pre-order traversal visits the current node before its subtrees.
        - Post-order traversal visits the subtrees before the current node.
    - **Time Complexity: O(n) - All three functions visit all nodes once.**
    - **Space Complexity: O(h) - Determined by the maximum depth of recursion (height of the tree).**
4. **Function: void destroyTree(Node* node)**
    - **Explanation:** This is a private recursive function used by the destructor to delete nodes and release memory.
        - Traverses the tree in a post-order manner (left subtree, right subtree, current node).
        - Deletes nodes as it goes.
    - **Time Complexity: O(n) - Visits and deletes all nodes once.**
    - **Space Complexity: O(h) - Determined by the maximum depth of recursion (height of the tree).**