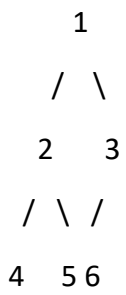
Complete Binary Tree

A complete binary tree is a special type of binary tree in which every level, except possibly the last, is completely filled, and all nodes are as left as possible. In other words, it's a binary tree where nodes are added from left to right on each level, and the last level may not be completely filled, but it should be filled from left to right.

Example:



1. **Node** Class:
    - **Node** represents a node in the binary tree.
    - It has three member variables:
        - **left**: A pointer to the left child node.
        - **right**: A pointer to the right child node.
        - **value**: An integer value stored in the node.
    - The constructor initializes the **value** and sets **left** and **right** to **nullptr**.

2. **CompleteBinaryTree** Class:
    - **CompleteBinaryTree** represents a complete binary tree and includes methods to manipulate and traverse the tree.
    - Member variables:
        - **root**: A pointer to the root node of the binary tree.
    - Public member functions:
        - **insert(int val)**: Inserts a value into the binary tree in a way that maintains the tree's completeness.
        - **levelOrderTraversal()**: Performs a level-order traversal of the tree and prints the nodes at each level.

- **inOrderTraversal()**: Initiates in-order traversal from the root and prints the nodes in sorted order.
- **getRoot()**: Gets the root node of the binary tree.
- **deleteTree(Node* root)**: Deletes the entire binary tree to free memory.

insert(int val):

- This function inserts a value into the binary tree in a way that maintains the completeness of the tree.
- It uses a level-order traversal approach with a queue to find the appropriate position for insertion.
- The function starts from the root and traverses the tree level by level.
- If a node has an empty left child, the new node is inserted as the left child; otherwise, it is inserted as the right child.
- **Time Complexity: O(N), where N is the number of nodes in the binary tree (worst case is O(N)).**
- **Space Complexity: O(M), where M is the maximum number of nodes at any level of the tree.**

levelOrderTraversal():

- This function performs a level-order traversal of the binary tree using a queue.
- It prints the nodes at each level, starting from the root.
- **Time Complexity: O(N), where N is the number of nodes in the binary tree.**
- **Space Complexity: O(M), where M is the maximum number of nodes at any level of the tree (space used by the queue).**

inOrderTraversal(Node* root):

- This is a recursive function that performs in-order traversal of the binary tree.
- It prints the nodes in sorted order.
- **Time Complexity: O(N), where N is the number of nodes in the binary tree.**
- **Space Complexity: O(H), where H is the height of the binary tree (space used by the recursive call stack).**

deleteTree(Node* root):

- This function recursively deletes the entire binary tree to free memory.
- It deletes nodes in a post-order fashion to ensure that child nodes are deleted before their parents.

- **Time Complexity: O(N), where N is the number of nodes in the binary tree.**
- **Space Complexity: O(H), where H is the height of the binary tree (space used by the recursive call stack).**