Revision Notes: Level Order Traversal of a Binary Tree

Problem Statement

Given the root of a binary tree, return the level order traversal of its nodes' values. (i.e., from left to right, level by level).

Example 1:

Input: root = [3,9,20,null,null,15,7]

Output: [[3],[9,20],[15,7]]

Example 2:

Input: root = [1]

Output: [[1]]

Example 3:

Input: root = []

Output: []

Constraints:

- The number of nodes in the tree is in the range [0, 2000].
- -1000 <= Node.val <= 1000

Code Implementation:

```
class Solution {
public:
    int level(TreeNode* root){
        if(root == NULL) return 0;
        return 1 + max(level(root->right), level(root->left));
    }

    void nth_level(TreeNode* root, int lvl, int t_lvl, vector<in
        if(root == NULL) return;
        if(lvl == t_lvl){
            v.push_back(root->val);
            return;
        }
        nth_level(root->left, lvl+1, t_lvl, v);
        nth_level(root->right, lvl+1, t_lvl, v);
    }

    void lOrder(TreeNode* root, vector<vector<int>>& ans){
        int n = level(root);
        for(int i = 0; i < n; i++){
            vector<int> v;
            nth_level(root, 0, i, v);
            ans.push_back(v);
        }
    }

    vector<vector<int>> levelOrder(TreeNode* root){
        vector<vector<int>> ans;
        lOrder(root, ans);
        return ans;
    }
};
```

Explanation & Logic

1. Finding the Tree Depth (level function)

- This function calculates the depth of the binary tree recursively.
- If the root is NULL, return 0.
- Otherwise, return 1 + max(depth of left subtree, depth of right subtree).

2. Traversing Nodes at a Given Level (nth_level function)

- This function is used to collect all node values at a specific level (t_lvl).
- If the current level (lvl) matches t_lvl, add the node's value to the vector.
- Recursively call this function for left and right subtrees with lvl + 1.

3. Collecting Nodes for Each Level (lOrder function)

- First, find the depth of the tree (n).
- Iterate from level 0 to n-1, calling nth_level to collect node values for each level.
- Store the collected values in ans.

4. Final Level Order Traversal (levelOrder function)

- Calls lOrder to populate the answer vector and returns it.

Time Complexity Analysis

- level() function runs in **O(N)** time (worst-case scenario for skewed trees).
- nth_level() is called for every level, leading to a **worst-case complexity of O(N^2)** for skewed trees.
- **Overall Complexity: O(N^2)** (not the most optimal approach but works well for small constraints).

Alternative Approach (Using Queue - BFS)

- A more efficient approach is to use **Breadth-First Search (BFS) with a queue**.
- This approach processes nodes level by level in **O(N)** time complexity.

Key Takeaways

- The given approach follows a recursive method to perform level order traversal.
- It uses a depth-first strategy to collect node values at each level.
- While it works correctly, a BFS-based approach using a queue is more efficient. And that too is mentioned in after this approach…
