 # Level Order Traversal of a Binary Tree (BFS Approach)

Problem Statement:

Given the root of a binary tree, return the level order traversal of its nodes' values. That is, traverse the tree level by level from left to right.

Example 1:

Input:

Output:

1 2 3 4 5 6 7

Explanation & Logic

The approach used here is **Breadth-First Search (BFS)** using a **queue**:

1. **Initialize a Queue:**
    - Push the root node into the queue.
2. **Process Nodes Level by Level:**
    - While the queue is not empty:
        - Extract the front node.
        - Print its value.
        - Push its left and right children into the queue (if they exist).

Code Implementation

```
#include<iostream>
#include<queue>
using namespace std;

class node {
public:
    int val;
    node* right;
    node* left;
    node(int val) {
        this->val = val;
        right = NULL;
        left = NULL;
    }
};

void level_wise_display(node* root, queue<node*>& q) {
    if (root == NULL) return;
    q.push(root);
    while (q.size()>0) {
        node* temp = q.front();
        q.pop();
        cout << temp->val << " ";
        if (temp->left != NULL) q.push(temp->left);
        if (temp->right != NULL) q.push(temp->right);
    }
};
```

```
int main() {
    node* a = new node(1);
    node* b = new node(2);
    node* c = new node(3);
    node* d = new node(4);
    node* e = new node(5);
    node* f = new node(6);
    node* g = new node(7);

    a->left = b;
    a->right = c;
    b->left = d;
    b->right = e;
    c->left = f;
    c->right = g;

    queue<node*> q;
    level_wise_display(a, q);

    return 0;
}
```

Dry Run of the Code

Step-by-Step Execution

1. **Initialization:**
    - q = {1}
    - Output: 1
    - Enqueue left child 2 and right child 3
    - q = {2,3}
2. **Processing Node 2:**
    - q = {3}
    - Output: 1 2
    - Enqueue left child 4 and right child 5

- q = {3,4,5}

3. **Processing Node 3:**
   - q = {4,5}
   - Output: 1 2 3
   - Enqueue left child 6 and right child 7
   - q = {4,5,6,7}
4. **Processing Node 4:**
   - q = {5,6,7}
   - Output: 1 2 3 4
5. **Processing Node 5:**
   - q = {6,7}
   - Output: 1 2 3 4 5
6. **Processing Node 6:**
   - q = {7}
   - Output: 1 2 3 4 5 6
7. **Processing Node 7:**
   - q = {} (empty queue, traversal ends)
   - Output: 1 2 3 4 5 6 7

Time Complexity Analysis

- Each node is **visited once** and pushed into the queue **once**.
- Each node is removed from the queue **once**.
- Thus, **O(N) time complexity**, where N is the number of nodes in the tree.

Conclusion

- This BFS-based approach is **optimal for level order traversal**.
- It ensures that all nodes are processed level by level.
- Uses **O(N) space complexity** due to queue usage.

This method is widely used in real-world applications like **finding shortest paths in unweighted graphs** and **handling hierarchical structures in databases**. 🚀