

13
.4

Level Order Traversal

MEDIUM

■ What You Will Learn

- ✓ BFS on trees using a Queue
- ✓ How to group nodes by level (2D output)
- ✓ Level-by-level processing pattern
- ✓ Why BFS is fundamentally different from DFS traversals

■ Intuition First

Level Order means visiting nodes level by level — all nodes on depth 0 first, then all on depth 1, then depth 2, and so on. Think of it like floors in a building — you visit every room on floor 1 before going to floor 2.

The secret weapon: use a Queue (FIFO). Add root, then process level by level. When you dequeue a node, enqueue its children. Children stay in order, level by level.

■ BFS with Queue

- Add root to queue
- While queue is not empty:
 - → Record queue size (= number of nodes at current level)
 - → Process exactly that many nodes (dequeue, add to level list)
 - → For each dequeued node: enqueue left child and right child
- Add level list to result after processing each level

■ Visual — Level by Level

Tree with 3 levels

```
3 ← Level 0
/ \
9 20 ← Level 1
/ \
15 7 ← Level 2
```

Level 0: [3]
Level 1: [9, 20]
Level 2: [15, 7]
Output: [[3],[9,20],[15,7]] ✓

■ Dry Run

Step 1	Queue: [3]. Size=1. Dequeue 3 → level=[3]. Enqueue 9, 20.
Step 2	Queue: [9,20]. Size=2. Dequeue 9 → level=[9]. No children. Dequeue 20 → level=[9,20]. Enqueue 15,7.
Step 3	Queue: [15,7]. Size=2. Dequeue 15 → level=[15]. No children. Dequeue 7 → level=[15,7]. No children.
Step 4	Queue empty → DONE. Result: [[3],[9,20],[15,7]] ✓

■ Java Code

```
public static List<List<Integer>> levelOrder(TreeNode root) {
    List<List<Integer>> result = new ArrayList<>();
    if (root == null) return result;
    Queue<TreeNode> queue = new LinkedList<>();
    queue.offer(root);
    while (!queue.isEmpty()) {
        int levelSize = queue.size(); // Nodes at current level
        List<Integer> level = new ArrayList<>();
        for (int i = 0; i < levelSize; i++) {
            TreeNode node = queue.poll();
            level.add(node.val);
            if (node.left != null) queue.offer(node.left);
            if (node.right != null) queue.offer(node.right);
        }
        result.add(level);
    }
    return result;
}
```

■ Complexity

Case	Time	Space	What it means
All Cases	O(n)	O(n)	Every node visited once. Queue holds at most O(w) nodes (w = max width)

■ Key: queue.size() Before the Inner Loop

- The queue at any point holds ALL nodes of the NEXT level

- Recording size BEFORE the inner loop gives the exact count for this level
- As you process this level, you're adding next level to queue — but size is already fixed
- This is the 'snapshot' trick that separates levels cleanly

■ Common Mistakes

- Not recording levelSize before inner loop — queue grows as you add children!
- Using Stack instead of Queue — Stack gives wrong (DFS) order
- Forgetting null check before enqueueing children

■ Pro Tips — Level Order Variations

- Zigzag level order: alternate left-right per level (use deque)
- Right side view: take last element of each level
- Left side view: take first element of each level
- Average of levels: sum each level / levelSize

■ Visual Thinking

Picture water filling a bowl with a tree submerged in it.

Water fills level 0 (root) first, then level 1, then level 2.

Queue is like a waiting room — everyone from the current floor waits, then all go up together.

Animation: nodes illuminate level by level, top to bottom, left to right.

The 'levelSize snapshot' is like closing the door on the current floor before going to next.

■ Practice Problems

[Med] Right side view of binary tree (last node per level)

[Med] Zigzag level order traversal

[Med] Average value of nodes on each level

[Hard] Connect all nodes at same level using next pointer

■ Quick Revision Box

- Level Order = BFS using Queue
- Record `queue.size()` = `levelSize` BEFORE inner loop
- Inner loop processes exactly `levelSize` nodes
- Enqueue left and right children for each processed node
- $O(n)$ time, $O(n)$ space (queue width)
- Basis for: right view, zigzag, level averages