

Explain from the very basics

Alright Sparsh, let's break this down step-by-step from the absolute basics so it's crystal clear.

The code you've shared is implementing **Level Order Traversal** (also called **Breadth-First Search, BFS**) of a **Binary Tree**.

### 1 Understanding the TreeNode class

In Java, a binary tree node is represented by:

```
java
public class TreeNode {
    int val; // The value/data stored in the node
    TreeNode left; // Pointer/reference to the left child
    // ... (Additional fields like right child) ...

    // Constructors
    TreeNode() {} // Empty constructor
    TreeNode(int val) { this.val = val; } // Constructor with value
    TreeNode(int val, TreeNode left, TreeNode right) {
        this.val = val;
        this.left = left;
        this.right = right;
    }
}
```

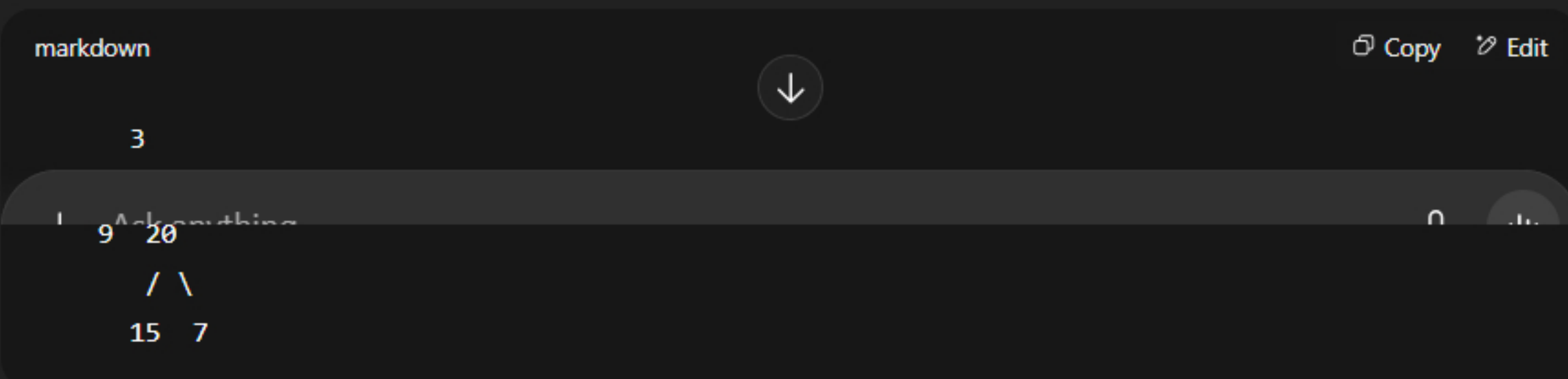
This structure basically means:



### 2 What is Level Order Traversal?

Level order means visiting all nodes **level by level** from **top to bottom** and **left to right**.

Example:



Level order output:

```
csharp
[
  [3],
  [9, 20],
  [15, 7]
]
```

### 3 The BFS (Queue-based) Idea

We use a **queue** (FIFO structure) to keep track of nodes we need to visit.

Steps:

1. Add root node to the queue.
2. While the queue is not empty:
  - Find how many nodes are in the current level (`queue.size()`).
  - Process each node:
    - Add its value to the current level list.
    - Add its left child to the queue (if not null).
    - Add its right child to the queue (if not null).
3. Add the current level list to the final answer.
4. Repeat until all levels are processed.

### 4 Code Walkthrough

```
java
public List<List<Integer>> levelOrder(TreeNode root) {
    Queue<TreeNode> queue = new LinkedList<>(); // Queue to store nodes
    List<List<Integer>> wrapList = new LinkedList<>(); // Final result

    if (root == null) return wrapList; // Edge case: empty tree

    queue.offer(root); // Start with the root node

    while (!queue.isEmpty()) {
        int levelNum = queue.size(); // Number of nodes in current level
        List<Integer> sublist = new LinkedList<>();

        for (int i = 0; i < levelNum; i++) {
            // Look at first node in queue without removing it
            if (queue.peek().left != null)
                queue.offer(queue.peek().left); // Add left child
            if (queue.peek().right != null)
                queue.offer(queue.peek().right); // Add right child

            // Remove the node from the queue and add its value to the current level list
            int val = queue.poll().val;
            sublist.add(val);
        }

        // Add the current level list to the final answer
        wrapList.add(sublist);
    }

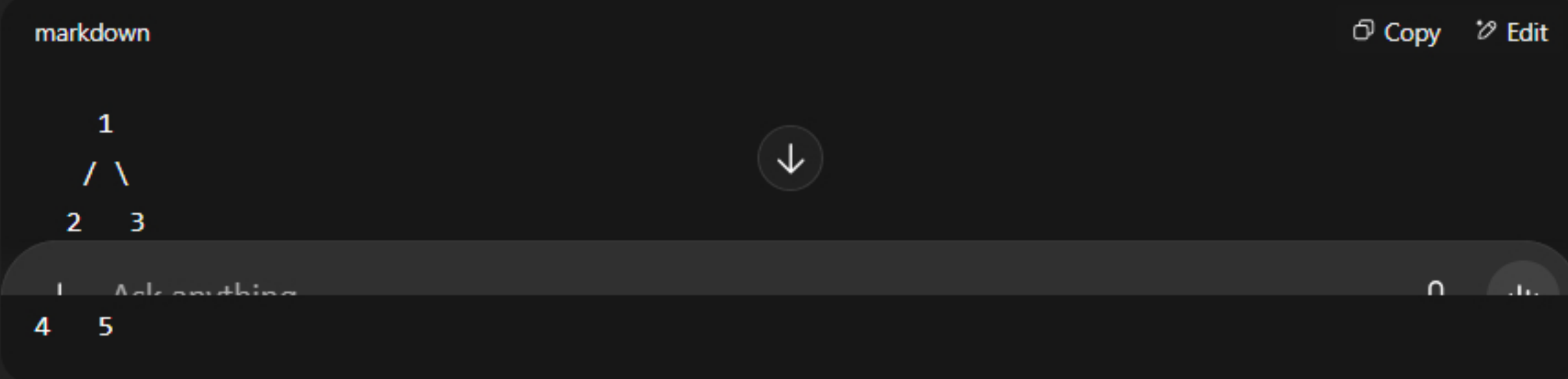
    return wrapList;
}
```

### 5 Time & Space Complexity

- **Time Complexity (TC):  $O(N)$** 
  - Every node is visited **once**.
  - $N$  = number of nodes in the tree.
- **Space Complexity (SC):  $O(N)$** 
  - Worst case: the queue holds all nodes in the largest level (last level of a complete binary tree).

### 6 Step-by-step Example

For tree:



Process:

1. Queue: [1] → Process 1 → Add 2, 3 → Result: [[1]]
2. Queue: [2, 3] → Process 2, 3 → Add 4, 5 → Result: [[1], [2, 3]]
3. Queue: [4, 5] → Process 4, 5 → Result: [[1], [2, 3], [4, 5]]
4. Queue empty → Stop.

If you want, I can also **redraw this exact example visually** showing the queue content at each step so you can see the BFS in action.

That would make the working of `queue.peek()` and `queue.poll()` extra clear.