📬 Queue Interview Quick-Check Pattern

A high-speed mental framework to crack queue and BFS-based problems with confidence.

---

1. 🧠 When to Use a Queue?

Ask yourself:

- ❓ Am I processing things **in the order they came** (FIFO)?
- ❓ Is the problem **level by level**, **step by step**, or **time-based**?
- ❓ Do I need to **spread**, **simulate** or **traverse a graph/tree**?
- ❓ Do I need a **sliding window**, **rate limiter**, or **task scheduler**?
- ❓ Am I tracking **shortest path**, **time to infect/rot**, or **distance**?

> 💡 Queue is your go-to when order matters and you're simulating **real-time processing** or **layered traversal**.

---

2. 🔍 Common Queue Use Cases

Problem Type	Queue Helps With…
BFS on Tree or Graph	Layered traversal
Level Order Traversal	Tree nodes level by level
Rotting Oranges / Infection	Multi-source BFS
Word Ladder / Shortest Path	BFS with steps
Sliding Window Max/Min	Monotonic Queue
First Non-Repeating Character	Queue + Frequency Map
Task Scheduling	Simulate delay/gaps using queue

---

3. 🔩 Must-Know Queue Templates

✅ Level Order Traversal (Tree BFS)

```
function levelOrder(root) {
  if (!root) return [];
  const queue = [root], res = [];

  while (queue.length) {
    const levelSize = queue.length, level = [];
    for (let i = 0; i < levelSize; i++) {
      const node = queue.shift();
      level.push(node.val);
      if (node.left) queue.push(node.left);
      if (node.right) queue.push(node.right);
    }
    res.push(level);
  }

  return res;
}
```

✅ Rotting Oranges (Multi-Source BFS)

```
function orangesRotting(grid) {
  const rows = grid.length, cols = grid[0].length;
  const queue = [], directions = [[1,0],[-1,0],[0,1],[0,-1]];
  let fresh = 0, minutes = 0;

  for (let r = 0; r < rows; r++) {
    for (let c = 0; c < cols; c++) {
      if (grid[r][c] === 2) queue.push([r, c]);
      if (grid[r][c] === 1) fresh++;
    }
  }

  while (queue.length && fresh > 0) {
```

```
    let size = queue.length;
    while (size--) {
      const [x, y] = queue.shift();
      for (let [dx, dy] of directions) {
        const nx = x + dx, ny = y + dy;
        if (nx >= 0 && ny >= 0 && nx < rows && ny < cols &&
grid[nx][ny] === 1) {
          grid[nx][ny] = 2;
          queue.push([nx, ny]);
          fresh--;
        }
      }
    }
    minutes++;
  }

  return fresh === 0 ? minutes : -1;
}
```

✅ Word Ladder (Shortest Path using Queue)

```
function ladderLength(beginWord, endWord, wordList) {
  const set = new Set(wordList);
  if (!set.has(endWord)) return 0;

  const queue = [[beginWord, 1]];
  while (queue.length) {
    const [word, depth] = queue.shift();
    if (word === endWord) return depth;

    for (let i = 0; i < word.length; i++) {
      for (let c = 97; c <= 122; c++) {
        const next = word.slice(0, i) + String.fromCharCode(c) +
word.slice(i + 1);
        if (set.has(next)) {
          queue.push([next, depth + 1]);
          set.delete(next);
        }
      }
```

```
    }
  }

  return 0;
}
```

✅ Monotonic Queue (Sliding Window Max)

```
function maxSlidingWindow(nums, k) {
  const deque = [], res = [];

  for (let i = 0; i < nums.length; i++) {
    while (deque.length && deque[0] <= i - k) deque.shift();
    while (deque.length && nums[i] >= nums[deque[deque.length - 1]])
deque.pop();
    deque.push(i);

    if (i >= k - 1) res.push(nums[deque[0]]);
  }

  return res;
}
```

---

4. 🧱 Edge Cases to Watch

- Empty input / edge size window
- Queue underflow (popping empty)
- Infinite loops from cyclic graphs (track visited)
- Skewed trees (queue may not be balanced)
- Multiple sources (add all sources at once)
- Off-by-one for time/level counting
- Monotonic queue mistakes (wrong direction or cleanup)

---

5. 🧠 Mental Model for Queue Problems

Trigger Words	Pattern Used
“Layer by layer”	BFS (Queue)
“Min steps”, “shortest path”	BFS with queue
“Spread infection/time”	Multi-source BFS
“Window” or “Range max”	Monotonic queue (deque)
“First in, first out”	Basic Queue
“Stream or Live Data”	Queue with constant cleanup

🔁 Problem Solving Loop

1. ❓ Am I processing **in order** (FIFO)?
2. 📦 Do I need to **track level/time/steps**?
3. 🔄 Should I push **once or multiple times** (multi-source)?
4. 🧠 Do I need a **visited set** to avoid cycles?
5. 🧪 What are the **window boundaries**?

✅ Final Interview Checklist

- Am I using queue to maintain **correct order**?
- Do I track levels/time if needed?
- Did I account for **multiple sources** in BFS?
- Is my queue **growing/shrinking correctly**?
- Do I handle **empty input / single element / no solution**?