1. Definition (BFS)

Breadth-First Search (BFS) is a graph/tree traversal algorithm that explores all nodes **level by level**, using a **queue**.
It guarantees the **shortest path** in **unweighted** graphs by visiting closer nodes before deeper ones.

2. Solution Approach (Short)

1. Create a **queue** and a **visited** array/set.
2. Mark the **start node** as visited and push it into the queue.
3. While the queue is not empty:
    - Pop the front node
    - Visit all its neighbors
    - If a neighbor is unvisited → mark visited + enqueue
4. Continue until all reachable nodes are processed (or target is found).

3. Hint (General BFS Tips)

- Use a **queue** (FIFO) — that's the core of BFS.
- Always maintain a **visited** set to avoid loops.
- BFS naturally finds the **shortest path** in unweighted graphs.
- Perfect for: levels, distances, grids, connectivity, shortest path.

4. Pseudocode (C++ Style)

```
void BFS(int start, vector<vector<int>>& graph) {
   int n = graph.size();
   vector<bool> visited(n, false);
   queue<int> q;

   visited[start] = true;
   q.push(start);

   while (!q.empty()) {
      int node = q.front();
```

```
        q.pop();

        // process node
        cout << node << " ";

        for (int nbr : graph[node]) {
            if (!visited[nbr]) {
                visited[nbr] = true;
                q.push(nbr);
            }
        }
    }
}
```

5. Hint :

- **Queue → FIFO → Level order traversal**
- Use adjacency list for efficiency (not matrix).
- In grids: check boundaries + use direction array.
- BFS = shortest path in unweighted graphs → No need for Dijkstra.

6. Time Complexity

Scenario	Time Complexity	Space Complexity
Graph BFS (V + E)	**O(V + E)**	**O(V)**
Tree (N nodes)	**O(N)**	**O(N)**
Grid (N × M)	**O(NM)**	**O(NM)**
Adjacency Matrix Graph	**$O(V^2)$**	**O(V)**