Quick notes — Unique Paths II (LeetCode 63)

Problem (short)

Given `m x n` grid with `0` (free) and `1` (obstacle). Robot starts at `grid[0][0]` and can move **right** or **down** only. Count unique paths to `grid[m-1][n-1]` avoiding obstacles. Answer ≤ 2e9.

Key ideas

- If a cell is `1` → cannot step on it → contributes `0` paths.
- Recurrence (DP):

  `dp[i][j] = 0` if obstacle else `dp[i-1][j] + dp[i][j-1]`.
- Base: start cell `dp[0][0] = 1` if it's not an obstacle.
- Use memo recursion or bottom-up DP. Can optimize space to **1D** (row or column).

Complexity

- Time: `O(m * n)` for DP (or exponential for plain recursion).
- Space: `O(m * n)` for 2D dp, `O(n)` for optimized 1D.

  Constraints `m,n ≤ 100` so DP is cheap.

Edge cases (must handle)

1. Start cell is obstacle → answer `0`.
2. End cell obstacle → answer `0`.
3. Single row / single column grids.
4. All zeros (no obstacles).

Your recursive + memo code — review

```
class Solution {
public:
    int helper(int sr,int sc,int er,int ec,vector<vector<int>>& dp,vector<vector<i
nt>>& arr){
        if(sr > er || sc > ec) return 0;
        if(arr[sr][sc] == 1) return 0;
        if(sr == er && sc == ec) return 1;
        if(dp[sr][sc] != -1) return dp[sr][sc];
        int rightway = helper(sr,sc+1,er,ec,dp,arr);
        int downway  = helper(sr+1,sc,er,ec,dp,arr);
        return dp[sr][sc] = rightway + downway;
    }
    int uniquePathsWithObstacles(vector<vector<int>>& arr) {
        int m = arr.size();
        int n = arr[0].size();
        vector<vector<int>>dp(m,vector<int>(n,-1));
        return helper(0,0,m-1,n-1,dp,arr);
    }
};
```

- ✅ Correct logic: checks obstacles, bounds, memoization.
- ✅ Handles start/end obstacles (since `arr[sr][sc]==1` checked before reaching end).
- Note: recursion depth ≈ `m+n` (safe for given constraints).
- Micro improvement: early return if `arr[0][0]==1` or `arr[m-1][n-1]==1` to avoid calling helper.

Preferred — Bottom-up DP (clean, iterative)

2D DP version (easy to read):

```
class Solution {
public:
    int uniquePathsWithObstacles(vector<vector<int>>& a) {
        int m = a.size(), n = a[0].size();
        if(a[0][0] == 1) return 0;
        vector<vector<int>> dp(m, vector<int>(n, 0));
        dp[0][0] = 1;
        for(int i = 0; i < m; ++i){
            for(int j = 0; j < n; ++j){
                if(a[i][j] == 1){ dp[i][j] = 0; continue; }
                if(i > 0) dp[i][j] += dp[i-1][j];
                if(j > 0) dp[i][j] += dp[i][j-1];
            }
        }
        return dp[m-1][n-1];
    }
};
```

Optimized — 1D DP (space `O(n)`)

Use a single row `dp[j]` that stores current row results:

```
class Solution {
public:
    int uniquePathsWithObstacles(vector<vector<int>>& a) {
        int m = a.size(), n = a[0].size();
        if(a[0][0] == 1) return 0;
        vector<int> dp(n, 0);
        dp[0] = 1;
        for(int i = 0; i < m; ++i){
            for(int j = 0; j < n; ++j){
                if(a[i][j] == 1) {
                    dp[j] = 0;  // obstacle → no ways to this column in this row
                } else {
```

```
                if(j > 0) dp[j] += dp[j-1];
            }
        }
    }
    return dp[n-1];
  }
};
```

- Explanation: `dp[j]` after processing row `i` = number of ways to reach `(i,j)`.
- At obstacle set `dp[j]=0` so future sums ignore it.

Quick examples

- `[[0,0,0],[0,1,0],[0,0,0]]` → `2`.
- `[[0,1],[0,0]]` → `1`.
- `[[1,...]]` or `[...,1]` with start/end blocked → `0`.

Final tip

- For contest / interview: mention both memo and bottom-up; provide 1D optimization when asked about space.
- Add early checks for start/end obstacles for micro-efficiency.
