LeetCode : 238

✅ Problem Summary:

- Given an array `nums` of length `n`.
- For each index `i`, compute **product of all elements except** `nums[i]`.
- **Constraints:**
  - Cannot use division.
  - Must run in **O(n) time**.
  - **Follow-up:** Achieve **O(1) extra space** (excluding output array).

✅ Key Idea:

Instead of multiplying all numbers and dividing, we compute:

- **Prefix product:** product of all elements **before** `i`.
- **Suffix product:** product of all elements **after** `i`.

Then:

answer[i]=prefix[i]×suffix[i]answer[i] = prefix[i] \times suffix[i]

✅ Your Code Explanation:

```
class Solution {
public:
    vector<int> productExceptSelf(vector<int>& arr) {
        int n = arr.size();

        // Step 1: Create prefix and suffix arrays (initialized to 1)
        vector<int> suffix(n, 1);
        vector<int> prefix(n, 1);
```

```
        // Step 2: Compute prefix products
        // prefix[i] = product of all elements before i
        for (int i = 1; i < n; i++) {
            prefix[i] = prefix[i - 1] * arr[i - 1];
        }

        // Step 3: Compute suffix products
        // suffix[i] = product of all elements after i
        for (int i = n - 2; i >= 0; i--) {
            suffix[i] = arr[i + 1] * suffix[i + 1];
        }

        // Step 4: Combine prefix and suffix for final result
        for (int i = 0; i < n; i++) {
            arr[i] = prefix[i] * suffix[i];
        }

        return arr;
    }
};
```

✅ Step-by-Step Dry Run:

Example:

`arr = [1, 2, 3, 4]`

- **Prefix:**

  `prefix[0] = 1`

  `prefix[1] = 1 * 1 = 1`

  `prefix[2] = 1 * 2 = 2`

  `prefix[3] = 2 * 3 = 6`

  So `prefix = [1, 1, 2, 6]`

- **Suffix:**

  `suffix[3] = 1`

`suffix[2] = 1 * 4 = 4`

`suffix[1] = 4 * 3 = 12`

`suffix[0] = 12 * 2 = 24`

So `suffix = [24, 12, 4, 1]`

- **Final Result:**

  `result[i] = prefix[i] * suffix[i]`

  `= [1*24, 1*12, 2*4, 6*1]`

  `= [24, 12, 8, 6]`

✅ Time & Space Complexity:

- **Time:** O(n)
- **Space:** O(n) (because of `prefix` and `suffix` arrays)
- **Follow-up:** Can be optimized to O(1) space (excluding result array).

✅ Optimized Approach (O(1) extra space):

Instead of two arrays, compute result in **two passes**:

1. First pass → store prefix products directly in `result`.
2. Second pass → multiply suffix product into `result`.

Final Notes:

✔️ Your solution is **perfect for clarity** (using two arrays).

✔️ For interviews, also mention **O(1) space approach** because it's asked as a follow-up.
