LeetCode: 238

✅ Problem Recap

We need an array `answer` such that:

$answer[i]=\prod_{j=0, j\neq i}^{n-1} nums[j]$answer[i] = \prod_{\substack{j=0 \\ j \neq i}}^{n-1} nums[j]

Constraints:

- **O(n)** time.
- **No division operation allowed**.
- **Prefer O(1) extra space** (excluding output array).

✅ Your Approach

Step 1: Initialization

```
int product = 1; // product of all elements
int p2 = 1;      // product of all non-zero elements
int noz = 0;     // count zeros
```

Step 2: First loop – Calculate products

```
for(int i=0; i<n; i++){
    if(arr[i] == 0) noz++;
    product *= arr[i];     // overall product (includes zeros)
    if(arr[i]!=0) p2 *= arr[i];  // product ignoring zeros
}
```

- `product` → full product including zeros.
- `p2` → product excluding zeros (useful when zeros exist).
- `noz` → counts how many zeros are present.

Step 3: Handle zeros and division

```
if(noz>1) p2 = 0; // more than 1 zero → all products are zero

for(int i=0;i<n;i++){
    if(arr[i] != 0) arr[i] = product / arr[i];  // division step
    else arr[i] = p2;
}
```

- If there are:
  - **0 zeros** → use `product / arr[i]`.
  - **1 zero** → only index with zero gets `p2`, others get `0`.
  - **>1 zero** → everything is `0`.

✅ Why is this method logically correct?

- **Mathematical logic works perfectly**:
  - For `nums[i] ≠ 0`,

    answer[i]=total productnums[i]answer[i] = \frac{\text{total product}}{\text{nums[i]}}
  - For `nums[i] = 0`,

    answer[i]=product of all non-zero elementsanswer[i] = \text{product of all non-zero elements}
- Handles **zero cases correctly**.
- Runs in **O(n)**.

❌ Why is it wrong for this problem?

1. **Uses Division**
   - Problem explicitly says:

     > You must write an algorithm without using the division operation.
2. **Doesn't meet the "Follow-up" constraint**
   - Extra variables (`product`, `p2`, `noz`) → still **O(1) extra space**, so that's fine.

- But **core violation** = usage of division.

✅ Why Division is Not Allowed?

- **Risk of overflow** if the product of all numbers is huge (although 32-bit limit helps here).
- **Division by zero** problem complicates direct approach.
- Forcing no-division makes the solution algorithmically interesting → requires **prefix & suffix products**.

✅ Correct Approach (as per constraints)

Idea:

- For each index `i`,

answer[i]=
(product of all elements to the left of i)×(product of all elements to the right of i)answer[i] = (\text{product of all elements to the left of i}) \times (\text{product of all elements to the right of i})

- Build **prefix product** and **suffix product** arrays, or compute in two passes with constant extra space.

Steps

1. Initialize `answer` array with `1`.
2. **First pass (left to right)** → store prefix product:

   answer[i]=product of all elements before ianswer[i] = \text{product of all elements before i}

3. **Second pass (right to left)** → multiply by suffix product:

   answer[i]∗=product of all elements after ianswer[i] *= \text{product of all elements after i}

Time: O(n)

Space: O(1) extra (if we ignore output array).

✅ Mathematical Expression for Correct Method

$$answer[i] = \left(\prod_{j=0}^{i-1} nums[j] \right) \times \left(\prod_{j=i+1}^{n-1} nums[j] \right)$$

✅ Explanatory Notes (Summary)

Your method:

- ✅ Correct mathematically.
- ❌ Violates problem constraint (used division).
- ✅ Handles zeros well.

Required method:

- Use **prefix & suffix multiplication**.
- No division allowed.

🔥 Do you want me to **create a clean, well-commented solution for the prefix-suffix method**, along with **a PDF explanatory note comparing both approaches with diagrams and formulas**? Or should I **just give you the commented code and summary in text format here**?