
Problem Statement:

Given an integer array nums, return an array result such that result[i] is equal to the product of all the elements of nums except nums[i].

Solution One – Brute Force

```python
def product_except_self_blind(nums: List[int]) -> List[int]:
    result = [0] * len(nums)
    i = 0
    while i < len(nums):
        product = 1
        j = 0
        while j < len(nums):
            if j != i: product *= nums[j]
            j += 1
        result[i] = product
        i += 1

    return result
```

Step-by-Step Breakdown

1. **Input:** [1, 2, 3, 4]
2. **Outer Loop (i):**
    - For each element in nums, compute the product of all other elements.
3. **Inner Loop (j):**
    - If j != i, multiply product by nums[j].

Step	i	j Exclusion	Intermediate Product	Result Update
1	0	0	2 * 3 * 4 = 24	24
2	1	1	1 * 3 * 4 = 12	12
3	2	2	1 * 2 * 4 = 8	8
4	3	3	1 * 2 * 3 = 6	6

4. **Output:** [24, 12, 8, 6]
5. **Efficiency Analysis:**
    - **Time Complexity: O(n^2) due to the nested loops.**
    - **Space Complexity: O(n) for the result list.**

Solution Two – Optimised Prefix Suffix Approach

```python
def product_except_self(nums: List[int]) -> List[int]:
    n = len(nums)
    result = [1] * n
    prefix = 1
    suffix = 1

    # Compute prefix products
    for i in range(n):
        result[i] = prefix
        prefix *= nums[i]

    # Compute suffix products and multiply with prefix results
    for i in range(n - 1, -1, -1): # for i in [3, 2, 1, 0]
        result[i] *= suffix
        suffix *= nums[i]

    return result
```

Step-by-Step Breakdown

1. **Input:** [1, 2, 3, 4]
2. **Prefix Calculation:**
    a. Traverse the list from left to right, computing cumulative products.
    b. Store the prefix product at each index.

Step	i	Prefix Value	Result[]	Prefix Update
1	0	1	[1, 1, 1, 1]	prefix *= 1 -> 1
2	1	1	[1, 1, 1, 1]	prefix *= 2 -> 2
3	2	2	[1, 1, 2, 1]	prefix *= 3 -> 6
4	3	6	[1, 1, 2, 6]	prefix *= 4 -> 24

3. **Suffix Calculation:**
    a. Traverse the list from right to left, computing cumulative products.
    b. Multiply each element of the result by the suffix product.

Step	i	Suffix Value	Result Progression	Suffix Update
1	3	1	[1, 1, 2, 6]	suffix *= 4 -> 4
2	2	4	[1, 1, 8, 6]	suffix *= 3 -> 12

3	1	12	[1, 12, 8, 6]	suffix *= 2 -> 24
4	0	24	[24, 12, 8, 6]	suffix *= 1 -> 24

4. **Output:** [24, 12, 8, 6]
5. **Efficiency Analysis:**
    a. Time Complexity: O(n) due to two linear traversals.
    b. Space Complexity: O(1) extra space excluding the result list.

Comparison

Aspect	Solution One	Solution Two
Time Complexity	O(n^2)	O(n)
Space Complexity	O(n)	O(1)
Ease of Implementation	Simple	Slightly Complex
Scalability	Poor	Excellent
