Product of Array Except Self :-

238. Product of Array Except Self

Medium Topics Companies Hint

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

The product of any prefix or suffix of `nums` is **guaranteed** to fit in a **32-bit** integer.

You must write an algorithm that runs in `O(n)` time and without using the division operation.

Example 1:

```
Input: nums = [1,2,3,4]
Output: [24,12,8,6]
```

Example 2:

```
Input: nums = [-1,1,0,-3,3]
Output: [0,0,9,0,0]
```

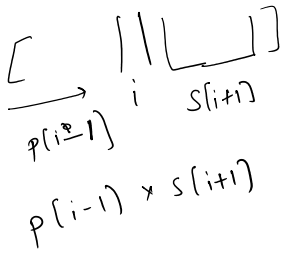
Constraints:

- `2 <= nums.length <= 10^5`
- `-30 <= nums[i] <= 30`
- The input is generated such that `answer[i]` is **guaranteed** to fit in a **32-bit** integer.

Follow up: Can you solve the problem in `O(1)` extra space complexity? (The output array **does not** count as extra space for space complexity analysis.)

Seen this question in a real interview before? 1/5

eg: I/p: nums = [1,2,3,4]
O/p: [24,12,8,6]



P(i-1) × S(i+1)

C++ Auto

```cpp
class Solution {
public:
    vector<int> productExceptSelf(vector<int>& a) {
        int n=a.size();
        vector<int> pre(n, 1), suf(n, 1), ans(n);
        pre[0]=a[0];
        for(int i=1;i<n;i++)pre[i]=a[i]*pre[i-1];
        suf[n-1]=a[n-1];
        for(int i=n-2;i>=0;i--)suf[i]=a[i]*suf[i+1];
        ans[0]=suf[1];
        ans[n-1]=pre[n-2];
        for(int i=1;i<n-1;i++){
         ans[i]=pre[i-1]*suf[i+1];
        }
        return ans;
    }
};
```

T.C. O(n)

S.C. O(1)

Approach 2 :- looping the input array twice

```
class Solution{
public:
  vector<int> productExceptSelf(vector<int>&nums){
     vector<int> output (nums.size(),1);
     int left = 1;
     for(int i= 0; i<nums.size(); i++){
```

```
        output[i] *= left;
        left *= nums[i]; }

    int right = 1;
for(int i = nums.size()-1; i>=0; i--){
        output[i] *= right;
        right *= nums[i];
    }
    return output;
    }
};
```