POTD - 15/03/2024

Product of Array Except Self (Leetcode-238)

Company Tag

Facebook Amazon Apple Microsoft
Asana Adobe Nutanix Lyft Nvidia
Intel VMware

LinkedIn@manojofficialmj

PRODUCT OF ARRAY EXCEPT SELF (Leetcode-238)

Nums	-1	2	3	4
	0	1	2	3
O/P	24	-12	-8	-6
	2×3×4	-1×3×4	-1×2×4	-1×3×2

{ N = Nums.size()
N = 4 }

[BRUTE FORCE APPROAC:] (Nested For Loop)

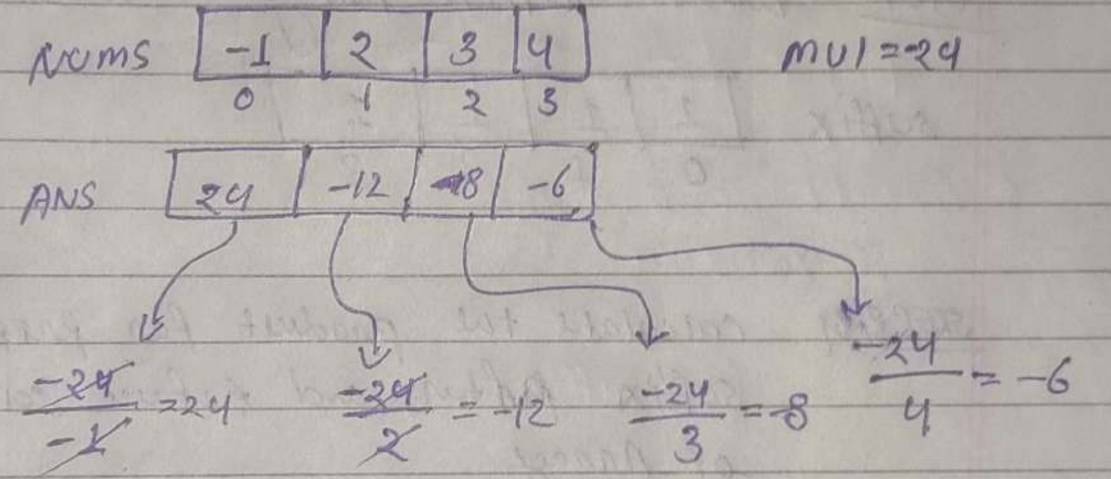
```
vector <int> Ans(N);
for (int i = 0; i<n ; i++) {
    int mul = 1;
    for (j = 0 ; j<n ; j++) {
        if (i != j) {
        Ans = mul * Nums[j]
        }
    }
    Ans[i] = mul;
}
return Ans;
```

$T.C. = O(N)^2$
$S.C. = O(1)$

[Approach : Division operation]

STEP1 Get product of All Elements ⇒ -24

STEP2 Traverse on Array to store the product on Each Index via division operation



$\frac{\cancel{-24}}{\cancel{-1}} = 24$ $\frac{\cancel{-24}}{\cancel{2}} = -12$ $\frac{-24}{3} = -8$ $\frac{-24}{4} = -6$

STEP1
```
int mul = 1;
for(int i=0 ; i<N ; i++){
        mul = mul * NUMS[i];
}
```

STEP2
```
vector<int> ans(N);

for( int i=0; i<N ; i++){
        Ans[i] = mul/NUMS[i];
}

return Ans;
```

T.C. = O(N)
S.C. = O(1)

[APProach : prefix and suffix]

STEP1 Create prefix Array of N size

prefix	1	1	1	1
	0	1	2	3

STEP2 Create suffix Array of N size

suffix	1	1	1	1
	0	1	2	3

STEP3, 4 Calculate the product for prefix & suffix after and before Each index of Array

STEP5 store product in Ans to get final output.

Logic

		i		
NUMS	-1	2	3	4
	0	1	2	3

$$\underbrace{(0 \text{ to } i-1)}_{\text{prefix}} \times \underbrace{(i+1 \text{ to } N)}_{\text{suffix}}$$

i.e. $(-1) \times (3 \times 4) \Rightarrow -12$

prefix
(=1)

Prufix $\overset{\circ}{i}=1$

Nums

-1	2	3	4
0	1	2	3

~~Prefix~~

~~[illegible]~~

$\boxed{prefix[i] = prefix[i-1] \times Nums[i-1]}$

prefix

1	-1	-2	-6
0	1	2	3

suffix
(=N-2)

$\overset{\circ}{i}=2$ suffix

Nums

-1	2	3	4
0	1	2	3

suffix

24	12	4	1
0	1	2	3

$\boxed{suffix[i] = suffix[i+1] \times Nums[i+1]}$

Ans

prefix

1	-1	-2	-6
0	1	2	3

suffix

24	12	4	1
0	1	2	3

T.C. = O(N)
S.C. = O(N)

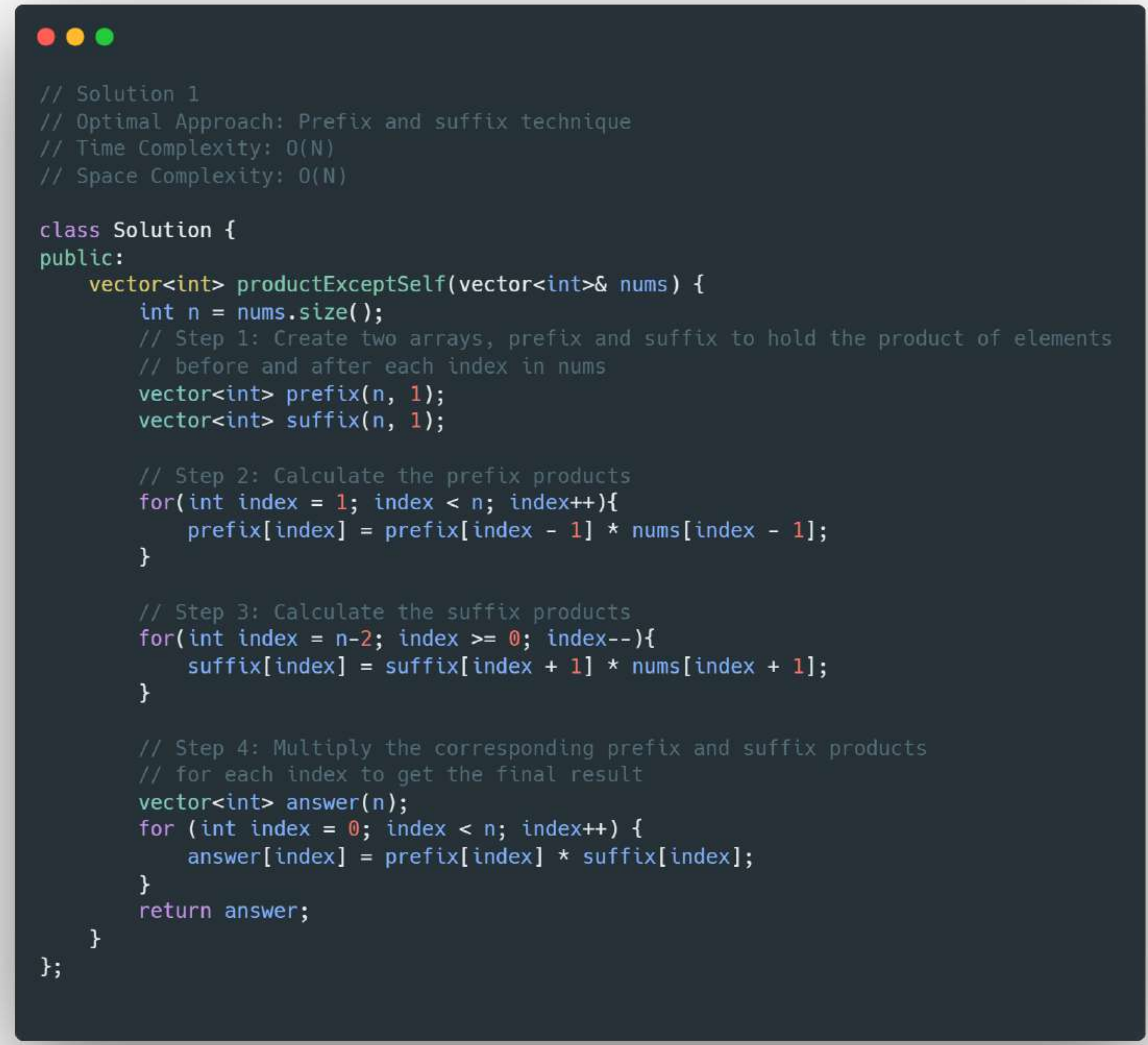
$\overset{\circ}{i}=0$

Ans

24	-12	-8	-6
0	1	2	3

Final output

$\boxed{Ans[i] = prefix[i] \times suffix[i]}$



```cpp
// Solution 1
// Optimal Approach: Prefix and suffix technique
// Time Complexity: O(N)
// Space Complexity: O(N)

class Solution {
public:
    vector<int> productExceptSelf(vector<int>& nums) {
        int n = nums.size();
        // Step 1: Create two arrays, prefix and suffix to hold the product of elements
        // before and after each index in nums
        vector<int> prefix(n, 1);
        vector<int> suffix(n, 1);

        // Step 2: Calculate the prefix products
        for(int index = 1; index < n; index++){
            prefix[index] = prefix[index - 1] * nums[index - 1];
        }

        // Step 3: Calculate the suffix products
        for(int index = n-2; index >= 0; index--){
            suffix[index] = suffix[index + 1] * nums[index + 1];
        }

        // Step 4: Multiply the corresponding prefix and suffix products
        // for each index to get the final result
        vector<int> answer(n);
        for (int index = 0; index < n; index++) {
            answer[index] = prefix[index] * suffix[index];
        }
        return answer;
    }
};
```

[Approach : Prefix & suffix]

Nums

-1	2	3	4
0	1	2	3

- Create prefix Array as Answer

prefix

1	-1	-2	-6
0	1	2	3

- Create a variable suffix as a product variable

int suffix = 1;

- Traverse on Entire array $_\wedge$ Nums & prefix from End (N-1)

Nums

-1	2	3	4
0	1	2	3

$j = N-1$
$= 3$

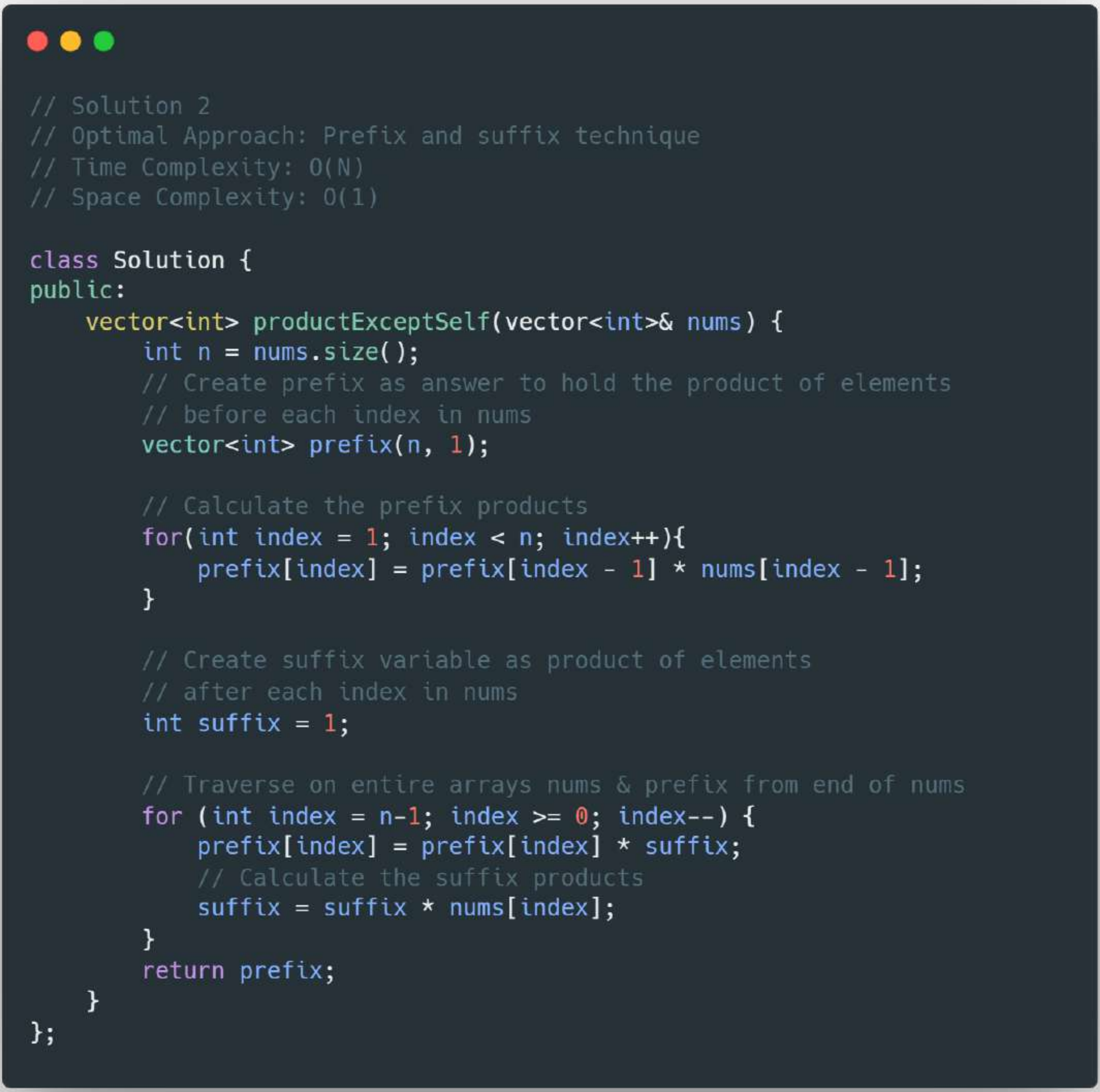
prefix

1	-1	-2	-6
0	1	2	3

```
for ( j = end ⟶ 0)
    prefix[j] = prefix[j] * suffix
    suffix = suffix * ~~prefix~~ Nums [j]
```

$\boxed{\begin{array}{l} T.C. = O(N) \\ S.C. = O(1) \end{array}}$

suffix = $\underbrace{1 \times 4}_{4} \times 3 \times 2$ (3, 2, -1)



```cpp
// Solution 2
// Optimal Approach: Prefix and suffix technique
// Time Complexity: O(N)
// Space Complexity: O(1)

class Solution {
public:
    vector<int> productExceptSelf(vector<int>& nums) {
        int n = nums.size();
        // Create prefix as answer to hold the product of elements
        // before each index in nums
        vector<int> prefix(n, 1);

        // Calculate the prefix products
        for(int index = 1; index < n; index++){
            prefix[index] = prefix[index - 1] * nums[index - 1];
        }

        // Create suffix variable as product of elements
        // after each index in nums
        int suffix = 1;

        // Traverse on entire arrays nums & prefix from end of nums
        for (int index = n-1; index >= 0; index--) {
            prefix[index] = prefix[index] * suffix;
            // Calculate the suffix products
            suffix = suffix * nums[index];
        }
        return prefix;
    }
};
```