Product of Aarray Except Self

arr = [1 2 3 4]

|| || || ||

(2 x 3 x 4) (1 x 3 x 4) (1 x 2 x 4) (1 x 2 x 3)

|| || || ||

modified array = [24 12 8 6]

arr = [1 2 3 4]

OIP = [24 12 8 6]

Let's do some observation Guy's

● → Prefix Product from right to left

● → Prefix Product from left to right

1 2 3 4 ⇒ 1 * 24 = 24

1 2 3 4 ⇒ 1 * 12 = 12

1 2 3 4 ⇒ 2 * 4 = 8

1 2 3 4 ⇒ 6 * 1 = 6

↖ Desired OIP

Algorithm :-

arr = [1 2 3 4]

Note

Step.1) Create two array L[] & R[]

→ L[] → Store the Product of left Prefix except self

→ R[] → Store the Product of Right Prefix except self

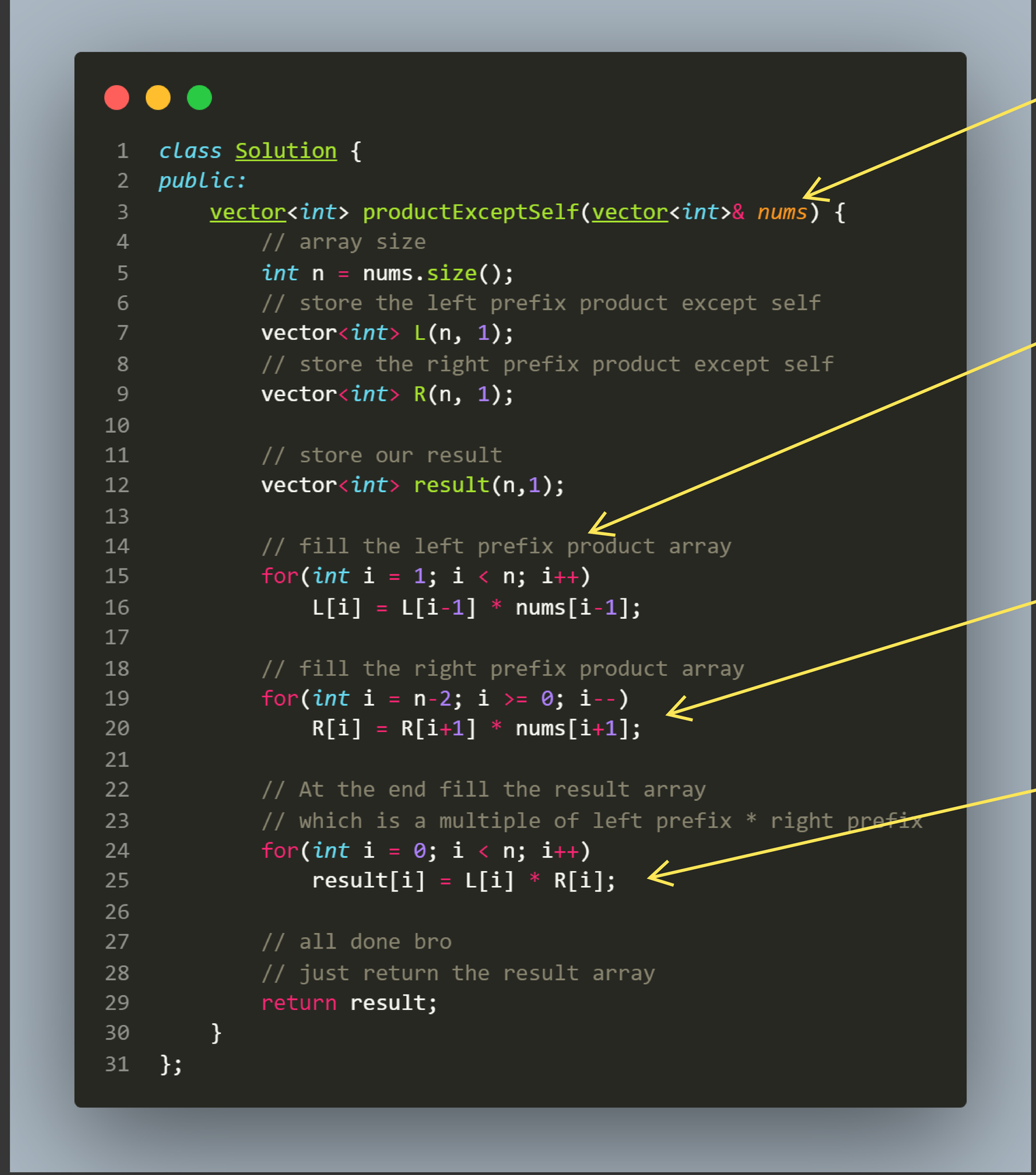
Step.2) Create a result array that store our O/P

L[] → [1 1 2 6]

R[] → [24 12 4 1]

result[i] = L[i] * R[i] → [24 12 8 6]

```cpp
class Solution {
public:
    vector<int> productExceptSelf(vector<int>& nums) {
        // array size
        int n = nums.size();
        // store the left prefix product except self
        vector<int> L(n, 1);
        // store the right prefix product except self
        vector<int> R(n, 1);

        // store our result
        vector<int> result(n,1);

        // fill the left prefix product array
        for(int i = 1; i < n; i++)
            L[i] = L[i-1] * nums[i-1];

        // fill the right prefix product array
        for(int i = n-2; i >= 0; i--)
            R[i] = R[i+1] * nums[i+1];

        // At the end fill the result array
        // which is a multiple of left prefix * right prefix
        for(int i = 0; i < n; i++)
            result[i] = L[i] * R[i];

        // all done bro
        // just return the result array
        return result;
    }
};
```



* Time Complexity = O(n)

* Space Complexity = O(n)

Follow up

Can we do it in O(1) Space

Note:- O/P Array not Considered as extra Space

Given in Question

Constant Space Approach

arr = [1 2 3 4]

result[] = [1 1 1 1] ← initially

Step-1 Fill the left Prefix Product in result array

```
for(int i=1 ; i<n ; i++)
    result[i] = result[i-1] * arr[i-1];
```

result[] = [1, 1, 2, 6]

Step-2 Fill the right Prefix Product in result array

```
int Product = arr[n-1] => (4)
for(int i = n-2; i >= 0; i--){
    result(i) = result[i] * product;
    product * = arr[i];
}
```

Step 3 return the result array

result[] = [1 1 2 6]

arr = [1 2 3 4]

product = arr[n-1];
= 4

for(n-2 .to 0){

result[i] *= product;

product *= arr[i];

}

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

result	Product
2 * 4 = 8	4
1 * 12 = 12	4 * 3 = 12
1 * 24 = 24	12 * 2 = 24

Time :- O(n)

Space :- O(1)

Source Code

```cpp
class Solution {
public:
    vector<int> productExceptSelf(vector<int>& nums) {
        // array size
        int n = nums.size();

        // our answer store here
        vector<int> ans(n,1);

        // calculate the prefix left product
        for(int i = 1; i < n; i++)
            ans[i] = ans[i-1] * nums[i-1];

        // calculate the prefix right product
        int product_right = nums[n-1];
        for(int i = n-2; i >= 0; i--){
            ans[i] *= product_right;
            product_right*=nums[i];
        }
    return ans;
    }
};
```

Pay My Fee :)

just kidding