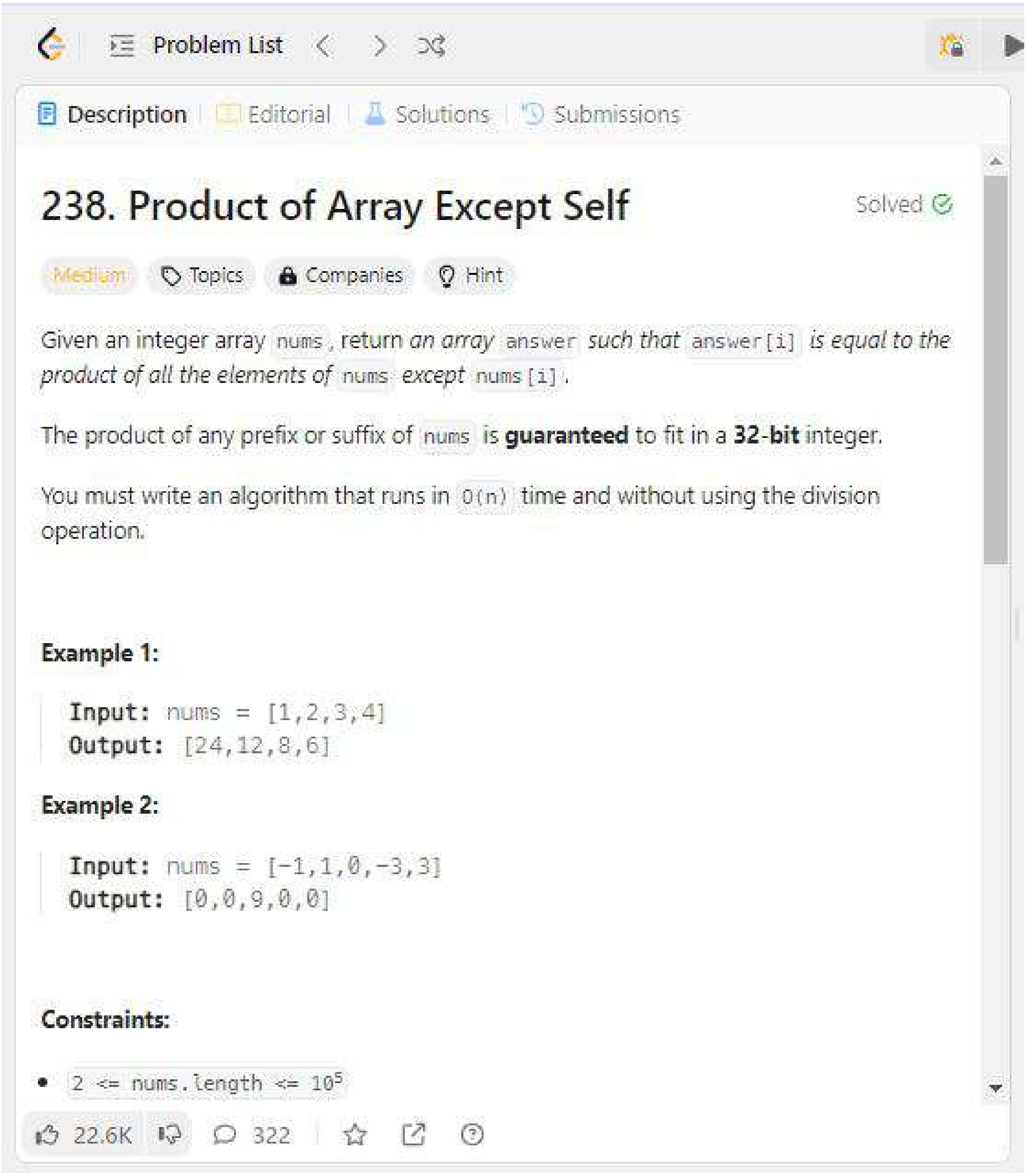


238. Product of Array Except Self

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:

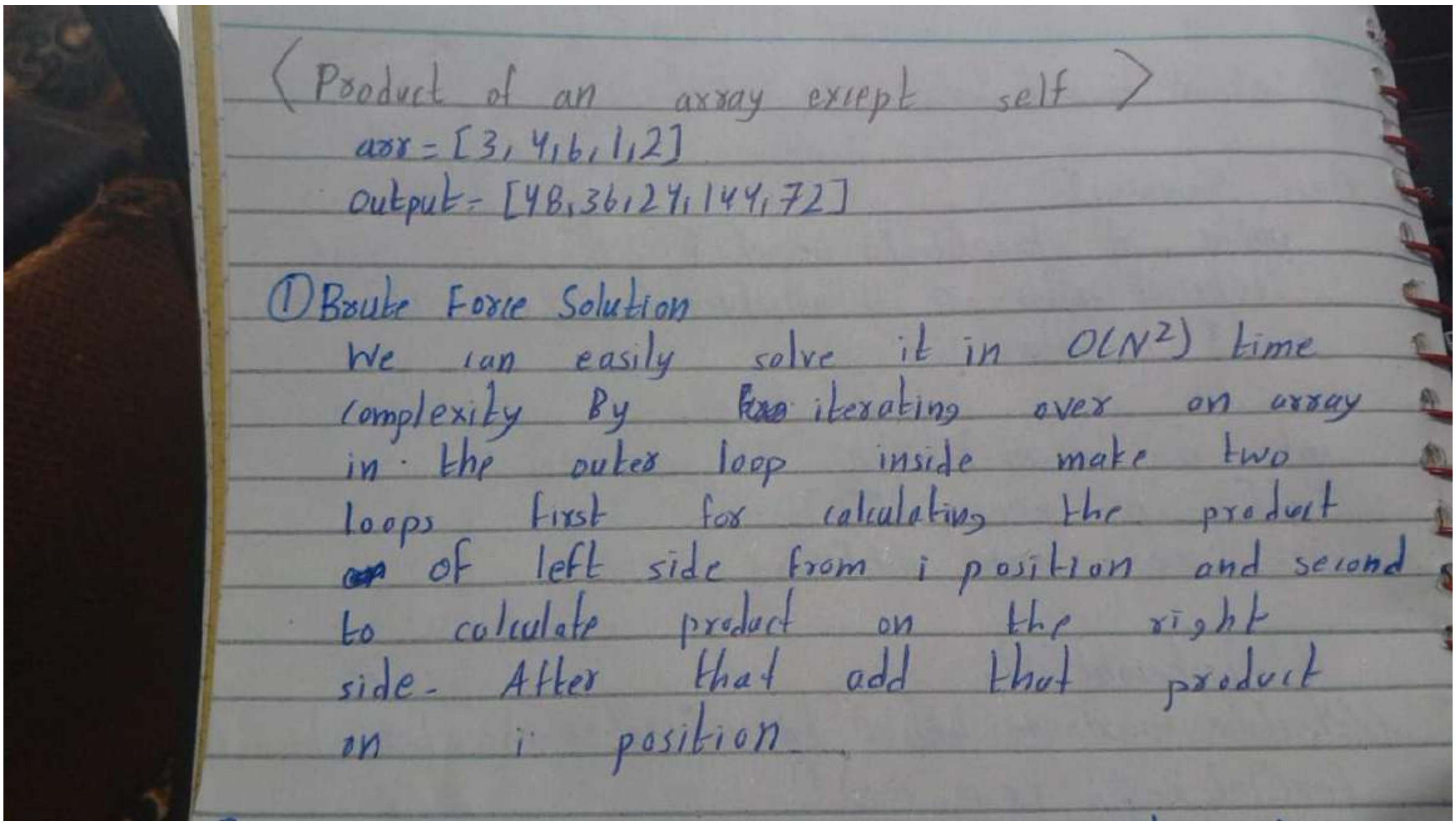
```
Input: nums = [1,2,3,4]
Output: [24,12,8,6]
```

Example 2:

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

Constraints:

- $2 <= \text{nums.length} <= 10^5$



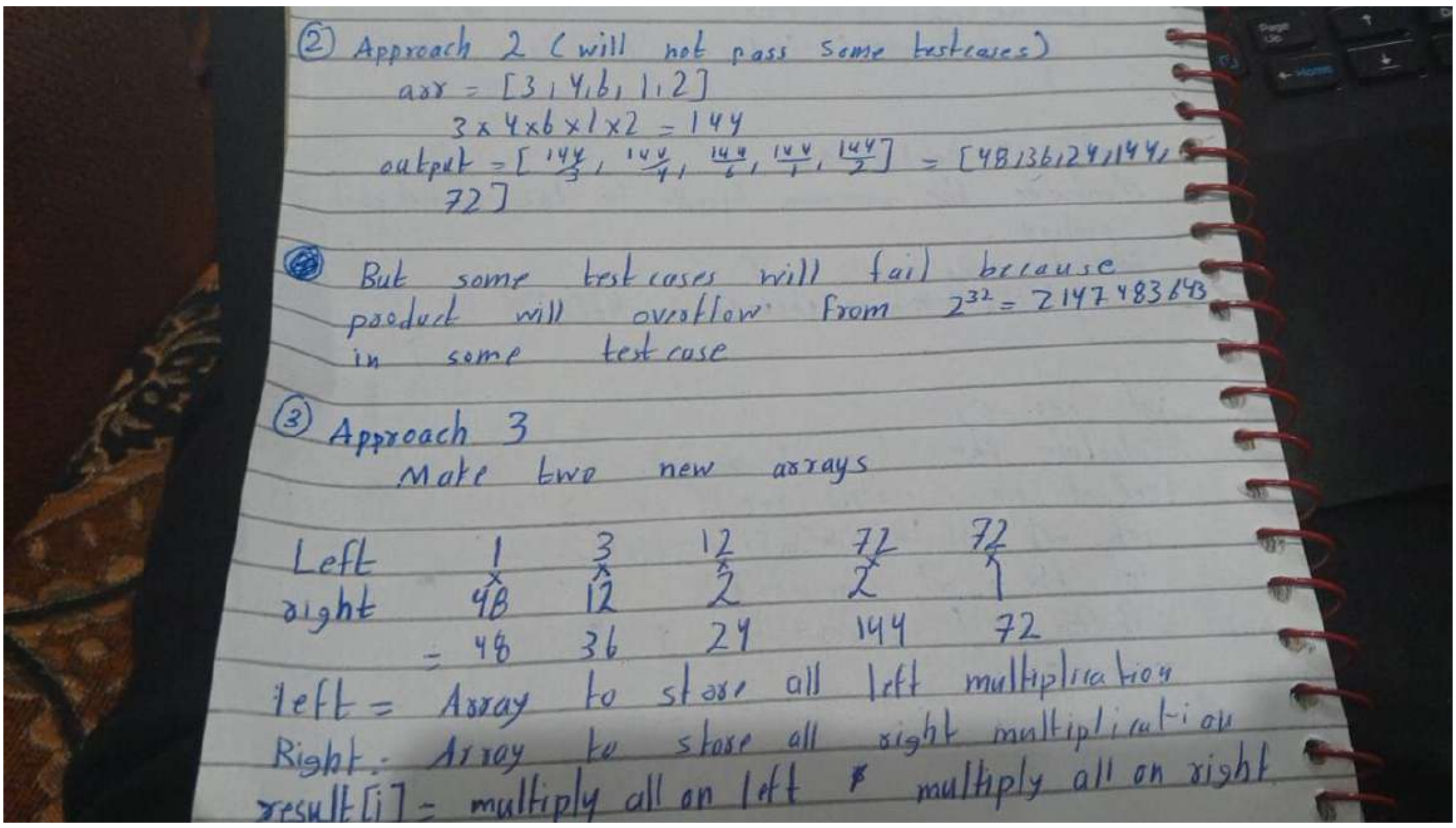
<Product of an array except self>

arr = [3, 4, 6, 1, 2]

Output = [48, 36, 24, 144, 72]

① Brute Force Solution

We can easily solve it in $O(N^2)$ time complexity By ~~[illegible]~~ iterating over on array in the outer loop inside make two loops first for calculating the product ~~on~~ of left side from i position and second to calculate product on the right side. After that add that product on i position.



② Approach 2 (will not pass some testcases)

arr = [3, 4, 6, 1, 2]

$3 \times 4 \times 6 \times 1 \times 2 = 144$

output = $[\frac{144}{3}, \frac{144}{4}, \frac{144}{6}, \frac{144}{1}, \frac{144}{2}]$ = [48, 36, 24, 144, 72]

But some test cases will fail because product will overflow from $2^{32} = 2147483648$ in some test case

③ Approach 3

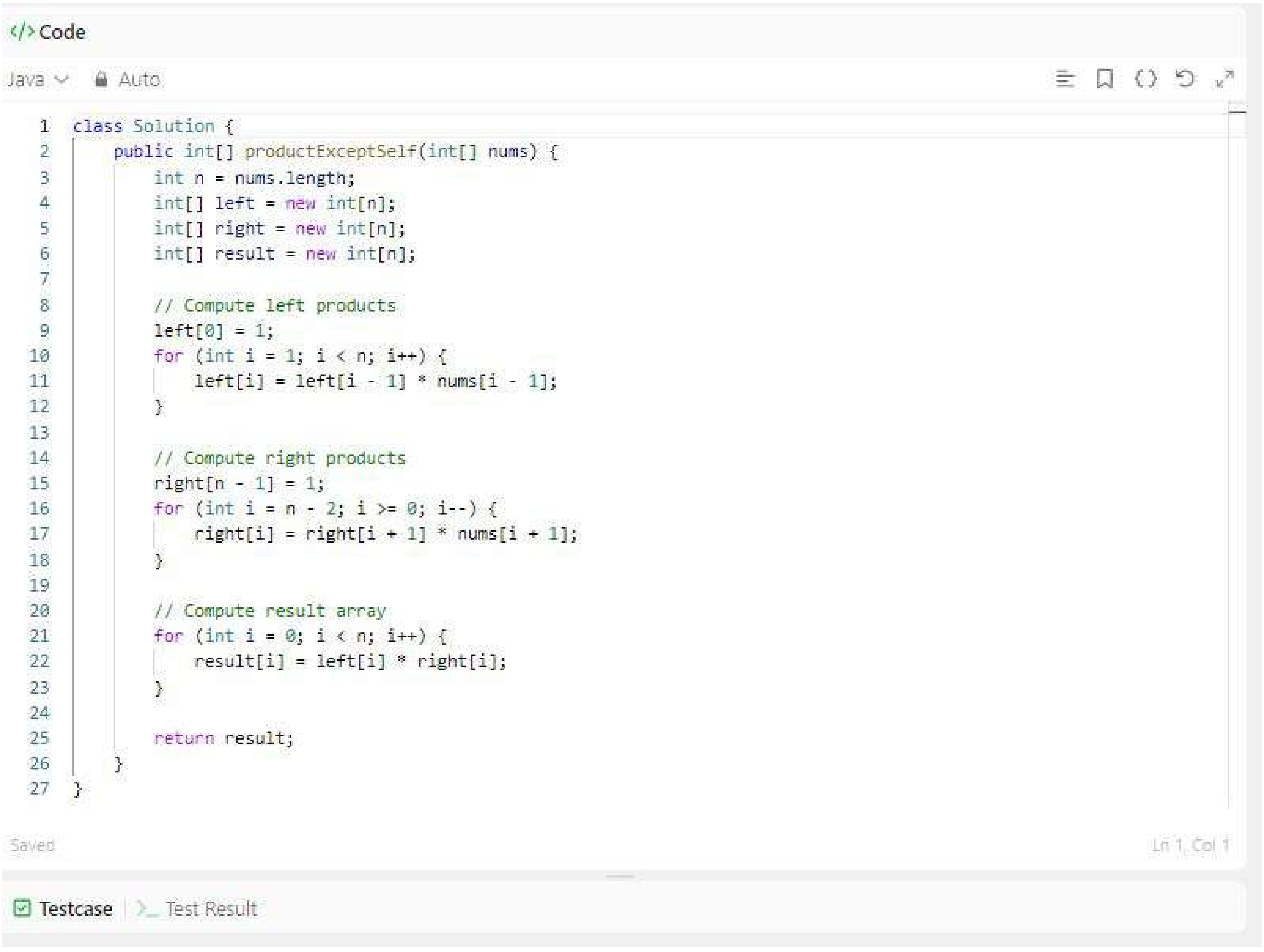
Make two new arrays

Left	1	3	12	72	72
	×	×	×	×	×
right	48	12	2	2	1
	= 48	36	24	144	72

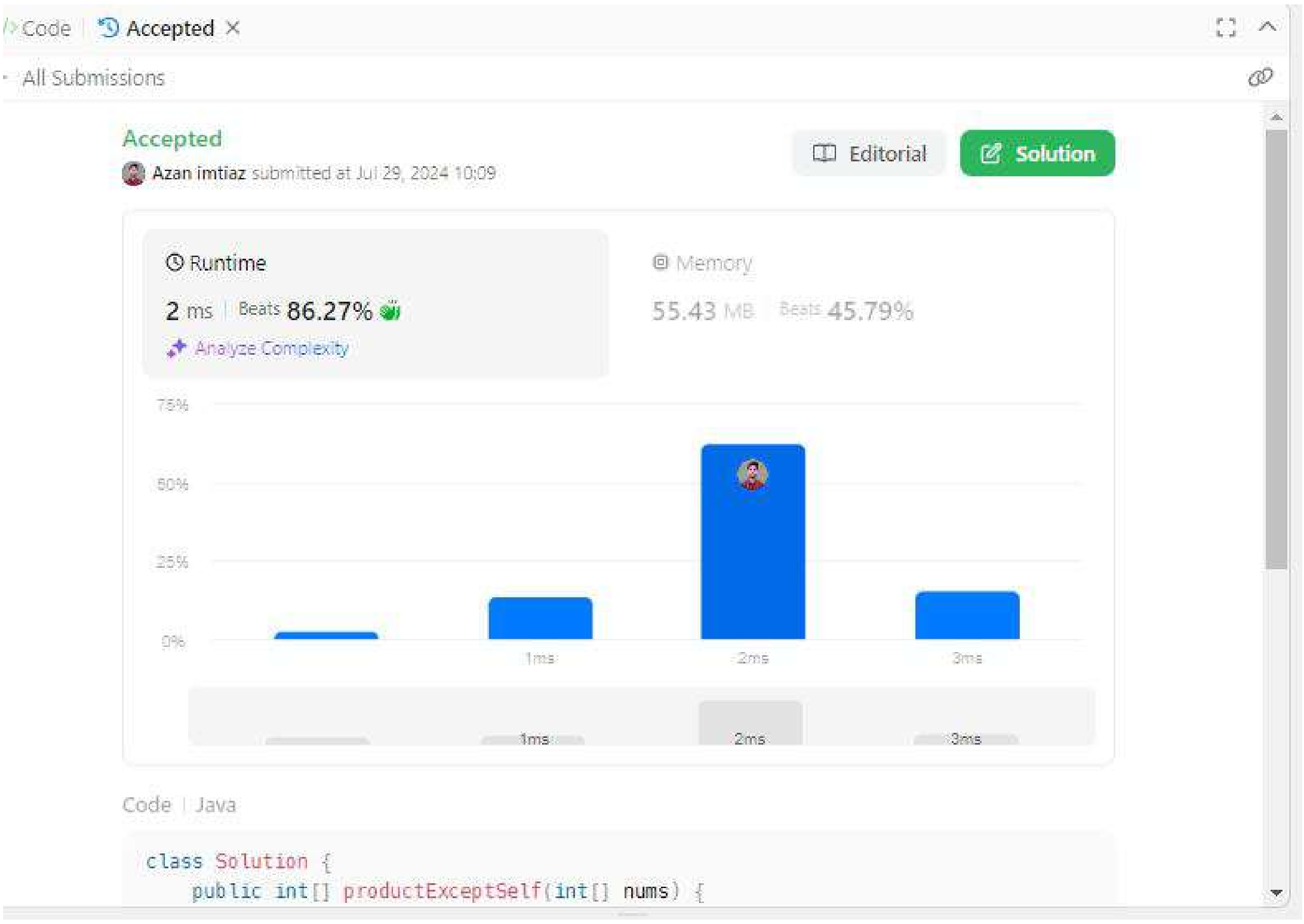
left = Array to store all left multiplication

Right = Array to store all right multiplication

result[i] = multiply all on left * multiply all on right



```java
class Solution {
    public int[] productExceptSelf(int[] nums) {
        int n = nums.length;
        int[] left = new int[n];
        int[] right = new int[n];
        int[] result = new int[n];

        // Compute left products
        left[0] = 1;
        for (int i = 1; i < n; i++) {
            left[i] = left[i - 1] * nums[i - 1];
        }

        // Compute right products
        right[n - 1] = 1;
        for (int i = n - 2; i >= 0; i--) {
            right[i] = right[i + 1] * nums[i + 1];
        }

        // Compute result array
        for (int i = 0; i < n; i++) {
            result[i] = left[i] * right[i];
        }

        return result;
    }
}
```


</> Code | Accepted ×

All Submissions

Accepted
Azan imtiaz submitted at Jul 29, 2024 10:09

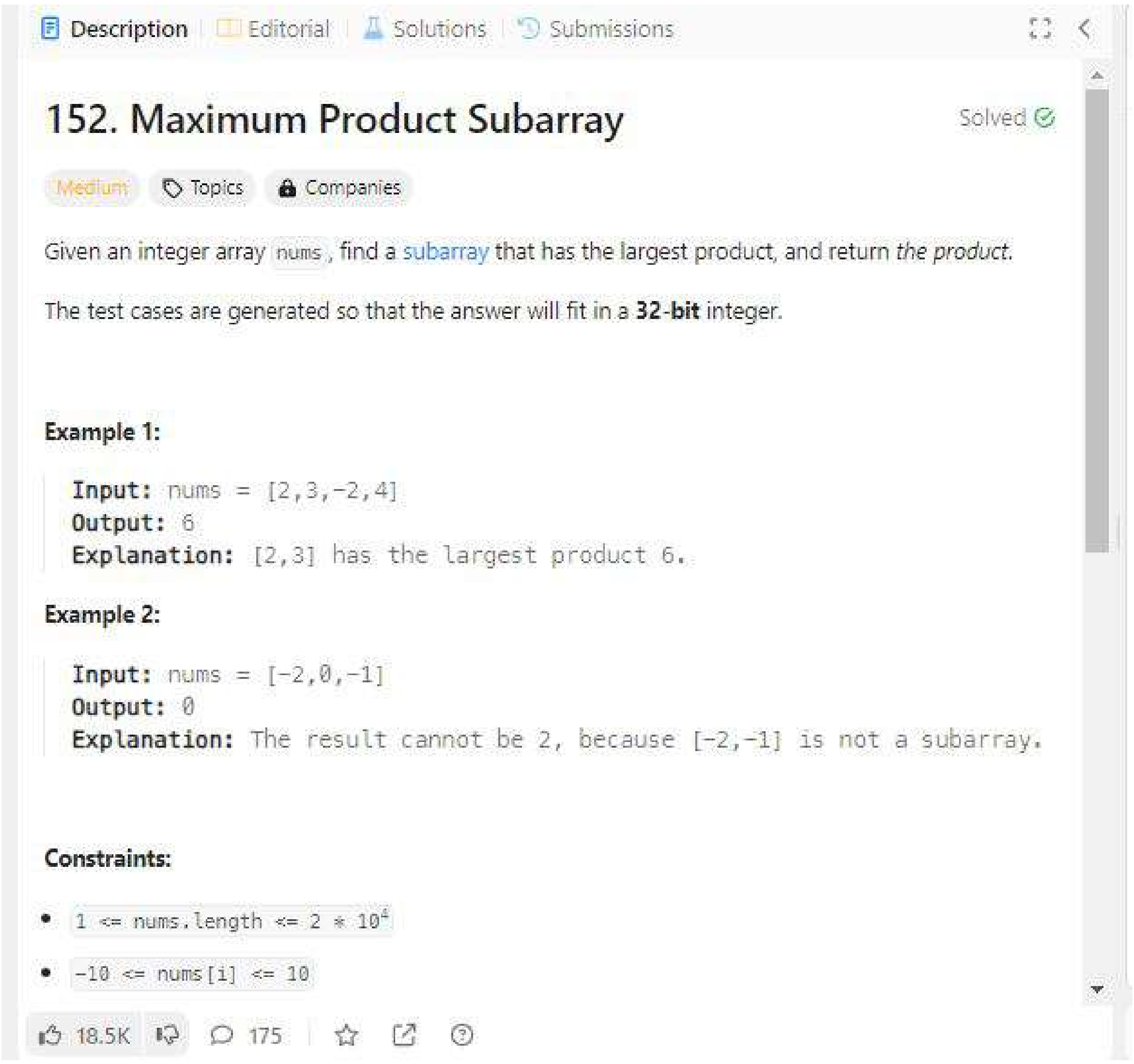
Editorial   Solution

Runtime
2 ms | Beats 86.27%
Analyze Complexity

Memory
55.43 MB | Beats 45.79%

Code | Java

```
class Solution {
    public int[] productExceptSelf(int[] nums) {
```



152. Maximum Product Subarray

Given an integer array `nums`, find a subarray that has the largest product, and return *the product*.

The test cases are generated so that the answer will fit in a **32-bit** integer.

Example 1:

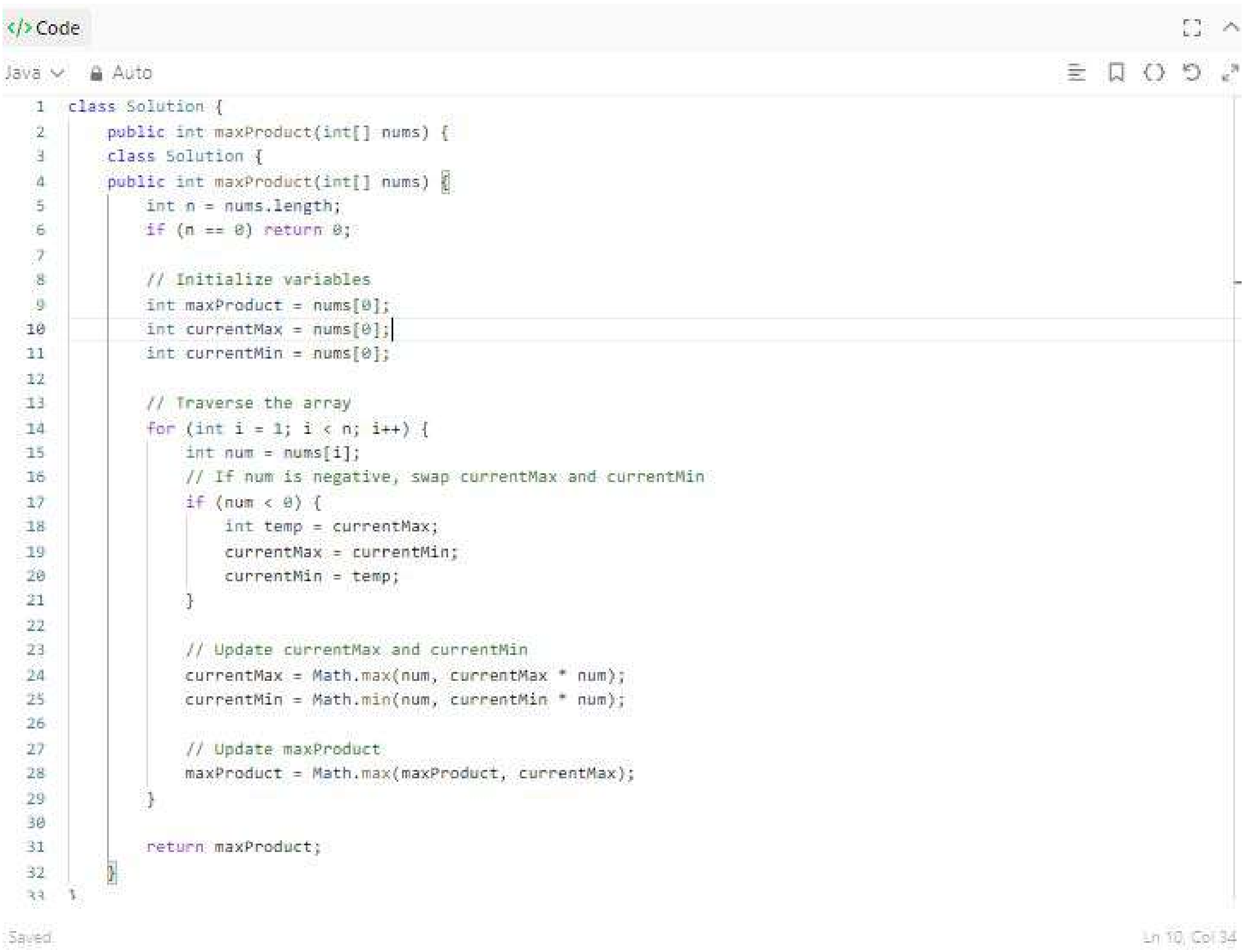
```
Input: nums = [2,3,-2,4]
Output: 6
Explanation: [2,3] has the largest product 6.
```

Example 2:

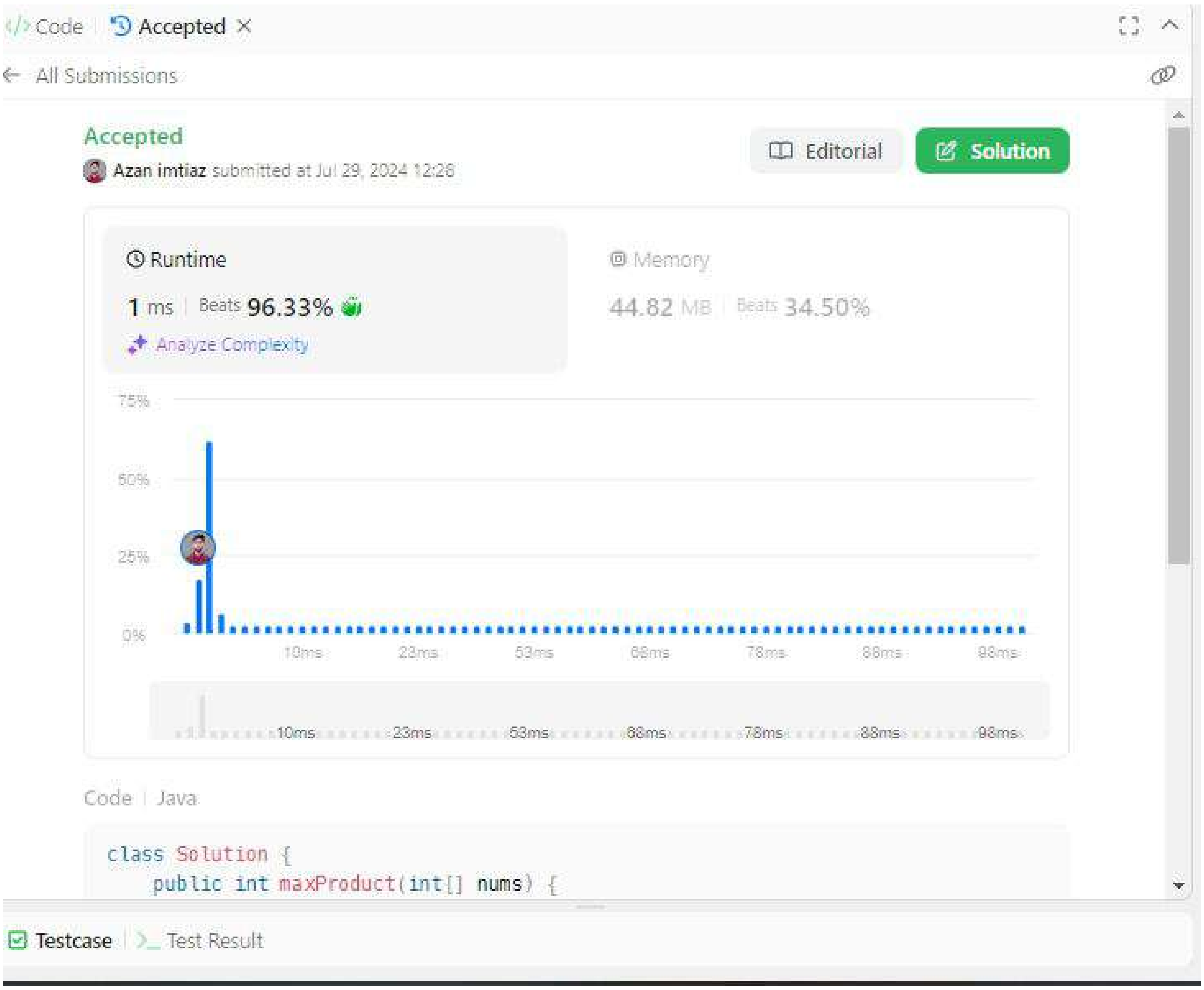
```
Input: nums = [-2,0,-1]
Output: 0
Explanation: The result cannot be 2, because [-2,-1] is not a subarray.
```

Constraints:

- `1 <= nums.length <= 2 * 10`^{4}
- `-10 <= nums[i] <= 10`



```java
class Solution {
    public int maxProduct(int[] nums) {
    class Solution {
    public int maxProduct(int[] nums) {
        int n = nums.length;
        if (n == 0) return 0;

        // Initialize variables
        int maxProduct = nums[0];
        int currentMax = nums[0];
        int currentMin = nums[0];

        // Traverse the array
        for (int i = 1; i < n; i++) {
            int num = nums[i];
            // If num is negative, swap currentMax and currentMin
            if (num < 0) {
                int temp = currentMax;
                currentMax = currentMin;
                currentMin = temp;
            }

            // Update currentMax and currentMin
            currentMax = Math.max(num, currentMax * num);
            currentMin = Math.min(num, currentMin * num);

            // Update maxProduct
            maxProduct = Math.max(maxProduct, currentMax);
        }

        return maxProduct;
    }
}
```



Code | Accepted

All Submissions

Accepted

Azan imtiaz submitted at Jul 29, 2024 12:28

Editorial

Solution

Runtime

1 ms | Beats 96.33%

Analyze Complexity

Memory

44.82 MB | Beats 34.50%

Code | Java

```
class Solution {
    public int maxProduct(int[] nums) {
```

Testcase | Test Result