Problem 2495: Number of Subarrays Having Even Product

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Given a

0-indexed

integer array

nums

, return

the number of

subarrays

of

nums

having an even product

.

Example 1:

Input:

nums = [9,6,7,13]

Output:

6

Explanation:

There are 6 subarrays with an even product: - nums[0..1] = 9 * 6 = 54. - nums[0..2] = 9 * 6 * 7 = 378. - nums[0..3] = 9 * 6 * 7 * 13 = 4914. - nums[1..1] = 6. - nums[1..2] = 6 * 7 = 42. - nums[1..3] = 6 * 7 * 13 = 546.

Example 2:

Input:

nums = [7,3,5]

Output:

0

Explanation:

There are no subarrays with an even product.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
long long evenProduct(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public long evenProduct(int[] nums) {

}
}
```

Python3:

```
class Solution:
def evenProduct(self, nums: List[int]) -> int:
```

Python:

```
class Solution(object):
def evenProduct(self, nums):
"""
:type nums: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @return {number}
*/
var evenProduct = function(nums) {

};
```

TypeScript:

```
function evenProduct(nums: number[]): number {

};
```

C#:

```
public class Solution {
public long EvenProduct(int[] nums) {

}
}
```

C:

```
long long evenProduct(int* nums, int numsSize) {

}
```

Go:

```
func evenProduct(nums []int) int64 {

}
```

Kotlin:

```
class Solution {
fun evenProduct(nums: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func evenProduct(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn even_product(nums: Vec<i32>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def even_product(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function evenProduct($nums) {

}
}
```

Dart:

```
class Solution {
int evenProduct(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def evenProduct(nums: Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec even_product(nums :: [integer]) :: integer
def even_product(nums) do

end
end
```

Erlang:

```
-spec even_product(Nums :: [integer()]) -> integer().
even_product(Nums) ->
.
```

Racket:

```
(define/contract (even-product nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Subarrays Having Even Product
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
long long evenProduct(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Subarrays Having Even Product
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public long evenProduct(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Subarrays Having Even Product
Difficulty: Medium
Tags: array, dp, math

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def evenProduct(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def evenProduct(self, nums):
"""
:type nums: List[int]
:rtype: int
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Subarrays Having Even Product
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} nums
* @return {number}
*/
var evenProduct = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Subarrays Having Even Product
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function evenProduct(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Subarrays Having Even Product
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public long EvenProduct(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Number of Subarrays Having Even Product
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

long long evenProduct(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Number of Subarrays Having Even Product
// Difficulty: Medium
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table
```

```
func evenProduct(nums []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun evenProduct(nums: IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func evenProduct(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Subarrays Having Even Product
// Difficulty: Medium
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn even_product(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def even_product(nums)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function evenProduct($nums) {

}
}
```

Dart Solution:

```
class Solution {
int evenProduct(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def evenProduct(nums: Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec even_product(nums :: [integer]) :: integer
def even_product(nums) do

end
end
```

Erlang Solution:

```
-spec even_product(Nums :: [integer()]) -> integer().
even_product(Nums) ->
.
```

Racket Solution:

```
(define/contract (even-product nums)
(-> (listof exact-integer?) exact-integer?)
)
```