Problem 3411: Maximum Subarray With Equal Products

Problem Information

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

Problem Description

You are given an array of

positive

integers

nums

.

An array

arr

is called

product equivalent

if

prod(arr) == lcm(arr) * gcd(arr)

, where:

prod(arr)

is the product of all elements of

arr

.

gcd(arr)

is the

GCD

of all elements of

arr

.

lcm(arr)

is the

LCM

of all elements of

arr

.

Return the length of the

longest

product equivalent

subarray

of

nums

.

Example 1:

Input:

nums = [1,2,1,2,1,1,1]

Output:

5

Explanation:

The longest product equivalent subarray is

[1, 2, 1, 1, 1]

, where

prod([1, 2, 1, 1, 1]) = 2

,

gcd([1, 2, 1, 1, 1]) = 1

, and

lcm([1, 2, 1, 1, 1]) = 2

.

Example 2:

Input:

nums = [2,3,4,5,6]

Output:

3

Explanation:

The longest product equivalent subarray is

[3, 4, 5].

Example 3:

Input:

nums = [1,2,3,1,4,5,1]

Output:

5

Constraints:

2 <= nums.length <= 100

1 <= nums[i] <= 10

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int MaxLength(int[] nums) {
}
}
```

C:

```
int maxLength(int* nums, int numsSize) {

}
```

Go:

```
func maxLength(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun maxLength(nums: IntArray): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```racket
(define/contract (max-length nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```cpp
/*
* Problem: Maximum Subarray With Equal Products
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

Java Solution:

```java
/**
* Problem: Maximum Subarray With Equal Products
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int maxLength(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Subarray With Equal Products
Difficulty: Easy
Tags: array, math

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Subarray With Equal Products
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Subarray With Equal Products
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

C# Solution:

```
/*
* Problem: Maximum Subarray With Equal Products
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

public class Solution {
public int MaxLength(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Subarray With Equal Products
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int maxLength(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximum Subarray With Equal Products
// Difficulty: Easy
// Tags: array, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func maxLength(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxLength(nums: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Subarray With Equal Products
// Difficulty: Easy
// Tags: array, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-length nums)
(-> (listof exact-integer?) exact-integer?)
)
```