Problem 1567: Maximum Length of Subarray With Positive Product

Problem Information

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

Problem Description

Given an array of integers

nums

, find the maximum length of a subarray where the product of all its elements is positive.

A subarray of an array is a consecutive sequence of zero or more values taken out of that array.

Return

the maximum length of a subarray with positive product

.

Example 1:

Input:

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

Output:

4

Explanation:

The array nums already has a positive product of 24.

Example 2:

Input:

nums = [0,1,-2,-3,-4]

Output:

3

Explanation:

The longest subarray with positive product is [1,-2,-3] which has a product of 6. Notice that we cannot include 0 in the subarray since that'll make the product 0 which is not positive.

Example 3:

Input:

nums = [-1,-2,-3,0,1]

Output:

2

Explanation:

The longest subarray with positive product is [-1,-2] or [-2,-3].

Constraints:

1 <= nums.length <= 10

5

-10

9

<= nums[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Length of Subarray With Positive Product
* Difficulty: Medium
* Tags: array, dp, greedy
*
```

```
* 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:
int getMaxLen(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Length of Subarray With Positive Product
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 int getMaxLen(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Length of Subarray With Positive Product
Difficulty: Medium
Tags: array, dp, greedy

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 getMaxLen(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Length of Subarray With Positive Product
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 getMaxLen = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Length of Subarray With Positive Product
* Difficulty: Medium
* Tags: array, dp, greedy
```

```
*
* 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 getMaxLen(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Length of Subarray With Positive Product
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 int GetMaxLen(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Length of Subarray With Positive Product
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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
*/

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

```
}
```

Go Solution:

```
// Problem: Maximum Length of Subarray With Positive Product
// Difficulty: Medium
// Tags: array, dp, greedy
//
// 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 getMaxLen(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Length of Subarray With Positive Product
// Difficulty: Medium
// Tags: array, dp, greedy
//
// 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 get_max_len(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

```
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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