Problem 152: Maximum Product Subarray

Problem Information

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

Problem Description

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.

Note

that the product of an array with a single element is the value of that element.

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

The product of any subarray of

nums

is

guaranteed

to fit in a

32-bit

integer.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}

}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Product Subarray
* Difficulty: Medium
* Tags: array, dp
*
```

```
* 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 maxProduct(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Product Subarray
* Difficulty: Medium
* Tags: array, dp
*
* 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 maxProduct(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Product Subarray
Difficulty: Medium
Tags: array, dp

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Product Subarray
* Difficulty: Medium
* Tags: array, dp
*
* 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 maxProduct = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Product Subarray
* Difficulty: Medium
* Tags: array, dp
```

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

};
```

C# Solution:

```
/*
* Problem: Maximum Product Subarray
* Difficulty: Medium
* Tags: array, dp
*
* 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 MaxProduct(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Product Subarray
* Difficulty: Medium
* Tags: array, dp
*
* 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 maxProduct(int* nums, int numsSize) {
}
```

Go Solution:

```
// Problem: Maximum Product Subarray
// Difficulty: Medium
// Tags: array, dp
//
// 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 maxProduct(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Product Subarray
// Difficulty: Medium
// Tags: array, dp
//
// 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 max_product(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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


}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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