Problem 1822: Sign of the Product of an Array

Problem Information

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

Problem Description

Implement a function

signFunc(x)

that returns:

1

if

x

is positive.

-1

if

x

is negative.

0

if

x

is equal to

0

.

You are given an integer array

nums

. Let

product

be the product of all values in the array

nums

.

Return

signFunc(product)

.

Example 1:

Input:

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

Output:

1

Explanation:

The product of all values in the array is 144, and signFunc(144) = 1

Example 2:

Input:

nums = [1,5,0,2,-3]

Output:

0

Explanation:

The product of all values in the array is 0, and signFunc(0) = 0

Example 3:

Input:

nums = [-1,1,-1,1,-1]

Output:

-1

Explanation:

The product of all values in the array is -1, and signFunc(-1) = -1

Constraints:

1 <= nums.length <= 1000

-100 <= nums[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (array-sign nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sign of the Product of an Array
* 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 arraySign(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Sign of the Product of an Array
* 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 arraySign(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Sign of the Product of an Array
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 arraySign(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Sign of the Product of an Array
* 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 arraySign = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Sign of the Product of an Array
* 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 arraySign(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Sign of the Product of an Array
* 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 ArraySign(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Sign of the Product of an Array
* 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 arraySign(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Sign of the Product of an Array
// 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 arraySign(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sign of the Product of an Array
// 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 array_sign(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (array-sign nums)
(-> (listof exact-integer?) exact-integer?)
)
```