Problem 3511: Make a Positive Array

Problem Information

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

Problem Description

You are given an array

nums

. An array is considered

positive

if the sum of all numbers in each

subarray

with

more than two

elements is positive.

You can perform the following operation any number of times:

Replace

one

element in

nums

with any integer between -10

18

and 10

18

.

Find the

minimum

number of operations needed to make

nums

positive

.

Example 1:

Input:

nums = [-10,15,-12]

Output:

1

Explanation:

The only subarray with more than 2 elements is the array itself. The sum of all elements is

(-10) + 15 + (-12) = -7

. By replacing

nums[0]

with 0, the new sum becomes

0 + 15 + (-12) = 3

. Thus, the array is now positive.

Example 2:

Input:

nums = [-1,-2,3,-1,2,6]

Output:

1

Explanation:

The only subarrays with more than 2 elements and a non-positive sum are:

Subarray Indices

Subarray

Sum

Subarray After Replacement (Set nums[1] = 1)

New Sum

nums[0...2]

[-1, -2, 3]

0

[-1, 1, 3]

3

nums[0...3]

[-1, -2, 3, -1]

-1

[-1, 1, 3, -1]

2

nums[1...3]

[-2, 3, -1]

0

[1, 3, -1]

3

Thus,

nums

is positive after one operation.

Example 3:

Input:

nums = [1,2,3]

Output:

0

Explanation:

The array is already positive, so no operations are needed.

Constraints:

3 <= nums.length <= 10

5

-10

9

<= nums[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
```

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Make a Positive Array
* Difficulty: Medium
* Tags: array, greedy
*
* 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 makeArrayPositive(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Make a Positive Array
* Difficulty: Medium
* Tags: array, greedy
*
* 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 makeArrayPositive(int[] nums) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Make a Positive Array
Difficulty: Medium
Tags: array, greedy

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Make a Positive Array
* Difficulty: Medium
* Tags: array, greedy
*
* 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 makeArrayPositive = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Make a Positive Array
* Difficulty: Medium
* Tags: array, greedy
*
* 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 makeArrayPositive(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Make a Positive Array
* Difficulty: Medium
* Tags: array, greedy
*
* 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 MakeArrayPositive(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Make a Positive Array
* Difficulty: Medium
* Tags: array, greedy
*
* 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 makeArrayPositive(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Make a Positive Array
// Difficulty: Medium
// Tags: array, greedy
//
// 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 makeArrayPositive(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Make a Positive Array
// Difficulty: Medium
// Tags: array, greedy
//
// 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 make_array_positive(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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