Problem 1413: Minimum Value to Get Positive Step by Step Sum

Problem Information

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

Problem Description

Given an array of integers

nums

, you start with an initial

positive

value

startValue

.

In each iteration, you calculate the step by step sum of

startValue

plus elements in

nums

(from left to right).

Return the minimum

positive

value of

startValue

such that the step by step sum is never less than 1.

Example 1:

Input:

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

Output:

5

Explanation:

If you choose startValue = 4, in the third iteration your step by step sum is less than 1.

step by step sum

startValue = 4 | startValue = 5 | nums

(4

-3

) = 1 | (5

-3

) = 2 | -3 (1

+2

) = 3 | (2

+2

) = 4 | 2 (3

-3

) = 0 | (4

-3

) = 1 | -3 (0

+4

) = 4 | (1

+4

) = 5 | 4 (4

+2

) = 6 | (5

+2

) = 7 | 2

Example 2:

Input:

nums = [1,2]

Output:

1

Explanation:

Minimum start value should be positive.

Example 3:

Input:

nums = [1,-2,-3]

Output:

5

Constraints:

1 <= nums.length <= 100

-100 <= nums[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-start-value nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Value to Get Positive Step by Step Sum
* Difficulty: Easy
* Tags: array
*
* 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 minStartValue(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Value to Get Positive Step by Step Sum
* Difficulty: Easy
* Tags: array
*
* 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 minStartValue(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Value to Get Positive Step by Step Sum
Difficulty: Easy
Tags: array

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Value to Get Positive Step by Step Sum
* Difficulty: Easy
* Tags: array
*
* 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 minStartValue = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Value to Get Positive Step by Step Sum
* Difficulty: Easy
* Tags: array
*
* 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 minStartValue(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Value to Get Positive Step by Step Sum
* Difficulty: Easy
* Tags: array
*
* 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 MinStartValue(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Value to Get Positive Step by Step Sum
* Difficulty: Easy
* Tags: array
*
* 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 minStartValue(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Value to Get Positive Step by Step Sum
// Difficulty: Easy
// Tags: array
//
// 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 minStartValue(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: Minimum Value to Get Positive Step by Step Sum
// Difficulty: Easy
// Tags: array
//
// 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 min_start_value(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-start-value nums)
(-> (listof exact-integer?) exact-integer?)
)
```