Problem 1785: Minimum Elements to Add to Form a Given Sum

Problem Information

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

Problem Description

You are given an integer array

nums

and two integers

limit

and

goal

. The array

nums

has an interesting property that

abs(nums[i]) <= limit

.

Return

the minimum number of elements you need to add to make the sum of the array equal to

goal

. The array must maintain its property that

abs(nums[i]) <= limit

.

Note that

abs(x)

equals

x

if

x >= 0

, and

-x

otherwise.

Example 1:

Input:

nums = [1,-1,1], limit = 3, goal = -4

Output:

2

Explanation:

You can add -2 and -3, then the sum of the array will be 1 - 1 + 1 - 2 - 3 = -4.

Example 2:

Input:

nums = [1,-10,9,1], limit = 100, goal = 0

Output:

1

Constraints:

1 <= nums.length <= 10

5

1 <= limit <= 10

6

-limit <= nums[i] <= limit

-10

9

<= goal <= 10

9

Code Snippets

C++:

```
class Solution {
public:
```

```
int minElements(vector<int>& nums, int limit, int goal) {

}
};
```

Java:

```
class Solution {
public int minElements(int[] nums, int limit, int goal) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} limit
* @param {number} goal
* @return {number}
*/
var minElements = function(nums, limit, goal) {

};
```

TypeScript:

```
function minElements(nums: number[], limit: number, goal: number): number {

};
```

C#:

```
public class Solution {
public int MinElements(int[] nums, int limit, int goal) {

}
}
```

C:

```
int minElements(int* nums, int numsSize, int limit, int goal) {

}
```

Go:

```
func minElements(nums []int, limit int, goal int) int {

}
```

Kotlin:

```
class Solution {
fun minElements(nums: IntArray, limit: Int, goal: Int): Int {

}
}
```

Swift:

```
class Solution {
func minElements(_ nums: [Int], _ limit: Int, _ goal: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_elements(nums: Vec<i32>, limit: i32, goal: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} limit
# @param {Integer} goal
# @return {Integer}
def min_elements(nums, limit, goal)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $limit
* @param Integer $goal
* @return Integer
*/
function minElements($nums, $limit, $goal) {

}
}
```

Dart:

```
class Solution {
int minElements(List<int> nums, int limit, int goal) {

}
}
```

Scala:

```
object Solution {
def minElements(nums: Array[Int], limit: Int, goal: Int): Int = {
```

```
}
}
```

Elixir:

```
defmodule Solution do
@spec min_elements(nums :: [integer], limit :: integer, goal :: integer) ::
integer
def min_elements(nums, limit, goal) do

end
end
```

Erlang:

```
-spec min_elements(Nums :: [integer()], Limit :: integer(), Goal ::
integer()) -> integer().
min_elements(Nums, Limit, Goal) ->
.
```

Racket:

```
(define/contract (min-elements nums limit goal)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Elements to Add to Form a Given Sum
* 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 minElements(vector<int>& nums, int limit, int goal) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Elements to Add to Form a Given Sum
* 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 minElements(int[] nums, int limit, int goal) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Elements to Add to Form a Given Sum
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 minElements(self, nums: List[int], limit: int, goal: int) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Elements to Add to Form a Given Sum
* 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
* @param {number} limit
* @param {number} goal
* @return {number}
*/
var minElements = function(nums, limit, goal) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Elements to Add to Form a Given Sum
* 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 minElements(nums: number[], limit: number, goal: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Elements to Add to Form a Given Sum
* 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 MinElements(int[] nums, int limit, int goal) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Elements to Add to Form a Given Sum
* 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 minElements(int* nums, int numsSize, int limit, int goal) {
```

```
}
```

Go Solution:

```
// Problem: Minimum Elements to Add to Form a Given Sum
// 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 minElements(nums []int, limit int, goal int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minElements(nums: IntArray, limit: Int, goal: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minElements(_ nums: [Int], _ limit: Int, _ goal: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Elements to Add to Form a Given Sum
// 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 min_elements(nums: Vec<i32>, limit: i32, goal: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} limit
# @param {Integer} goal
# @return {Integer}
def min_elements(nums, limit, goal)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $limit
* @param Integer $goal
* @return Integer
*/
function minElements($nums, $limit, $goal) {

}
}
```

Dart Solution:

```
class Solution {
int minElements(List<int> nums, int limit, int goal) {

}
}
```

Scala Solution:

```
object Solution {
def minElements(nums: Array[Int], limit: Int, goal: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_elements(nums :: [integer], limit :: integer, goal :: integer) ::
integer
def min_elements(nums, limit, goal) do

end
end
```

Erlang Solution:

```
-spec min_elements(Nums :: [integer()], Limit :: integer(), Goal ::
integer()) -> integer().
min_elements(Nums, Limit, Goal) ->
.
```

Racket Solution:

```
(define/contract (min-elements nums limit goal)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```