Problem 2567: Minimum Score by Changing Two Elements

Problem Information

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

Problem Description

You are given an integer array

nums

.

The

low

score of

nums

is the

minimum

absolute difference between any two integers.

The

high

score of

nums

is the

maximum

absolute difference between any two integers.

The

score

of

nums

is the sum of the

high

and

low

scores.

Return the

minimum score

after

changing two elements

of

nums

.

Example 1:

Input:

nums = [1,4,7,8,5]

Output:

3

Explanation:

Change

nums[0]

and

nums[1]

to be 6 so that

nums

becomes [6,6,7,8,5].

The low score is the minimum absolute difference: |6 - 6| = 0.

The high score is the maximum absolute difference: |8 - 5| = 3.

The sum of high and low score is 3.

Example 2:

Input:

nums = [1,4,3]

Output:

0

Explanation:

Change

nums[1]

and

nums[2]

to 1 so that

nums

becomes [1,1,1].

The sum of maximum absolute difference and minimum absolute difference is 0.

Constraints:

3 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
```

```
int minimizeSum(vector<int>& nums) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (minimize-sum nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Score by Changing Two Elements
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 minimizeSum(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Score by Changing Two Elements
```

```
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 minimizeSum(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Score by Changing Two Elements
Difficulty: Medium
Tags: array, greedy, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Score by Changing Two Elements
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 minimizeSum = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Score by Changing Two Elements
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 minimizeSum(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Score by Changing Two Elements
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 MinimizeSum(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Score by Changing Two Elements
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 minimizeSum(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Score by Changing Two Elements
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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 minimizeSum(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Score by Changing Two Elements
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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 minimize_sum(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (minimize-sum nums)
(-> (listof exact-integer?) exact-integer?)
)
```