Problem 1085: Sum of Digits in the Minimum Number

Problem Information

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

Problem Description

Given an integer array

nums

, return

0

if the sum of the digits of the minimum integer in

nums

is odd, or

1

otherwise

.

Example 1:

Input:

nums = [34,23,1,24,75,33,54,8]

Output:

0

Explanation:

The minimal element is 1, and the sum of those digits is 1 which is odd, so the answer is 0.

Example 2:

Input:

nums = [99,77,33,66,55]

Output:

1

Explanation:

The minimal element is 33, and the sum of those digits is 3 + 3 = 6 which is even, so the answer is 1.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 100

Code Snippets

C++:

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

}
```

```
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Sum of Digits in the Minimum Number
* 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 sumOfDigits(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Digits in the Minimum Number
* 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 sumOfDigits(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of Digits in the Minimum Number
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 sumOfDigits(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Digits in the Minimum Number
* 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 sumOfDigits = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Digits in the Minimum Number
* 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 sumOfDigits(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Sum of Digits in the Minimum Number
* 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 SumOfDigits(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Sum of Digits in the Minimum Number
* 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 sumOfDigits(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Sum of Digits in the Minimum Number
// 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 sumOfDigits(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sum of Digits in the Minimum Number
// 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 sum_of_digits(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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