Problem 1991: Find the Middle Index in Array

Problem Information

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

Problem Description

Given a

0-indexed

integer array

nums

, find the

leftmost

middleIndex

(i.e., the smallest amongst all the possible ones).

A

middleIndex

is an index where

nums[0] + nums[1] + ... + nums[middleIndex-1] == nums[middleIndex+1] + nums[middleIndex+2] + ... + nums[nums.length-1]

.

If

middleIndex == 0

, the left side sum is considered to be

0

. Similarly, if

middleIndex == nums.length - 1

, the right side sum is considered to be

0

.

Return

the

leftmost

middleIndex

that satisfies the condition, or

-1

if there is no such index

.

Example 1:

Input:

nums = [2,3,-1,

8

,4]

Output:

3

Explanation:

The sum of the numbers before index 3 is: 2 + 3 + -1 = 4 The sum of the numbers after index 3 is: 4 = 4

Example 2:

Input:

nums = [1,-1,

4

]

Output:

2

Explanation:

The sum of the numbers before index 2 is: 1 + -1 = 0 The sum of the numbers after index 2 is: 0

Example 3:

Input:

nums = [2,5]

Output:

-1

Explanation:

There is no valid middleIndex.

Constraints:

1 <= nums.length <= 100

-1000 <= nums[i] <= 1000

Note:

This question is the same as 724:

https://leetcode.com/problems/find-pivot-index/

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
```

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (find-middle-index nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Middle Index in Array
* 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 findMiddleIndex(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Middle Index in Array
* 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 findMiddleIndex(int[] nums) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Find the Middle Index in Array
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 findMiddleIndex(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the Middle Index in Array
* 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 findMiddleIndex = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Middle Index in Array
* 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 findMiddleIndex(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Find the Middle Index in Array
* 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 FindMiddleIndex(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Find the Middle Index in Array
* 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 findMiddleIndex(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Find the Middle Index in Array
// 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 findMiddleIndex(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Find the Middle Index in Array
// 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 find_middle_index(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-middle-index nums)
(-> (listof exact-integer?) exact-integer?)
)
```