Problem 805: Split Array With Same Average

Problem Information

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

Problem Description

You are given an integer array

nums

.

You should move each element of

nums

into one of the two arrays

A

and

B

such that

A

and

B

are non-empty, and

average(A) == average(B)

.

Return

true

if it is possible to achieve that and

false

otherwise.

Note

that for an array

arr

,

average(arr)

is the sum of all the elements of

arr

over the length of

arr

.

Example 1:

Input:

nums = [1,2,3,4,5,6,7,8]

Output:

true

Explanation:

We can split the array into [1,4,5,8] and [2,3,6,7], and both of them have an average of 4.5.

Example 2:

Input:

nums = [3,1]

Output:

false

Constraints:

1 <= nums.length <= 30

0 <= nums[i] <= 10

4

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean splitArraySameAverage(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function splitArraySameAverage(nums: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool SplitArraySameAverage(int[] nums) {
}
}
```

C:

```
bool splitArraySameAverage(int* nums, int numsSize) {

}
```

Go:

```
func splitArraySameAverage(nums []int) bool {

}
```

Kotlin:

```
class Solution {
fun splitArraySameAverage(nums: IntArray): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool splitArraySameAverage(List<int> nums) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (split-array-same-average nums)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Split Array With Same Average
* Difficulty: Hard
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
bool splitArraySameAverage(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Split Array With Same Average
* Difficulty: Hard
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public boolean splitArraySameAverage(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Split Array With Same Average
Difficulty: Hard
Tags: array, dp, math

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def splitArraySameAverage(self, nums: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Split Array With Same Average
* Difficulty: Hard
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Split Array With Same Average
* Difficulty: Hard
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function splitArraySameAverage(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Split Array With Same Average
* Difficulty: Hard
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

public class Solution {
public bool SplitArraySameAverage(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Split Array With Same Average
* Difficulty: Hard
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

bool splitArraySameAverage(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Split Array With Same Average
// Difficulty: Hard
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func splitArraySameAverage(nums []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun splitArraySameAverage(nums: IntArray): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Split Array With Same Average
// Difficulty: Hard
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn split_array_same_average(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

```
class Solution {
bool splitArraySameAverage(List<int> nums) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (split-array-same-average nums)
(-> (listof exact-integer?) boolean?)
)
```