Problem 548: Split Array with Equal Sum

Problem Information

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

Problem Description

Given an integer array

nums

of length

n

, return

true

if there is a triplet

(i, j, k)

which satisfies the following conditions:

0 < i, i + 1 < j, j + 1 < k < n - 1

The sum of subarrays

(0, i - 1)

,

(i + 1, j - 1)

,

(j + 1, k - 1)

and

(k + 1, n - 1)

is equal.

A subarray

(l, r)

represents a slice of the original array starting from the element indexed

l

to the element indexed

r

.

Example 1:

Input:

nums = [1,2,1,2,1,2,1]

Output:

true

Explanation:

i = 1, j = 3, k = 5. sum(0, i - 1) = sum(0, 0) = 1 sum(i + 1, j - 1) = sum(2, 2) = 1 sum(j + 1, k - 1) = sum(4, 4) = 1 sum(k + 1, n - 1) = sum(6, 6) = 1

Example 2:

Input:

nums = [1,2,1,2,1,2,1,2]

Output:

false

Constraints:

n == nums.length

1 <= n <= 2000

-10

6

<= nums[i] <= 10

6

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Split Array with Equal Sum
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Split Array with Equal Sum
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

```
}
}
```

Python3 Solution:

```
"""
Problem: Split Array with Equal Sum
Difficulty: Hard
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Split Array with Equal Sum
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Split Array with Equal Sum
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

C# Solution:

```
/*
* Problem: Split Array with Equal Sum
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
```

```
}
```

C Solution:

```
/*
* Problem: Split Array with Equal Sum
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

bool splitArray(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Split Array with Equal Sum
// Difficulty: Hard
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func splitArray(nums []int) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Split Array with Equal Sum
// Difficulty: Hard
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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