Problem 3698: Split Array With Minimum Difference

Problem Information

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

Problem Description

You are given an integer array

nums

.

Split the array into

exactly

two

subarrays

,

left

and

right

, such that

left

is

strictly increasing

and

right

is

strictly decreasing

.

Return the

minimum possible absolute difference

between the sums of

left

and

right

. If no valid split exists, return

-1

.

Example 1:

Input:

nums = [1,3,2]

Output:

2

Explanation:

i

left

right

Validity

left

sum

right

sum

Absolute difference

0

[1]

[3, 2]

Yes

1

5

|1 - 5| = 4

1

[1, 3]

[2]

Yes

4

2

|4 - 2| = 2

Thus, the minimum absolute difference is 2.

Example 2:

Input:

nums = [1,2,4,3]

Output:

4

Explanation:

i

left

right

Validity

left

sum

right

sum

Absolute difference

0

[1]

[2, 4, 3]

No

1

9

-

1

[1, 2]

[4, 3]

Yes

3

7

|3 - 7| = 4

2

[1, 2, 4]

[3]

Yes

7

3

|7 - 3| = 4

Thus, the minimum absolute difference is 4.

Example 3:

Input:

nums = [3,1,2]

Output:

-1

Explanation:

No valid split exists, so the answer is -1.

Constraints:

2 <= nums.length <= 10

5

1 <= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

```
};
```

C#:

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

}
}
```

C:

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

}
```

Go:

```
func splitArray(nums []int) int64 {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn split_array(nums: Vec<i32>) -> i64 {
```

```
}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Split Array With Minimum Difference
* Difficulty: Medium
* 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:
long long splitArray(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Split Array With Minimum Difference
* Difficulty: Medium
* 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 long splitArray(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Split Array With Minimum Difference
Difficulty: Medium
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 splitArray(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Split Array With Minimum Difference
* Difficulty: Medium
* 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 splitArray = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Split Array With Minimum Difference
* Difficulty: Medium
* 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 splitArray(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Split Array With Minimum Difference
* Difficulty: Medium
* 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 long SplitArray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Split Array With Minimum Difference
* Difficulty: Medium
* 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
*/

long long splitArray(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Split Array With Minimum Difference
// Difficulty: Medium
// 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 splitArray(nums []int) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Split Array With Minimum Difference
// Difficulty: Medium
// 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 split_array(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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