Problem 413: Arithmetic Slices

Problem Information

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

Problem Description

An integer array is called arithmetic if it consists of

at least three elements

and if the difference between any two consecutive elements is the same.

For example,

[1,3,5,7,9]

,

[7,7,7,7]

, and

[3,-1,-5,-9]

are arithmetic sequences.

Given an integer array

nums

, return

the number of arithmetic

subarrays

of

nums

.

A

subarray

is a contiguous subsequence of the array.

Example 1:

Input:

nums = [1,2,3,4]

Output:

3

Explanation:

We have 3 arithmetic slices in nums: [1, 2, 3], [2, 3, 4] and [1,2,3,4] itself.

Example 2:

Input:

nums = [1]

Output:

0

Constraints:

1 <= nums.length <= 5000

-1000 <= nums[i] <= 1000

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (number-of-arithmetic-slices nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Arithmetic Slices
* Difficulty: Medium
* Tags: array, dp
*
* 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:
int numberOfArithmeticSlices(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Arithmetic Slices
* Difficulty: Medium
* Tags: array, dp
*
* 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 int numberOfArithmeticSlices(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Arithmetic Slices
Difficulty: Medium
Tags: array, dp

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 numberOfArithmeticSlices(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Arithmetic Slices
* Difficulty: Medium
* Tags: array, dp
*
* 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 {number}
*/
var numberOfArithmeticSlices = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Arithmetic Slices
* Difficulty: Medium
* Tags: array, dp
*
* 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 numberOfArithmeticSlices(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Arithmetic Slices
* Difficulty: Medium
* Tags: array, dp
*
* 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 int NumberOfArithmeticSlices(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Arithmetic Slices
* Difficulty: Medium
* Tags: array, dp
*
* 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
*/

int numberOfArithmeticSlices(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Arithmetic Slices
// Difficulty: Medium
// Tags: array, dp
//
// 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 numberOfArithmeticSlices(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Arithmetic Slices
// Difficulty: Medium
// Tags: array, dp
//
// 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 number_of_arithmetic_slices(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (number-of-arithmetic-slices nums)
(-> (listof exact-integer?) exact-integer?)
)
```