Problem 1630: Arithmetic Subarrays

Problem Information

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

Problem Description

A sequence of numbers is called

arithmetic

if it consists of at least two elements, and the difference between every two consecutive elements is the same. More formally, a sequence

s

is arithmetic if and only if

s[i+1] - s[i] == s[1] - s[0]

for all valid

i

.

For example, these are

arithmetic

sequences:

1, 3, 5, 7, 9 7, 7, 7, 7 3, -1, -5, -9

The following sequence is not

arithmetic

:

1, 1, 2, 5, 7

You are given an array of

n

integers,

nums

, and two arrays of

m

integers each,

l

and

r

, representing the

m

range queries, where the

i

th

query is the range

[l[i], r[i]]

. All the arrays are

0-indexed

.

Return

a list of

boolean

elements

answer

, where

answer[i]

is

true

if the subarray

nums[l[i]], nums[l[i]+1], ... , nums[r[i]]

can be

rearranged

to form an

arithmetic

sequence, and

false

otherwise.

Example 1:

Input:

nums =

[4,6,5,9,3,7]

, l =

[0,0,2]

, r =

[2,3,5]

Output:

[true,false,true]

Explanation:

In the 0

th

query, the subarray is [4,6,5]. This can be rearranged as [6,5,4], which is an arithmetic sequence. In the 1

st

query, the subarray is [4,6,5,9]. This cannot be rearranged as an arithmetic sequence. In the 2

nd

query, the subarray is

[5,9,3,7]. This

can be rearranged as

[3,5,7,9]

, which is an arithmetic sequence.

Example 2:

Input:

nums = [-12,-9,-3,-12,-6,15,20,-25,-20,-15,-10], l = [0,1,6,4,8,7], r = [4,4,9,7,9,10]

Output:

[false,true,false,false,true,true]

Constraints:

n == nums.length

m == l.length

m == r.length

2 <= n <= 500

1 <= m <= 500

0 <= l[i] < r[i] < n

-10

5

<= nums[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
vector<bool> checkArithmeticSubarrays(vector<int>& nums, vector<int>& l,
vector<int>& r) {

}
};
```

Java:

```
class Solution {
public List<Boolean> checkArithmeticSubarrays(int[] nums, int[] l, int[] r) {

}
}
```

Python3:

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

Python:

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

```
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number[]} l
* @param {number[]} r
* @return {boolean[]}
*/
var checkArithmeticSubarrays = function(nums, l, r) {

};
```

TypeScript:

```
function checkArithmeticSubarrays(nums: number[], l: number[], r: number[]):
boolean[] {

};
```

C#:

```
public class Solution {
public IList<bool> CheckArithmeticSubarrays(int[] nums, int[] l, int[] r) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
bool* checkArithmeticSubarrays(int* nums, int numsSize, int* l, int lSize,
int* r, int rSize, int* returnSize) {

}
```

Go:

```
func checkArithmeticSubarrays(nums []int, l []int, r []int) []bool {

}
```

Kotlin:

```
class Solution {
fun checkArithmeticSubarrays(nums: IntArray, l: IntArray, r: IntArray):
List<Boolean> {

}
}
```

Swift:

```
class Solution {
func checkArithmeticSubarrays(_ nums: [Int], _ l: [Int], _ r: [Int]) ->
[Bool] {

}
}
```

Rust:

```
impl Solution {
pub fn check_arithmetic_subarrays(nums: Vec<i32>, l: Vec<i32>, r: Vec<i32>)
-> Vec<bool> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[]} l
# @param {Integer[]} r
# @return {Boolean[]}
def check_arithmetic_subarrays(nums, l, r)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $l
* @param Integer[] $r
* @return Boolean[]
*/
function checkArithmeticSubarrays($nums, $l, $r) {

}
}
```

Dart:

```
class Solution {
List<bool> checkArithmeticSubarrays(List<int> nums, List<int> l, List<int> r)
{

}
}
```

Scala:

```
object Solution {
def checkArithmeticSubarrays(nums: Array[Int], l: Array[Int], r: Array[Int]):
List[Boolean] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec check_arithmetic_subarrays(nums :: [integer], l :: [integer], r ::
[integer]) :: [boolean]
def check_arithmetic_subarrays(nums, l, r) do

end
end
```

Erlang:

```
-spec check_arithmetic_subarrays(Nums :: [integer()], L :: [integer()], R ::
[integer()]) -> [boolean()].
check_arithmetic_subarrays(Nums, L, R) ->
.
```

Racket:

```
(define/contract (check-arithmetic-subarrays nums l r)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?)
(listof boolean?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Arithmetic Subarrays
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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:
vector<bool> checkArithmeticSubarrays(vector<int>& nums, vector<int>& l,
vector<int>& r) {

}
};
```

Java Solution:

```
/**
* Problem: Arithmetic Subarrays
* Difficulty: Medium
* Tags: array, hash, sort
*
```

```
* 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 List<Boolean> checkArithmeticSubarrays(int[] nums, int[] l, int[] r) {

}
}
```

Python3 Solution:

```
"""
Problem: Arithmetic Subarrays
Difficulty: Medium
Tags: array, hash, sort

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 checkArithmeticSubarrays(self, nums: List[int], l: List[int], r:
List[int]) -> List[bool]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Arithmetic Subarrays
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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
* @param {number[]} l
* @param {number[]} r
* @return {boolean[]}
*/
var checkArithmeticSubarrays = function(nums, l, r) {

};
```

TypeScript Solution:

```
/**
* Problem: Arithmetic Subarrays
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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 checkArithmeticSubarrays(nums: number[], l: number[], r: number[]):
boolean[] {

};
```

C# Solution:

```
/*
* Problem: Arithmetic Subarrays
* Difficulty: Medium
```

```
* Tags: array, hash, sort
*
* 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 IList<bool> CheckArithmeticSubarrays(int[] nums, int[] l, int[] r) {

}
}
```

C Solution:

```
/*
* Problem: Arithmetic Subarrays
* Difficulty: Medium
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
bool* checkArithmeticSubarrays(int* nums, int numsSize, int* l, int lSize,
int* r, int rSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Arithmetic Subarrays
// Difficulty: Medium
// Tags: array, hash, sort
//
// 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 checkArithmeticSubarrays(nums []int, l []int, r []int) []bool {

}
```

Kotlin Solution:

```
class Solution {
fun checkArithmeticSubarrays(nums: IntArray, l: IntArray, r: IntArray):
List<Boolean> {

}
}
```

Swift Solution:

```
class Solution {
func checkArithmeticSubarrays(_ nums: [Int], _ l: [Int], _ r: [Int]) ->
[Bool] {

}
}
```

Rust Solution:

```
// Problem: Arithmetic Subarrays
// Difficulty: Medium
// Tags: array, hash, sort
//
// 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 check_arithmetic_subarrays(nums: Vec<i32>, l: Vec<i32>, r: Vec<i32>)
-> Vec<bool> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[]} l
# @param {Integer[]} r
# @return {Boolean[]}
def check_arithmetic_subarrays(nums, l, r)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $l
* @param Integer[] $r
* @return Boolean[]
*/
function checkArithmeticSubarrays($nums, $l, $r) {

}
}
```

Dart Solution:

```
class Solution {
List<bool> checkArithmeticSubarrays(List<int> nums, List<int> l, List<int> r)
{

}
}
```

Scala Solution:

```
object Solution {
def checkArithmeticSubarrays(nums: Array[Int], l: Array[Int], r: Array[Int]):
List[Boolean] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec check_arithmetic_subarrays(nums :: [integer], l :: [integer], r ::
[integer]) :: [boolean]
def check_arithmetic_subarrays(nums, l, r) do

end
end
```

Erlang Solution:

```
-spec check_arithmetic_subarrays(Nums :: [integer()], L :: [integer()], R ::
[integer()]) -> [boolean()].
check_arithmetic_subarrays(Nums, L, R) ->
.
```

Racket Solution:

```
(define/contract (check-arithmetic-subarrays nums l r)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?)
(listof boolean?))
)
```