Problem 3676: Count Bowl Subarrays

Problem Information

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

Problem Description

You are given an integer array

nums

with

distinct

elements.

A

subarray

nums[l...r]

of

nums

is called a

bowl

if:

The subarray has length at least 3. That is,

r - l + 1 >= 3

.

The

minimum

of its two ends is

strictly greater

than the

maximum

of all elements in between. That is,

min(nums[l], nums[r]) > max(nums[l + 1], ..., nums[r - 1])

.

Return the number of

bowl

subarrays in

nums

.

Example 1:

Input:

nums = [2,5,3,1,4]

Output:

2

Explanation:

The bowl subarrays are

[3, 1, 4]

and

[5, 3, 1, 4]

.

[3, 1, 4]

is a bowl because

min(3, 4) = 3 > max(1) = 1

.

[5, 3, 1, 4]

is a bowl because

min(5, 4) = 4 > max(3, 1) = 3

.

Example 2:

Input:

nums = [5,1,2,3,4]

Output:

3

Explanation:

The bowl subarrays are

[5, 1, 2]

,

[5, 1, 2, 3]

and

[5, 1, 2, 3, 4]

.

Example 3:

Input:

nums =

[1000000000,999999999,999999998]

Output:

0

Explanation:

No subarray is a bowl.

Constraints:

3 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

nums

consists of distinct elements.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

```
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (bowl-subarrays nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Bowl Subarrays
* Difficulty: Medium
* Tags: array, stack
*
* 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 bowlSubarrays(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Count Bowl Subarrays
* Difficulty: Medium
* Tags: array, stack
*
* 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 bowlSubarrays(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Bowl Subarrays
Difficulty: Medium
Tags: array, stack
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Bowl Subarrays
* Difficulty: Medium
* Tags: array, stack
*
* 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 bowlSubarrays = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Bowl Subarrays
* Difficulty: Medium
* Tags: array, stack
*
* 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 bowlSubarrays(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Bowl Subarrays
* Difficulty: Medium
* Tags: array, stack
*
* 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 BowlSubarrays(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Count Bowl Subarrays
* Difficulty: Medium
* Tags: array, stack
*
* 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 bowlSubarrays(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Count Bowl Subarrays
// Difficulty: Medium
// Tags: array, stack
//
// 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 bowlSubarrays(nums []int) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Bowl Subarrays
// Difficulty: Medium
// Tags: array, stack
```

```
//
// 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 bowl_subarrays(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (bowl-subarrays nums)
(-> (listof exact-integer?) exact-integer?)
)
```