Problem 2012: Sum of Beauty in the Array

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

. For each index

i

(

1 <= i <= nums.length - 2

) the

beauty

of

nums[i]

equals:

2

, if

nums[j] < nums[i] < nums[k]

, for

all

0 <= j < i

and for

all

i < k <= nums.length - 1

.

1

, if

nums[i - 1] < nums[i] < nums[i + 1]

, and the previous condition is not satisfied.

0

, if none of the previous conditions holds.

Return

the

sum of beauty

of all

nums[i]

where

1 <= i <= nums.length - 2

.

Example 1:

Input:

nums = [1,2,3]

Output:

2

Explanation:

For each index i in the range 1 <= i <= 1: - The beauty of nums[1] equals 2.

Example 2:

Input:

nums = [2,4,6,4]

Output:

1

Explanation:

For each index i in the range 1 <= i <= 2: - The beauty of nums[1] equals 1. - The beauty of nums[2] equals 0.

Example 3:

Input:

nums = [3,2,1]

Output:

0

Explanation:

For each index i in the range 1 <= i <= 1: - The beauty of nums[1] equals 0.

Constraints:

3 <= nums.length <= 10

5

1 <= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (sum-of-beauties nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of Beauty in the Array
* 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:
int sumOfBeauties(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Beauty in the Array
* 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 int sumOfBeauties(int[] nums) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Sum of Beauty in the Array
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 sumOfBeauties(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Beauty in the Array
* 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 sumOfBeauties = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Beauty in the Array
* 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 sumOfBeauties(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Sum of Beauty in the Array
* 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 int SumOfBeauties(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Sum of Beauty in the Array
* 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
*/

int sumOfBeauties(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Sum of Beauty in the Array
// 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 sumOfBeauties(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Sum of Beauty in the Array
// 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 sum_of_beauties(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (sum-of-beauties nums)
(-> (listof exact-integer?) exact-integer?)
)
```