Problem 2210: Count Hills and Valleys in an Array

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

. An index

i

is part of a

hill

in

nums

if the closest non-equal neighbors of

i

are smaller than

nums[i]

. Similarly, an index

i

is part of a

valley

in

nums

if the closest non-equal neighbors of

i

are larger than

nums[i]

. Adjacent indices

i

and

j

are part of the

same

hill or valley if

nums[i] == nums[j]

.

Note that for an index to be part of a hill or valley, it must have a non-equal neighbor on

both

the left and right of the index.

Return

the number of hills and valleys in

nums

.

Example 1:

Input:

nums = [2,4,1,1,6,5]

Output:

3

Explanation:

At index 0: There is no non-equal neighbor of 2 on the left, so index 0 is neither a hill nor a valley. At index 1: The closest non-equal neighbors of 4 are 2 and 1. Since 4 > 2 and 4 > 1, index 1 is a hill. At index 2: The closest non-equal neighbors of 1 are 4 and 6. Since 1 < 4 and 1 < 6, index 2 is a valley. At index 3: The closest non-equal neighbors of 1 are 4 and 6. Since 1 < 4 and 1 < 6, index 3 is a valley, but note that it is part of the same valley as index 2. At index 4: The closest non-equal neighbors of 6 are 1 and 5. Since 6 > 1 and 6 > 5, index 4 is a hill. At index 5: There is no non-equal neighbor of 5 on the right, so index 5 is neither a hill nor a valley. There are 3 hills and valleys so we return 3.

Example 2:

Input:

nums = [6,6,5,5,4,1]

Output:

0

Explanation:

At index 0: There is no non-equal neighbor of 6 on the left, so index 0 is neither a hill nor a valley. At index 1: There is no non-equal neighbor of 6 on the left, so index 1 is neither a hill nor a valley. At index 2: The closest non-equal neighbors of 5 are 6 and 4. Since 5 < 6 and 5 > 4, index 2 is neither a hill nor a valley. At index 3: The closest non-equal neighbors of 5 are 6 and 4. Since 5 < 6 and 5 > 4, index 3 is neither a hill nor a valley. At index 4: The closest non-equal neighbors of 4 are 5 and 1. Since 4 < 5 and 4 > 1, index 4 is neither a hill nor a valley. At index 5: There is no non-equal neighbor of 1 on the right, so index 5 is neither a hill nor a valley. There are 0 hills and valleys so we return 0.

Constraints:

3 <= nums.length <= 100

1 <= nums[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (count-hill-valley nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Hills and Valleys in an Array
* Difficulty: Easy
* 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 countHillValley(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Count Hills and Valleys in an Array
* Difficulty: Easy
* 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 countHillValley(int[] nums) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Count Hills and Valleys in an Array
Difficulty: Easy
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 countHillValley(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Hills and Valleys in an Array
* Difficulty: Easy
* 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 countHillValley = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Hills and Valleys in an Array
* Difficulty: Easy
* 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 countHillValley(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Hills and Valleys in an Array
* Difficulty: Easy
* 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 CountHillValley(int[] nums) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Count Hills and Valleys in an Array
* Difficulty: Easy
* 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 countHillValley(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Count Hills and Valleys in an Array
// Difficulty: Easy
// 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 countHillValley(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Hills and Valleys in an Array
// Difficulty: Easy
// 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 count_hill_valley(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (count-hill-valley nums)
(-> (listof exact-integer?) exact-integer?)
)
```