Problem 3748: Count Stable Subarrays

Problem Information

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

Problem Description

You are given an integer array

nums

.

A

subarray

of

nums

is called

stable

if it contains

no inversions

, i.e., there is no pair of indices

i < j

such that

nums[i] > nums[j]

.

You are also given a

2D integer array

queries

of length

q

, where each

queries[i] = [l

i

, r

i

]

represents a query. For each query

[l

i

, r

i

]

, compute the number of

stable subarrays

that lie entirely within the segment

nums[l

i

..r

i

]

.

Return an integer array

ans

of length

q

, where

ans[i]

is the answer to the

i

th

query.

Note

:

A single element subarray is considered stable.

Example 1:

Input:

nums = [3,1,2], queries = [[0,1],[1,2],[0,2]]

Output:

[2,3,4]

Explanation:

For

queries[0] = [0, 1]

, the subarray is

[nums[0], nums[1]] = [3, 1]

.

The stable subarrays are

[3]

and

[1]

. The total number of stable subarrays is 2.

For

queries[1] = [1, 2]

, the subarray is

[nums[1], nums[2]] = [1, 2]

.

The stable subarrays are

[1]

,

[2]

, and

[1, 2]

. The total number of stable subarrays is 3.

For

queries[2] = [0, 2]

, the subarray is

[nums[0], nums[1], nums[2]] = [3, 1, 2]

.

The stable subarrays are

[3]

,

[1]

,

[2]

, and

[1, 2]

. The total number of stable subarrays is 4.

Thus,

ans = [2, 3, 4]

.

Example 2:

Input:

nums = [2,2], queries = [[0,1],[0,0]]

Output:

[3,1]

Explanation:

For

queries[0] = [0, 1]

, the subarray is

[nums[0], nums[1]] = [2, 2]

.

The stable subarrays are

[2]

,

[2]

, and

[2, 2]

. The total number of stable subarrays is 3.

For

queries[1] = [0, 0]

, the subarray is

[nums[0]] = [2]

.

The stable subarray is

[2]

. The total number of stable subarrays is 1.

Thus,

ans = [3, 1]

.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

5

1 <= queries.length <= 10

5

queries[i] = [l

i

, r

i

]

0 <= l

i

<= r

i

<= nums.length - 1

Code Snippets

C++:

```
class Solution {
public:
```

```
vector<long long> countStableSubarrays(vector<int>& nums,
vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public long[] countStableSubarrays(int[] nums, int[][] queries) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function countStableSubarrays(nums: number[], queries: number[][]): number[]
{

};
```

C#:

```
public class Solution {
public long[] CountStableSubarrays(int[] nums, int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* countStableSubarrays(int* nums, int numsSize, int** queries, int
queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go:

```
func countStableSubarrays(nums []int, queries [][]int) []int64 {

}
```

Kotlin:

```
class Solution {
fun countStableSubarrays(nums: IntArray, queries: Array<IntArray>): LongArray
{

}
}
```

Swift:

```
class Solution {
func countStableSubarrays(_ nums: [Int], _ queries: [[Int]]) -> [Int] {
}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[][]} queries
# @return {Integer[]}
def count_stable_subarrays(nums, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[][] $queries
* @return Integer[]
*/
function countStableSubarrays($nums, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> countStableSubarrays(List<int> nums, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def countStableSubarrays(nums: Array[Int], queries: Array[Array[Int]]):
Array[Long] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_stable_subarrays(nums :: [integer], queries :: [[integer]]) ::
[integer]
def count_stable_subarrays(nums, queries) do

end
end
```

Erlang:

```
-spec count_stable_subarrays(Nums :: [integer()], Queries :: [[integer()]])
-> [integer()].
count_stable_subarrays(Nums, Queries) ->
.
```

Racket:

```
(define/contract (count-stable-subarrays nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Stable Subarrays
* Difficulty: Hard
* Tags: array, search
```

```
*
* 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:
vector<long long> countStableSubarrays(vector<int>& nums,
vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Count Stable Subarrays
* Difficulty: Hard
* Tags: array, search
*
* 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[] countStableSubarrays(int[] nums, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Stable Subarrays
Difficulty: Hard
Tags: array, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Stable Subarrays
* Difficulty: Hard
* Tags: array, search
*
* 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
* @param {number[][]} queries
* @return {number[]}
*/
var countStableSubarrays = function(nums, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Stable Subarrays
* Difficulty: Hard
* Tags: array, search
*
* 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 countStableSubarrays(nums: number[], queries: number[][]): number[]
{

};
```

C# Solution:

```
/*
* Problem: Count Stable Subarrays
* Difficulty: Hard
* Tags: array, search
*
* 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[] CountStableSubarrays(int[] nums, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Count Stable Subarrays
* Difficulty: Hard
* Tags: array, search
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* countStableSubarrays(int* nums, int numsSize, int** queries, int queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Count Stable Subarrays
// Difficulty: Hard
// Tags: array, search
//
// 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 countStableSubarrays(nums []int, queries [][]int) []int64 {

}
```

Kotlin Solution:

```
class Solution {
fun countStableSubarrays(nums: IntArray, queries: Array<IntArray>): LongArray {

}
}
```

Swift Solution:

```
class Solution {
func countStableSubarrays(_ nums: [Int], _ queries: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Count Stable Subarrays
// Difficulty: Hard
// Tags: array, search
//
// 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_stable_subarrays(nums: Vec<i32>, queries: Vec<Vec<i32>>) ->
Vec<i64> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[][]} queries
# @return {Integer[]}
def count_stable_subarrays(nums, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[][] $queries
* @return Integer[]
*/
function countStableSubarrays($nums, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> countStableSubarrays(List<int> nums, List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def countStableSubarrays(nums: Array[Int], queries: Array[Array[Int]]):
Array[Long] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_stable_subarrays(nums :: [integer], queries :: [[integer]]) ::
[integer]
def count_stable_subarrays(nums, queries) do

end
end
```

Erlang Solution:

```
-spec count_stable_subarrays(Nums :: [integer()], Queries :: [[integer()]])
-> [integer()].
count_stable_subarrays(Nums, Queries) ->
.
```

Racket Solution:

```
(define/contract (count-stable-subarrays nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```