Problem 2848: Points That Intersect With Cars

Problem Information

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

Problem Description

You are given a

0-indexed

2D integer array

nums

representing the coordinates of the cars parking on a number line. For any index

i

,

nums[i] = [start

i

, end

i

]

where

start

i

is the starting point of the

i

th

car and

end

i

is the ending point of the

i

th

car.

Return

the number of integer points on the line that are covered with

any part

of a car.

Example 1:

Input:

nums = [[3,6],[1,5],[4,7]]

Output:

7

Explanation:

All the points from 1 to 7 intersect at least one car, therefore the answer would be 7.

Example 2:

Input:

nums = [[1,3],[5,8]]

Output:

7

Explanation:

Points intersecting at least one car are 1, 2, 3, 5, 6, 7, 8. There are a total of 7 points, therefore the answer would be 7.

Constraints:

1 <= nums.length <= 100

nums[i].length == 2

1 <= start

i

<= end

i

<= 100

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int numberOfPoints(List<List<Integer>> nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int NumberOfPoints(IList<IList<int>> nums) {

}
}
```

C:

```
int numberOfPoints(int** nums, int numsSize, int* numsColSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun numberOfPoints(nums: List<List<Int>>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

```
object Solution {
def numberOfPoints(nums: List[List[Int]]): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (number-of-points nums)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Points That Intersect With Cars
* Difficulty: Easy
* Tags: array, hash
*
* 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:
int numberOfPoints(vector<vector<int>>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Points That Intersect With Cars
* Difficulty: Easy
* Tags: array, hash
*
* 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 int numberOfPoints(List<List<Integer>> nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Points That Intersect With Cars
Difficulty: Easy
Tags: array, hash

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

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Points That Intersect With Cars
* Difficulty: Easy
* Tags: array, hash
*
* 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
* @return {number}
*/
var numberOfPoints = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Points That Intersect With Cars
* Difficulty: Easy
* Tags: array, hash
*
* 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 numberOfPoints(nums: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Points That Intersect With Cars
* Difficulty: Easy
* Tags: array, hash
*
* 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 int NumberOfPoints(IList<IList<int>> nums) {

}
}
```

C Solution:

```
/*
* Problem: Points That Intersect With Cars
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int numberOfPoints(int** nums, int numsSize, int* numsColSize) {

}
```

Go Solution:

```
// Problem: Points That Intersect With Cars
// Difficulty: Easy
// Tags: array, hash
//
// 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 numberOfPoints(nums [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numberOfPoints(nums: List<List<Int>>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Points That Intersect With Cars
// Difficulty: Easy
// Tags: array, hash
//
// 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 number_of_points(nums: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

```
object Solution {
def numberOfPoints(nums: List[List[Int]]): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (number-of-points nums)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```