

Problem 853: Car Fleet

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

There are

n

cars at given miles away from the starting mile 0, traveling to reach the mile

target

.

You are given two integer arrays

position

and

speed

, both of length

n

, where

position[i]

is the starting mile of the

i

th

car and

speed[i]

is the speed of the

i

th

car in miles per hour.

A car cannot pass another car, but it can catch up and then travel next to it at the speed of the slower car.

A

car fleet

is a single car or a group of cars driving next to each other. The speed of the car fleet is the

minimum

speed of any car in the fleet.

If a car catches up to a car fleet at the mile

target

, it will still be considered as part of the car fleet.

Return the number of car fleets that will arrive at the destination.

Example 1:

Input:

target = 12, position = [10,8,0,5,3], speed = [2,4,1,1,3]

Output:

3

Explanation:

The cars starting at 10 (speed 2) and 8 (speed 4) become a fleet, meeting each other at 12.
The fleet forms at

target

.

The car starting at 0 (speed 1) does not catch up to any other car, so it is a fleet by itself.

The cars starting at 5 (speed 1) and 3 (speed 3) become a fleet, meeting each other at 6. The
fleet moves at speed 1 until it reaches

target

.

Example 2:

Input:

target = 10, position = [3], speed = [3]

Output:

1

Explanation:

There is only one car, hence there is only one fleet.

Example 3:

Input:

target = 100, position = [0,2,4], speed = [4,2,1]

Output:

1

Explanation:

The cars starting at 0 (speed 4) and 2 (speed 2) become a fleet, meeting each other at 4. The car starting at 4 (speed 1) travels to 5.

Then, the fleet at 4 (speed 2) and the car at position 5 (speed 1) become one fleet, meeting each other at 6. The fleet moves at speed 1 until it reaches

target

.

Constraints:

$n == \text{position.length} == \text{speed.length}$

$1 \leq n \leq 10$

5

$0 < \text{target} \leq 10$

6

$0 \leq \text{position}[i] < \text{target}$

All the values of

position

are

unique

.

$0 < \text{speed}[i] \leq 10$

6

Code Snippets

C++:

```
class Solution {  
public:  
    int carFleet(int target, vector<int>& position, vector<int>& speed) {  
  
    }  
};
```

Java:

```
class Solution {  
    public int carFleet(int target, int[] position, int[] speed) {  
  
    }  
}
```

Python3:

```
class Solution:  
    def carFleet(self, target: int, position: List[int], speed: List[int]) ->  
        int:
```

Python:

```

class Solution(object):
    def carFleet(self, target, position, speed):
        """
        :type target: int
        :type position: List[int]
        :type speed: List[int]
        :rtype: int
        """

```

JavaScript:

```

/**
 * @param {number} target
 * @param {number[]} position
 * @param {number[]} speed
 * @return {number}
 */
var carFleet = function(target, position, speed) {

};

```

TypeScript:

```

function carFleet(target: number, position: number[], speed: number[]):
number {

};

```

C#:

```

public class Solution {
    public int CarFleet(int target, int[] position, int[] speed) {

    }
}

```

C:

```

int carFleet(int target, int* position, int positionSize, int* speed, int
speedSize) {

}

```

Go:

```
func carFleet(target int, position []int, speed []int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun carFleet(target: Int, position: IntArray, speed: IntArray): Int {  
  
    }  
}
```

Swift:

```
class Solution {  
    func carFleet(_ target: Int, _ position: [Int], _ speed: [Int]) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn car_fleet(target: i32, position: Vec<i32>, speed: Vec<i32>) -> i32 {  
  
    }  
}
```

Ruby:

```
# @param {Integer} target  
# @param {Integer[]} position  
# @param {Integer[]} speed  
# @return {Integer}  
def car_fleet(target, position, speed)  
  
end
```

PHP:

```

class Solution {

    /**
     * @param Integer $target
     * @param Integer[] $position
     * @param Integer[] $speed
     * @return Integer
     */
    function carFleet($target, $position, $speed) {

    }

}

```

Dart:

```

class Solution {
  int carFleet(int target, List<int> position, List<int> speed) {

  }

}

```

Scala:

```

object Solution {
  def carFleet(target: Int, position: Array[Int], speed: Array[Int]): Int = {

  }

}

```

Elixir:

```

defmodule Solution do
  @spec car_fleet(target :: integer, position :: [integer], speed :: [integer])
    :: integer
  def car_fleet(target, position, speed) do

  end

end

```

Erlang:

```

-spec car_fleet(Target :: integer(), Position :: [integer()], Speed ::
[integer()]) -> integer().

```

```
car_fleet(Target, Position, Speed) ->  
.
```

Racket:

```
(define/contract (car-fleet target position speed)  
  (-> exact-integer? (listof exact-integer?) (listof exact-integer?)  
        exact-integer?)  
  )
```

Solutions

C++ Solution:

```
/*  
 * Problem: Car Fleet  
 * Difficulty: Medium  
 * Tags: array, sort, 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:  
    int carFleet(int target, vector<int>& position, vector<int>& speed) {  
  
    }  
};
```

Java Solution:

```
/**  
 * Problem: Car Fleet  
 * Difficulty: Medium  
 * Tags: array, sort, 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 int carFleet(int target, int[] position, int[] speed) {

}
}

```

Python3 Solution:

```

"""
Problem: Car Fleet
Difficulty: Medium
Tags: array, sort, 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 carFleet(self, target: int, position: List[int], speed: List[int]) ->
int:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

class Solution(object):
def carFleet(self, target, position, speed):
"""
:type target: int
:type position: List[int]
:type speed: List[int]
:rtype: int
"""

```

JavaScript Solution:

```

/**
 * Problem: Car Fleet
 * Difficulty: Medium
 * Tags: array, sort, 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} target
 * @param {number[]} position
 * @param {number[]} speed
 * @return {number}
 */
var carFleet = function(target, position, speed) {

};

```

TypeScript Solution:

```

/**
 * Problem: Car Fleet
 * Difficulty: Medium
 * Tags: array, sort, 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 carFleet(target: number, position: number[], speed: number[]):
number {

};

```

C# Solution:

```

/*
 * Problem: Car Fleet
 * Difficulty: Medium

```

```

* Tags: array, sort, 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 int CarFleet(int target, int[] position, int[] speed) {

}
}

```

C Solution:

```

/*
* Problem: Car Fleet
* Difficulty: Medium
* Tags: array, sort, 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
*/

int carFleet(int target, int* position, int positionSize, int* speed, int
speedSize) {

}

```

Go Solution:

```

// Problem: Car Fleet
// Difficulty: Medium
// Tags: array, sort, 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 carFleet(target int, position []int, speed []int) int {

```

```
}
```

Kotlin Solution:

```
class Solution {  
    fun carFleet(target: Int, position: IntArray, speed: IntArray): Int {  
  
    }  
}
```

Swift Solution:

```
class Solution {  
    func carFleet(_ target: Int, _ position: [Int], _ speed: [Int]) -> Int {  
  
    }  
}
```

Rust Solution:

```
// Problem: Car Fleet  
// Difficulty: Medium  
// Tags: array, sort, 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 car_fleet(target: i32, position: Vec<i32>, speed: Vec<i32>) -> i32 {  
  
    }  
}
```

Ruby Solution:

```
# @param {Integer} target  
# @param {Integer[]} position  
# @param {Integer[]} speed  
# @return {Integer}
```

```
def car_fleet(target, position, speed)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer $target
     * @param Integer[] $position
     * @param Integer[] $speed
     * @return Integer
     */
    function carFleet($target, $position, $speed) {

    }

}
```

Dart Solution:

```
class Solution {
  int carFleet(int target, List<int> position, List<int> speed) {

  }
}
```

Scala Solution:

```
object Solution {
  def carFleet(target: Int, position: Array[Int], speed: Array[Int]): Int = {

  }
}
```

Elixir Solution:

```
defmodule Solution do
  @spec car_fleet(target :: integer, position :: [integer], speed :: [integer])
    :: integer
  def car_fleet(target, position, speed) do
```

```
end  
end
```

Erlang Solution:

```
-spec car_fleet(Target :: integer(), Position :: [integer()], Speed ::  
[integer()]) -> integer().  
car_fleet(Target, Position, Speed) ->  
.
```

Racket Solution:

```
(define/contract (car-fleet target position speed)  
  (-> exact-integer? (listof exact-integer?) (listof exact-integer?)  
        exact-integer?)  
  )
```