Description | Editorial | Solutions (850) | Submissions

853. Car Fleet

Medium 2.8K 661

Companies

There are `n` cars going to the same destination along a one-lane road. The destination is `target` miles away.

You are given two integer array `position` and `speed`, both of length `n`, where `position[i]` is the position of the i^{th} car and `speed[i]` is the speed of the i^{th} car (in miles per hour).

A car can never pass another car ahead of it, but it can catch up to it and drive bumper to bumper **at the same speed**. The faster car will **slow down** to match the slower car's speed. The distance between these two cars is ignored (i.e., they are assumed to have the same position).

A **car fleet** is some non-empty set of cars driving at the same position and same speed. Note that a single car is also a car fleet.

If a car catches up to a car fleet right at the destination point, it will still be considered as one 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.
```

Java | Auto

```java
class Solution {
    public int carFleet(int target, int[] position, int[] speed) {
        // Step 1
        float[] moves = new float[position.length];

        for(int i=0; i<position.length;i++){
            moves[i] = (((float)target - position[i]) / speed[i] );
        }

        // Step 2
        int fleet = 0;
        int counter =0;
        int maxPosition = -1, maxIndex = -1;
        float maxPositionMoves;

        while(counter < position.length){

            // Step 2a
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

target =
12

position =
[10,8,0,5,3]

Console | Run | Submit

Description | Editorial | Solutions (4.7K) | Submissions

852. Peak Index in a Mountain Array

Medium 6K 1.8K

Companies

An array `arr` a **mountain** if the following properties hold:

- `arr.length >= 3`
- There exists some `i` with `0 < i < arr.length - 1` such that:
  - `arr[0] < arr[1] < ... < arr[i - 1] < arr[i]`
  - `arr[i] > arr[i + 1] > ... > arr[arr.length - 1]`

Given a mountain array `arr`, return the index `i` such that `arr[0] < arr[1] < ... < arr[i - 1] < arr[i] > arr[i + 1] > ... > arr[arr.length - 1]`.

You must solve it in `O(log(arr.length))` time complexity.

Example 1:

```
Input: arr = [0,1,0]
Output: 1
```

Example 2:

```
Input: arr = [0,2,1,0]
Output: 1
```

Example 3:

Java | Auto

```java
class Solution {
    public int peakIndexInMountainArray(int[] arr) {
        int i = 0;
        int j = arr.length - 1;
        int n = arr.length;
        while (i <= j) {
            int mid = (i + j) / 2;
            if ((mid == 0 || arr[mid - 1] < arr[mid]) && (mid == n - 1 || arr[mid + 1] < arr[mid]))
                return mid;
            else if (mid > 0 && arr[mid - 1] > arr[mid])
                j = mid - 1;
            else
                i = mid + 1;
        }
        return -1;
    }
}
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

arr =
[0,1,0]

Output

1

Console | Run | Submit