853. Car Fleet

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<sup>th</sup>` car and `speed[i]` is the speed of the `i<sup>th</sup>` 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 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 and 8 become a fleet, meeting each other at 12.
The car starting at 0 doesn't catch up to any other car, so it is a fleet by itself.
The cars starting at 5 and 3 become a fleet, meeting each other at 6.
Note that no other cars meet these fleets before the destination, so the answer is 3.
```

Example 2:

```
Input: target = 10, position = [3], speed = [3]
Output: 1
```

Constraints:

- `n == position.length == speed.length`

- `1 <= n <= 10<sup>5</sup>`
- `0 < target <= 10<sup>6</sup>`
- `0 <= position[i] < target`
- **All the values of `position` are unique.**
- `0 < speed[i] <= 10<sup>6</sup>`

```
class Solution:
    def carFleet(self, target: int, position: List[int], speed: List[int]) -
> int:
        ans = []
        for i in range(len(position)):
            ans.append((position[i],speed[i]))

        ans = sorted(ans,key=lambda x:(x[0],x[1]), reverse = True)
        stack = []

        for i in range(len(ans)):
            time = (target-ans[i][0])/ans[i][1]

            if not stack:
                stack.append(time)
            elif time>stack[-1]:
                stack.append(time)
        return len(stack)
```

The pattern:

- Many greedy problems require sorting and processing things in order while checking if the current item overlaps/dissolves into its predecessor (the previous item).
- This is usually needed to determine the most optimimum (max/min) number of something (for example a resource, as in how many rooms needed given this timeline of meetings). Usually the problem gives us criteria upon which we can group some of these items. The grouping/clustering nature is what allows us to obtain the optimum result (for ex: the minimum number of rooms needed, or the the number of fleets)
- This problem fits the pattern of what I like to call ***"allocating resources to overlapping events"***
- In this kind of problems, its usually the case that you have to sort the items with respect to some feature and process them one at at time while constantly checking the previous items by popping from a stack.
- Problems that follow a somewhat simialr pattern:

56. Merge Intervals
57. Non-overlapping Intervals

58. Minimum Number of Arrows to Burst Balloons

59. Meeting Rooms

60. Meeting Rooms II

61. My Calendar I

62. My Calendar II

63. My Calendar III (Meeting rooms II)

64. Course Schedule III

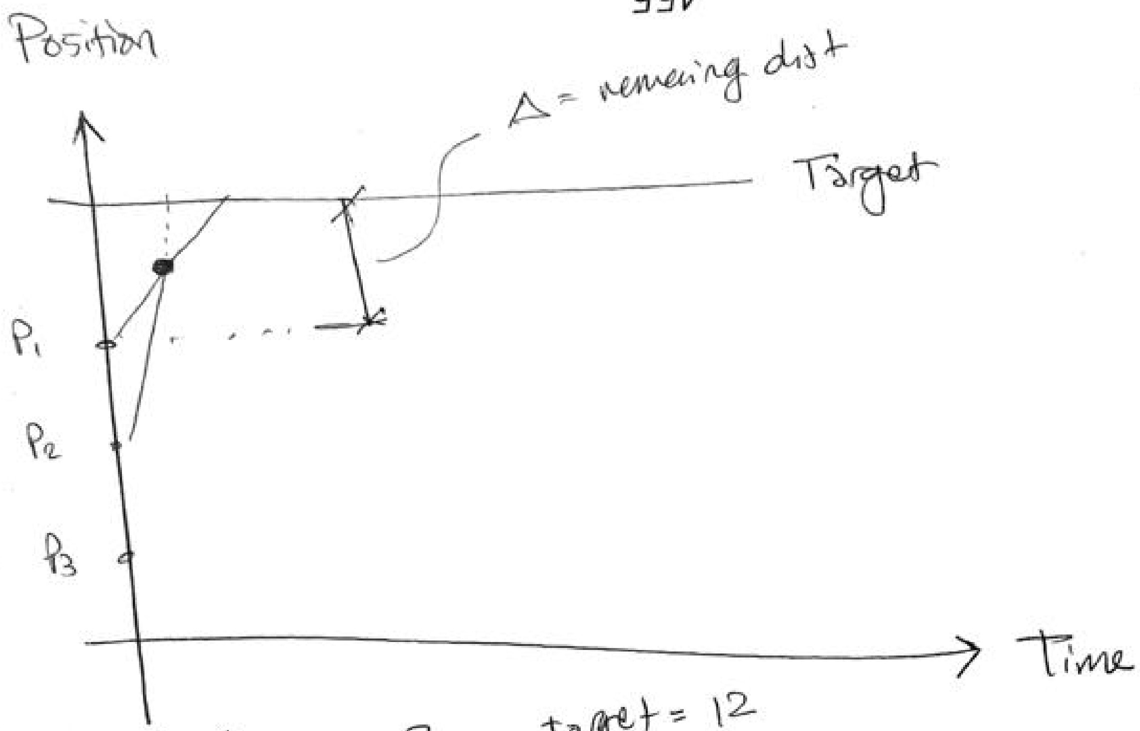
65. Car Pooling (Meeting Rooms II with a twist)

.

.

Sketches:

455



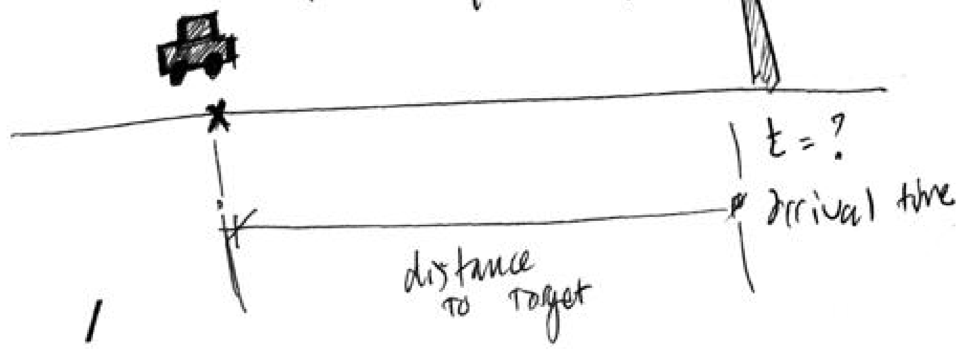
target = 12

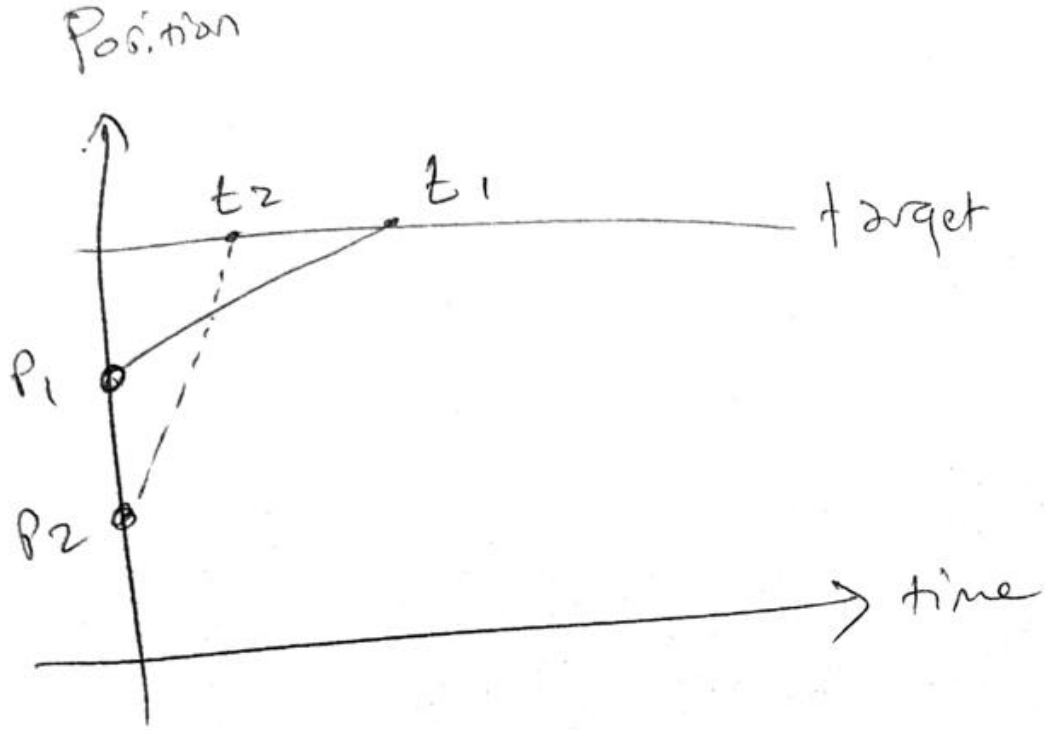
x = [10, 8, 0, 5, 3]

v = [2, 4, 1, 1, 3]

sorted = [(10,2), (8,4), (5,1), (3,3), (0,1)]

• sorted =	(10,2)	(8,4)	(5,1)	(3,3)	(0,1)
• distance to target =	12 − 10 = 2	12 − 8 = 4	12 − 5 = 7	12 − 3 = 9	12 − 0 = 12
• arrival Time =	2/2 = 1	4/4 = 1	7/1 = 7	9/3 = 3	12/1 = 12





t_1 : Arrival time of car1
t_2 : Arrival time of car2

if $t_2 <= t_1$
then car2 joins the same fleet as car 1

else :
car 2 forms its own fleet
(yes, fleet can be of size 1)