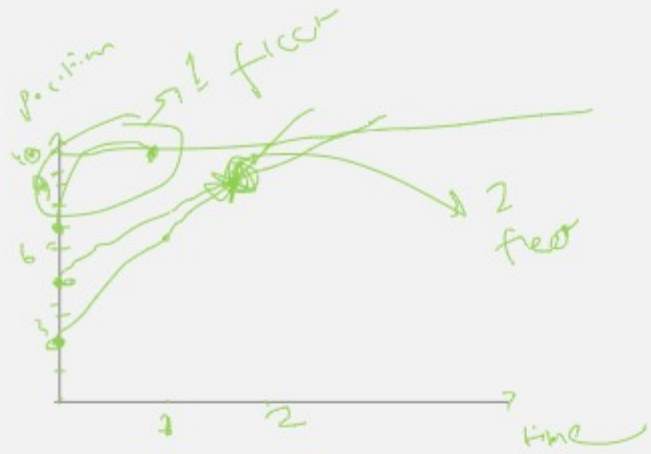
(3,3) (5,1) (7,3)
target = 10

853. Car Fleet

In this problem, it is given that there is a one lane road, where n cars (pos and speed given) are going with different speeds of their own. The road has an endpoint t where the cars will stop. Now it is needed to find the number of car fleets the cars will form on reaching the target t.



Now what is a car fleet. Suppose car A is 2 unit behind car B, and it's speed is faster than car B. So, if the road was infinite they would have must collided and the cars would go with the speed of the slower car B. But since there is a target t, that is end of road, we have to find out if they collide before the target, if they do collide, we remove the concern of the car behind, as now both of them are going with the speed of the car that was ahead.

So now number of car fleet is required.

If T(x-1) < Tx, that is, the time required for C(x-1) to reach target is less than time required for Cx to reach target, then they are meant to collide. So, I form them 1 fleet, and after forming one fleet, they move with speed Sx after Tx amount of time.

Tx, Time needed for Cx to reach target is (target-Cx)/Sx

Again I compare T(x-2) with Tx, now if they form fleet, I go and do the same, else, I start comparing the rest of the array with T(x-2) and this algorithm goes on.

The reason why I started from right is because, the speeds are determined by the slower ones, so if I find a very slow vehicle at start of the cars, then that is going to determine my number of car fleets.

The Algorithm:

1. Make vector<pair of car pos and speed> (Data structure for keeping the car pos and speed connected and changing the order they occur in the problem)
2. Sort vector<pair of car pos and speed> (To order them for comparing)
3. Init empty stack. (For keeping record of limited cars, or fleet leaders/heads)
4. Set curr_index to last index of vector. (It goes as a fleet either way, as head of everyone)
5. Push curr_index of vector to stack. (last fleet head)
6. Do this while we don't reach the beginning of the stack. curr_index > 0 (compare loop for finding fleet heads)
    a. curr_index-- (compare former with current fleet head i.e. stack top)
    b. Compare time needed for vector[curr_index] to reach target with time needed by stack top. (If less time needed by former position then they meet and form fleet, else we got another fleet head, who makes for a fleet).
        i. If T_vec_curr_index <= T_stack_top (they become a fleet)
        ii. Else push vector_curr_index. to stack. (fleet head found)
7. Return the stack size. (Number of fleet heads i.e. Fleets)

Code:

```
class Solution {
public:
   double t_reach_target(pair<int,int>& car, int target){
      cout << (target-car.first)/(double)car.second << endl;
      return (target-car.first)/(double)car.second;
   }
   int carFleet(int target, vector<int>& position, vector<int>& speed) {
      vector<pair<int,int>> pos_speed_vec;
      for(int i=0;i<position.size();i++){
         pos_speed_vec.push_back(make_pair(position[i],speed[i]));
      }
      sort(pos_speed_vec.begin(),pos_speed_vec.end());

      stack<pair<int,int>> st_fleet_head;
      int last_index = position.size()-1;
      int curr_index = last_index;
      st_fleet_head.push(pos_speed_vec[last_index]);

      while(curr_index > 0) {
         curr_index--;
         if(t_reach_target(pos_speed_vec[curr_index],target) <=
t_reach_target(st_fleet_head.top(),target)){
            //they became one fleet.
         }
         else{
            st_fleet_head.push(pos_speed_vec[curr_index]);
         }
      }
      return st_fleet_head.size();
   }
};
```