Max Performance of Team.

1st example

(n=6)

speed	2	10	3	1	5	8
efficiency	5	4	3	9	7	2

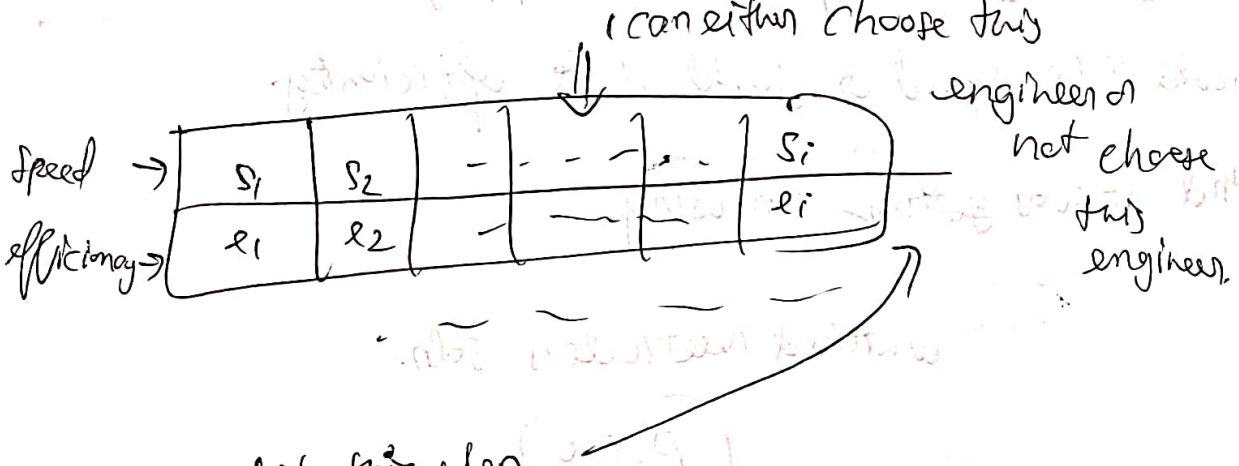
Our goal is to select atmost k ~~degrees~~ engineers such that performance is maximized

$$\text{performance} = \frac{(\text{Out of the chosen engineers})}{(\text{min Performance}) * [\text{Sum of Speed of all Engineers}]}$$

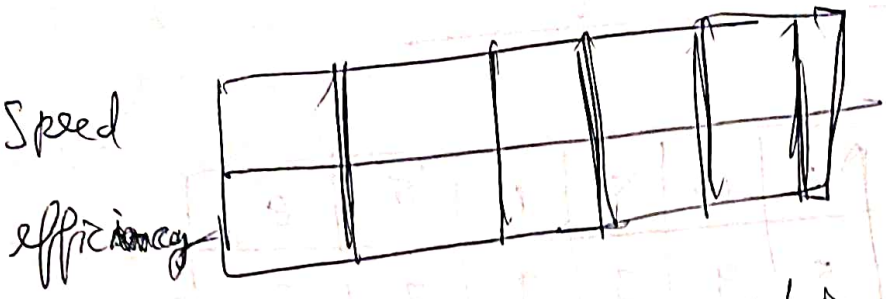
What my Brain thought

(1st thought)

My mind went in to thinking of whether to choose or not to choose an engineer.



Like this when I reached the end, I can calculate the performance of the team I formed.



But for this in dp I have to maintain

a state as → (startPosition, No of Engineers Picked)

n

But in constraints I have,

$1 \leq k \leq n \leq 10^5$

So dp cannot be formed.

(2nd thought)

Next I thought about priority queue, somehow priority queue or some sorting needs to be done to solve the problem.

Now I was thinking what to apply the condition on?

Should it be speed or should it be efficiency.

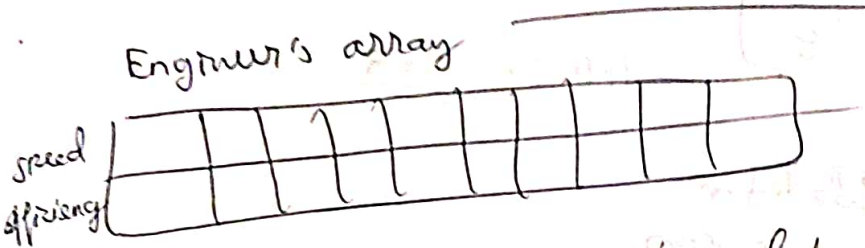
And it was getting confusing.

Watched Neetcode's soln.

(P.T.O)

To understand intuition behind it.

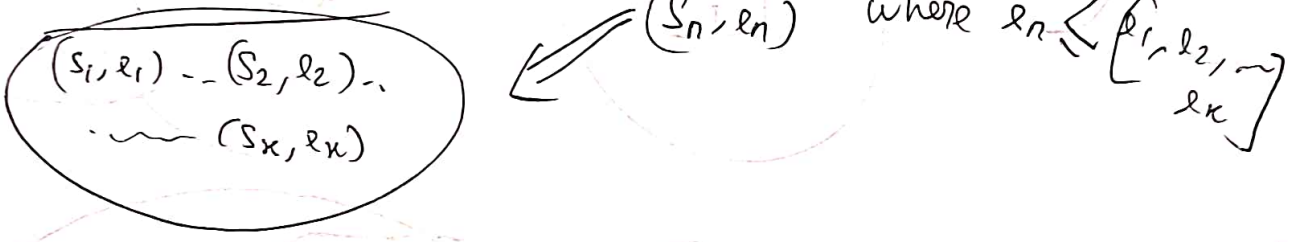
(NeetCode's Soln & Intuition)



Let's say this is the Engineer's array,

Now lets say you already have a team with some k members.

Now you add a new member whose efficiency is lesser than all the k members of the team.



Note: If am adding this guy to the team, will the efficiencies of the previous engineers matter at all?

No. Because we want the minimum efficiency, after sort speed matters.

So new Speed = $(s_1 + s_2 + \ldots + s_k + s_n) *$ multiplied by e_n

Building upon this intuition, we sort the engineers array by efficiencies only.

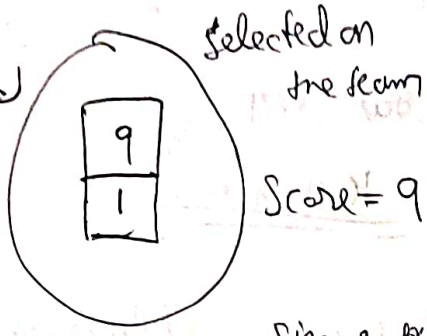
(P.T.O)

i=0 Sort

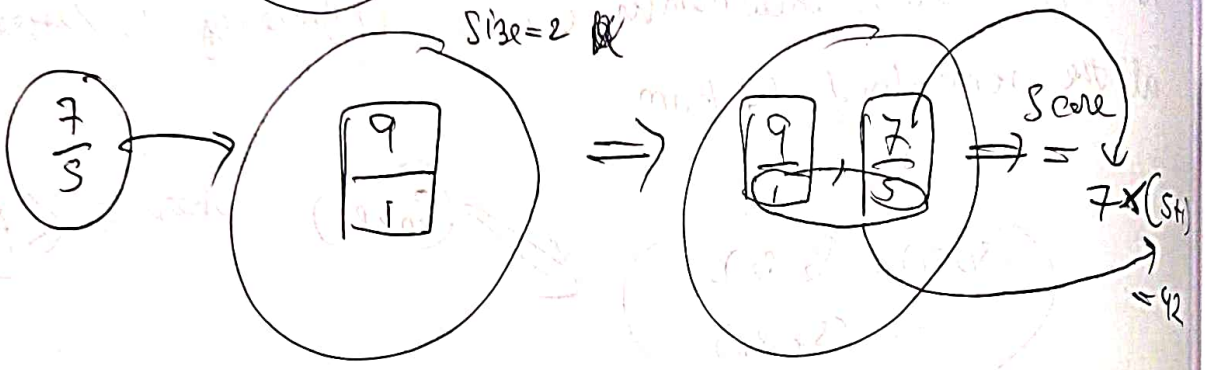
Eff	9	7	5	4	3	2
Speed	1	5	2	10	3	8

Let's say K=3

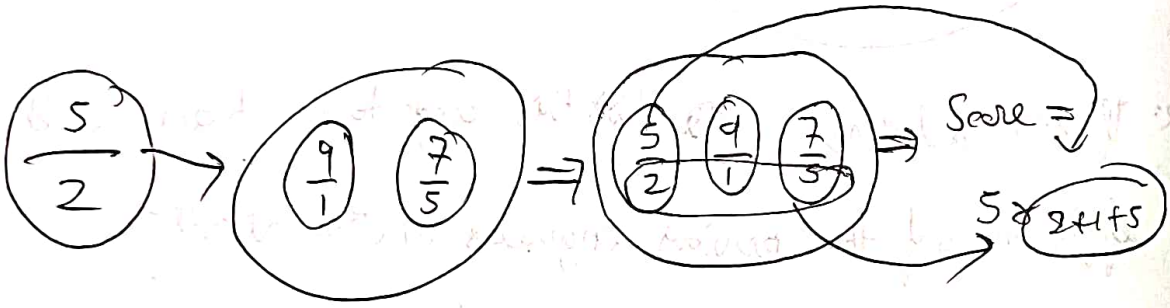
Simulating the process, i=0



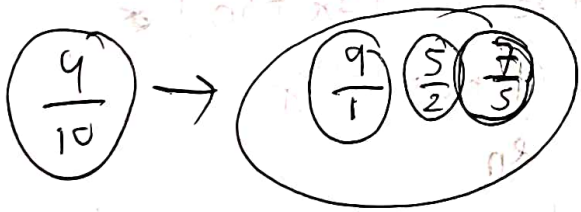
i=1



i=2



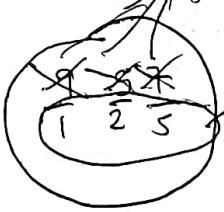
i=3 this is where the whole game happens.



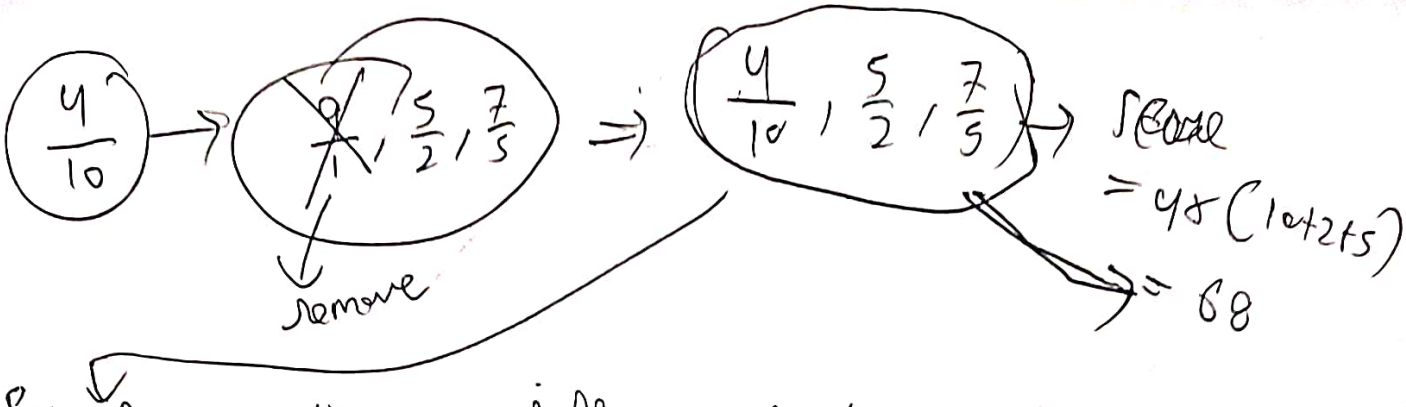
team is full and to evaluate the score using the new member, we have to remove someone

Since efficiency of the newest member is only going to be taken into consideration

So... Don't matter



matters. Remove the one with lowest speed and add member



So if you notice carefully we don't have to maintain efficiency, we just need to keep speeds and when a new member comes and we wanna to evaluate if ~~we want to~~ that new member can increase the score, then we simply pop the member with least speed.

which we can maintain using a min heap.