Scientific Computing

Mon, Jan. 26

Announcements

* Homework 1 due Friday night, 11:59pm.

Office Hours:

Mon, 9:30-10:30
Fri, 2:00-3:00

Cudahy 307

Greedy Algorithms

Vague definition: A greedy algorithm is a way of solving a problem that builds up a solution bit by bit, always picking the next bit that is the best, even that leads to a suboptimal full solution.

"neurotic"

They are:
- normally lightning fast
- much better than random solutions
- sometimes pretty bad, sometimes pretty good, sometimes provably optimal, depending on the problem.

Ex: Giving change - How does a cashier give change? Suppose you owe \$3.27 and pay with \$20. They start giving you bills and coins from largest to smallest.

You get \$16.73

"Cashier's Algorithm" - As you give change, just give denomination possible at each step.

~~\$100~~	~~\$50~~	~~\$20~~	\$10	\$5	\$1	\$0.25	\$0.10	\$0.05	\$0.01
			1	1	1	2	2	0	3
			6.73	1.73	\$0.73	\$0.23	\$0.03		\$0

$1+1+1+2+2+3 = 10$ coins/bills

Is this the combination with the <u>least</u> # of bills/coins that adds to \$16.73?

Problem #1: Interval Scheduling (Algorithm Design, by Kleinberg + Tardos)

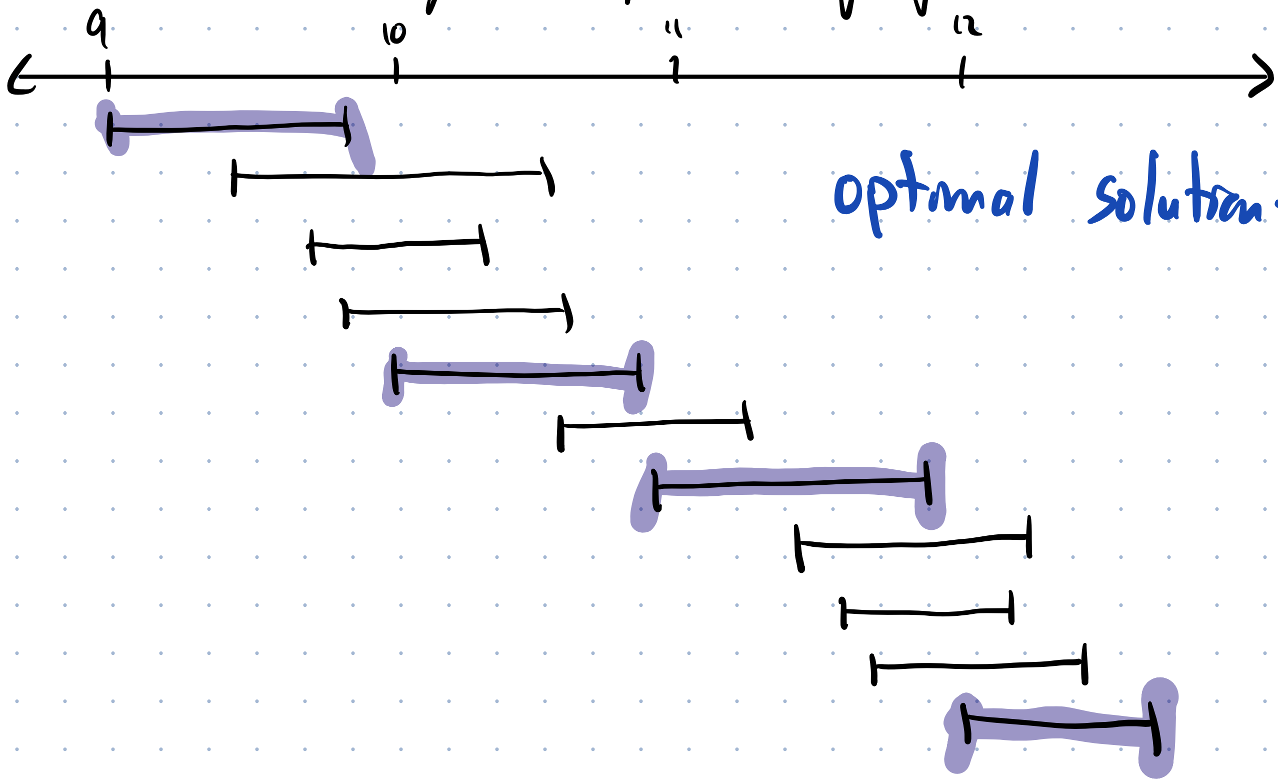
Suppose you are in charge of a conference room that a lot of people want to use to hold meetings. A bunch of people tell you the times they want to book the room for, and your goal is to accomodate as many groups as possible.

Ex:

Reservations:

9am – 9:50am
9:30am – 10:30am
9:45am – 10:15am
9:50am – 10:30am
10:00am – 10:50am
10:30am – 11:15am
11:00am – 11:50am
11:30am – 12:15pm
11:35am – 12:10pm
11:40am – 12:20pm
12:00pm – 12:30pm

What is the largest # of meetings you can book?



optimal solution: 4 meetings

Formal setup:

- n requests
- each request has a start time s_i and a finish time f_i (real numbers), with $s_i < f_i$.

Goal: find a maximal size subset of nonoverlapping requests

Two requests (s_i, f_i) and (s_j, f_j) overlap if:

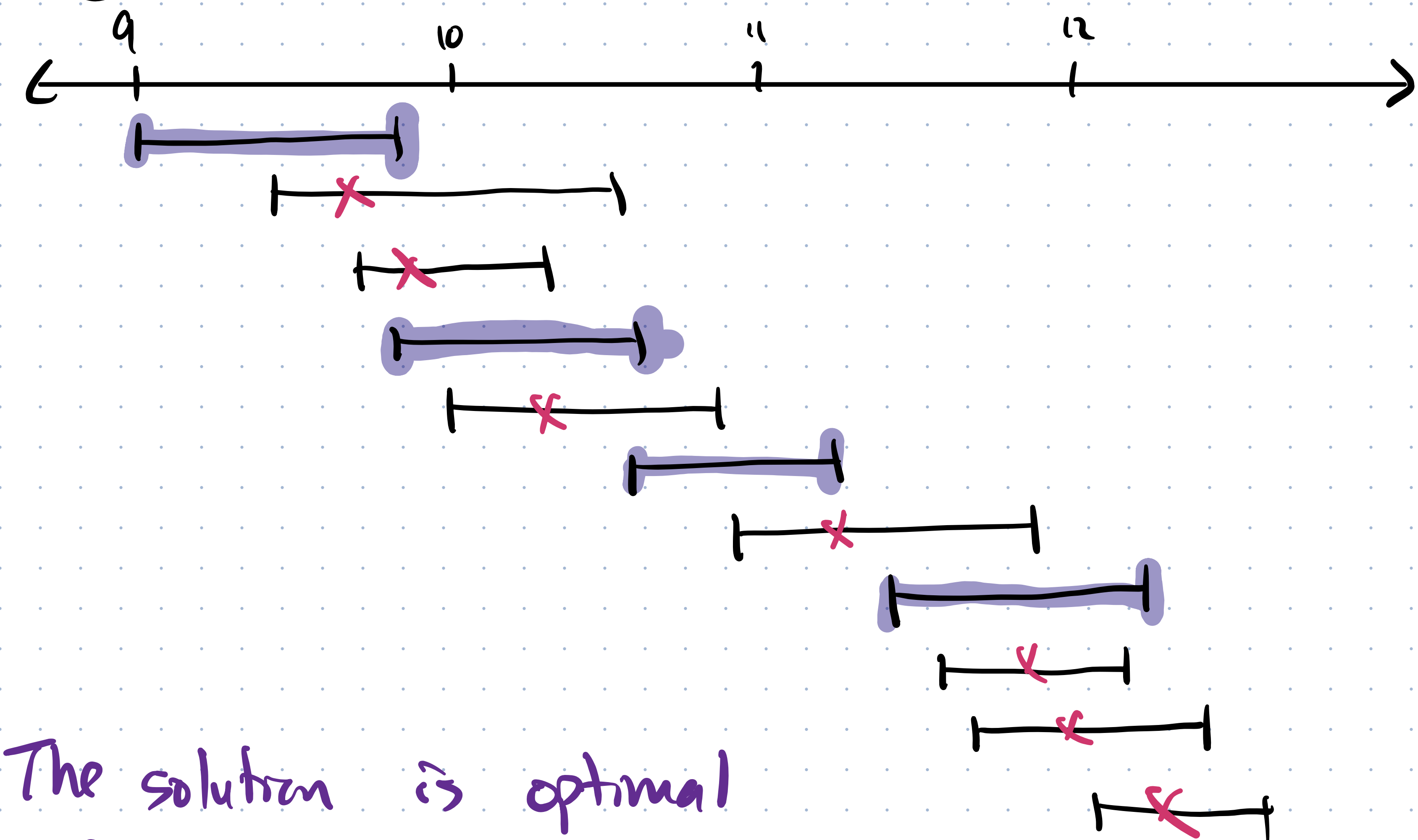
$s_j < f_i$ and $s_i < f_j$

Let's think about possible greedy approaches.

General idea: * decide on a rule for which meeting is "best"

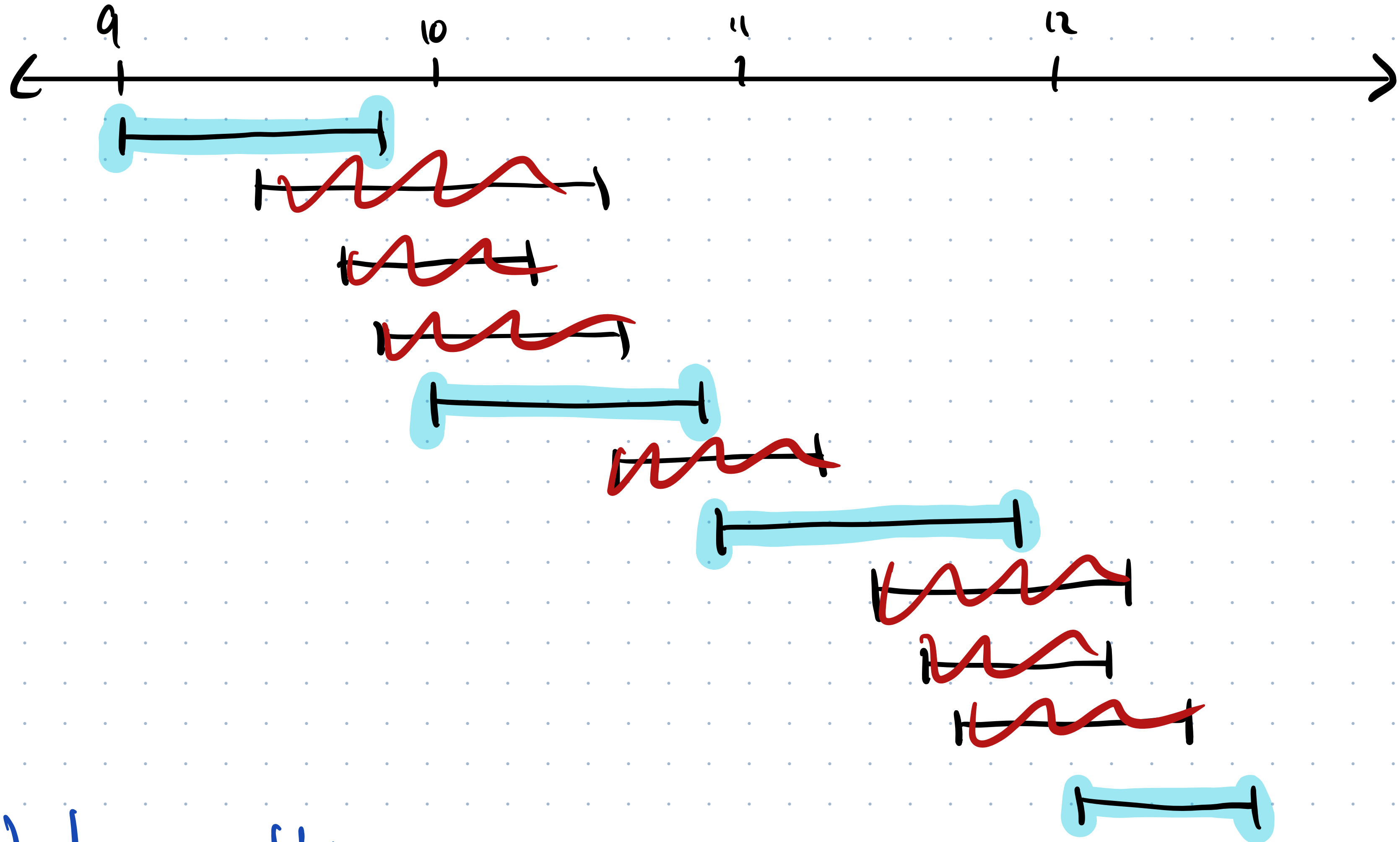
* pick it, then eliminate conflicts
* repeat

Idea #1: best = earliest start time



The solution is optimal for this example

Idea #1: best = earliest start time



Works in this case.

Can we break it?

Idea #1: best = earliest start time

Can we break it?

This greedy algorithm is not optimal.

Idea #2: best = "shortest"

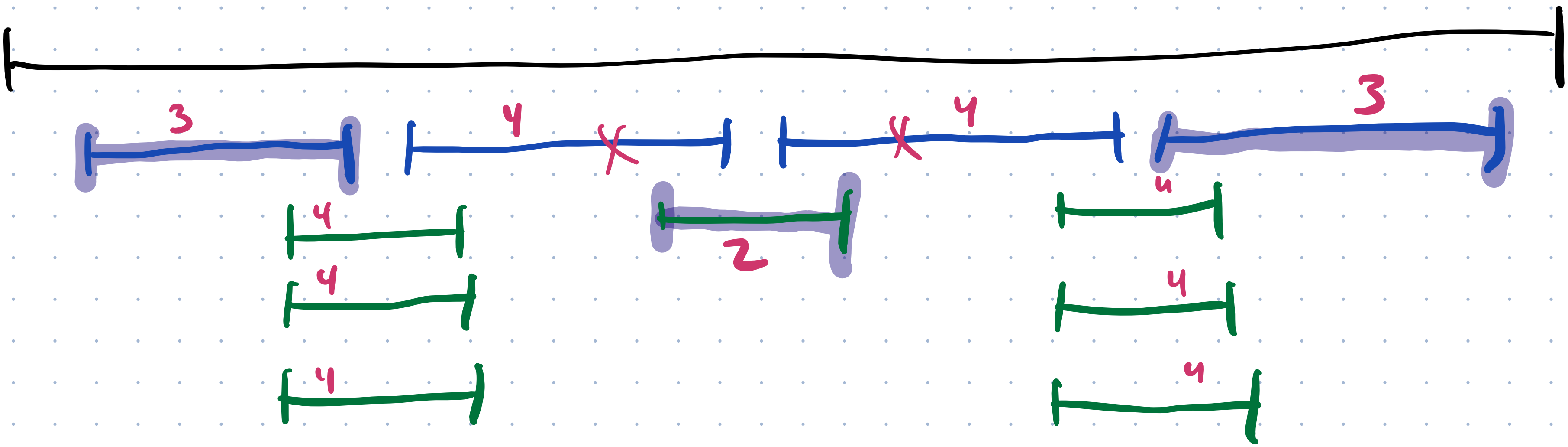
Can we break this?

Greedy solution: 1 red meeting

Optimal solution: 2 purple meetings

Idea #3: best = "least conflicts"

Can we break it?



Greedy: 3 meetings

Optimal: 4 meetings

Idea #4: best = "earliest ending time"

This works on all our previous examples.

Can we break it? No, this is an optimal greedy algorithm

Idea #4: best = "earliest ending time"

This works on all our previous examples.
Can we break it?

Intuition: Picking the one that ends earliest gets you credit for a meeting that gets out of the room as quickly as possible.

Algorithm:

Let R be the set of requests.

Let A be the empty set.

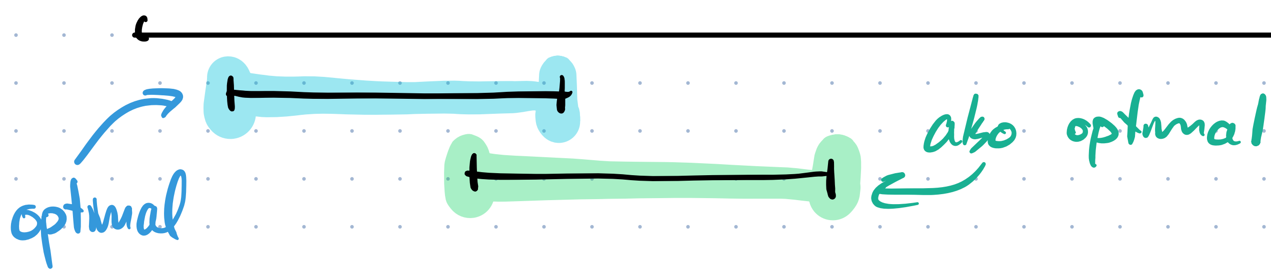
While R is non-empty:

- Find the request with earliest end time.
- Add it to A.
- Remove it from R and remove all other requests that are not compatible.

A is the solution

Theorem: The greedy algorithm produces an optimal solution.

Note: There could be other optimal solutions too.



Proof: Let R be a set of requests, and let A be the output of our greedy algorithm. Let $\mathcal{O}$ be an optimal solution. We want to show $|A| = |\mathcal{O}|$.

Since $\mathcal{O}$ is optimal, $|A| \leq |\mathcal{O}|$.

A common strategy when proving that your greedy algo. is optimal is showing that the answer it produces **stays ahead** of any optimal sol.

Suppose the requests in A are

$$A = \{(s_1, f_1), (s_2, f_2), \ldots, (s_k, f_k)\}$$

and in $\mathcal{O}$

$$\mathcal{O} = \{(s_1', f_1'), (s_2', f_2'), \ldots, (s_m', f_m')\}$$

and that we have listed them in order:

$$s_1 < f_1 \leq s_2 < f_2 \ldots$$

$$s_1' < f_1' \leq s_2' < f_2' \ldots.$$

Note that $m \geq k$ since $|\mathcal{O}| \geq |A|$.

Now we'll show that A "stays ahead" of $\mathcal{O}$:

$$f_r \leq f_r' \quad \text{for } r = 1, 2, \ldots k.$$

In English, the r^{th} task of A finishes before the r^{th} task of $\mathcal{O}$.

We'll prove this by induction.

Base case: $r=1$, $f_1 \le f_1'$.

This is true because we defined our greedy algorithm to start by picking the earliest ending time.

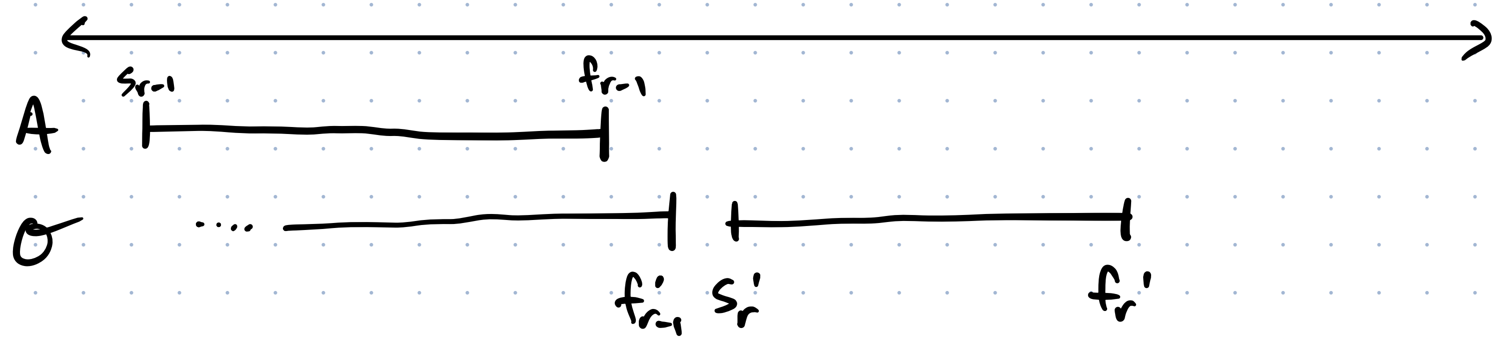
Induction step: Assume $f_i \le f_i'$ for $i = 1, 2, \dots, r-1$

We will show $f_r \le f_r'$. We know:

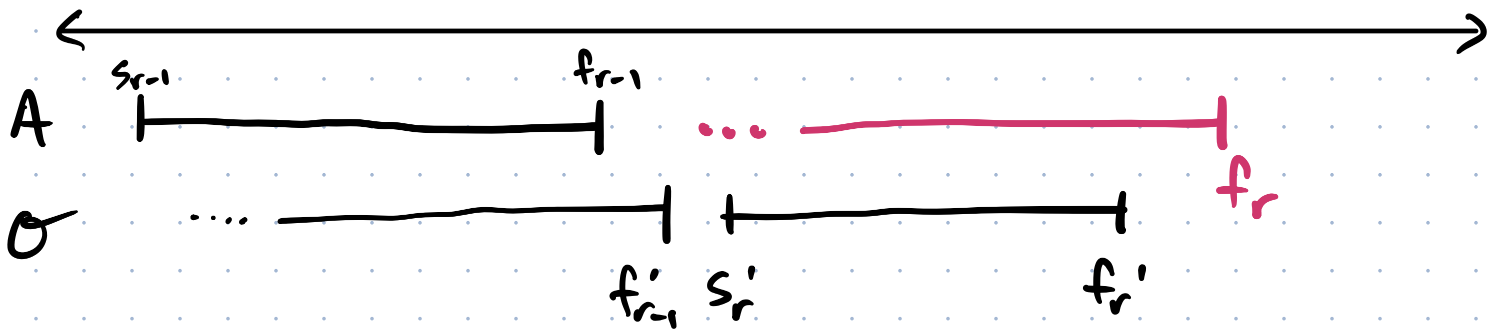
* $f_{r-1} \le f_{r-1}'$ (induction)
* $f_{r-1}' \le s_r'$ (otherwise $\mathcal{O}$ wouldn't make sense)

$\Rightarrow f_{r-1} \le s_r'$

This means:



If it's not true that $f_r \le f_r'$, then



This is impossible because our greedy algo. would never have picked (s_r, f_r) – it would have picked (s_r', f_r') instead.

Thus, $f_r \le f_r'$, completing the induction.

Not done yet!

So now we know A "stays ahead" of $\mathcal{O}$, and we know $|A| \le |\mathcal{O}|$. Last thing to show is $|A| = |\mathcal{O}|$. What would happen if it wasn't true, if $|A| < |\mathcal{O}|$?

Recall $|A| = k$.

Then, $\mathcal{O}$ has a task (s'_{k+1}, f'_{k+1}), and obviously $f'_k \le s'_{k+1}$. Since A "stays ahead", $f_k \le f'_k \le s'_{k+1}$, which means (s'_{k+1}, f'_{k+1}) doesn't conflict with the requests in A and so our greedy algorithm would have picked it. ■

* Coding the greedy algorithm!

    * Python lesson on functions and sort keys

    * Demo