Algorithms

Greedy algorithms

Recap

- HW3 - due in 1 week
  - ↳ Don't forget groups!
- No office hours today - sorry!
- Make up office hour:
  Friday (9/25) at 3pm
  (Still Tuesday & Friday as usual.)
- Reading as usual, & grades have (finally!!) been reset.
- New zoom still coming...

Dynamic Programming vs Greedy

Dyn. pro: try all possibilites
↳ but intelligently!

In greedy algorithms, we avoid building all possibilites.

How?
- Some part of the problem's structure lets us pick a local "best" and have it lead to a global best.

But - be careful!

Students often design a greedy strategy, but don't check that it yields the best global one.

Overall greedy strategy:

↳ for a proof

- Assume optimal is different than greedy
- Find the "first" place they differ.
- Argue that we can exchange the two without making optimal worse.

⟹ there is no "first place" where they must differ, so greedy in fact is an optimal solution.

Note: There may be many solutions. We're finding one.

First example in the book:

Storing files on tape.
(Seemed to make sense?)

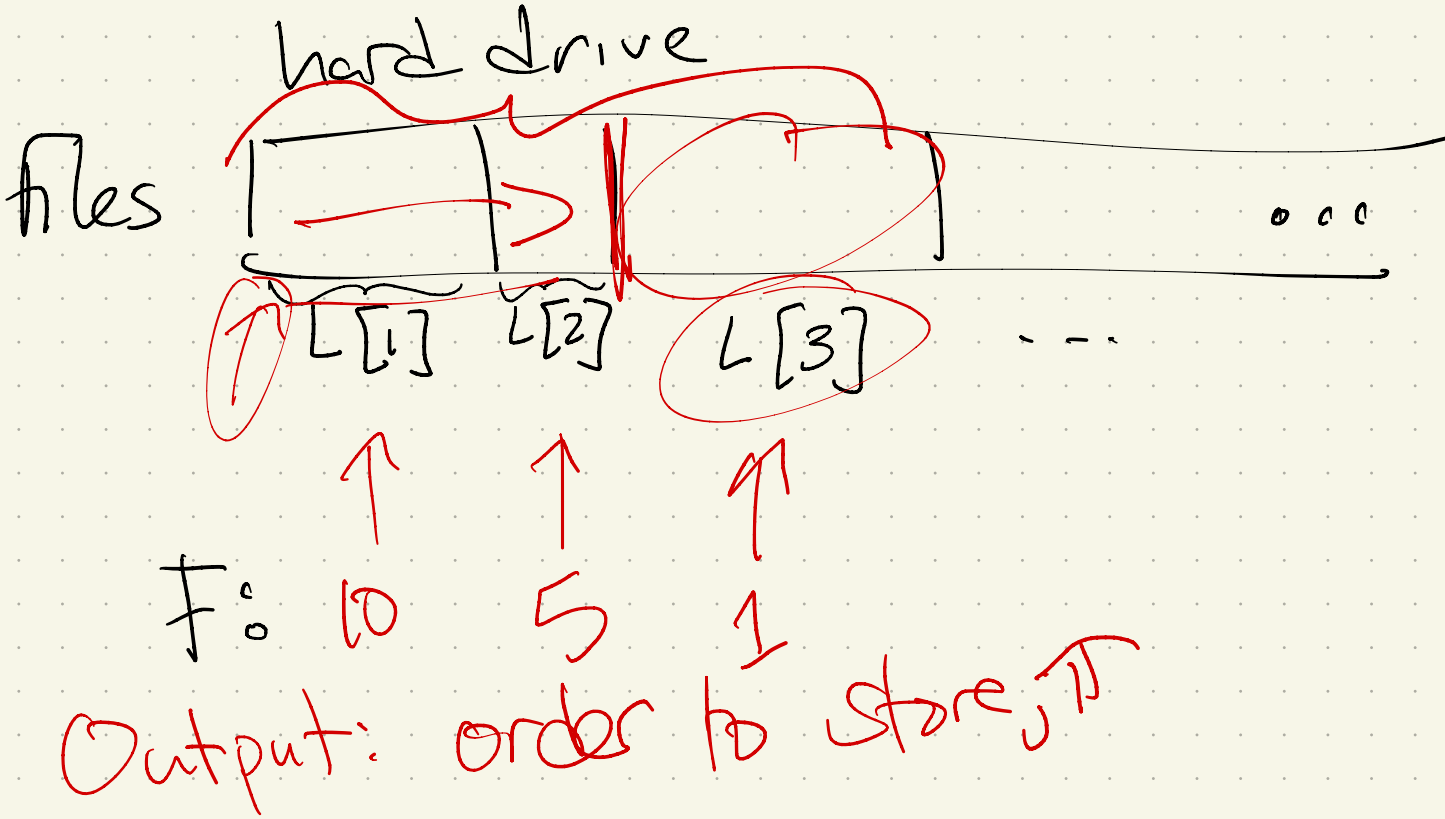
Input: n files, each with a length & # times it will be accessed:

$L[1..n]$ & $F[1..n]$

length

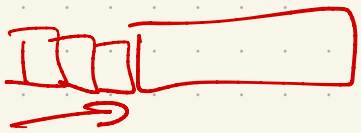
frequency

Goal: Minimize access time:



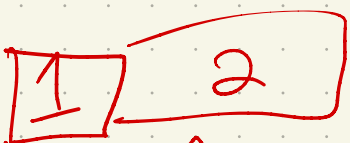
How to be greedy? (Not immediately clear!)

Try smallest first:



Correct?

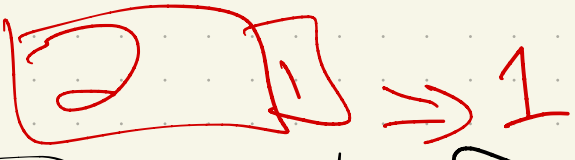
2 jobs: L = [1, 2], F = [1, 100]



ignore frequent

$1 \cdot 0 + 1 \cdot 100$

counterexample



$\Rightarrow 1$

Try most frequent first: ✗

L

F = 50, 1

highest freq

pays: 100

otherway:

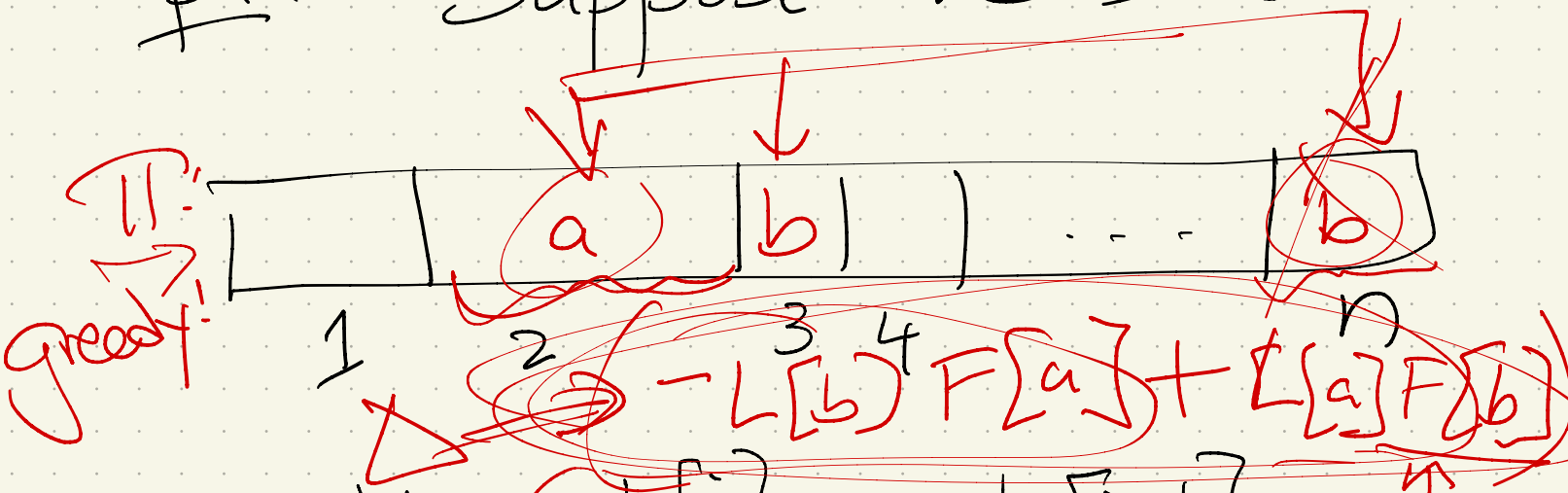
$1 \cdot 0 \cdot 50 \cdot 1$

counterexample

Lemma: Sort by $\frac{L[i]}{F[i]}$

↳ will ~~get~~ be tied with optimal.

pf: Suppose we sort:



↳ $\forall i, \frac{L[i]}{F[i]} \leq \frac{L[i+1]}{F[i+1]}$

Suppose this is not optimal.

What does that mean?

opt must be different ⇒

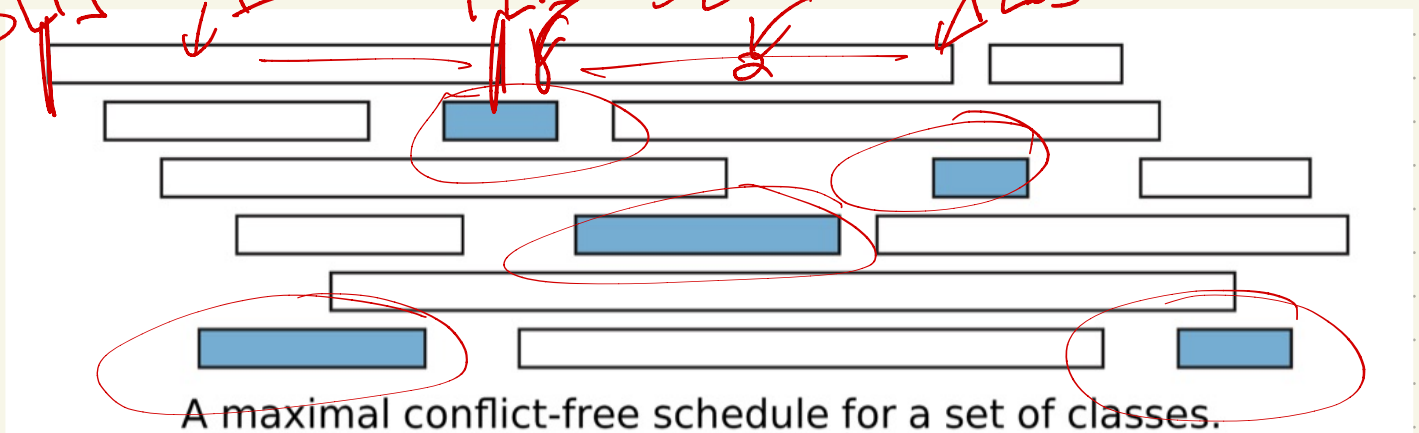
some pair must be swapped

but $\frac{L[b]}{F[b]} > \frac{L[a]}{F[a]}$

Problem: Interval Scheduling

Given a set of events (intervals with a start + end time), select as many as possible so that no 2 chosen will overlap.

"max"



A maximal conflict-free schedule for a set of classes.

note: not the greedy

More formally:

Two arrays

S[1..n]: start time

F[1..n]: end times

Goal: A subset $X \subseteq \{1..n\}$ as big as possible s.t.

$\forall i,j \in X_{s.t.}$ $F[i] \leq S[j]$ or $F[j] \leq S[i]$

How would we formalize a dynamic programming approach?

Recursive structure:

Class is either in or out.

Look at class 1, & try both!

func(S[1..n], F[1..n], X): {}

include 1 in X

→ & recurse on classes that don't overlap

not include →

func(S, F, X)

choose best & return

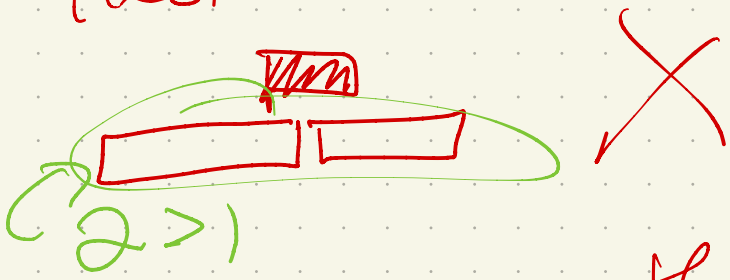
LIS style

Intuition for greedy:

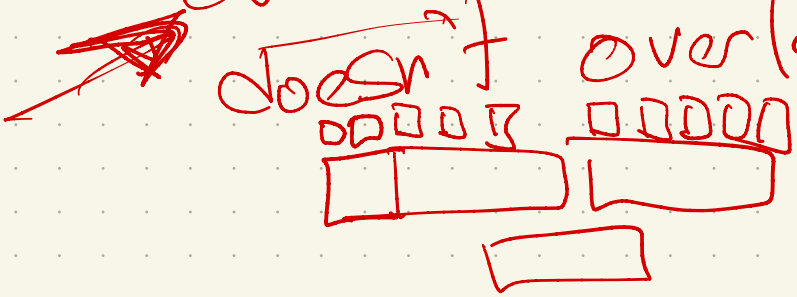
Consider what might be a good first one to choose.

Ideas? (+ break them)

- smallest class

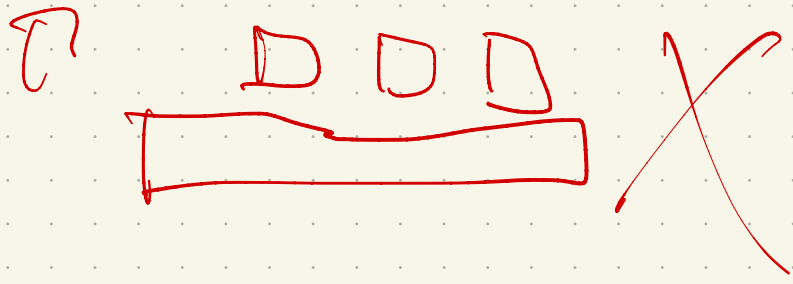


- choose one that doesn't overlap many ??



come back...

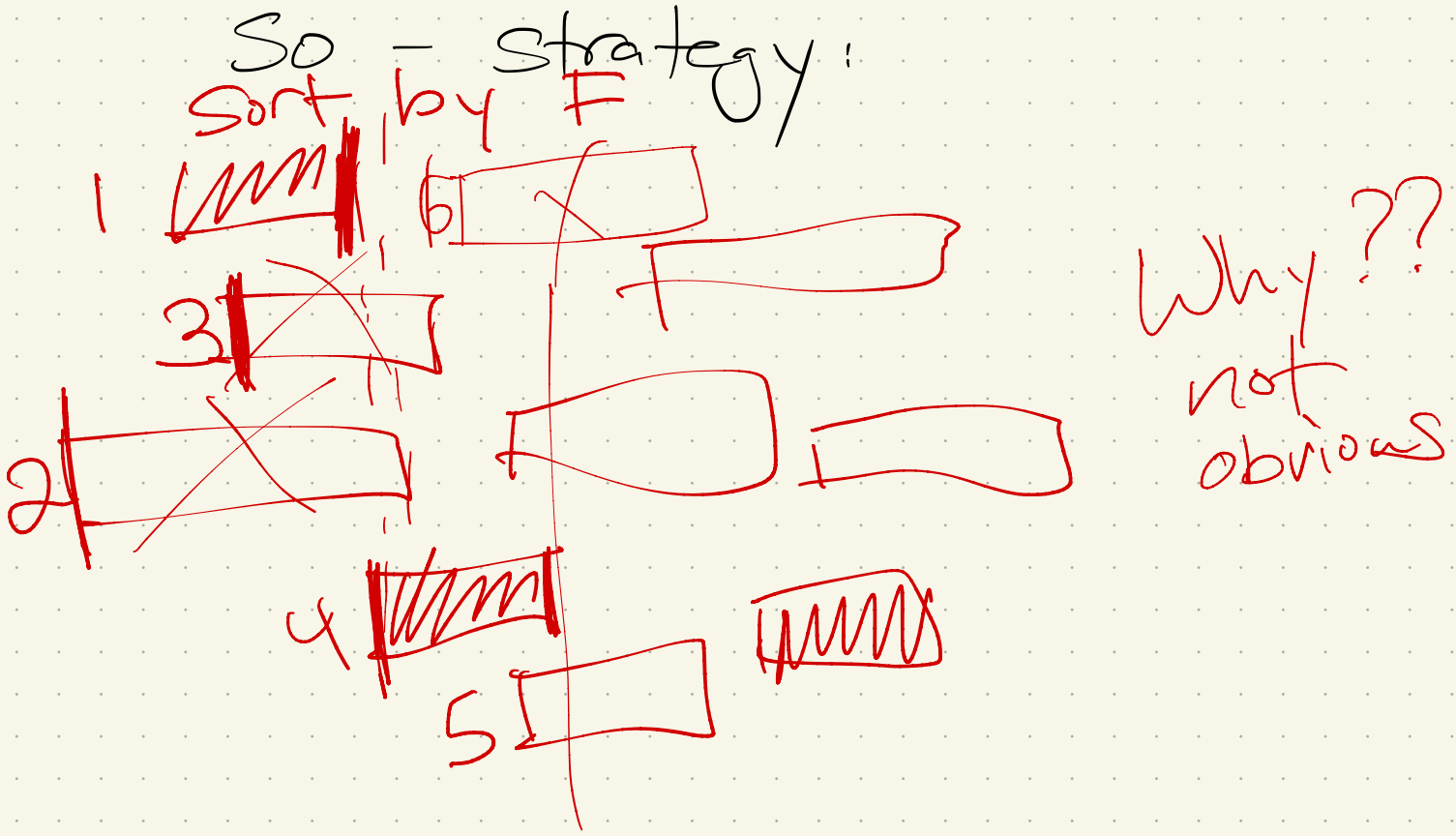
- choose earliest class



Key intuition:

If it finishes as early as possible, we can fit more things in!

So - strategy:



Why?? not obvious

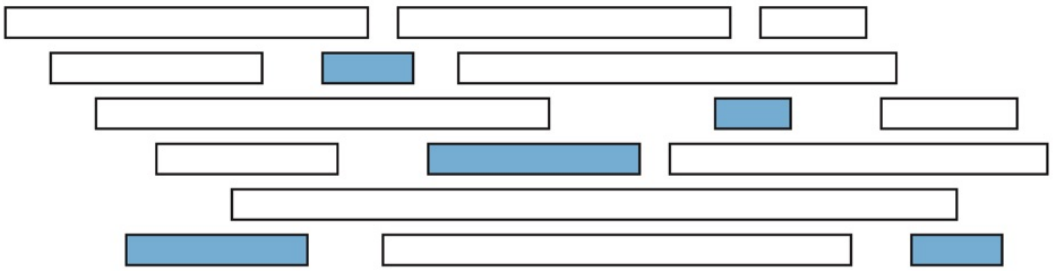
The code:

```
GreedySchedule(S[1..n], F[1..n]):
    sort F and permute S to match
    count ← 1
    X[count] ← 1
    for i ← 2 to n
        if S[i] > F[X[count]]
            count ← count + 1
            X[count] ← i
    return X[1..count]
```

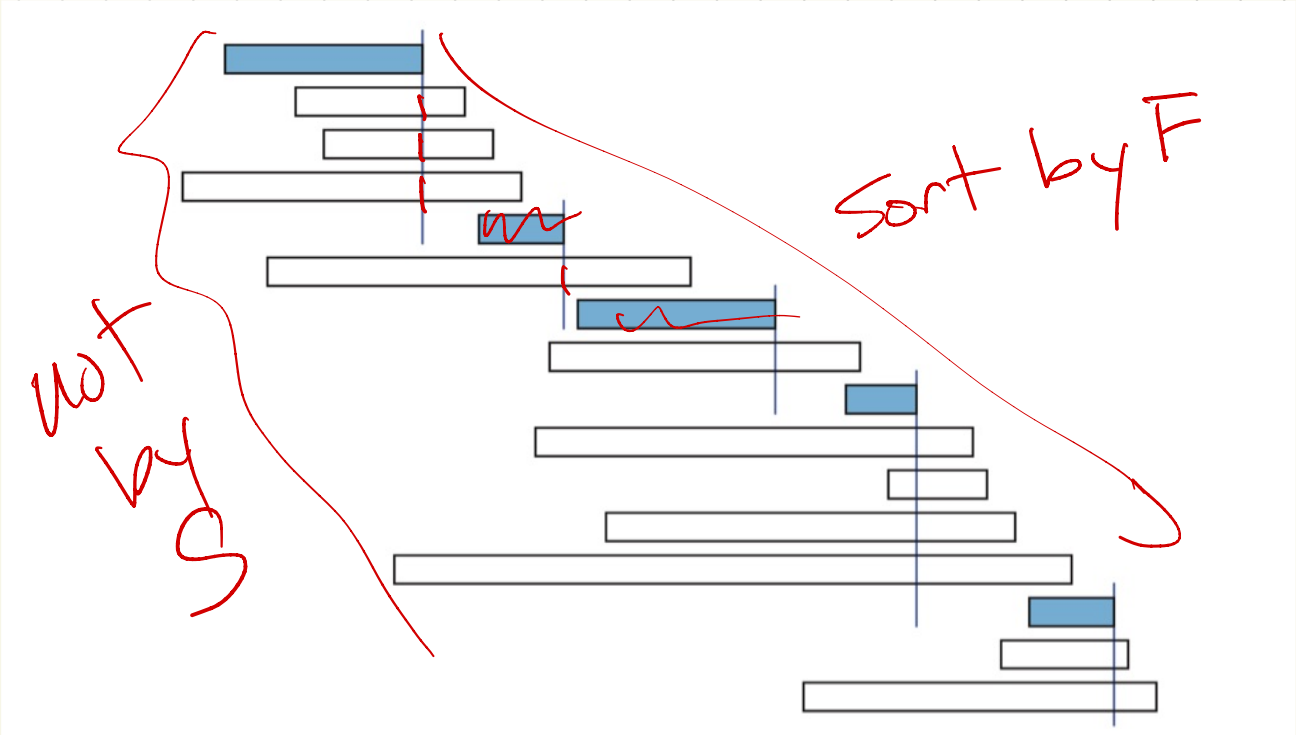
O(n log n)

compute X (& its size)

Picture:



A maximal conflict-free schedule for a set of classes.



The same classes sorted by finish times and the greedy schedule.

Correctness:

Why does this work?

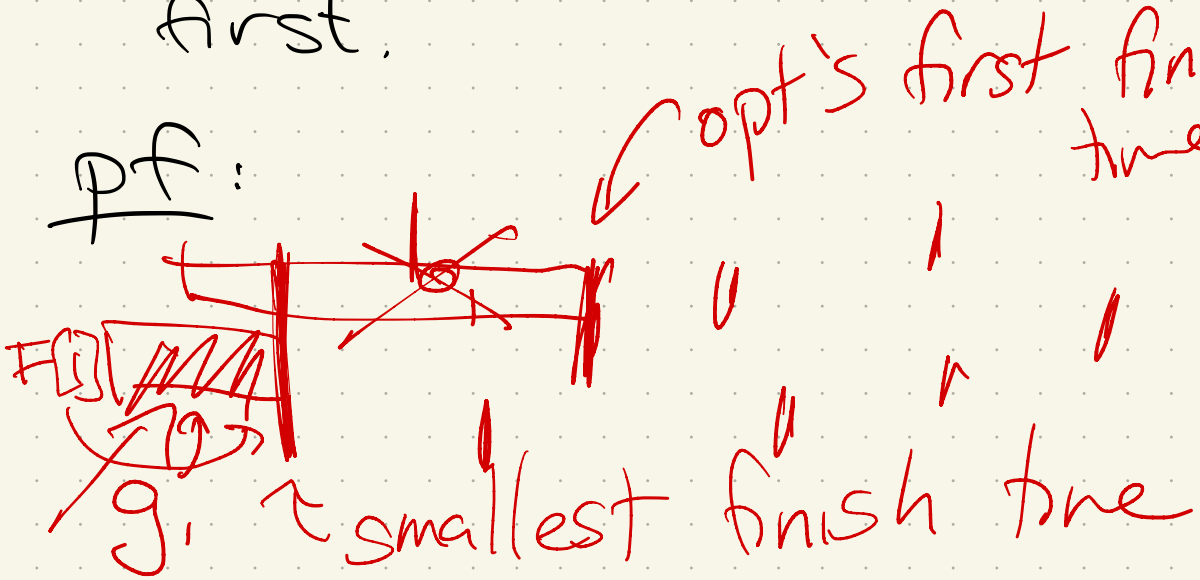
Note: No longer trying all possibilities or relying on optimal substructure!

So we need to be very careful on our proofs.

(Clearly, intuition can be wrong!)

Lemma: We may assume the optimal schedule includes the class that finishes first.

pf:



If opt (O) is different, then F of opt's first element is bigger

$\{g_1 \dots g_k\}$ $F[g_1] < F[o_1]$ happy!

$\{o_1, o_2 \dots o_l\}$ still started after $F[o_1]$

$S[o_i] > F[o_1] > F[g_1]$

Thm: The greedy schedule is optimal. (tied)

Pf: Suppose not.

Then ∃ an optimal schedule that has more intervals than the greedy one.

Consider first time they differ:

greedy: $\langle g_1, g_2, g_3, \ldots, g_i, g_e \rangle$

optimal: agree

$\langle g_1, g_2, \ldots, g_{i-1}, o_i, o_k \rangle$

$o_i \neq g_i$ but

o_j w/ $j < i$ is $= g_j$

use prev lemma

no diffs! $F[o_i] > F[g_i]$

& $S[o_{i+1}] \ldots S[o_k] > F[o_i]$

so swap.

Next time:

Huffman codes & Stable matchings.