Algorithms - Spring '25

Greedy Algs:
File Access
Scheduling

Recap

- Next week:
  - Sub on Monday, no office hours
  - Tuesday office hours 3-4pm
  - Wednesday office hours 2-3pm

    (plus usual TA times)

Dynamic Programing vs Greedy

Dyn. pro: try all possibilities
↳ but intelligently!

In greedy algorithms, we avoid building all possibilities.

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:

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

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

First example in the book:

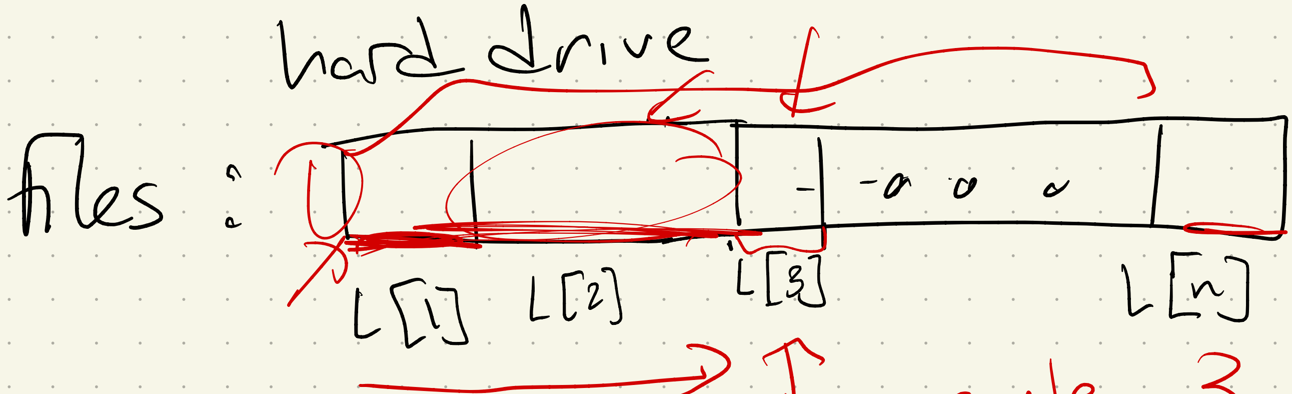
Storing files on tape.

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

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

length — frequency

Goal: Minimize access time:

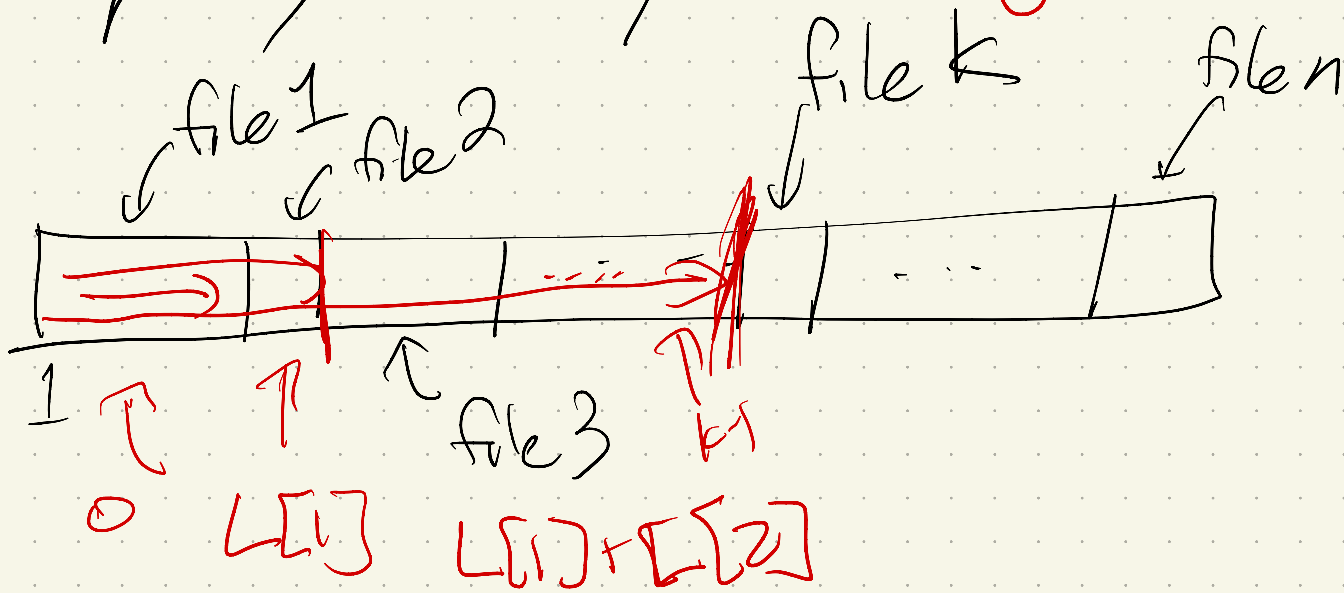


access item 3 $F[3]$ times

pay $L[1]+L[2]$

Cost? ⟶

If equally likely: order as given 1...n



Cost to access file k:

$$L[1]+L[2]+\cdots L[k-1] = \sum_{i=1}^{k-1} L[i]$$

If equally likely to access any file:

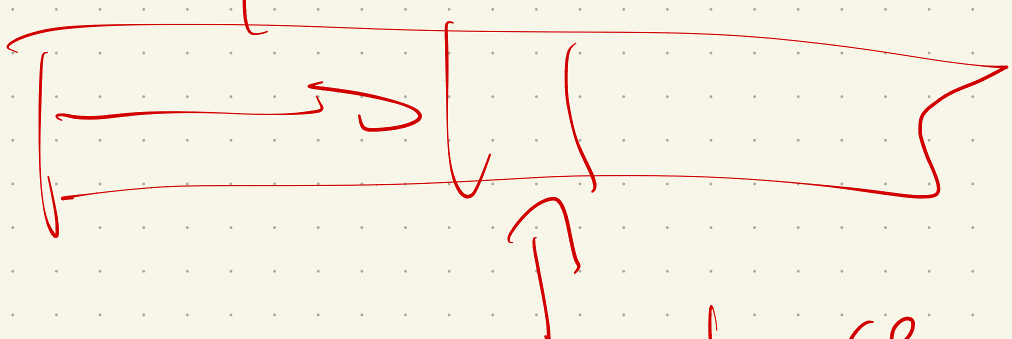
$$E[\text{cost}] = \sum_{k=1}^{n} (\text{prob of file } k)\cdot(\text{cost of file } k)$$

$$= \sum_{k=1}^{n} \left(\frac{1}{n}\right)\cdot\left(\sum_{i=1}^{k-1} L[i]\right)$$

$$E[\text{cost}] = \sum_{k=1}^{n} (\text{prob of file } k) \cdot (\text{cost of file } k)$$

$$= \sum_{k=1}^{n} \left(\frac{1}{n}\right) \cdot \left(\sum_{i=1}^{k-1} L[i]\right)$$

$$= \frac{1}{n} \sum_{k=1}^{n} \sum_{i=1}^{k-1} L[i]$$

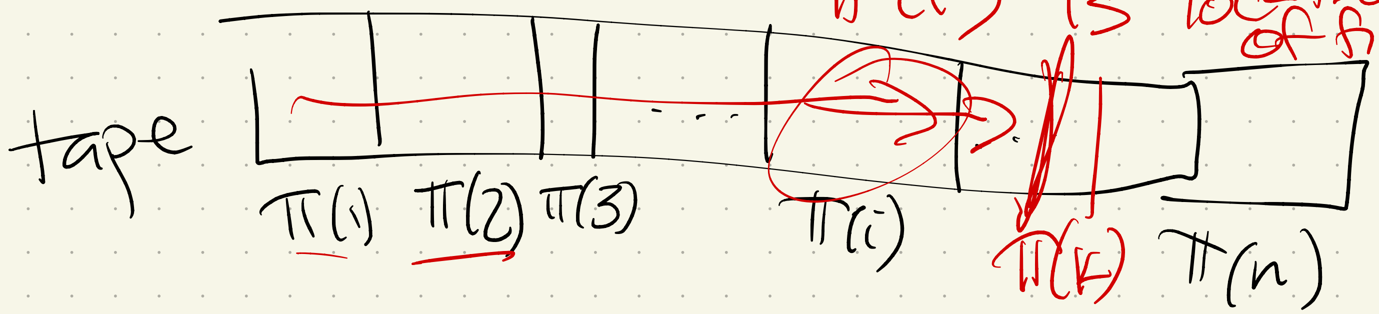


$1/n$ chance

Files: We can re-order:

permutation π $[1..n]$

$\pi(i)$ is location of file i



cost to access k^{th} one:

$$L(\pi(1)) + L(\pi(2)) \ldots L[\pi[k-1]]$$

$$= \sum_{i=1}^{k-1} L[\pi(i)]$$

And: how many times do I acess it?

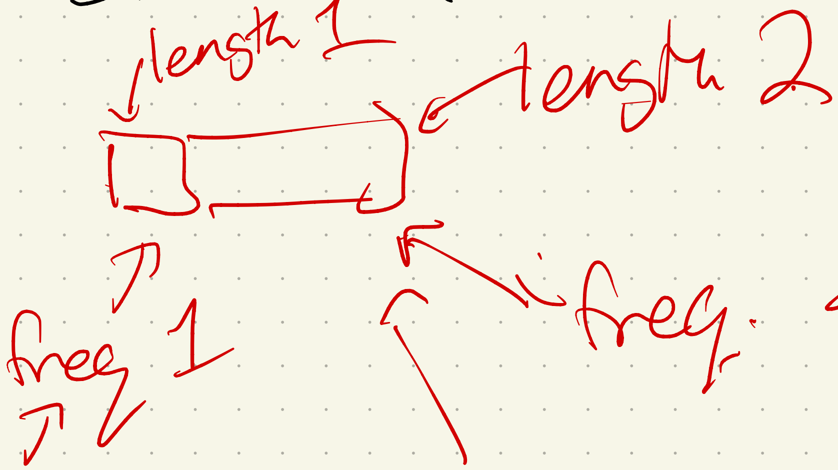
$F[\pi(k)]$ times

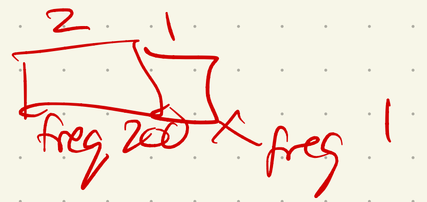
Total:

$$\Sigma cost(\pi) = \sum_{k=1}^{n}\left(F[\pi(k)] \cdot \sum_{i=1}^{k} L[\pi(i)]\right) = \sum_{k=1}^{n}\sum_{i=1}^{k}\left(F[\pi(k)] \cdot L[\pi(i)]\right).$$

How to be greedy?
(Not immediately clear!)

Try smallest first: no





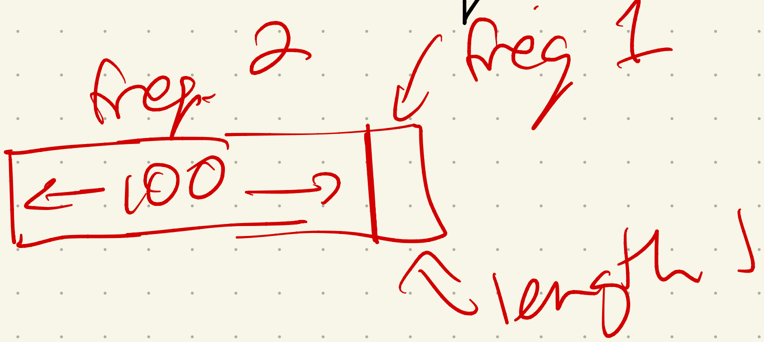
$0 \cdot 1 + 1 \cdot 200$

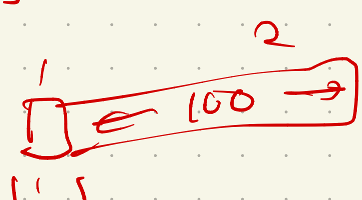
$= 200$

$200 \cdot 0 + 2 \cdot 1$

$= 2$

Try most frequent first: no





$2 \cdot 0 + 1 \cdot 100$

$= 100$

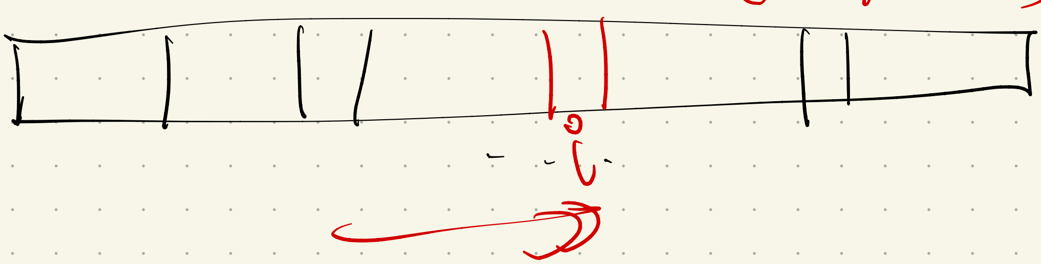
$1 \cdot 0 + 2 \cdot 1$

$= 2$ X

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

& will get optimal schedule.

pf: Suppose we sort: (by L/F)



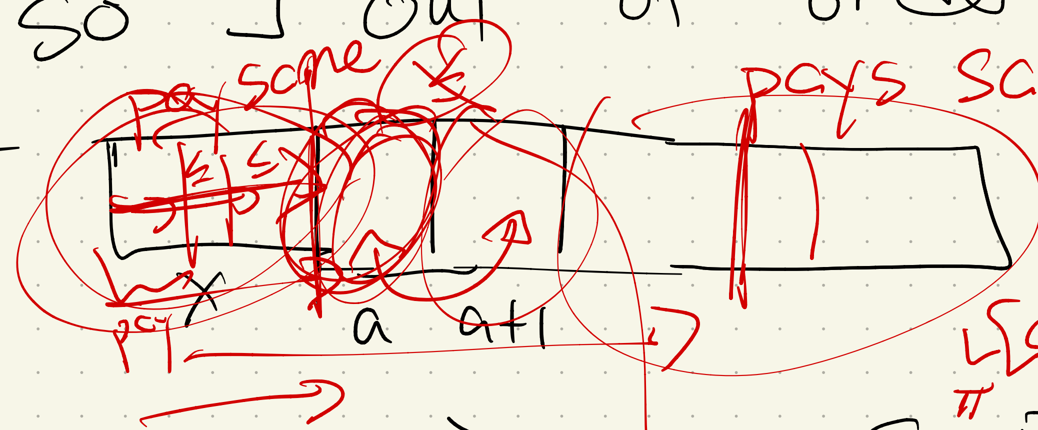
& $\forall i, \quad \frac{L[i]}{F[i]} \leq \frac{L[i+1]}{F[i+1]}$

Suppose this is <u>not</u> optimal.
What does that mean?

& opt must <u>not</u> be in this order

Well, OPT must be different,
so $\exists$ out of order pair:

OPT



$L[a]\cdot F[a+1] > L[a+1]\cdot F[a]$

with $\dfrac{L[a]}{F[a]} > \dfrac{L[a+1]}{F[a+1]}$

If OPT, must beat our "sorted" solution.

What if we swap a & a+1?

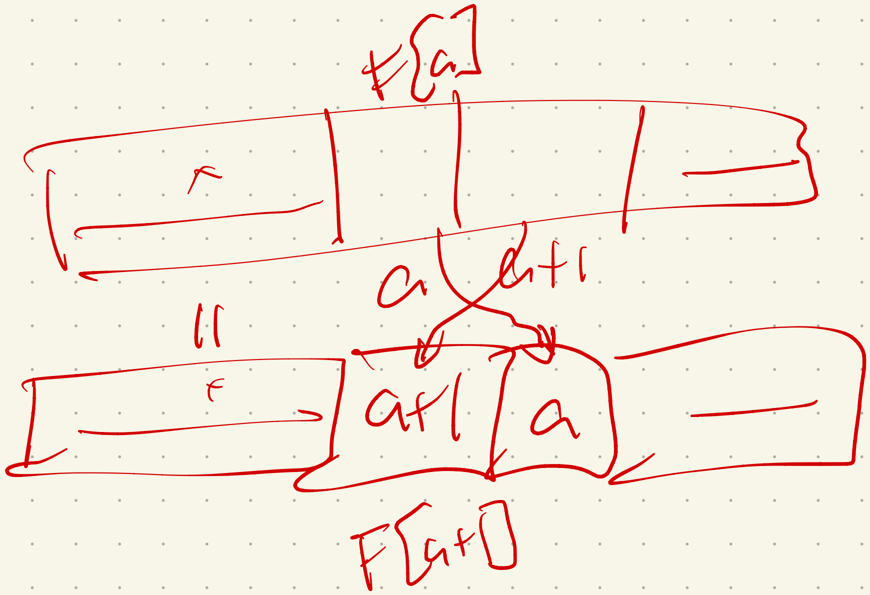
Before: $\left(\overbrace{\sum_{i=1}^{a-1} L[i]}^{x}\right)\cdot F[a] + \left(\overbrace{\sum_{i=1}^{a} L[i]}^{x+L[a]}\right) F[a+1]$

After: $\left(\overbrace{\sum_{i=1}^{a-1} L[i]}^{x}\right)\cdot F[a+1] +$

$\left(\left(\overbrace{\sum_{i=1}^{a-1} L[i]}^{x}\right) + L[a+1]\right)\cdot F[a]$

Pf (cont):

Before



After

difference:

Before

$$x \cdot F[a] + (x + L[a]) \cdot F[a+1]$$

$$- \left[x \cdot F[a+1] + (x + L[a+1]) \cdot F[a] \right]$$

after

$$x \overset{?}{=} \sum_{i=1}^{a-1} L[i]$$

$$\cancel{x \cdot F[a]} + \cancel{x \cdot F[a+1]} + L[a] F[a+1]$$

$$- \left(\cancel{x \cdot F[a+1]} + \cancel{x \cdot F[a]} + L[a+1] \cdot F[a] \right)$$

$$= L[a] \cdot F[a+1] - L[a+1] \cdot F[a] < 0$$

So: algorithm

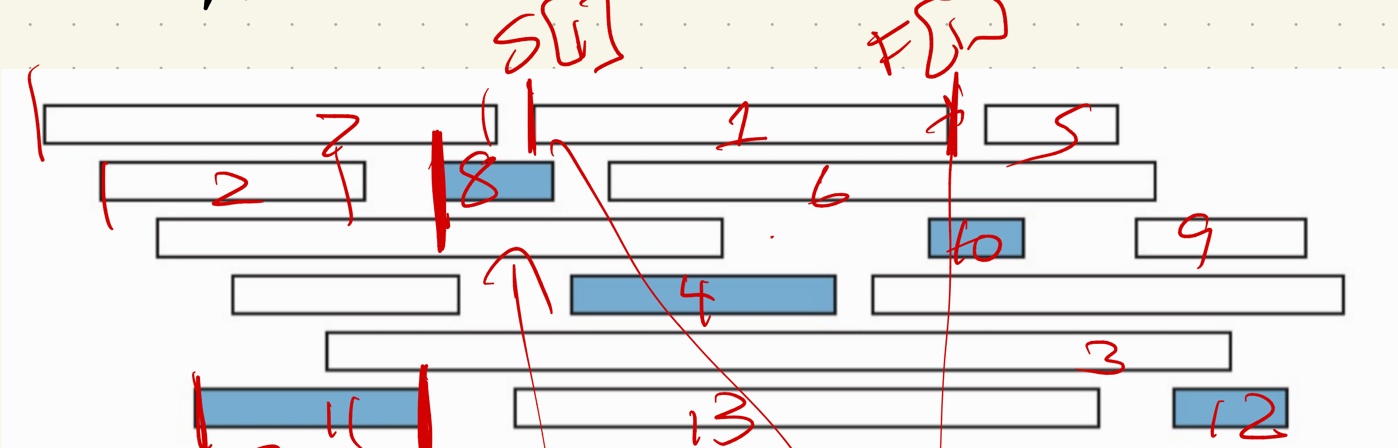
order by this

- Calculate $\frac{L[a]}{F[a]}$ for all a.
- Sort, + permute order of jobs to match.

Runtime: $O(n \log n)$

Problem: Interval Scheduling

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



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

$X[1] = 11$ $\quad$ $X[2]$

More formally:

Two arrays

$S[1..n]$:

$F[1..n]$:

$X[i]$ interval

↳ $S[X[i]]$

of $F[X[i]]$

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

$$F[i] \leq S[i+1]$$

How would we formalize a dynamic programing approach?

Recursive structure:

Consider job 1:

take it

↳ add to X

recurse on 2..n

don't

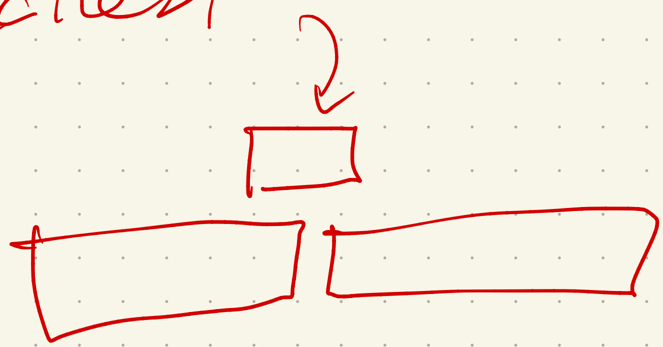
recurse on 2..n

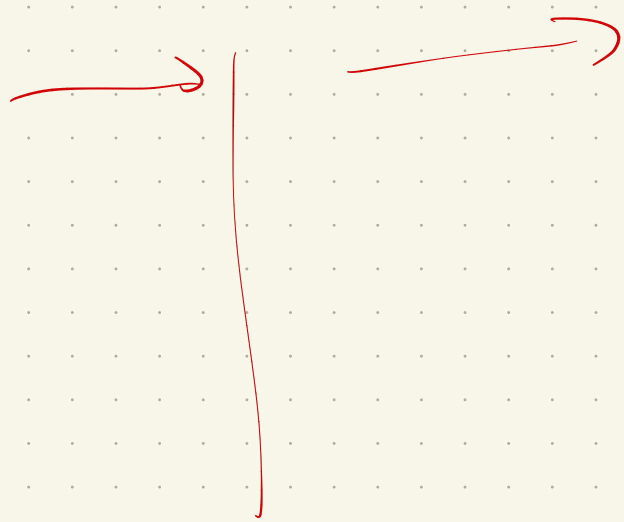
Intuition for greedy:

Consider what might be a good first one to choose.

Ideas?

Smallest





Key intuition:

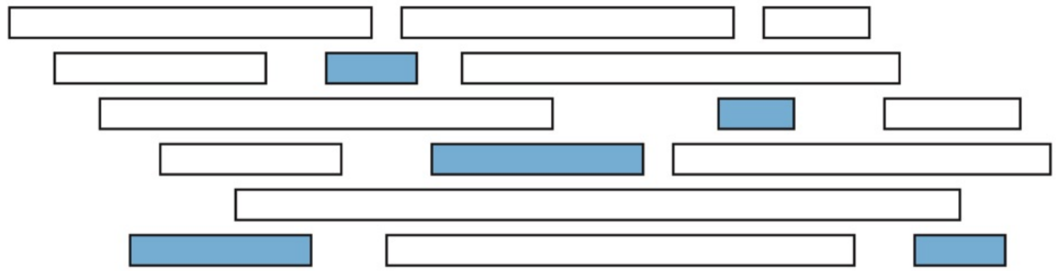
If it finishes as early as possible, we can fit more things in!

So - strategy:

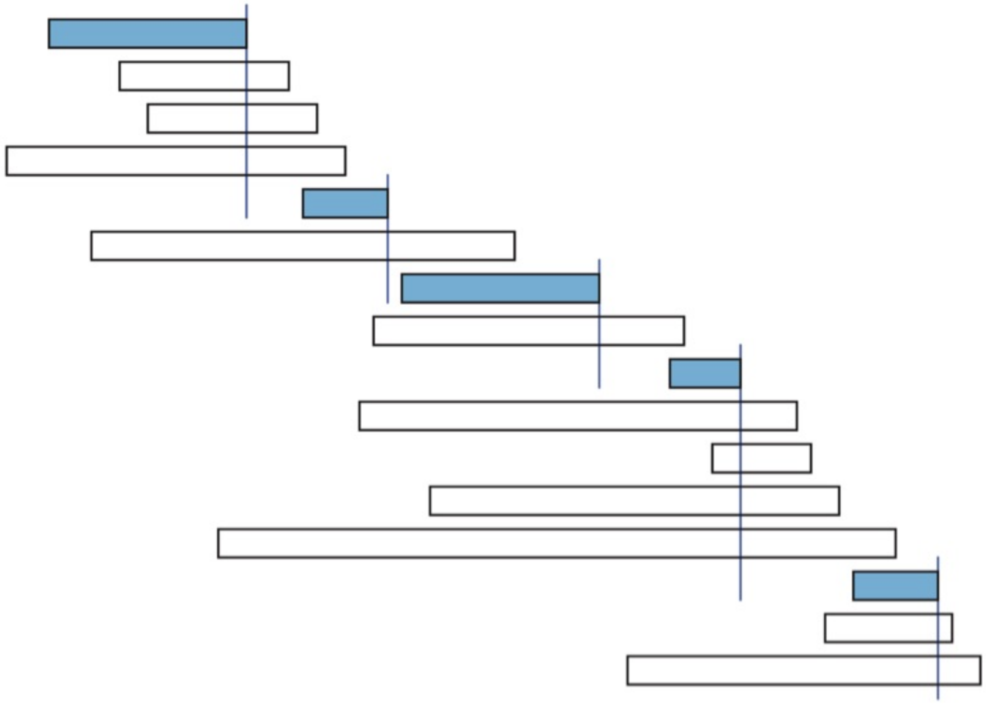
The code:

```
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]
```

Picture:



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



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

Correctnes:

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:

Thm: The greedy schedule is optimal.

pf: Suppose not.

Then $\exists$ an optimal schedule that has more intervals than the greedy one.

Consider first time they differ:

Greedy: $g_1 \; g_2 \cdots g_i \cdots g_k$

OPT: $O_1 \; O_2 \cdots O_i \cdots O_\ell$