CMSC 451 - Algorithm Design
Lecture 5 - Greedy Algorithms for Scheduling

Discrete Optimization:

Compute a discrete structure of a given class (e.g., subset, tree, path, partition) to maximize/minimize a given objective function (e.g., cost, distance, size) subject to a given set of constraints (e.g., disjointness, connectedness, completeness).

A feasible solution satisfies all constraints
An optimal solution is feasible and max/minimizes the objective function

Example:

Min. Spanning Tree: Given a connected, weighted graph compute:

- structure – subset of edges
- objective – min. total weight
- constraints – connected, acyclic, cover all vertices

Common Strategies:

- **Brute-force search** - Try all posibilities - Slow!!
- **Local search** - Find an init. feasible solution + repeatedly make small improvements
- **Dynamic programming** - (Future lectures)
- **Greedy** - Build a solution by repeated additions (never revoked/reversed) each based on best choice subject to constraints

  E.g., Kruskal's MST algorithm (add min. weight edge that doesn't cause a cycle)
- ...

This lecture: Three scheduling problems

- Interval scheduling
- Interval partioning
- Schedule to minimize lateness

Interval Scheduling:

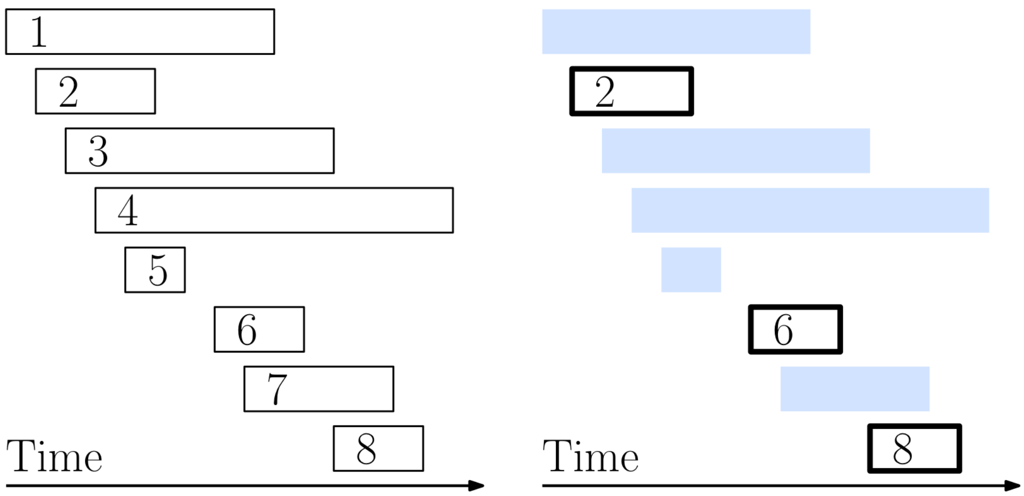
Given a set $R = \{r_1, \ldots, r_n\}$ of requests, each being an interval $r_i = [s_i, f_i]$ compute a subset of non-overlapping requests of max. cardinality

Application: Scheduling events at a facility

Example:

3 requests satisfied

Requests: Possible solution: $\{2, 6, 8\}$ Also: $\{5, 6, 8\}$



Greedy Approach - Select a request that does not conflict with prior selection + min. some measure:

- **Earliest Start** - request with smallest start, s_i
- **Earliest Finish** - " " " finish, f_i
- **Shortest Duration** - " " " duration, $f_i - s_i$
- **Min conflicts** - overlaps fewest among remaining requests

Optimal?

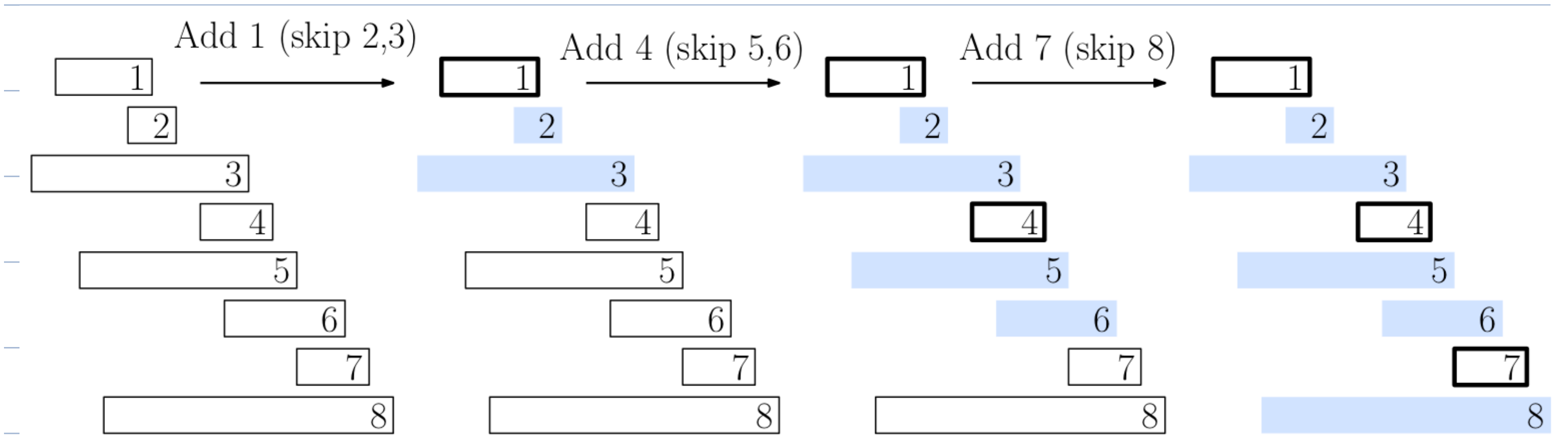
- Earliest Start - ✗
- Earliest Finish - ✓
- Shortest Duration - ✗
- Min conflicts - ✗

Exercise: Find counterexamples

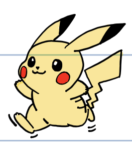
Earliest Finish First:

```
greedy-interval-sched(s, f)
    sort requests by f-times
    S ← ∅                          // init empty schedule
    prevFinish ← -∞
    for (i ← 1 to n)
        if (s[i] > prevFinish)     // no conflict?
            append i to S          // ... schedule it
            prevFinish ← f[i]
```

Example: (Presorted by finish times)



Running time:



$O(n \log n)$ - sort

$O(n)$ - remaining processing

$\left.\begin{array}{l} O(n \log n) \text{ - sort} \\ O(n) \text{ - remaining processing} \end{array}\right\} O(n \log n)$ total

Correctness:

Must show: **Feasibility** - A valid schedule (no conflicts)

Optimality - Maximizes no. of requests

Feasibility: Easy - No request is scheduled until after prev request finishes

Optimality: Not so easy - Let's do this rigorously.

Let $O = \langle x_1, \ldots, x_k \rangle$ be any optimal schedule (may be many)

Let $G = \langle g_1, \ldots, g_{k'} \rangle$ be EFF greedy schedule ($k' \leq k$)

If $G = O$, we're done!

otherwise, let j be smallest index s.t. $x_j \neq g_j$

- By definition of EFF, $f[g_j] \leq f[x_j]$
- Form a new schedule O' by replacing x_j by g_j — Still feasible + same size

$O:$	x_1	x_2	$\ldots$	x_{j-1}	x_j	x_{j+1}	x_{j+2}	$\ldots$
$G:$	x_1	x_2	$\ldots$	x_{j-1}	g_j	g_{j+1}	g_{j+2}	$\ldots$
$O':$	x_1	x_2	$\ldots$	x_{j-1}	g_j	x_{j+1}	x_{j+2}	$\ldots$

Repeat (hidden induction) until $O'''''' = G \Rightarrow G$ is optimal

$\square$

Next problem:

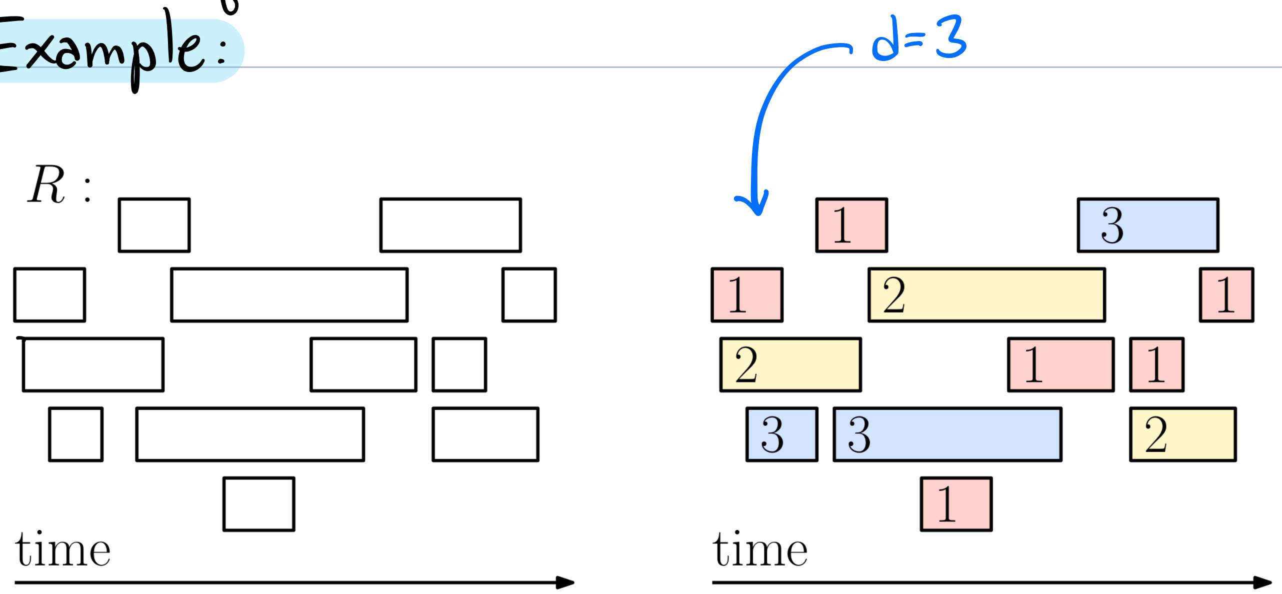
Interval Partitioning: Suppose we have ∞-many resources, but they cost \$\$\$

Want to satisfy all requests with fewest resources

Problem:

Given a set $R = \{r_1, \ldots, r_n\}$ of requests, each with start/finish times s_i / f_i, partition R into the smallest num. of sets $R_1, \ldots, R_d$ such that all requests in R_i are pairwise non-conflicting

Example:

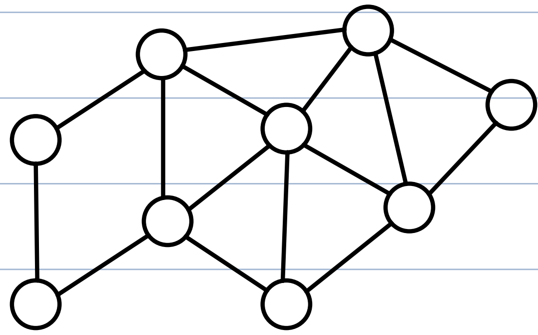


This is an example coloring –

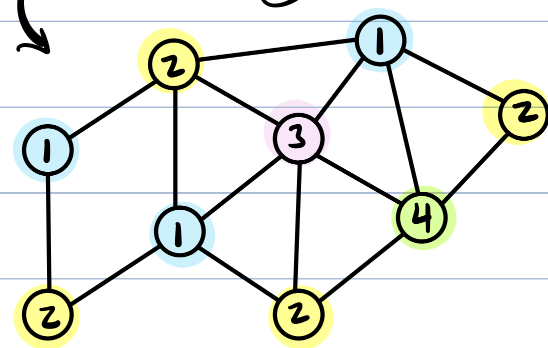
Given a set of objects + a conflict relation, partition into smallest number of conflict-free subsets

Number of subsets = coloring number

Graph coloring:



4-coloring



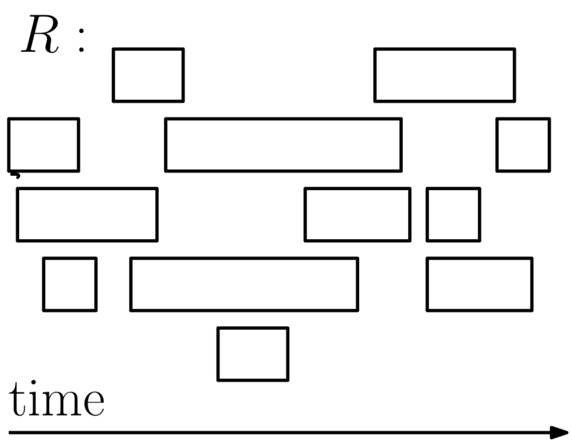
Graph coloring is NP-hard, but interval coloring is much easier.

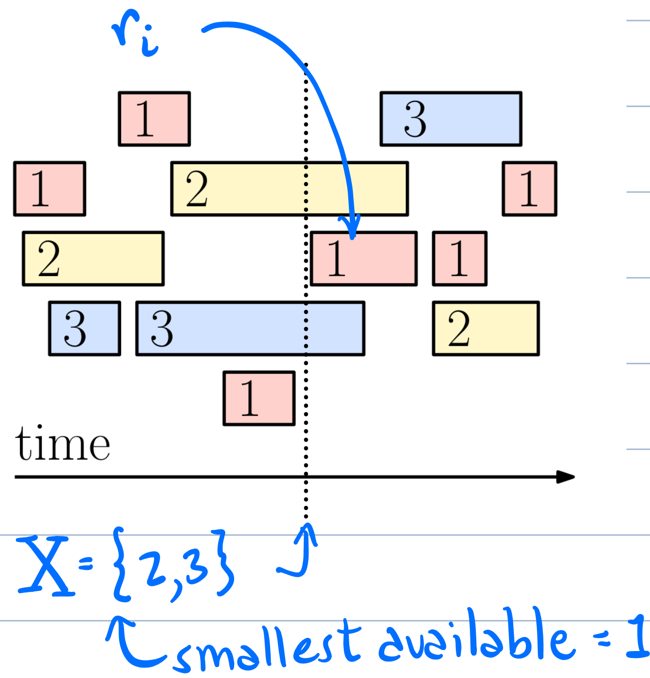
Greedy Strategy for Interval Partitioning

Lowest Available Color

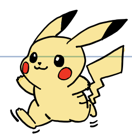
```
greedy-interval-partition(s, f)           // s = start
    sort by start times                   // f = finish
    for (i ← 1 to n)
        X ← ∅                             // X = excluded colors
        for (j ← 1 to i-1)
            if ([s_j, f_j] overlaps [s_i, f_i])
                add color[j] to X
        color[i] ← smallest color not in X
    return color array
```

Example:





Running Time:



- Sorting - $O(n \log n)$
- 2 nested loops $1 \ldots n + 1 \ldots i-1$

$$= \sum_{i=1}^{n} (i-1) = O(n^2)$$

$O(n^2)$ total

There is smarter approach that runs in $O(n)$ time after sorting - See lecture notes.

Correctness:
Need to show
- Feasibility - Obvious (Avoid conflicting colors)
- Optimality - ??

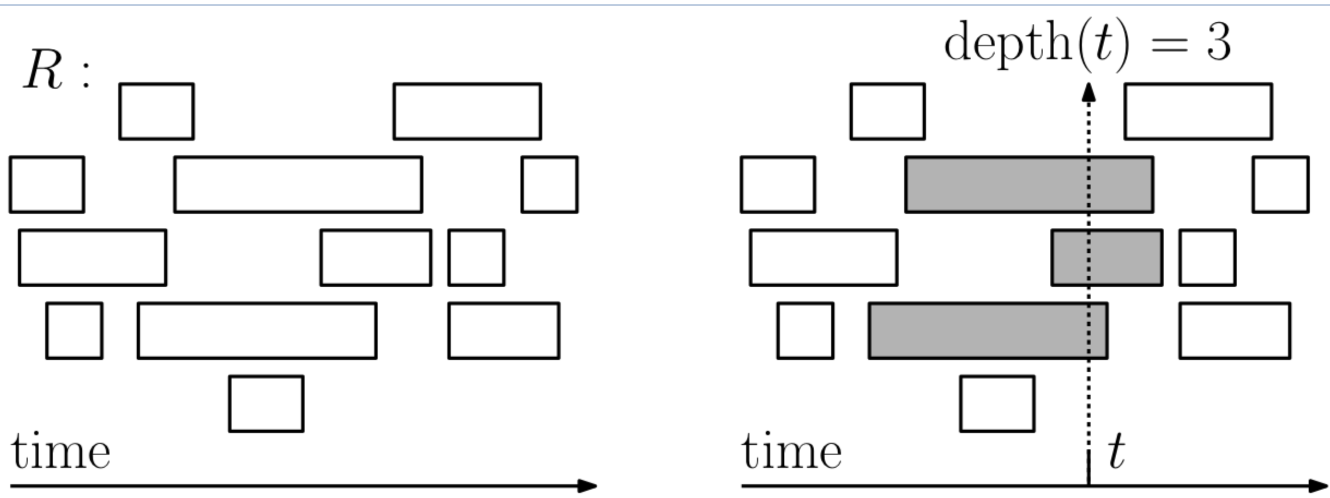
Optimality - Approach

- Define a statistic - depth
- Lower bound - Any solution must use $\geq$ depth colors
- Upper bound - Greedy algorithm uses $\leq$ depth colors

$\Rightarrow$ Greedy is optimal

Depth - Given request set $R = \{r_1, \dots, r_n\}$, $r_i = [s_i, f_i]$ + time t define

$$\text{depth}_R(t) = \text{num. intervals overlap } t = \left| \{ i \mid t \in [s_i, f_i] \} \right|$$



and: $$\text{depth}(R) = \max_{t \geq 0} \text{depth}_R(t)$$

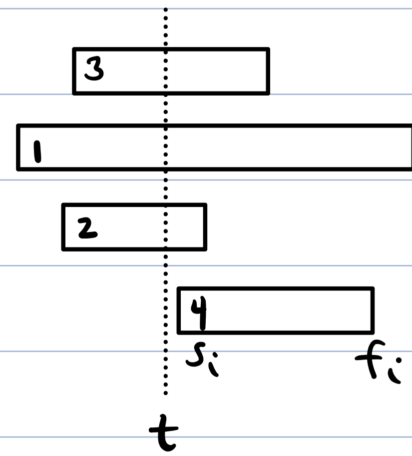
Clearly, all requests contibuting to depth conflict...

Lemma: For any d-coloring of R, $d \geq \text{depth}(R)$

The following implies that greedy is optimal

Lemma: The greedy algorithm generates a d-coloring, where $d \le depth(R)$

Proof: Suppose towards a contradiction that there is a first time s_i where greedy uses more than $depth_R(s_i)$ colors.



Consider time t just prior to s_i

$$depth_R(s_i) = depth_R(t) + 1 \quad (a)$$

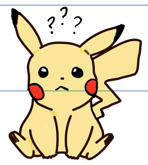
Let d = num. greedy colors at time t. Since s_i is the first error:

$$d \le depth_R(t). \quad (b)$$

Thus, there are d excluded colors in X when s_i is processed. Implies that greedy uses

$$d+1 \overset{(b)}{\le} depth_R(t)+1 \overset{(a)}{=} depth_R(s_i)$$

Contradicting hyp. that error at s_i $\square$



Q: If requests were sorted by a different criterion, would the algorithm still be optimal?

Q: Can we modify this to color graphs? (NP-hard!)

Scheduling to Minimize Lateness

- Tasks rather than requests - $X = \{x_1 \dots x_n\}$
  - Execution time - t_i
  - Deadline - d_i
- Application - Homework assignments

Objective:

- Compute start times $S = \{s_1, \dots, s_n\}$ s.t.
  - Tasks do not overlap $[s_i, f_i] \cap [s_j, f_j] = \emptyset$
    where $f_i = s_i + t_i$
  - Minimize lateness - max. deadline excess

$$l_i = \max(0, f_i - d_i)$$

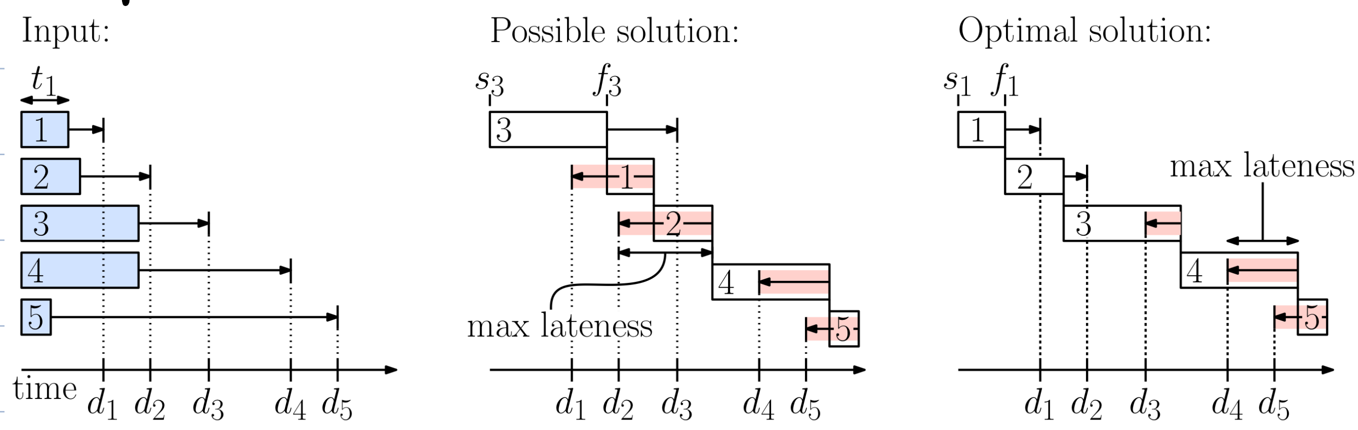
How far beyond the deadline did you finish?

max. lateness:

$$L(S) = \max_{1 \le i \le n} l_i$$

your worst deadline excess (not sum)

Example:



Greedy approach

- Schedule based on some criterion

- Shortest duration first - Sort by t_i ✗
- Earliest deadline first - Sort by d_i ✓
- Smallest slack first - Sort by $d_i - t_i$ ✗

```
greedy-lateness-sched(t, d)
    sort by deadlines (d_1 ≤ ... ≤ d_n)
    prevFinish ← 0                        // when prev task finished
    for (i ← 1 to n)
        s_i ← prevFinish                  // start after prev finish
        prevFinish ← f_i ← s_i + t_i      // update finish
        l_i ← max(0, f_i - d_i)           // lateness
    return [s_1, ..., s_n]   L = max_i l_i   // return start times
```

Running Time:

- Sorting - $O(n \log n)$
- Processing - $O(n)$

$\Big\}$ Total: $O(n \log n)$

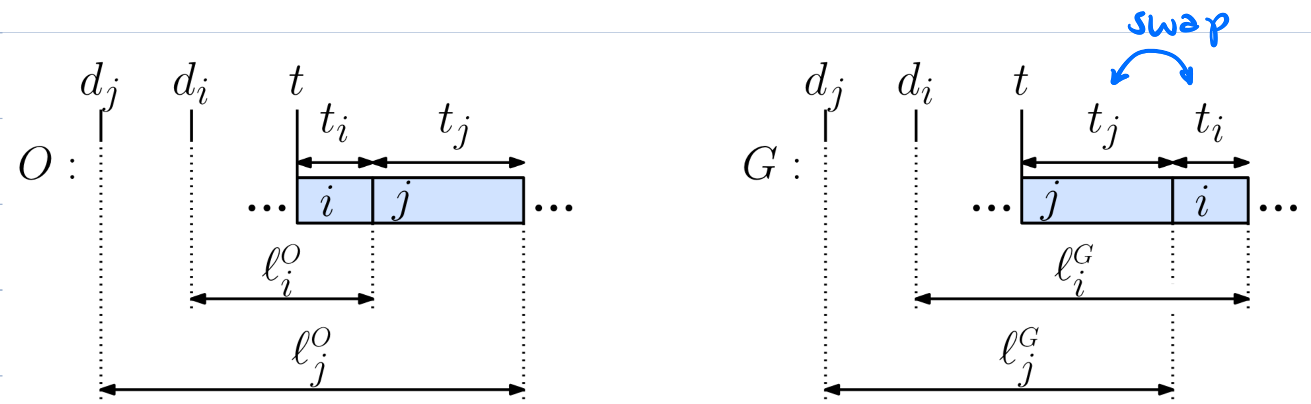
Correctness: Need to show

- Feasibility - Easy - No conflicting tasks
- Optimality - ??

> Lemma: The greedy algorithm minimizes max. lateness.

Proof: We may limit consideration to schedules that are "slack-free" - no gaps between tasks. (Greedy is slack-free)

- Let O be any lateness-optimal, slack-free schedule.
- If O is deadline sorted - we're done! $O = G$
- O.W. let x_i + x_j be first consecutive pair not in deadline order $d_i > d_j$
- We'll swap them + show that max lateness can only decrease.



- Let $\ell_i^O + \ell_j^O$ be latenesses before swap +
  $\ell_i^G + \ell_j^G$ " " after swap
- These are the only latenesses affected
- Want to show:

  $$\max(\ell_i^G, \ell_j^G) \le \max(\ell_i^O, \ell_j^O)$$

  swap improves max lateness
- Let t be current time
- For simplicity, assume $d_i + d_j$ passed: $d_j < d_i \le t$
  (Excercise- Fix this)

  We can ignore max
- Observe:

  $$\ell_i^O = (t + t_i) - d_i \qquad \ell_j^O = (t + t_i + t_j) - d_j$$

  Since $d_i > d_j$ + $t_j \ge 0$

  $$\ell_j^O = (t + t_i) + t_j - d_j > (t + t_i) - d_i = \ell_i^O$$

  $$\Rightarrow \max(\ell_i^O, \ell_j^O) = \ell_j^O$$
- Also:

  $$\ell_i^G = (t + t_i + t_j) - d_i < (t + t_i + t_j) - d_j = \ell_j^O$$

  $$\ell_j^G = (t + t_j) - d_j \le (t + t_i + t_j) - d_j = \ell_j^O$$
- Thus:

  $$\max(\ell_i^G, \ell_j^G) \le \max(\ell_j^O, \ell_j^O) = \max(\ell_i^O, \ell_j^O) \quad \square$$

Summary: 3 examples of greedy solutions to simple scheduling problems.

Interval scheduling - Earliest finish first

Interval partitioning - Smallest available color

Minimizing Max Lateness - Earliest deadline first

↳ All $O(n \log n)$