Objectives

- Wrap Up: Interval Partitioning
- Minimizing Lateness
  - Greedy exchange

Review

- Problem: Interval Scheduling
  - Solution?
  - How proved algorithm optimal?
- Problem: Interval Partitioning
  - Solution?

Review: Greedy Stays Ahead Proofs

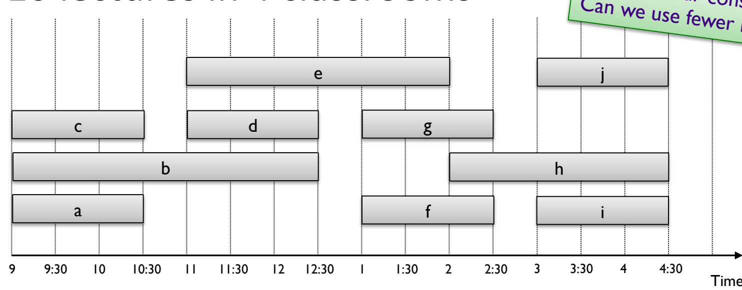
1. Define your solutions
   - Describe the form of your greedy solution (**A**) and of some other solution (possibly the optimal solution, **O**)
2. Find a measure
   - Find a measure by which greedy ***stays ahead*** of the optimal solution
     - Ex: Let $a_1, \ldots, a_k$ be the first *k* measures of greedy algorithm and $o_1, \ldots, o_m$ be the first *m* measures of other solution (sometimes $m = k$)
3. Prove greedy stays ahead
   - Show that greedy's partial solutions constructed are always just as good as the optimal solution's initial segments based on the measure
     - Ex: for all indices $r \leq \min(k,m)$, prove by induction that $a_r \geq o_r$ or $a_r \leq o_r$
   - Use the greedy algorithm to help you argue the inductive step
4. Prove optimality
   - Prove that since greedy stays ahead of the other solution with respect to the measure, then the greedy solution is optimal

→ Make sure maps back to measure of optimality

Review: Interval Partitioning

- Lecture j starts at s_j and finishes at f_j
- Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.
- Ex: 10 lectures in 4 classrooms

What are our constraints?
Can we use fewer rooms?



Review:
Interval Partitioning: Greedy Algorithm

- Consider lectures in increasing order of start time: assign lecture to any compatible classroom

```
Sort intervals by starting time so that s1 ≤ s2 ≤ ... ≤ sn
d = 0    ← number of allocated classrooms
for j = 1 to n
   if (lecture j is compatible with some classroom k)
      schedule lecture j in classroom k
   else
      allocate a new classroom d + 1
      schedule lecture j in classroom d + 1
      d = d + 1
```

- Implementation: O(n log n)
  - For each classroom *k*, maintain the finish time of the last job added
  - Keep the classrooms in a priority queue by last job finish time

Interval Partitioning: Greedy Analysis

- **Defn.** The *depth **d*** of a set of open intervals (lectures) is the maximum number that contain any given time.
- **Observation**. Greedy algorithm never schedules two incompatible lectures in the same classroom
- **Theorem**. Greedy algorithm is optimal
- Pf Intuition
  - When do we add more classrooms?
  - When would we add the d+1 classroom?

Interval Partitioning: Greedy Analysis

- **Observation**. Greedy algorithm never schedules two incompatible lectures in the same classroom
- **Theorem**. Greedy algorithm is ***optimal***
- Pf.
  - Let d = number of classrooms that the greedy algorithm allocates
  - Classroom d is opened because we needed to schedule a job, say j, that is incompatible with all d-1 other classrooms
  - Since we sorted by start time, all these incompatibilities are caused by lectures that start no later than s_j
  - Thus, we have d lectures overlapping at time $s_j + \varepsilon$
  - d is the depth of the set of lectures

Structural argument

Exchange argument

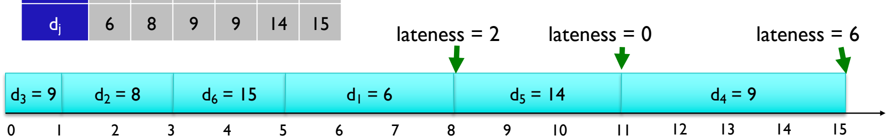
SCHEDULING TO MINIMIZE MAX LATENESS

Scheduling to Minimizing Max Lateness

- Single resource processes one job at a time
- Job j requires t_j units of processing time and is due at time d_j (its deadline)
- If j starts at time s_j, it finishes at time $f_j = s_j + t_j$
- Lateness: $\ell_j = \max \{ 0, \ f_j - d_j \}$
- **Goal**: schedule all jobs to ***minimize*** **maximum lateness** $L = \max \ell_j$

Input:

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



One schedule

Note: **not** a sum total

Greedy Algorithms

- Greedy template.
  Consider jobs in some order
- What do we want to optimize?
- What order?
  - Intuition of order?
  - Counter examples for order being optimal?

Minimizing Lateness: Greedy Algorithms

- Greedy template. Consider jobs in some order.
  - Shortest processing time first. Consider jobs in ascending order of processing time t_j.

Counter example

	1	2
t_j	1	10
d_j	100	10

  - Smallest slack. Consider jobs in ascending order of slack $d_j - t_j$.

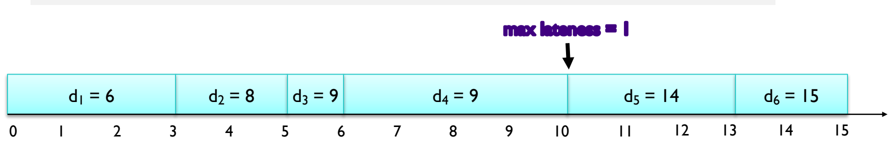
Counter example

	1	2
t_j	1	10
d_j	2	10

Minimizing Lateness: Greedy Algorithm

- Earliest deadline first.

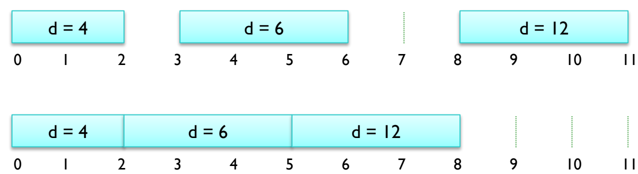
```
Sort n jobs by deadline so that d1 ≤ d2 ≤ … ≤ dn
t = 0
for j = 1 to n
   Assign job j to interval [t, t + tj]
   sj = t
   fj = t + tj
   t = t + tj
output intervals [sj, fj]
```



What can we say about this algorithm/its results?

Minimizing Lateness: No Idle Time

- Observation. There exists an optimal schedule with no idle time



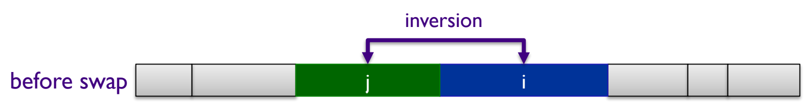
- Observation. The greedy schedule has no idle time

Proving Optimality

- Goal: Prove greedy algorithm produces optimal solution
- Approach: **Exchange argument**
  - Start with an optimal schedule Opt
  - Gradually modify Opt, *preserving* its optimality
  - Transform into a schedule identical to greedy's schedule

Minimizing Lateness: Inversions

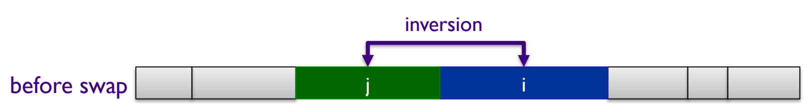
- Def. An ***inversion*** in schedule S is a pair of jobs *i* and *j* such that:
  $d_i < d_j$ (*i*'s deadline is before *j*)
  but *j* scheduled before *i*



Can Greedy's solution have any inversions?

Minimizing Lateness: Inversions

- Def. An ***inversion*** in schedule S is a pair of jobs *i* and *j* such that:
  $d_i < d_j$ (*i*'s deadline is before *j*)
  but *j* scheduled before *i*



Greedy's schedule has no inversions!

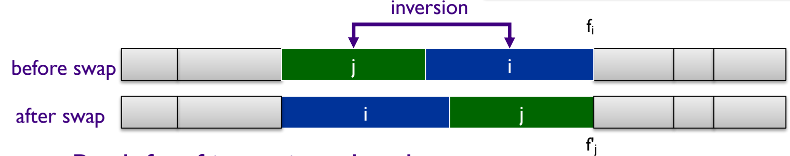
Minimizing Lateness: Inversions

- Claim. Swapping two adjacent, inverted jobs reduces the number of inversions by one and does *not increase* the ***max*** *lateness*

How do we know inversions are adjacent?

- Pf Setup. Let ℓ be the lateness before the swap, and let ℓ' be it afterwards

What can we say about how i's, j's, and other jobs' lateness changes?

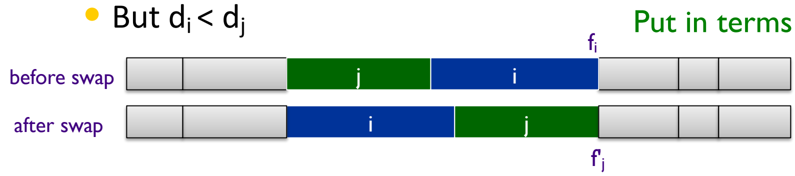


By defn of inversion, $d_i < d_j$

Minimizing Lateness: Inversions

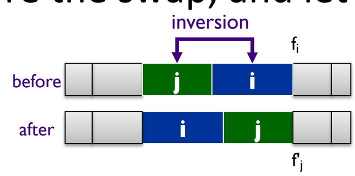
- Claim. Swapping two adjacent jobs with the same deadline does not increase the max lateness
- Pf. Let ℓ be the lateness before the swap, and let ℓ' be it afterwards
  - Lateness remains the same for all other jobs:
    - $\ell'_k = \ell_k$ for all $k \neq i, j$
  - $\ell_j \leq \ell_i$ because $d_i < d_j$
  - Lateness of *i* before is $\ell_i = f_i - d_i = T_{i-1} + t_i + t_j - d_i$
  - Lateness of *j* after is $\ell'_j = f_j' - d_j = T_{i-1} + t_i + t_j - d_j$
    - But $d_i < d_j$

Put in terms of ℓ_i



Minimizing Lateness: Inversions

- Claim. Swapping two adjacent, inverted jobs reduces the number of inversions by one and does *not increase the max lateness*.
- Pf. Let ℓ be the lateness before the swap, and let ℓ' be it afterwards
  - $\ell'_k = \ell_k$ for all $k \neq i, j$
  - $\ell'_i \leq \ell_i$
  - If job j is late:



$$
\begin{aligned}
\ell'_j &= f'_j - d_j && \text{(definition)} \\
&= f_i - d_j && (j \text{ finishes at time } f_i) \\
&\leq f_i - d_i && (i < j) \\
&\leq \ell_i && \text{(definition)}
\end{aligned}
$$

$d_i < d_j$

Shows that the maximum lateness of jobs does not increase after swap

Greedy Exchange Proofs

1. Label your algorithm's solution and a general solution.
   - Example: let A = {$a_1, a_2, \ldots, a_k$} be the solution generated by your algorithm, and let O = {$o_1, o_2, \ldots, o_m$} be an optimal feasible solution.
2. Compare greedy with other solution.
   - Assume that the optimal solution is not the same as your greedy solution (since otherwise, you are done).
   - Typically, can isolate a simple example of this difference, such as:
     1. There is an element $e \in O$ that $\notin A$ and an element $f \in A$ that $\notin O$
     2. 2 consecutive elements in O are in a different order than in A
        - i.e., there is an *inversion*
3. Exchange.
   - Swap the elements in question in O (either ① swap one element out and another in or ② swap the order of the elements) and argue that solution is no worse than before.
   - Argue that if you continue swapping, you eliminate all differences between O and A in a *finite* # of steps *without worsening the solution's quality*.
   - Thus, the greedy solution produced is just as good as any optimal solution, and hence is optimal itself.

Minimizing Lateness: Analysis of Greedy Algorithm

- Theorem. Greedy schedule S is optimal
- Pf idea. Convert Opt to Greedy
  - Does opt schedule have idle time?
  - What if opt schedule has no inversions?
  - What if opt schedule has inversions?

Minimizing Lateness: Analysis of Greedy Algorithm

- Theorem. Greedy schedule S is optimal
- Pf. Define S* to be an optimal schedule that has the fewest number of inversions, and let's see what happens
  - Can assume S* has no idle time
  - If S* has no inversions, then S = S*
  - If S* has an inversion, let i-j be an adjacent inversion
    - Swapping i and j does not increase the maximum lateness and strictly decreases the number of inversions
    - This contradicts definition of S* ▪

Looking Ahead

- Exam due Friday
- No wiki this week