Objectives

- Greedy Algorithms
  - Interval partitioning
  - Minimizing Lateness
- Greedy stays ahead
- Exchange argument

Review: Greedy Algorithm Template

- Consider jobs (or whatever) in some order
  - Decision: What order is best?
- Take each job provided it's compatible with the ones already taken

Greedy Algorithms

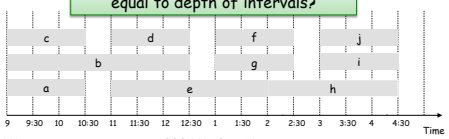
- At each step, take as much as you can get
  - Feasible – satisfy problem's constraints
  - Locally optimal – best local choice among available feasible choices
  - Irrevocable – after decided, no going back

Interval Partitioning: Lower Bound on Optimal Solution

- Def. The depth of a set of open intervals is the maximum number that contain any given time.
- Key observation. # of classrooms needed ≥ depth.
- Ex: Depth of schedule below = 3 ⇒ schedule below is optimal.

a, b, c all contain 9:30

Does there always exist a schedule equal to depth of intervals?



Interval Partitioning Discussion

- Does there always exist a schedule equal to depth of intervals?
- Can we make decisions locally to get a global optimum?
  - Or are there long-range obstacles that require more resources?

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
```

Analyze algorithm

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
```

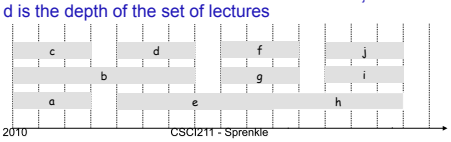
- Implementation: O(n log n)
  - For each classroom k, maintain the finish time of the last job added.
  - Keep the classrooms in a priority queue.

Interval Partitioning: Greedy Analysis

- 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 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



Proving Greedy Algorithms Work

- Specifically, produce an **optimal** solution
- Two approaches:
  - Greedy algorithm stays ahead
    - Does better than any other algorithm at each step
  - Exchange argument
    - Transform any solution into a greedy solution

Greedy Stays Ahead Proofs

1. Define your solutions
   - Describe the form your greedy solution takes and what form some other solution takes (possibly the optimal solution)
   - Example: Let A be the solution constructed by the greedy algorithm and O be an optimal solution.
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 initial segments of the optimal solution, based on the measure
   - Ex: for all indices $r \le \min(k,m)$, prove by induction that $a_r \ge o_r$ or $a_r \le o_r$
   - 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.

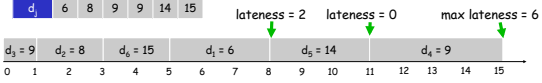
Exchange argument

SCHEDULING TO MINIMIZE LATENESS

Scheduling to Minimizing 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** $L = \max \ell_j$

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



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 $d_j - t_j$.

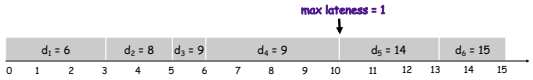
Counter example

	1	2
t_j	1	10
d_j	2	10

Minimizing Lateness: Greedy Algorithm

- Earliest deadline first.

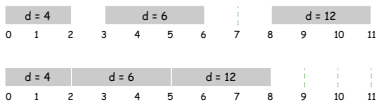
```
Sort n jobs by deadline so that d1 ≤ d2 ≤ … ≤ dn
t = 0
for j = 1 to n
   Assign job j to interval [t, t + tj]
   sj = t
   fj = t + tj
   t = t + tj
output intervals [sj, fj]
```



What can we say about this algorithm/its results?

Minimizing Lateness: No Idle Time

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



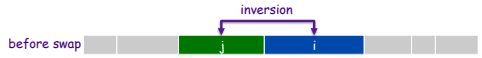
- Observation. The greedy schedule has no idle time

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

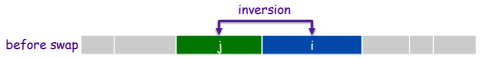
- Def. An ***inversion*** in schedule S is a pair of jobs i and j such that:
  $d_i < d_j$ but j scheduled before i



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:
  $d_i < d_j$ but j scheduled before i



Greedy's schedule has no inversions!

Minimizing Lateness: Inversions

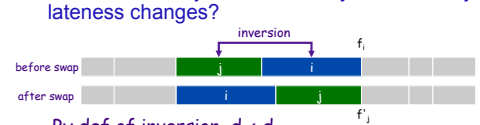
- Claim. Swapping two adjacent jobs with the same deadline does not increase the max lateness
- Pf Sketch. Let ℓ be the lateness before the swap, and let ℓ' be it afterwards
  - Lateness of other jobs?
  - Lateness of i? j?

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 $k \neq i, j$
  - Lateness of i before is $f_i - d_i = t_i + t_j - d_i$
  - Lateness of j after is $f'_j - d_j = t_i + t_j - d_j$
    - But $d_i = d_j$

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 def of inversion, $d_i < d_j$

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:

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

Assignments

- Read Chapter 4
  - Wiki due next Wednesday
- Exam 1
  - Open book, open notes, open lecture notes
  - **NO OTHER RESOURCES**
  - I mention explicitly to analyze your algorithms' running times. I will not do that in the future.
  - Wed: half lecture, half questions