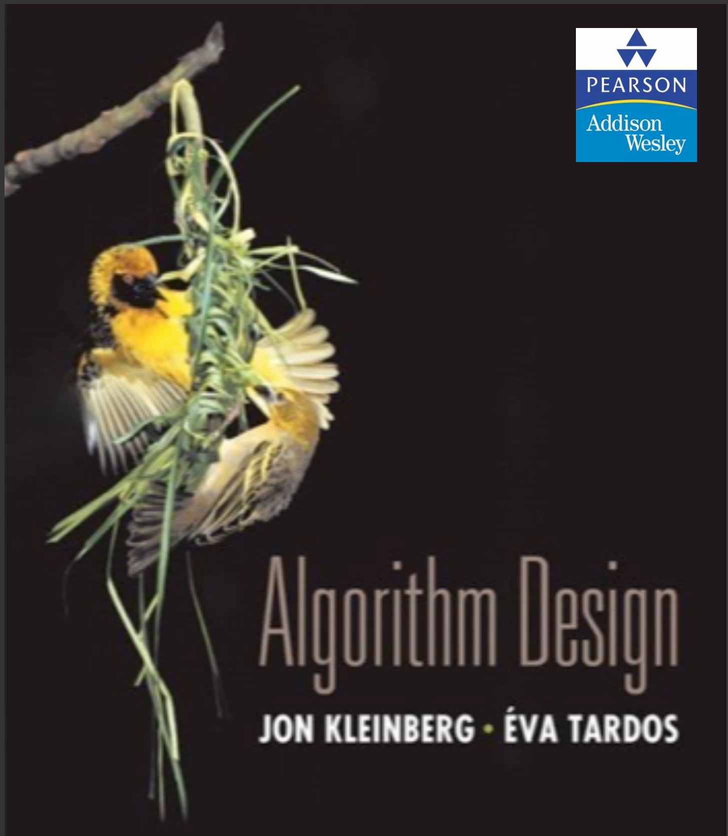


Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

4. Greedy Algorithms I

- *coin changing*
- *interval scheduling*
- *scheduling to minimize lateness*
- *optimal caching*

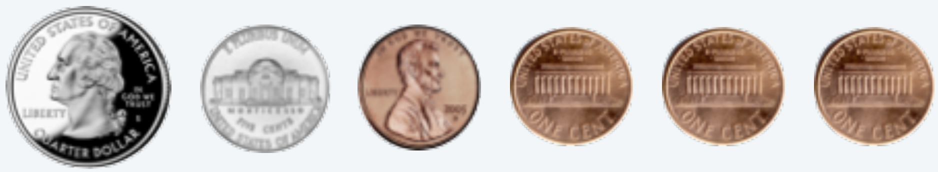
4. GREEDY ALGORITHMS I

- *coin changing*
- *interval scheduling*
- *scheduling to minimize lateness*
- *optimal caching*

Coin changing

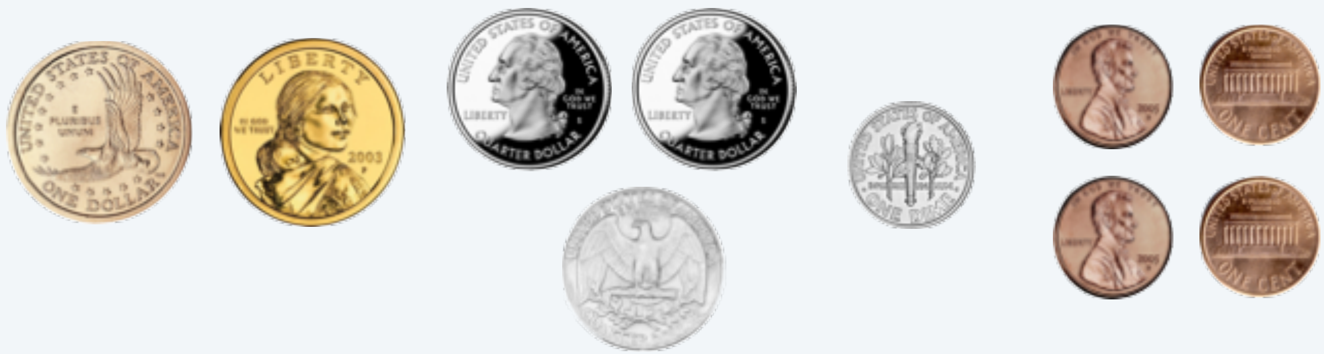
Goal. Given currency denominations: 1, 5, 10, 25, 100, devise a method to pay amount to customer using fewest number of coins.

Ex. 34¢.



Cashier's algorithm. At each iteration, add coin of the largest value that does not take us past the amount to be paid.

Ex. $2.89.



Cashier's algorithm

At each iteration, add coin of the largest value that does not take us past the amount to be paid.

CASHIERS-ALGORITHM $(x, c_1, c_2, \ldots, c_n)$

SORT n coin denominations so that $c_1 < c_2 < \ldots < c_n$

$S \leftarrow \varphi$ ⟵ set of coins selected

WHILE $x > 0$

  $k \leftarrow$ largest coin denomination c_k such that $c_k \leq x$

  IF no such k, RETURN "no solution"

  ELSE

    $x \leftarrow x - c_k$

    $S \leftarrow S \cup \{ k \}$

RETURN S

Improvement

Integer division to select more than one coin at a time.

Q. Is cashier's algorithm optimal?

Properties of optimal solution

Property. Number of pennies $\leq$ 4.

Pf. Replace 5 pennies with 1 nickel.

Property. Number of nickels $\leq$ 1.

Property. Number of quarters $\leq$ 3.

Property. Number of nickels + number of dimes $\leq$ 2.

Pf.

- Replace 3 dimes and 0 nickels with 1 quarter and 1 nickel;
- Replace 2 dimes and 1 nickel with 1 quarter.
- Recall: at most 1 nickel.

Analysis of cashier's algorithm

Theorem. Cashier's algorithm is optimal for U.S. coins: 1, 5, 10, 25, 100.

Pf. [by induction on x]

- Consider optimal way to change $c_k \leq x < c_{k+1}$: greedy takes coin k.
- We claim that any optimal solution must also take coin k.
  - if not, it needs enough coins of type $c_1, \ldots, c_{k-1}$ to add up to x
  - table below indicates no optimal solution can do this
- Problem reduces to coin-changing $x - c_k$ cents, which, by induction, is optimally solved by cashier's algorithm. ▪

k	c_k	all optimal solutions must satisfy	max value of coins $c_1, c_2, \ldots, c_{k-1}$ in any OPT
1	1	$P \leq 4$	–
2	5	$N \leq 1$	4
3	10	$N + D \leq 2$	4 + 5 = 9
4	25	$Q \leq 3$	20 + 4 = 24
5	100	*no limit*	75 + 24 = 99

Cashier's algorithm for other denominations

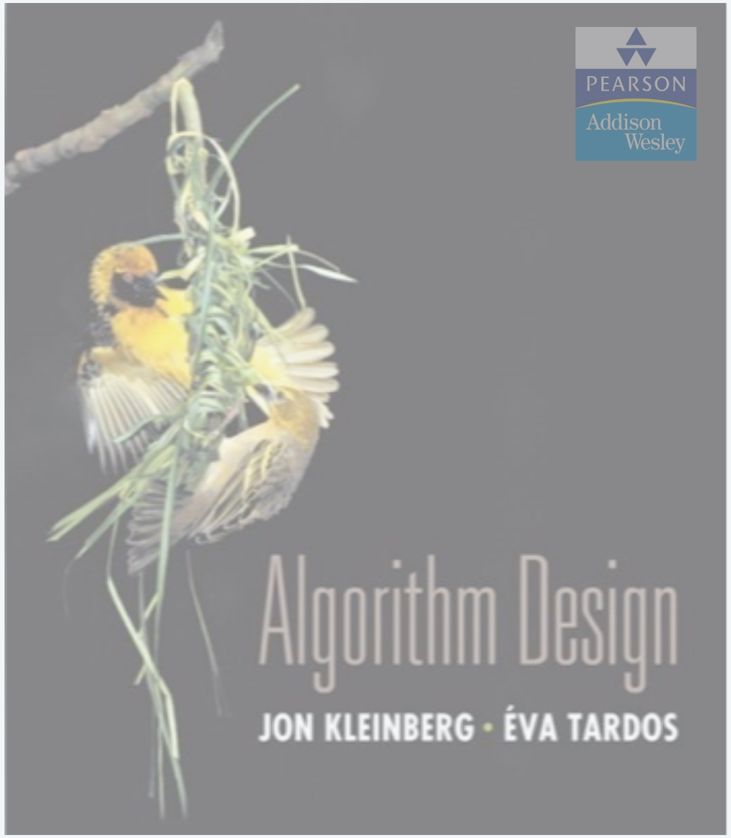
Q. Is cashier's algorithm for any set of denominations?

A. No. Consider U.S. postage: 1, 10, 21, 34, 70, 100, 350, 1225, 1500.

- Cashier's algorithm: 140¢ = 100 + 34 + 1 + 1 + 1 + 1 + 1 + 1.
- Optimal: 140¢ = 70 + 70.

A. No. It may not even lead to a feasible solution if $c_1 > 1$: 7, 8, 9.

- Cashier's algorithm: 15¢ = 9 + ???.
- Optimal: 15¢ = 7 + 8.

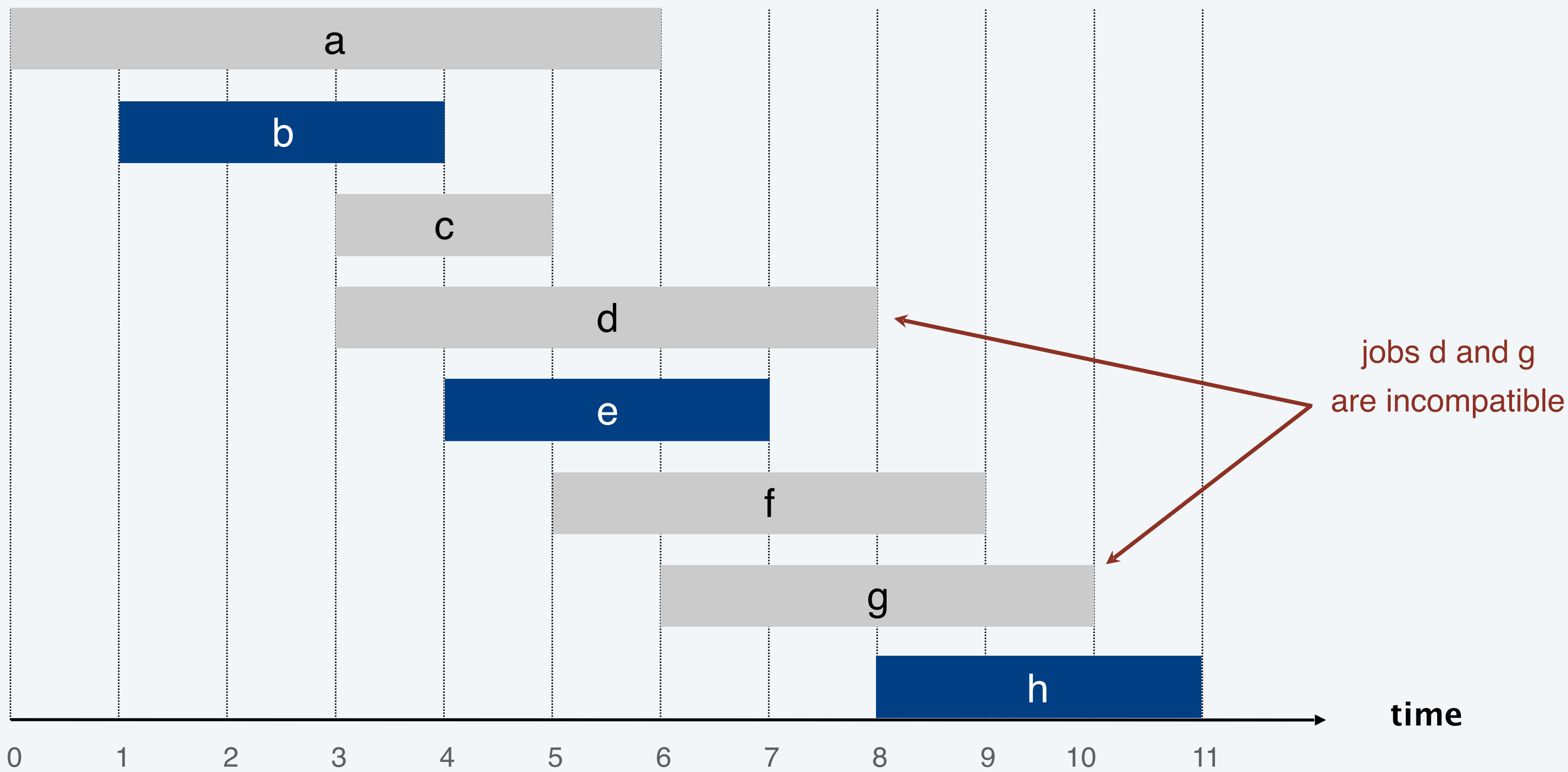


4. GREEDY ALGORITHMS I

- *coin changing*
- *interval scheduling*
- *scheduling to minimize lateness*
- *optimal caching*

Interval scheduling

- Job j starts at s_j and finishes at f_j.
- Two jobs compatible if they don't overlap.
- Goal: find maximum subset of mutually compatible jobs.



Interval scheduling: greedy algorithms

Greedy template. Consider jobs in some natural order.
Take each job provided it's compatible with the ones already taken.

- [Earliest start time] Consider jobs in ascending order of s_j.
- [Earliest finish time] Consider jobs in ascending order of f_j.
- [Shortest interval] Consider jobs in ascending order of $f_j - s_j$.
- [Fewest conflicts] For each job j, count the number of conflicting jobs c_j. Schedule in ascending order of c_j.

Interval scheduling: greedy algorithms

Greedy template. Consider jobs in some natural order.
Take each job provided it's compatible with the ones already taken.

counterexample for earliest start time

counterexample for shortest interval

counterexample for fewest conflicts

Interval scheduling: earliest-finish-time-first algorithm

EARLIEST-FINISH-TIME-FIRST $(n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n)$

SORT jobs by finish time so that $f_1 \leq f_2 \leq \ldots \leq f_n$

$A \leftarrow \varphi$ ⟵ set of jobs selected

FOR $j = 1$ TO n

IF job j is compatible with A

$A \leftarrow A \cup \{ j \}$

RETURN A

Proposition. Can implement earliest-finish-time first in $O(n \log n)$ time.

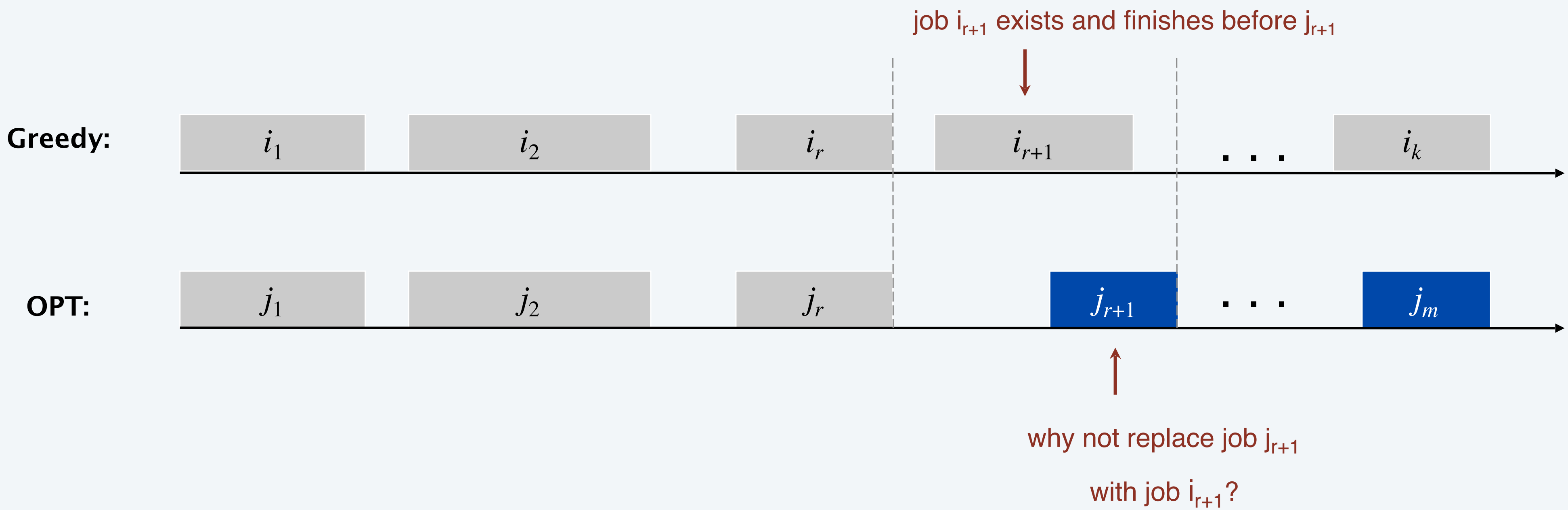
- Keep track of job j^* that was added last to A.
- Job j is compatible with A iff $s_j \geq f_{j^*}$.
- Sorting by finish time takes $O(n \log n)$ time.

Interval scheduling: analysis of earliest-finish-time-first algorithm

Theorem. The earliest-finish-time-first algorithm is optimal.

Pf. [by contradiction]

- Assume greedy is not optimal, and let's see what happens.
- Let i_1, i_2, ... i_k denote set of jobs selected by greedy.
- Let j_1, j_2, ... j_m denote set of jobs in an optimal solution with $i_1 = j_1$, $i_2 = j_2$, ..., $i_r = j_r$ for the largest possible value of r.
- By induction, for all indices r <= k we have $f(i_r) <= f(j_r)$

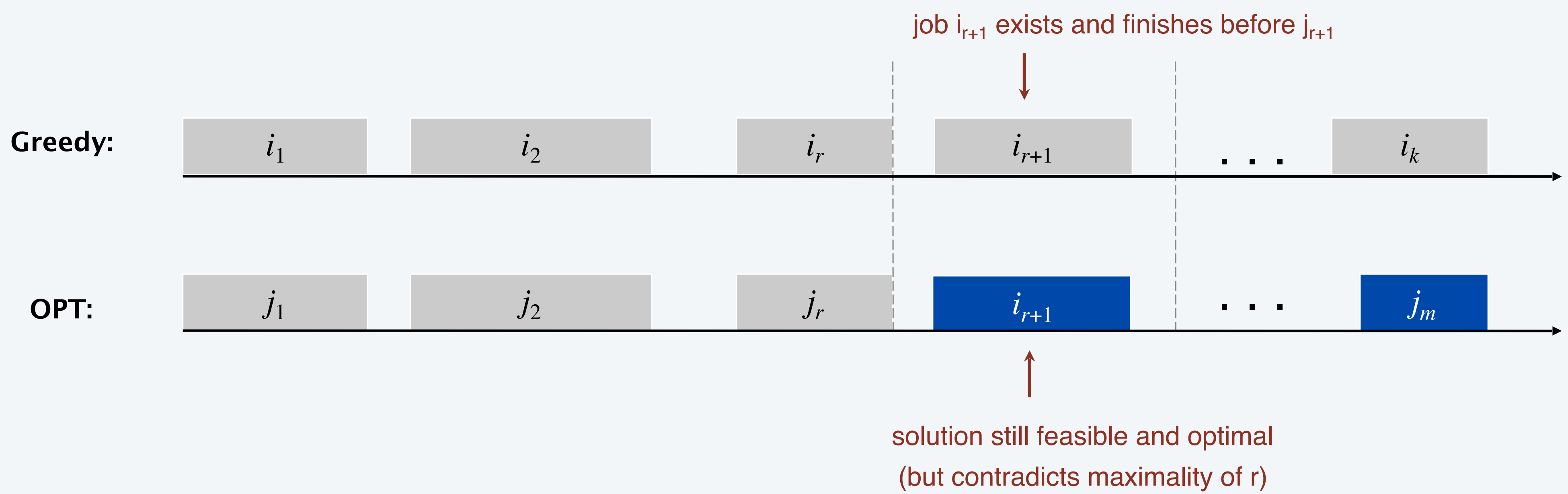


Interval scheduling: analysis of earliest-finish-time-first algorithm

Theorem. The earliest-finish-time-first algorithm is optimal.

Pf. [by contradiction]

- Assume greedy is not optimal, and let's see what happens.
- Let $i_1, i_2, \ldots\ i_k$ denote set of jobs selected by greedy.
- Let $j_1, j_2, \ldots\ j_m$ denote set of jobs in an optimal solution with $i_1 = j_1,\ i_2 = j_2,\ \ldots,\ i_r = j_r$ for the largest possible value of r.

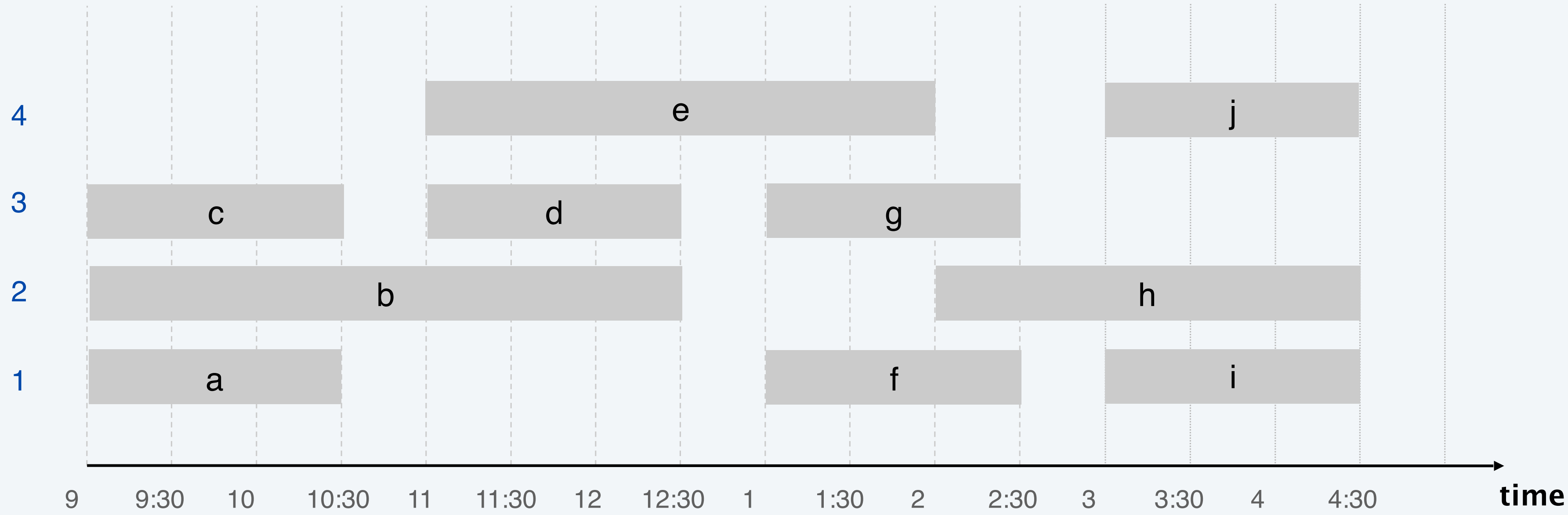


Interval partitioning

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 lectures occur at the same time in the same room.

Ex. This schedule uses 4 classrooms to schedule 10 lectures.

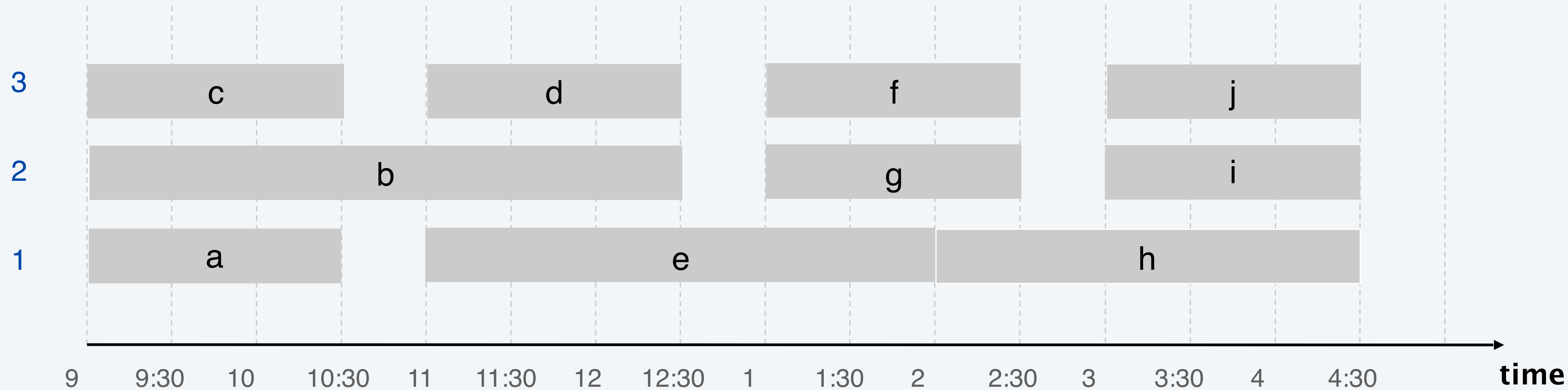


Interval partitioning

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 lectures occur at the same time in the same room.

Ex. This schedule uses 3 classrooms to schedule 10 lectures.



Interval partitioning: greedy algorithms

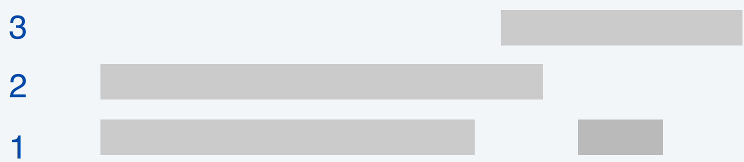
Greedy template. Consider lectures in some natural order.
Assign each lecture to an available classroom (which one?);
allocate a new classroom if none are available.

- [Earliest start time] Consider lectures in ascending order of s_j.
- [Earliest finish time] Consider lectures in ascending order of f_j.
- [Shortest interval] Consider lectures in ascending order of $f_j - s_j$.
- [Fewest conflicts] For each lecture j, count the number of conflicting lectures c_j. Schedule in ascending order of c_j.

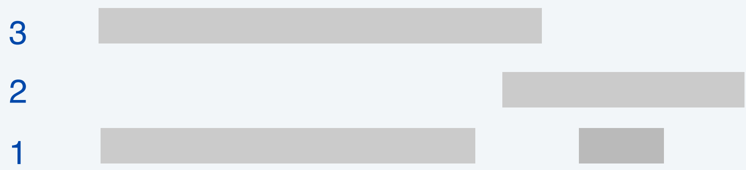
Interval partitioning: greedy algorithms

Greedy template. Consider lectures in some natural order. Assign each lecture to an available classroom (which one?); allocate a new classroom if none are available.

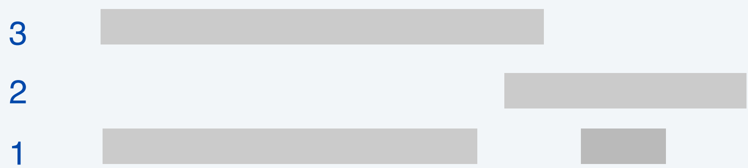
counterexample for earliest finish time



counterexample for shortest interval



counterexample for fewest conflicts



Interval partitioning: earliest-start-time-first algorithm

EARLIEST-START-TIME-FIRST $(n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n)$

SORT lectures by start time so that $s_1 \leq s_2 \leq \ldots \leq s_n$.

$d \leftarrow 0$ ⟵ number of allocated classrooms

FOR $j = 1$ TO n

IF lecture j is compatible with some classroom

Schedule lecture j in any such classroom k.

ELSE

Allocate a new classroom $d + 1$.

Schedule lecture j in classroom $d + 1$.

$d \leftarrow d + 1$

RETURN schedule.

Interval partitioning: earliest-start-time-first algorithm

Proposition. The earliest-start-time-first algorithm can be implemented in $O(n \log n)$ time.

Pf. Store classrooms in a priority queue (key = finish time of its last lecture).

- To determine whether lecture j is compatible with some classroom, compare s_j to key of min classroom k in priority queue.
- To add lecture j to classroom k, increase key of classroom k to f_j.
- Total number of priority queue operations is $O(n)$.
- Sorting by start time takes $O(n \log n)$ time. ▪

Remark. This implementation chooses the classroom k whose finish time of its last lecture is the earliest.

Interval partitioning: lower bound on optimal solution

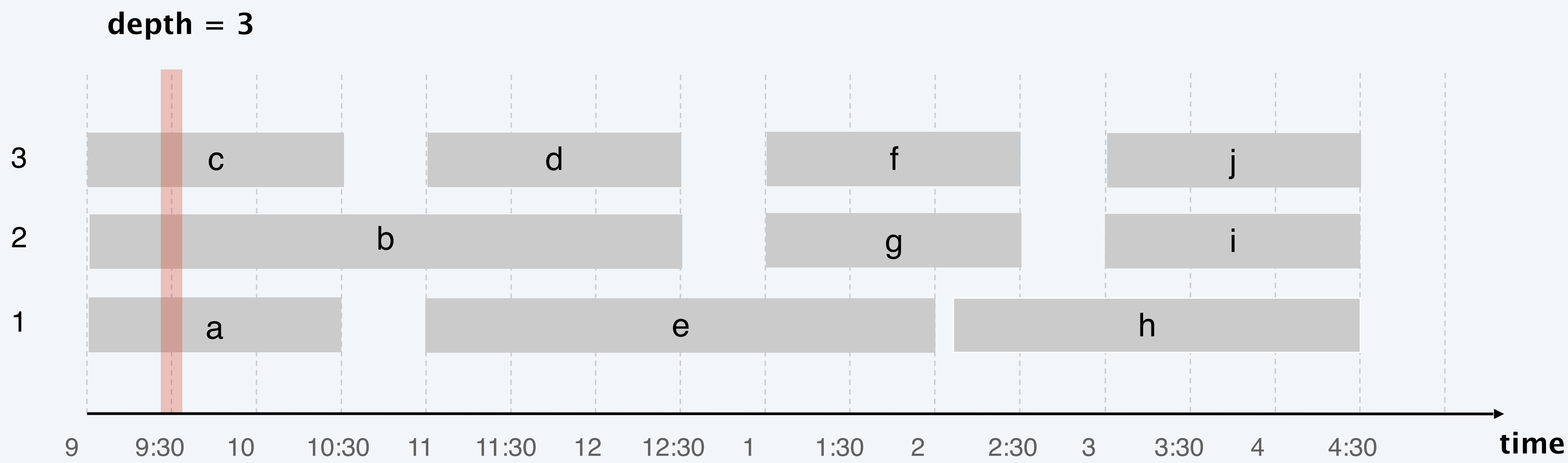
Def. The depth of a set of open intervals is the maximum number that contain any given time.

Key observation. Number of classrooms needed ≥ depth.

Q. Does number of classrooms needed always equal depth?

A. Yes! Moreover, earliest-start-time-first algorithm finds one.

Let's reexamine the original problem sets in previous slide.



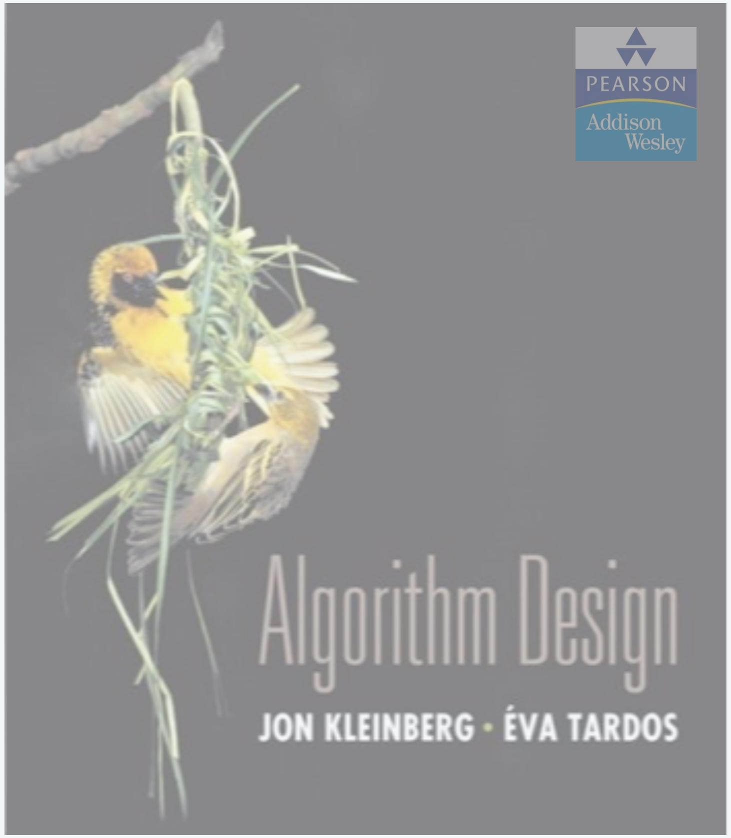
Interval partitioning: analysis of earliest-start-time-first algorithm

Observation. The earliest-start-time first algorithm never schedules two incompatible lectures in the same classroom.

Theorem. Earliest-start-time-first algorithm is optimal.

Pf.

- Let d = number of classrooms that the algorithm allocates.
- Classroom d is opened because we needed to schedule a lecture, say j, that is incompatible with all $d - 1$ other classrooms.
- These d lectures each end after s_j.
- 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 $s_j + \varepsilon$.
- Key observation $\Rightarrow$ all schedules use $\geq d$ classrooms. ▪



4. Greedy Algorithms I

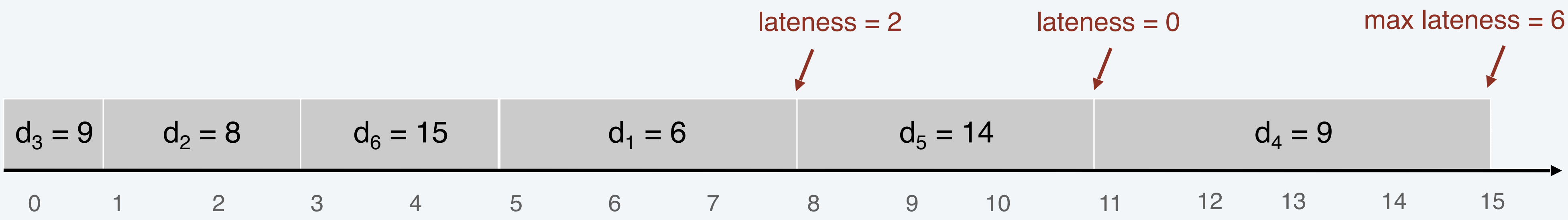
- *coin changing*
- *interval scheduling*
- *scheduling to minimize lateness*
- *optimal caching*

Scheduling to minimizing lateness

Minimizing lateness problem.

- Single resource processes one job at a time.
- Job j requires t_j units of processing time and is due at time d_j.
- 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 $L = \max_j \ell_j$.

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



Minimizing lateness: greedy algorithms

Greedy template. Schedule jobs according to some natural order.

- [Shortest processing time first] Schedule jobs in ascending order of processing time t_j.

- [Earliest deadline first] Schedule jobs in ascending order of deadline d_j.

- [Smallest slack] Schedule jobs in ascending order of slack $d_j - t_j$.

Minimizing lateness: greedy algorithms

Greedy template. Schedule jobs according to some natural order.

- [Shortest processing time first] Schedule jobs in ascending order of processing time t_j.

	1	2
t_j	1	10
d_j	100	10

counterexample

- [Smallest slack] Schedule jobs in ascending order of slack $d_j - t_j$.

	1	2
t_j	1	10
d_j	2	10

counterexample

Minimizing lateness: earliest deadline first

EARLIEST-DEADLINE-FIRST $(n, t_1, t_2, \ldots, t_n, d_1, d_2, \ldots, d_n)$

SORT n jobs so that $d_1 \leq d_2 \leq \ldots \leq d_n$.

$t \leftarrow 0$

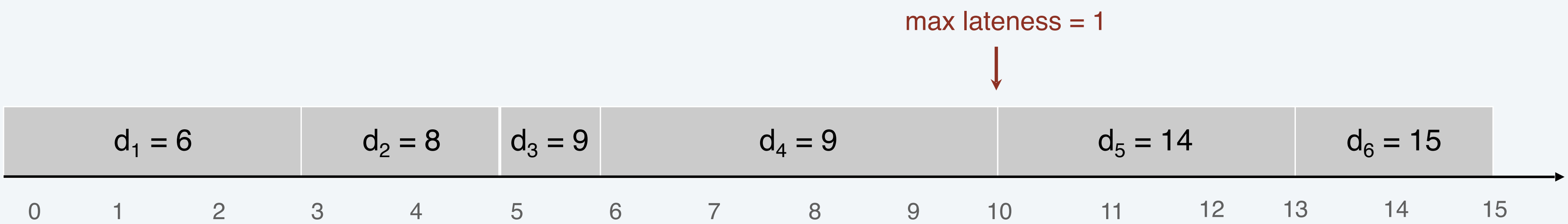
FOR $j = 1$ TO n

  Assign job j to interval $[t, t + t_j]$.

  $s_j \leftarrow t$; $f_j \leftarrow t + t_j$

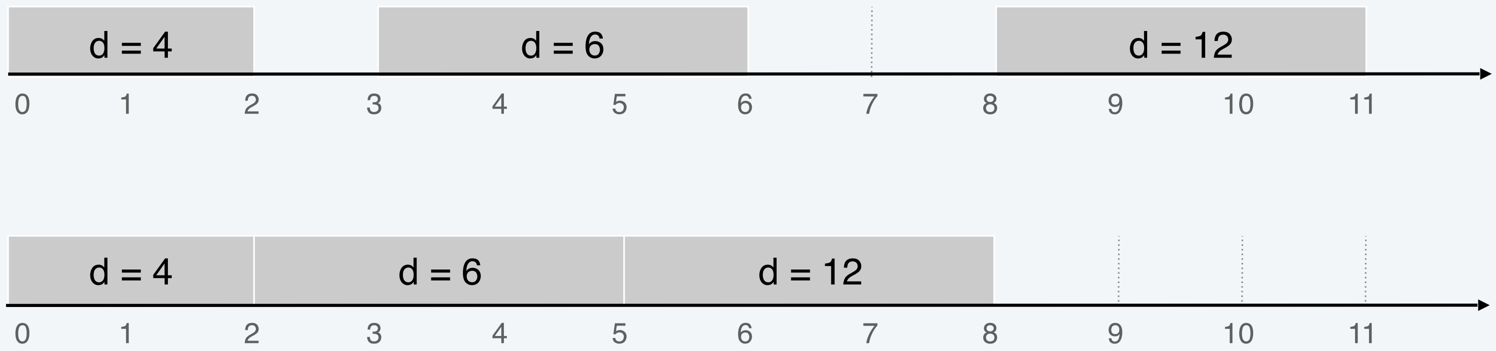
  $t \leftarrow t + t_j$

RETURN intervals $[s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n]$.



Minimizing lateness: no idle time

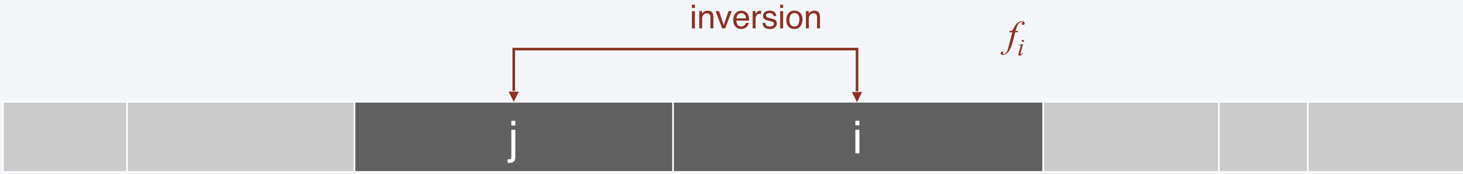
Observation 1. There exists an optimal schedule with no idle time.



Observation 2. The earliest-deadline-first schedule has no idle time.

Minimizing lateness: inversions

Def. Given a schedule S, an inversion is a pair of jobs i and j such that:
$i < j$ but j scheduled before i.



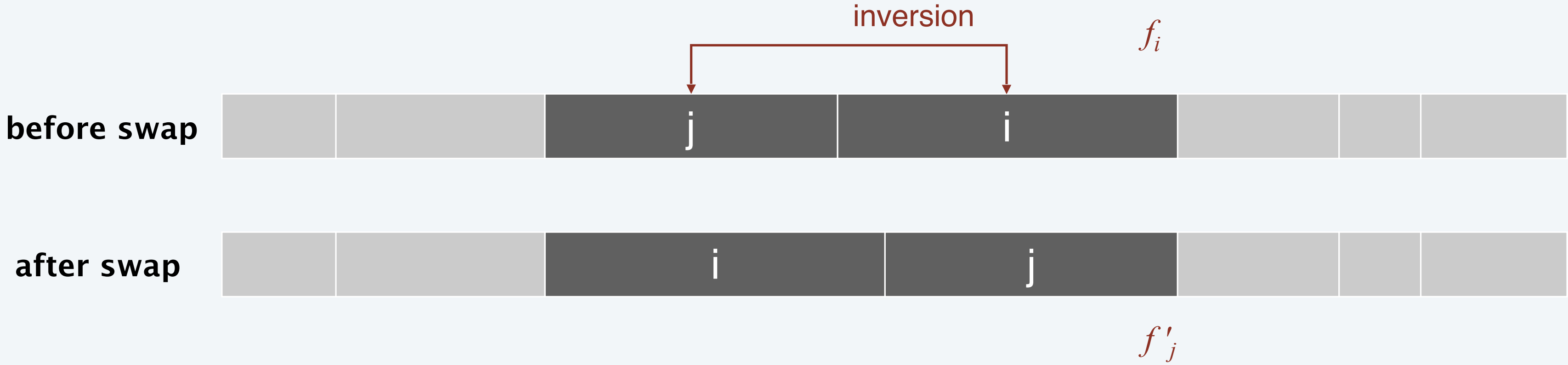
[as before, we assume jobs are numbered so that $d_1 \le d_2 \le \ldots \le d_n$]

Observation 3. The earliest-deadline-first schedule has no inversions.

Observation 4. If a schedule (with no idle time) has an inversion,
it has one with a pair of inverted jobs scheduled consecutively.

Minimizing lateness: inversions

Def. Given a schedule S, an inversion is a pair of jobs i and j such that: $i < j$ but j scheduled before i.



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 $k \neq i, j$. (No idle time)
- $\ell_i \leq \ell_i$.
- If job j is late, ℓ_j

$$
\begin{aligned}
\ell_j &= f'_j - d_j && \text{(definition)} \\
&= f_i - d_j && (j \text{ now finishes at time } f_i) \\
&\leq f_i - d_i && (\text{since } i \text{ and } j \text{ inverted}) \\
&\leq \ell_i\,. && \text{(definition)}
\end{aligned}
$$

Minimizing lateness: analysis of earliest-deadline-first algorithm

Theorem. The earliest-deadline-first schedule S is optimal.

Pf. [by contradiction]

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 $i\text{–}j$ be an adjacent inversion.
- Swapping i and j
  - does not increase the max lateness
  - strictly decreases the number of inversions
- This contradicts definition of S^* ▪

Greedy analysis strategies

Greedy algorithm stays ahead. Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's.

Structural. Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound.

Exchange argument. Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.

Other greedy algorithms. Gale-Shapley, Kruskal, Prim, Dijkstra, Huffman, …