

CSE 421 Winter 2025

Lecture 6: Greedy

Nathan Brunelle

<http://www.cs.uw.edu/421>

Greedy Algorithms

Hard to define exactly but can give general properties

- Solution is built in small steps
- Decisions on how to build the solution are made to maximize some criterion without looking to the future
 - Want the 'best' current partial solution as if the current step were the last step

May be more than one greedy algorithm using different criteria to solve a given problem

- Not obvious which criteria will actually work

Greedy Algorithms

- Greedy algorithms
 - Easy to describe
 - Fast running times
 - Work only on certain classes of problems
 - Hard part is showing that they are correct
- Focus on methods for proving that greedy algorithms do work

Interval Scheduling

Interval Scheduling:

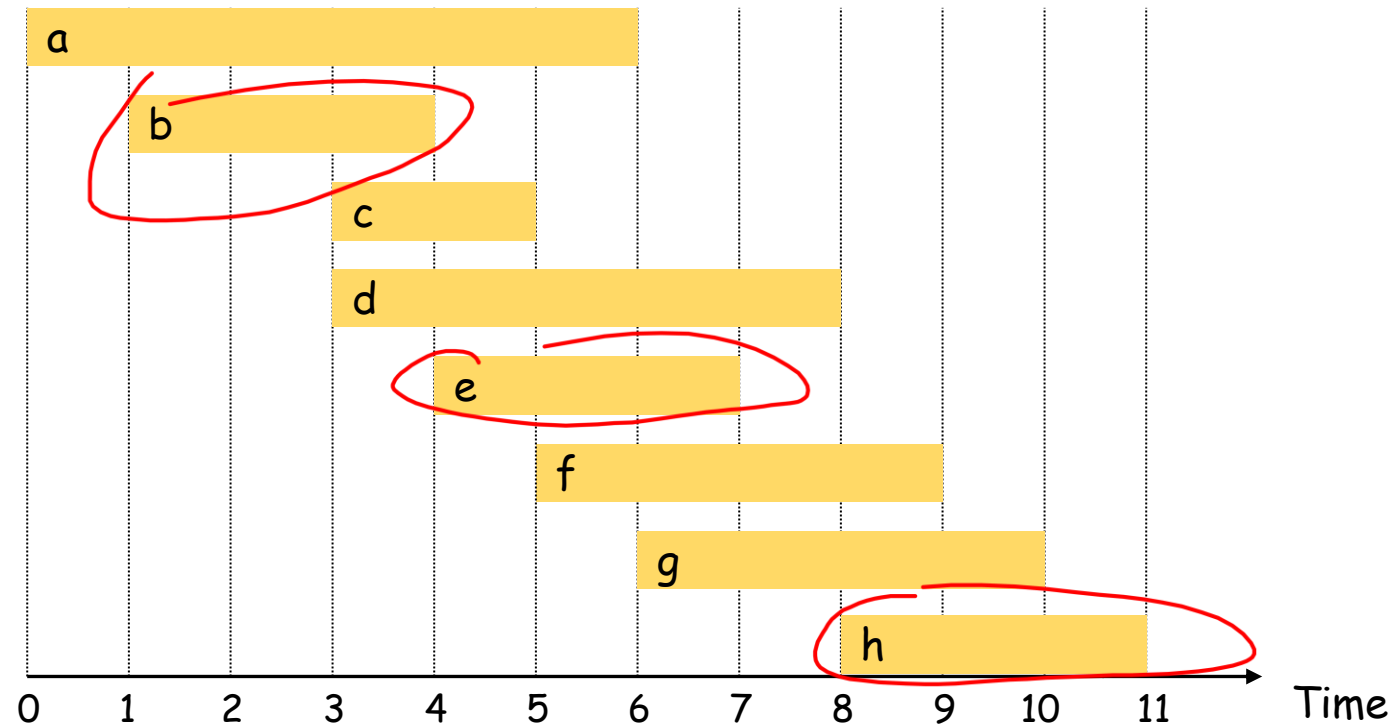
- Single resource
- Reservation requests of form:
“Can I reserve it from start time s to finish time f ?”
 $s < f$



Interval Scheduling

Interval scheduling:

- Job j starts at s_j and finishes at $f_j > s_j$.
- Two jobs i and j are **compatible** if they don't overlap: $f_i \leq s_j$ or $f_j \leq s_i$
- **Goal:** find maximum size subset of mutually compatible jobs.



Greedy Algorithms for Interval Scheduling

- What criterion should we try?

- Shortest interval first

- Earliest ending time

- ~~Maximize the longest free time~~

- Fewest conflicts first

- ~~Pick from the hour with the most concurrent intervals~~

- Earliest start time

Greedy Algorithms for Interval Scheduling

- What criterion should we try?

- Earliest start time s_i



- Shortest request time $f_i - s_i$



- Fewest conflicts



Greedy Algorithms for Interval Scheduling

- What criterion should we try?

- Earliest start time s_i
 - Doesn't work

- Shortest request time $f_i - s_i$
 - Doesn't work

- Fewest conflicts
 - Doesn't work

- Earliest finish time f_i
 - Works!



Greedy (by finish time) Algorithm for Interval Scheduling

R = set of all requests

A = \emptyset

while $R \neq \emptyset$ do:

Choose request $i \in R$ with smallest finish time f_i ←

Add request i to A

Delete all requests in R not compatible with request i

return A

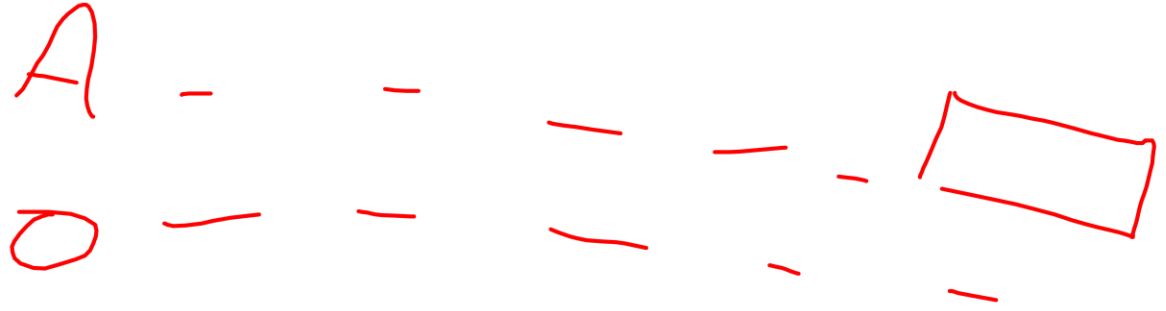
Greedy Analysis Strategies

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

For interval scheduling: Show that after the greedy algorithm selects each interval, any alternative schedule's selection would have also been non-conflicting.

Conclusion: Each choice from the alternative selections can be swapped with a greedy choice, making greedy no worse off.

Interval Scheduling: Analysis



Claim: A is a compatible set of requests and requests are added to A in order of finish time

- When we add a request to A we delete all incompatible ones from R

Name the finish times of requests in A as a_1, a_2, \dots, a_t in order.



Claim: Let $O \subseteq R$ be a set of compatible requests whose finish times in order are o_1, o_2, \dots, o_s . Then for every integer $k \geq 1$ we have:

- if O contains a k^{th} request then A does too, and
- $a_k \leq o_k$ " A is ahead of O "

Note that a) alone implies that $t \geq s$ which means that A is optimal but we also need b) "stays ahead" to keep the induction going.

Inductive Proof of Claim

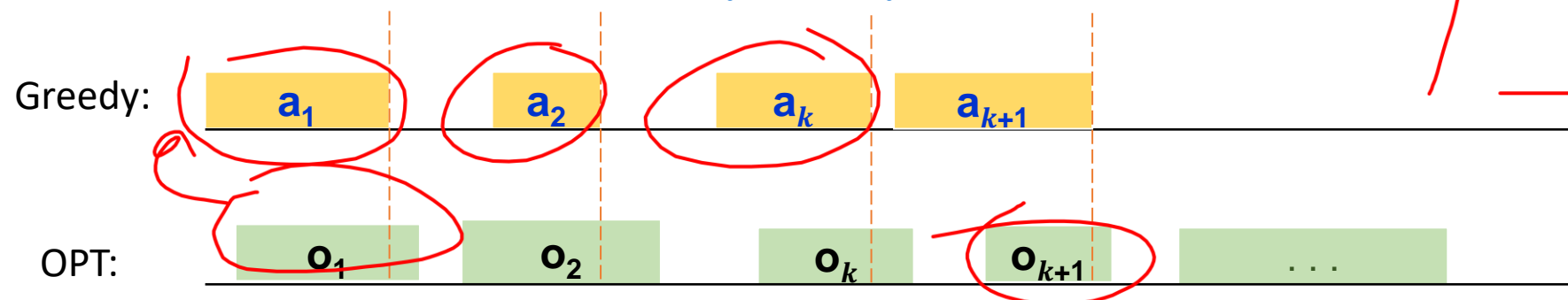
Base Case $k = 1$: A includes the request with smallest finish time, so if O is not empty then $a_1 \leq o_1$

Inductive Step: Suppose that $a_k \leq o_k$ and there is a $(k+1)^{\text{st}}$ request in O .

Then $(k+1)^{\text{st}}$ request in O is compatible with a_1, a_2, \dots, a_k since $a_k \leq o_k$ and $o_k \leq$ start time of $(k+1)^{\text{st}}$ request in O whose finish time is o_{k+1}

\Rightarrow There is a $(k+1)^{\text{st}}$ request in A whose finish time is named a_{k+1} .

Also, since A would have considered both requests and chosen the one with the earlier finish time, $a_{k+1} \leq o_{k+1}$.



Interval Scheduling: Greedy Algorithm Implementation

Sort jobs by finish times so that $0 \leq f_1 \leq f_2 \leq \dots \leq f_n$.

```
A =  $\phi$ 
last = 0
for j = 1 to n {
    if (last  $\leq$  sj)
        A = A  $\cup$  {j}
        last = fj
}
return A
```

$O(n \log n)$

$O(n)$

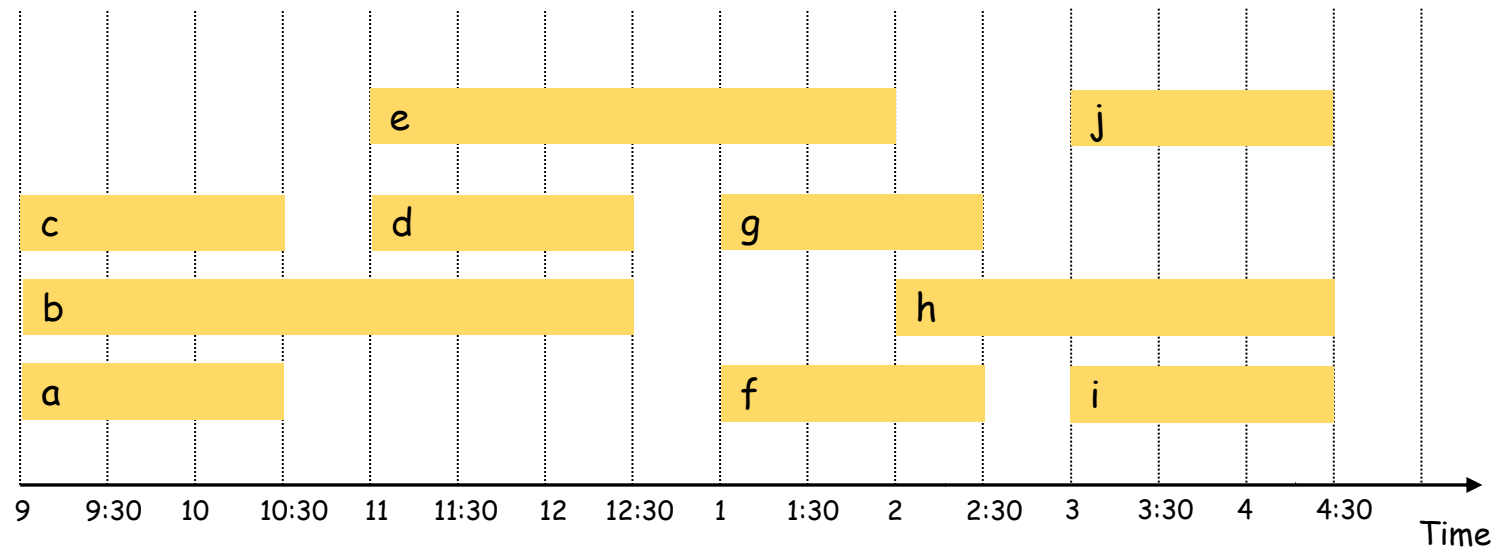
Scheduling All Intervals: Interval Partitioning

Interval Partitioning:

- Lecture j starts at s_j and finishes at f_j .

Goal: find **minimum number of rooms** to schedule all lectures so that no two occur at the same time in the same room.

Example: This schedule uses 4 rooms to schedule 10 lectures.



Can you do better?

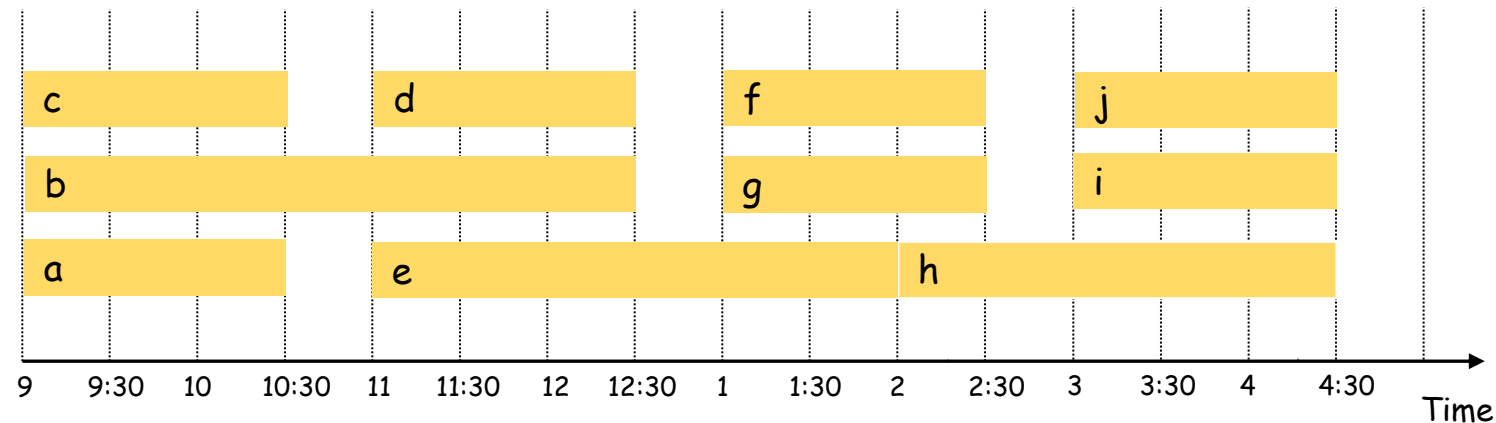
Scheduling All Intervals: Interval Partitioning

Interval Partitioning:

- Lecture j starts at s_j and finishes at f_j .

Goal: find **minimum number of rooms** to schedule all lectures so that no two occur at the same time in the same room.

Example: This schedule uses only 3 rooms.

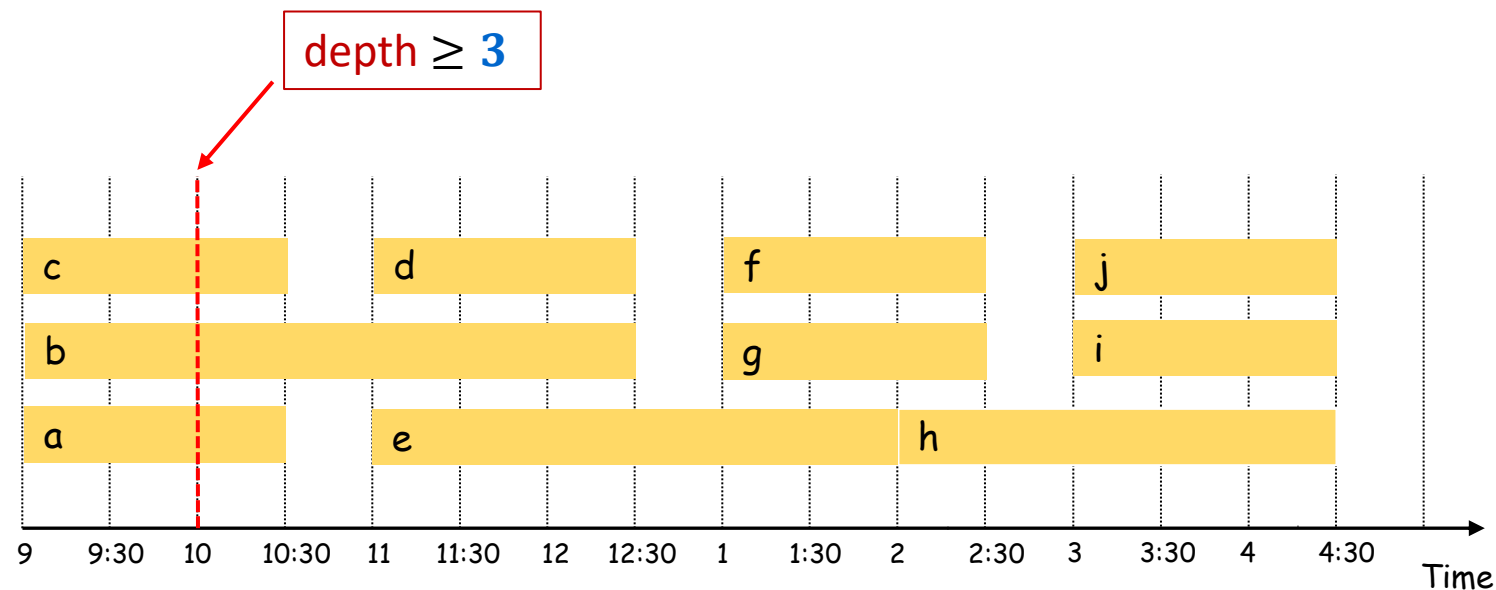


Scheduling All Intervals: Interval Partitioning

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

Key observation: # of rooms needed \geq depth.

Example: This schedule uses only **3** rooms. Since depth \geq **3** this is optimal.



A simple greedy algorithm

Sort requests in increasing order of start times $(s_1, f_1), \dots, (s_n, f_n)$

$last_1 = 0$ // finish time of last request currently scheduled in room **1**

for $i = 1$ to n {

$j = 1$

 while (request i not scheduled) {

 if $s_i \geq last_j$ then

 schedule request i in room j

$last_j = f_i$

$j = j + 1$

 if $last_j$ undefined then $last_j = 0$

 }

}

Look for the first room where the request will fit, opening a new room if all the others used so far are full.

Interval Partitioning: Greedy Analysis

Observation: Greedy algorithm never schedules two incompatible lectures in the same room

- Only schedules request i in room j if $s_i \geq last_j$

Theorem: Greedy algorithm is optimal.

Proof:

Let d = number of rooms that the greedy algorithm allocates.

- Room d is allocated because we needed to schedule a request, say j , that is incompatible with some request in each of the other $d - 1$ rooms.
- Since we sorted by start time, these incompatibilities are caused by requests that start no later than s_j and finish after s_j .

So... we have d requests overlapping at time $s_j + \epsilon$ for some (maybe tiny) $\epsilon > 0$.

Key observation \Rightarrow all schedules use $\geq d$ rooms.

A simple greedy algorithm

Sort requests in increasing order of start times $(s_1, f_1), \dots, (s_n, f_n)$

$last_1 = 0$ // finish time of last request currently scheduled in room 1

for $i = 1$ to n {

$j \leftarrow 1$

 while (request i not scheduled) {

 if $s_i \geq last_j$ then

 schedule request i in room j

$last_j = f_i$

$j = j + 1$

 if $last_j$ undefined then $last_j = 0$

 }

}

Runtime analysis

$O(n \log n)$

Might need to try all d rooms to schedule a request

$O(n d)$

d might be as big as n

Worst case $\Theta(n^2)$

A more efficient implementation: Priority queue

$O(n \log n)$

Sort requests in increasing order of start times $(s_1, f_1), \dots, (s_n, f_n)$

$d = 1$

schedule request **1** in room **1**

$last_1 = f_1$ $O(1)$

insert **1** into priority queue Q with key = $last_1$

for $i = 2$ to n {

$j = \text{findmin}(Q)$

 if $s_i \geq last_j$ then { $O(\log d)$

 schedule request i in room j

$last_j = f_i$

 increasekey(j, Q) to $last_j$ }

 else {

$d = d + 1$ $O(\log d)$

 schedule request i in room d

$last_d = f_i$

 insert d into priority queue Q with key = $last_d$ }

}

$O(n \log d)$

$\Theta(n \log n)$ total

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.

Scheduling to Minimize Lateness

Scheduling to minimize lateness:

- Single resource as in interval scheduling but, instead of start and finish times, request i has
 - Time requirement t_i which must be scheduled in a contiguous block
 - Target deadline d_i by which time the request would like to be finished
- Overall start time s for all jobs

Requests are scheduled by the algorithm into time intervals $[s_i, f_i]$ s.t. $t_i = f_i - s_i$

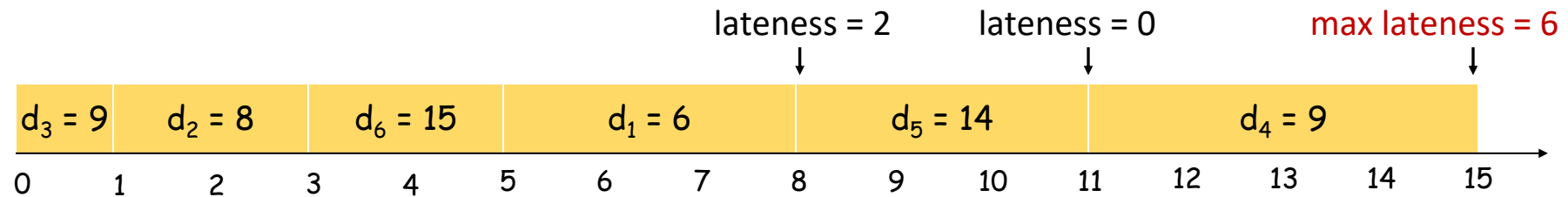
- Lateness of schedule for request i is
 - If $f_i > d_i$ then request i is late by $L_i = f_i - d_i$; otherwise its lateness $L_i = 0$
- **Maximum lateness** $L = \max_i L_i$

Goal: Find a schedule for **all** requests (values of s_i and f_i for each request i) to minimize the **maximum lateness**, L .

Scheduling to Minimizing Lateness

- Example:

	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: Consider jobs in some order.

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

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

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

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 .

	1	2
t_j	1	10
d_j	100	10

counterexample

Will schedule 1 (length 1) before 2 (length 10).
2 can only be scheduled at time 1
1 will finish at time 11 > 10. Lateness 1.
Lateness 0 possible if 1 goes last.

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

	1	2
t_j	1	10
d_j	2	10

counterexample

Will schedule 2 (slack 0) before 1 (slack 1).
1 can only be scheduled at time 10
1 will finish at time 11 > 10. Lateness 9.
Lateness 1 possible if 1 goes first.

Minimizing Lateness: Greedy Algorithms

Greedy template: Consider jobs in some order.

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

Greedy Algorithm: Earliest Deadline First

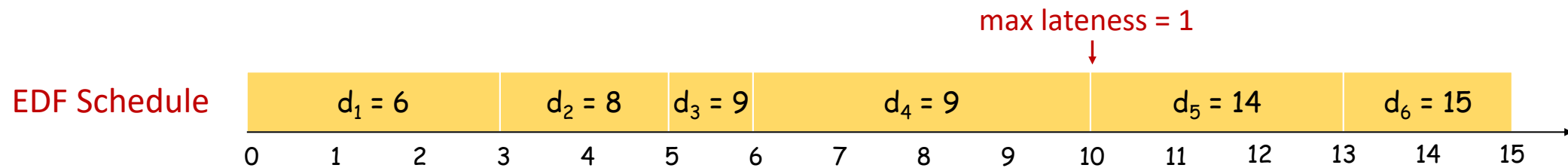
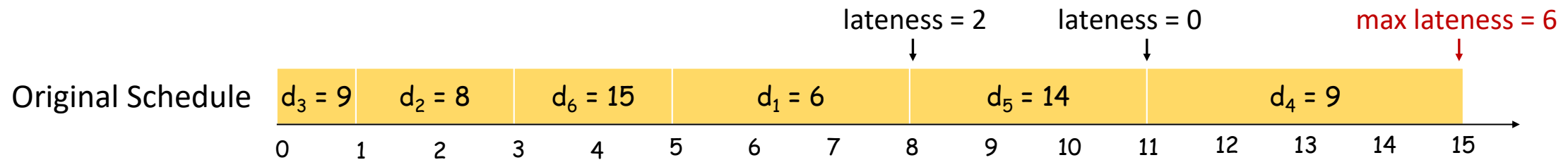
Consider requests in increasing order of deadlines

Schedule the request with the earliest deadline as soon as the resource is available

Scheduling to Minimizing Lateness

- Example:

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



Proof for Greedy EDF Algorithm: Exchange Argument

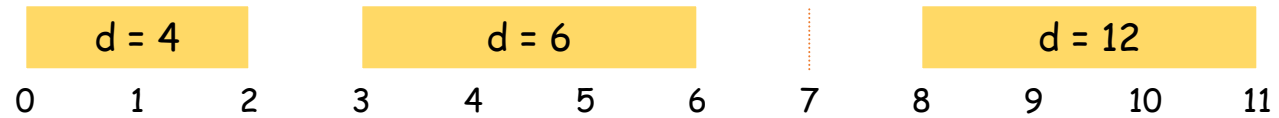
Show that if there is another schedule **O** (think optimal schedule) then we can gradually change **O** so that...

- at each step the maximum lateness in **O** never gets worse
- it eventually becomes the same cost as **A**

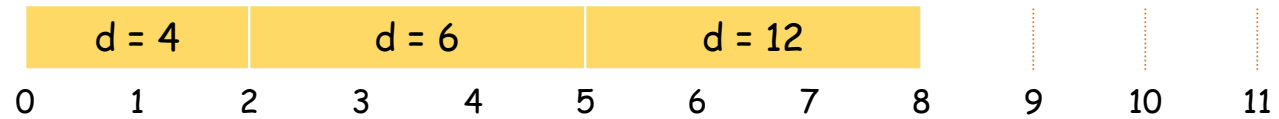
This means that **A** is at least as good as **O**, so **A** is also optimal!

Minimizing Lateness: No Idle Time

Observation: There exists an optimal schedule with no **idle time**



At least as good



Observation: The greedy EDF schedule has no idle time.

Minimizing Lateness: Inversions

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



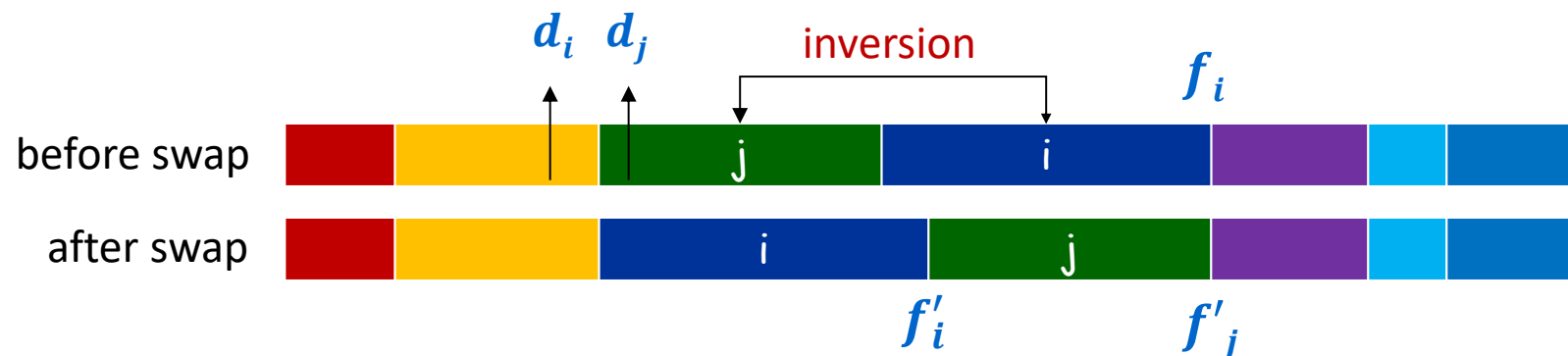
Observation: Greedy EDF schedule has no inversions.

Observation: If schedule S (with no idle time) has an inversion
it has two adjacent jobs that are inverted

- Any job in between would be inverted w.r.t. one of the two ends

Minimizing Lateness: Inversions

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

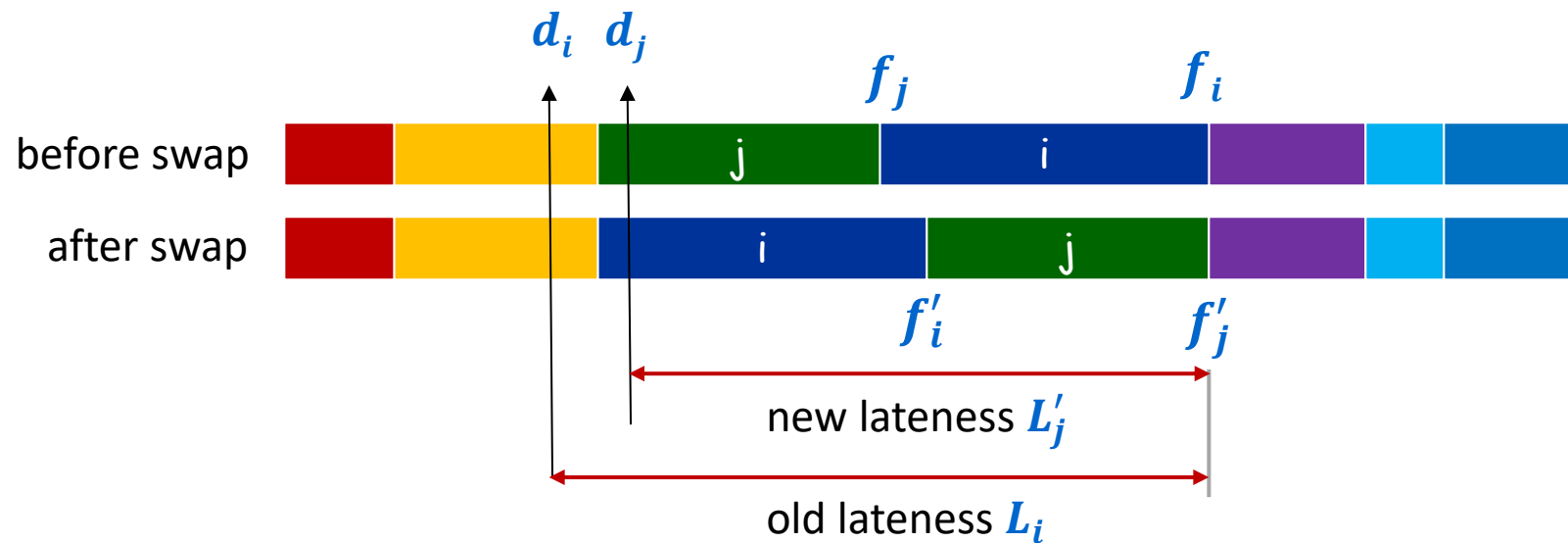


Claim: Swapping two adjacent, inverted jobs

- reduces the # of inversions by **1**
- does not increase the max lateness.

Minimizing Lateness: Inversions

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



Claim: Maximum lateness does not increase

Optimal schedules and inversions

Claim: There is an optimal schedule with no idle time and no inversions

Proof:

By previous argument there is an optimal schedule \mathcal{O} with no idle time

If \mathcal{O} has an inversion then it has an **adjacent** pair of requests in its schedule that are inverted and can be swapped without increasing lateness

... we just need to show one more claim that eventually this swapping stops

Optimal schedules and inversions

Claim: Eventually these swaps will produce an optimal schedule with no inversions.

Proof:

Each swap decreases the # of inversions by **1**

There are a bounded # of inversions possible in the worst case

- at most $n(n - 1)/2$ but we only care that this is finite.

The # of inversions can't be negative so this must stop.

Idleness and Inversions are the only issue

Claim: All schedules with no inversions and no idle time have the same maximum lateness.

Proof:

Schedules can differ only in how they order requests with equal deadlines

Consider all requests having some common deadline d .

- Maximum lateness of these jobs is based only on finish time of the last one ... and the set of these requests occupies the same time segment in both schedules.

⇒ The last of these requests finishes at the same time in any such schedule.

Earliest Deadline First is optimal

We know that

- There is an optimal schedule with no idle time or inversions
- All schedules with no idle time or inversions have the same maximum lateness
- EDF produces a schedule with no idle time or inversions

So ...

- EDF produces an optimal schedule

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.