# 4.2 Scheduling to Minimize Maximum Lateness

# Scheduling to Minimizing Maximum Lateness

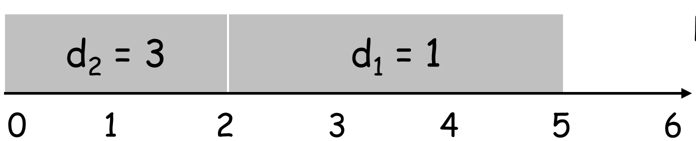
## Minimizing lateness problem.

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

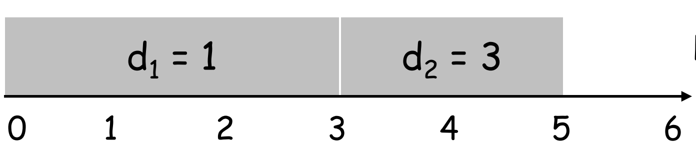
- If j starts at time $s_j$, it finishes at time $f_j = s_j + t_j$.
- Lateness: $\ell_j = \max \{ 0, \; f_j - d_j \}$.
- Goal: schedule all jobs to minimize maximum lateness $L = \max \ell_j$.

Ex:

| | 1 | 2 | job number |
|---|---|---|---|
| $t_j$ | 3 | 2 | time required |
| $d_j$ | 1 | 3 | deadline |




lateness = 4




lateness = 2

Q. Which schedule is better?

# Scheduling to Minimizing Maximum Lateness

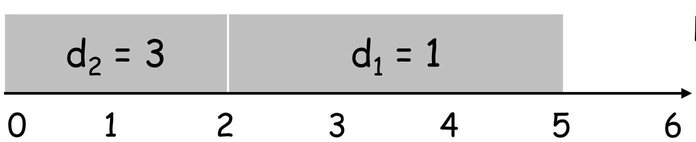
## Minimizing lateness problem.

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

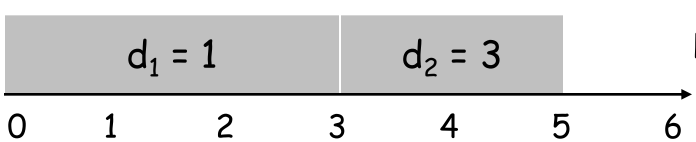
- If j starts at time $s_j$, it finishes at time $f_j = s_j + t_j$.
- Lateness: $\ell_j = \max \{ 0, f_j - d_j \}$.
- Goal: schedule all jobs to minimize maximum lateness $L = \max \ell_j$.

Ex:

| | 1 | 2 | job number |
|---|---|---|---|
| $t_j$ | 3 | 2 | time required |
| $d_j$ | 1 | 3 | deadline |




lateness = 4




lateness = 2

Q. Which schedule is better?
A. The lower; lateness 2.

# Scheduling to Minimizing Maximum Lateness

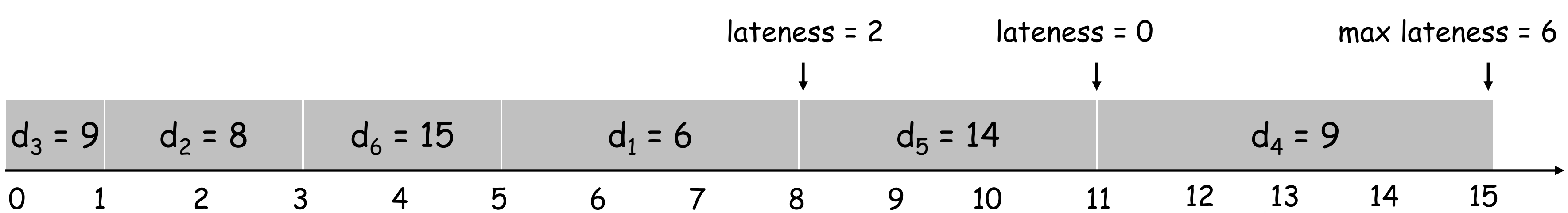
## Minimizing lateness problem.

- 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 $L = \max \ell_j$.

Ex:

| | 1 | 2 | 3 | 4 | 5 | 6 | ← job number |
|---|---|---|---|---|---|---|---|
| $t_j$ | 3 | 2 | 1 | 4 | 3 | 2 | ← time required |
| $d_j$ | 6 | 8 | 9 | 9 | 14 | 15 | ← deadline |



## Minimizing Maximum Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time $t_j$ (least work first).

- [Earliest deadline first] Consider jobs in ascending order of deadline $d_j$ (nearest deadline).

- [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$ (least time to start to make deadline).

Q. Which one do you think may work? (1 min)

# Minimizing Maximum Lateness: Greedy Algorithms

**Greedy template.** Consider jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time $t_j$ (least work first).

| | 1 | 2 |
|---|---|---|
| $t_j$ | 1 | 10 |
| $d_j$ | 100 | 10 |

counterexample

- [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$ (least time to start to make deadline).

| | 1 | 2 |
|---|---|---|
| $t_j$ | 1 | 10 |
| $d_j$ | 2 | 10 |

counterexample

# Minimizing Maximum Lateness: Greedy Algorithm

**Greedy algorithm.** Earliest deadline first.

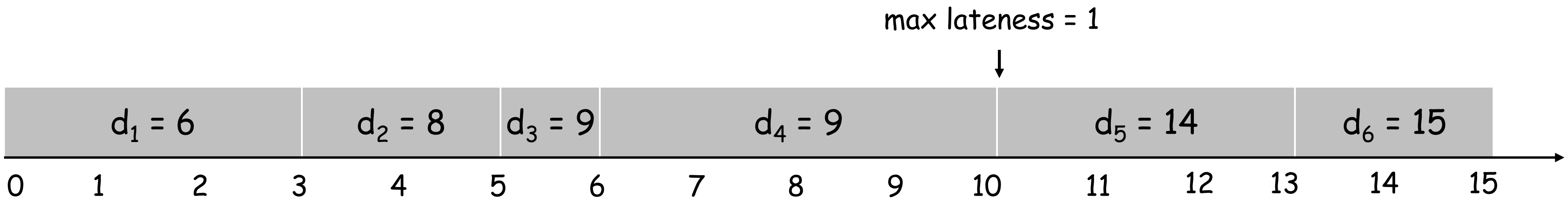
```
Sort n jobs by deadline so that d₁ ≤ d₂ ≤ … ≤ dₙ

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]
```

TU Delft

# Minimizing Maximum Lateness: Greedy Algorithm

**Greedy algorithm.** Earliest deadline first.

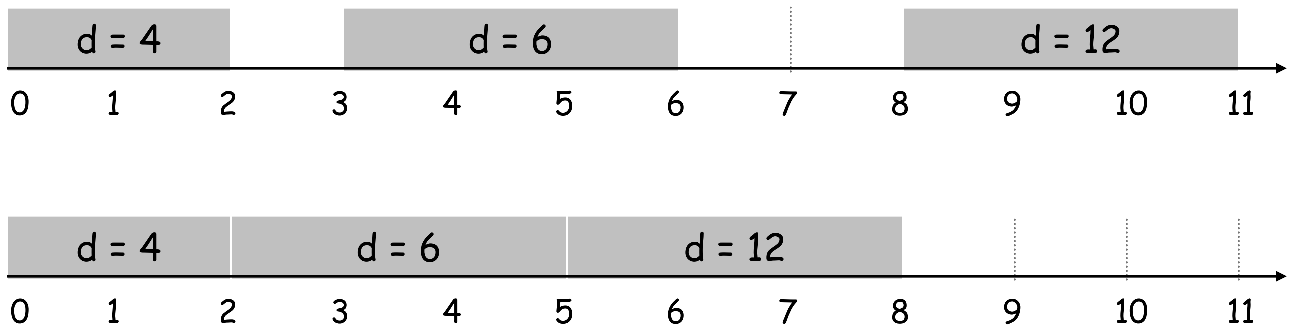
| | 1 | 2 | 3 | 4 | 5 | 6 | ← job number |
|---|---|---|---|---|---|---|---|
| $t_j$ | 3 | 2 | 1 | 4 | 3 | 2 | ← time required |
| $d_j$ | 6 | 8 | 9 | 9 | 14 | 15 | ← deadline |




**Observation.** The greedy schedule has no idle time.

# Minimizing Maximum Lateness: No Idle Time

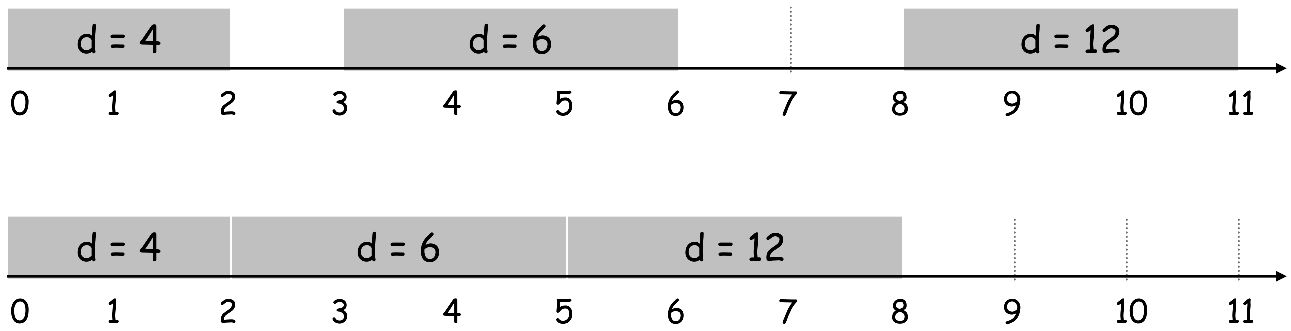
Observation. There exists an optimal schedule with no idle time.




Q*. How to prove that earliest-deadline-first greedy algorithm is optimal?

# Minimizing Maximum Lateness: No Idle Time

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



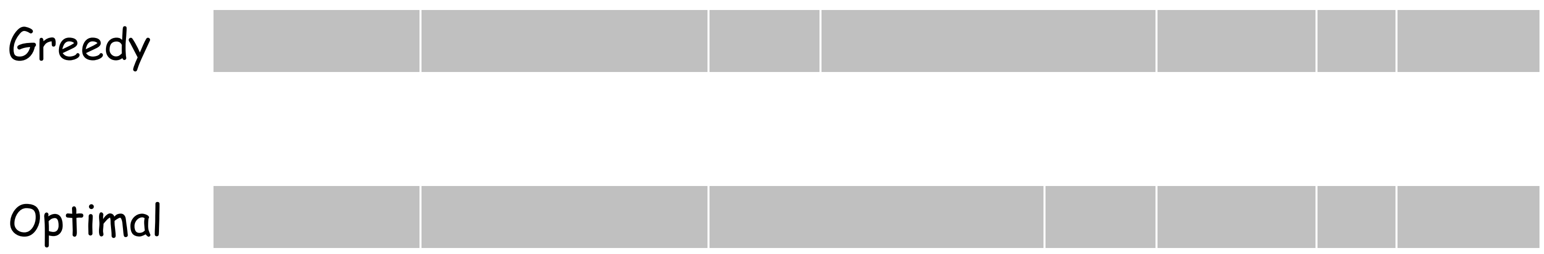

Q*. How to prove that earliest-deadline-first greedy algorithm is optimal?

A. Idea of proof: exchange argument:

- Take an optimal schedule.
- Change into greedy schedule without losing optimality…. but how?

# Towards proving greedy is optimal

Idea. Change optimal schedule to greedy without losing optimality.



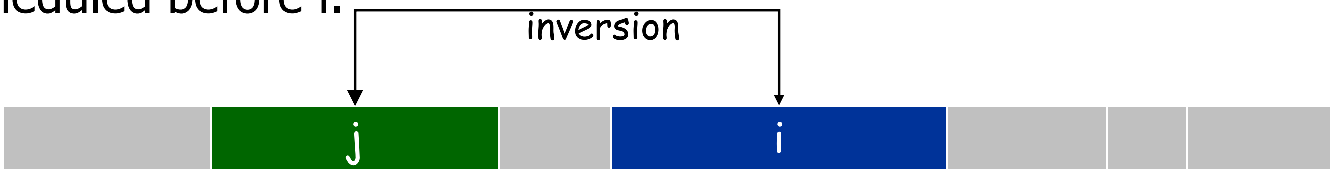

Problems to solve first:

- What do we know about the greedy schedule?
- How can we change the optimal to be more like that without losing optimality?

# Minimizing Maximum 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.

before swap

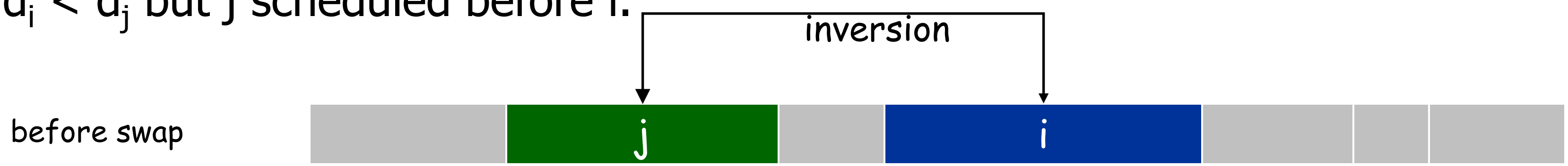



*We'll now study a number of properties of such an inversion.*

Q. How many inversions can a schedule from our Greedy algorithm have?
(0, 1, or more than 1)

# Minimizing Maximum 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.



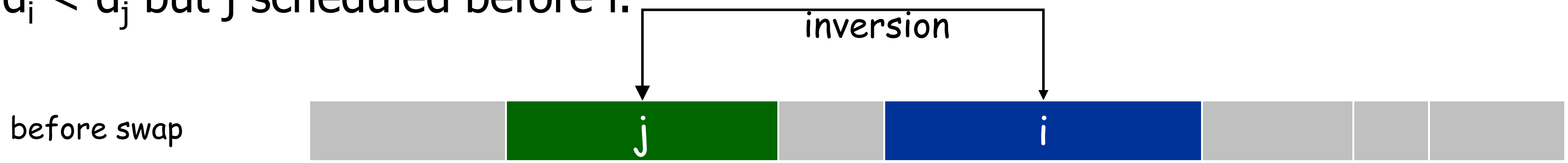

Observation. Greedy schedule has no inversions.

Q. What is the difference in maximum lateness between two schedules without inversions and without idle time? (1 min)

TU Delft

# Minimizing Maximum 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.




**Observation.** Greedy schedule has no inversions.

**Q.** What is the difference in maximum lateness between two schedules without inversions and without idle time?

**Pf.**

Only difference is an "inversion" of i and j with equal deadline ($d_i$=$d_j$).

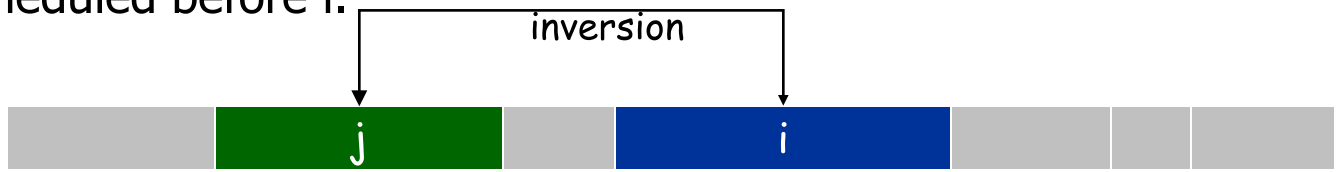
Maximum lateness of i and j is only influenced by last job ($f_i$ - $d_i$).

Maximum lateness of i and j is the same if i and j are swapped.

# Minimizing Maximum 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.

before swap



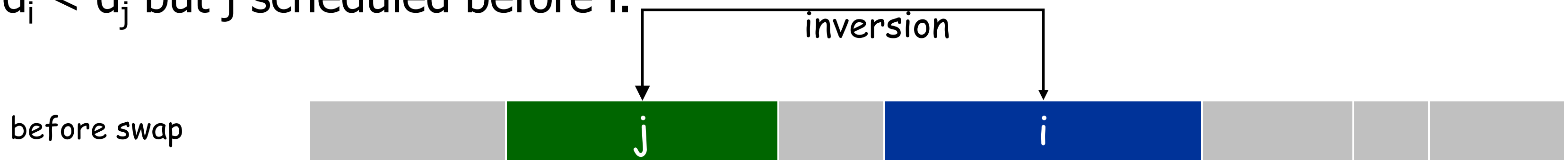

Observation. Greedy schedule has no inversions.

Observation. All schedules without inversions have same lateness.

Q. If a schedule (with no idle time) has an inversion, how can we find it?

# Minimizing Maximum 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.




Observation. Greedy schedule has no inversions.

Observation. All schedules without inversions have same lateness.

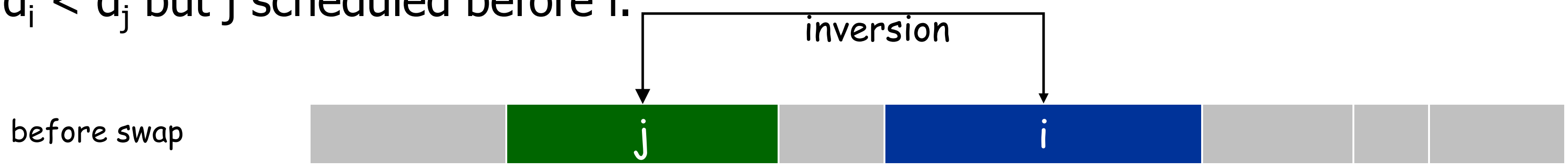
Observation. If a schedule (with no idle time) has an inversion, then it has one with a pair of inverted jobs scheduled consecutively.

Pf.

Q. How do we proof "If …, then …" ?

# Minimizing Maximum 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.




Observation. Greedy schedule has no inversions.

Observation. All schedules without inversions have same lateness.

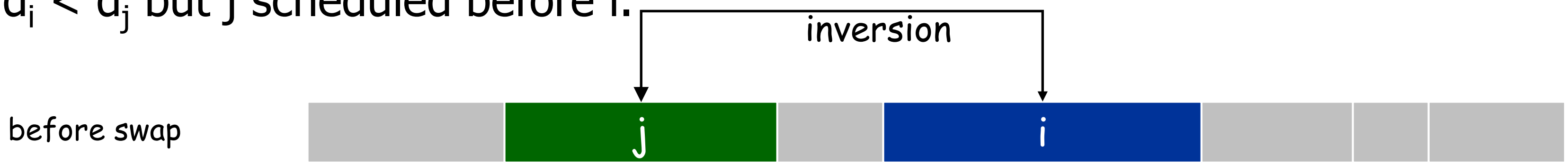
Observation. If a schedule (with no idle time) has an inversion, then it has one with a pair of inverted jobs scheduled consecutively.

Pf.

- Suppose there is an inversion.
- …
- … going through schedule, at some point deadline decreases.

# Minimizing Maximum 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.




Observation. Greedy schedule has no inversions.

Observation. All schedules without inversions have same lateness.

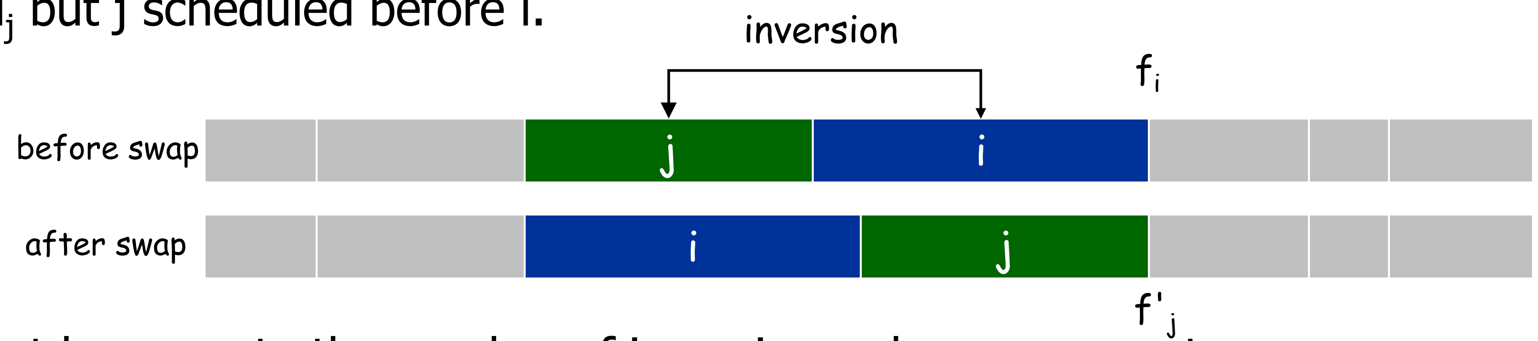
Observation. If a schedule (with no idle time) has an inversion, then it has one with a pair of inverted jobs scheduled consecutively.

Pf.

- Suppose there is an inversion.
- There is a pair of jobs i and j such that: $d_i < d_j$ but j scheduled before i.
- Walk through the schedule from j to i.
- Increasing deadlines (= no inversions), at some point deadline decreases.

# Minimizing Maximum 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.

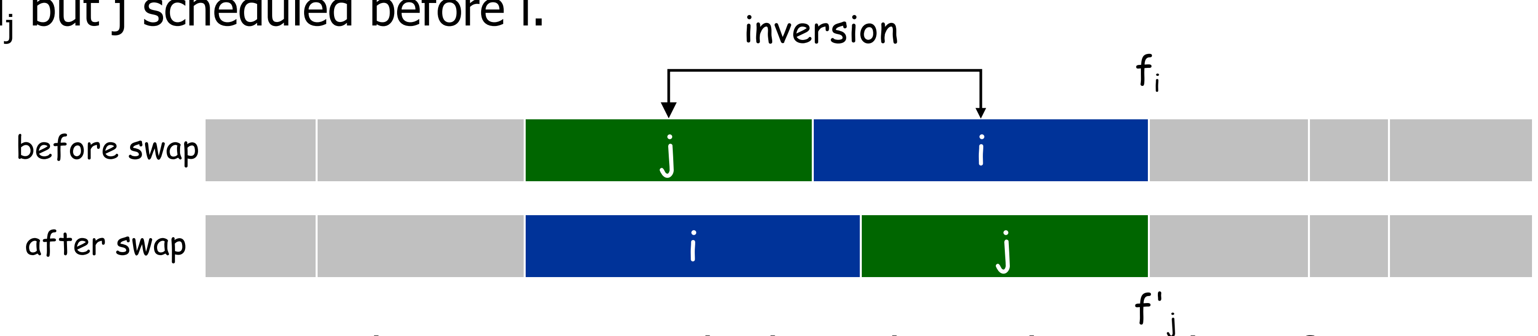



Q. What happens to the number of inversions when we swap two adjacent, inverted jobs? (reduces, stays equal, increases)

Q. Can we swap two adjacent, inverted jobs without increasing the maximum lateness?

# Minimizing Maximum 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.

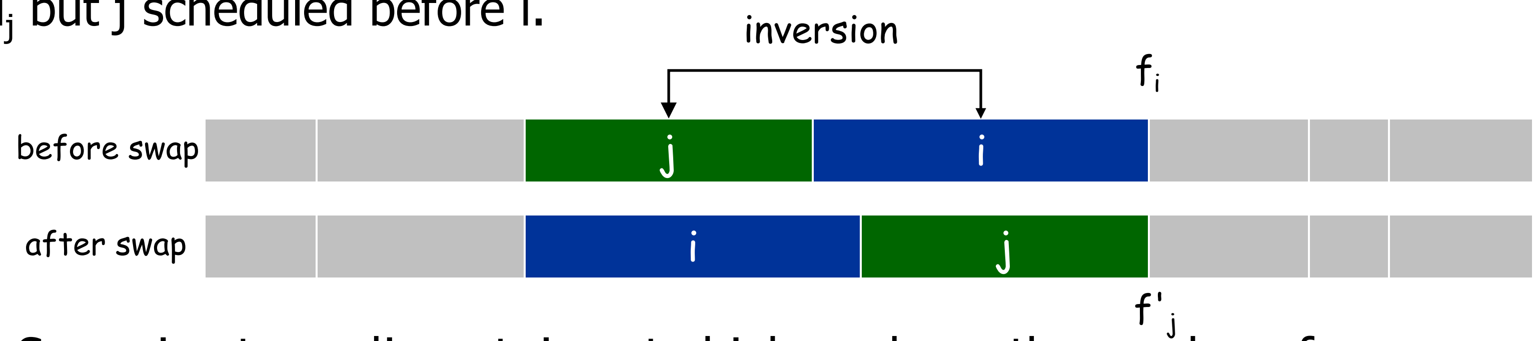



**Claim.** Swapping two adjacent, inverted jobs reduces the number of inversions by one and does not increase the maximum lateness.

**Pf.**

# Minimizing Maximum 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.



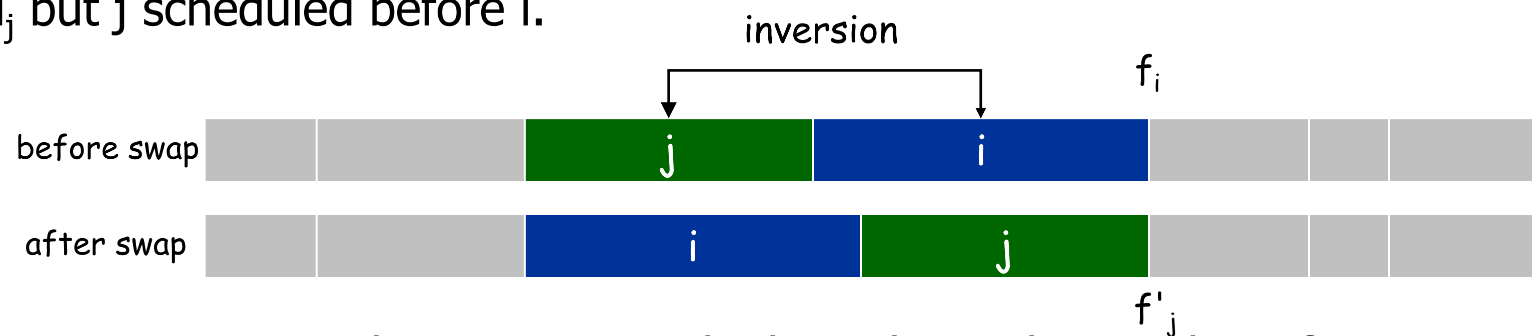

Claim. Swapping two adjacent, inverted jobs reduces the number of inversions by one and does not increase the maximum lateness.

Pf. Let $\ell$ be the max lateness before the swap, and let $\ell'$ be it afterwards.

- Q. What happens with the lateness of other jobs?
- Q. What happens with the lateness of i?
- Q. What happens with the lateness of j?

# Minimizing Maximum 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.




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

**Pf.** Let $\ell$ be the max lateness before the swap, and let $\ell'$ be it afterwards.

- $\ell'_k = \ell_k$ for all $k \neq i, j$ (lateness other jobs the same)
- $\ell'_i \leq \ell_i$ (new lateness for i smaller)
- If job j is late:

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

# Minimizing Lateness: Analysis of Greedy Algorithm

Theorem. Greedy schedule S is optimal.

Pf. (by contradiction)

Idea of proof:

- Suppose S is not optimal.
- Take a specific optimal schedule S*.
- Change to look like greedy schedule (less inversions) without losing optimality.

# Minimizing Lateness: Analysis of Greedy Algorithm

Theorem. Greedy schedule S is optimal.

Pf. (by contradiction)

Suppose S is not optimal.

Define S* to be an optimal schedule that has the fewest number of inversions (of all optimal schedules) and has no idle time.

Clearly S≠S*.

# Minimizing Lateness: Analysis of Greedy Algorithm

Theorem. Greedy schedule S is optimal.

Pf. (by contradiction)

Suppose S is not optimal.

Define S* to be an optimal schedule that has the fewest number of inversions (of all optimal schedules) and has no idle time.

Clearly S≠S*. Case analysis:

- If S* has no inversions
- If S* has an inversion

TU Delft

## Minimizing Lateness: Analysis of Greedy Algorithm

Theorem. Greedy schedule S is optimal.

Pf. (by contradiction)

Suppose S is not optimal.

Define S* to be an optimal schedule that has the fewest number of inversions (of all optimal schedules) and has no idle time.

Clearly S≠S*. Case analysis:

- If S* has no inversions. Q. How can we derive a contradiction?
- If S* has an inversion

# Minimizing Lateness: Analysis of Greedy Algorithm

Greedy has no inversions.

All schedules without inversions have same lateness.

**Theorem.** Greedy schedule S is optimal.

**Pf.** (by contradiction)

Suppose S is not optimal.

Define S* to be an optimal schedule that has the fewest number of inversions (of all optimal schedules) and has no idle time.

Clearly $S \neq S^*$.

- If S* has no inversions, then $L_S = L_{S^*}$. Contradiction.
- If S* has an inversion, Q. How can we derive a contradiction?

# Minimizing Lateness: Analysis of Greedy Algorithm

Greedy has no inversions.

All schedules without inversions have same lateness.

**Theorem.** Greedy schedule S is optimal.

**Pf.** (by contradiction)

Suppose S is not optimal.

Define S* to be an optimal schedule that has the fewest number of inversions (of all optimal schedules) and has no idle time.

Clearly S≠S*.

- If S* has no inversions, then maxl(S) = maxl(S*). Contradiction.
- 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*

So S is an optimal schedule. ▪

This proof can be found on pages 128-131.

TU Delft

# 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.

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

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.

Q. Which strategy did we use for the problems in this chapter (interval scheduling, interval partitioning, minimizing lateness) ?

# 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. [Interval scheduling]

Exchange argument. Gradually transform an optimal solution to the one found by the greedy algorithm without hurting its quality. [Minimizing lateness, Interval scheduling]

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. [Interval partitioning]

# Example exam exercise

Planning a mini-triathlon:

1. swim 20 laps (one at a time)
2. bike 10 km (can be done simultaneously)
3. run 3 km (can be done simultaneously)

expected times are given for each contestant

Def. The completion time is the earliest time all contestants are finished.

Q. In what order should they start to minimize the completion time?

Q. Proof that this order is optimal (minimal).

Ex.

| | 1 | 2 | 3 | ← contestant |
|---|---|---|---|---|
| $s_j$ | 3 | 2 | 4 | ← time required for swimming |
| $b_j$ | 5 | 4 | 6 | ← time required for biking |
| $r_j$ | 3 | 3 | 3 | ← time required for running |

Come to the instruction (Friday 13:45) if you do not know how to answer this.

# Variant: Scheduling to Minimizing *Total* Lateness

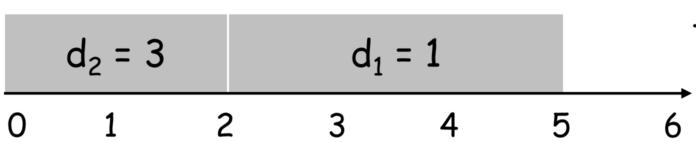
## Minimizing total lateness problem.

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

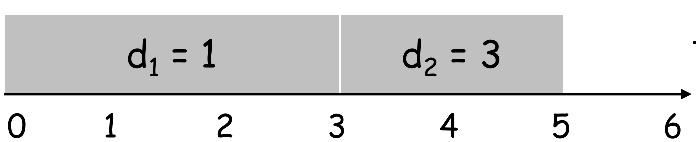
- If j starts at time $s_j$, it finishes at time $f_j = s_j + t_j$.
- Lateness: $\ell_j = \max \{ 0, \ f_j - d_j \}$.
- Goal: schedule all jobs to minimize total lateness $L = \Sigma_j \ell_j$.

Ex:

| | 1 | 2 | ← job number |
|---|---|---|---|
| $t_j$ | 3 | 2 | ← time required |
| $d_j$ | 1 | 3 | ← deadline |




total lateness = 0+4




total lateness = 2+2

No polynomial algorithm can compute optimal schedule (unless P=NP)
(see Master's course Advanced Algorithms)