CSE 421

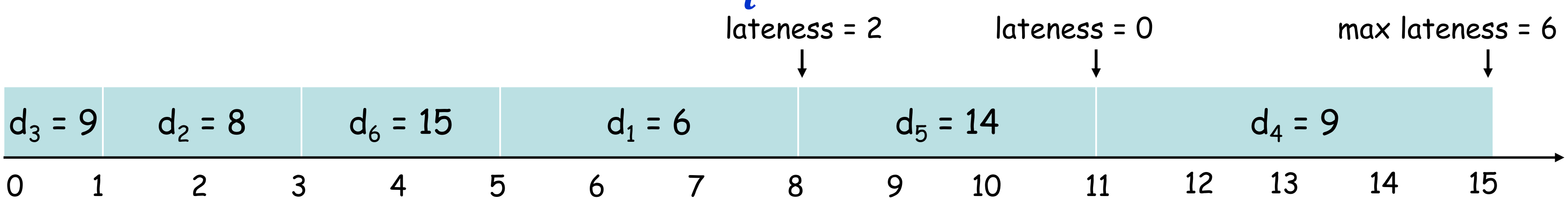
Greedy Algorithms / Minimizing Lateness

Yin Tat Lee

Scheduling to Minimizing Lateness

- Similar to interval scheduling.
- Instead of start and finish times, request $\boldsymbol{i}$ has
  - ➢ Time Requirement $\boldsymbol{t_i}$ which must be scheduled in a contiguous block
  - ➢ Deadline $\boldsymbol{d_i}$ by which time the request would like to be finished
- Requests are scheduled into time intervals $[\boldsymbol{s_i}, \boldsymbol{f_i}]$ s.t. $\boldsymbol{t_i = f_i - s_i}$.
- Lateness for request $\boldsymbol{i}$ is
  - If $\boldsymbol{d_i < f_i}$ then request $\boldsymbol{i}$ is late by $\boldsymbol{L_i = f_i - d_i}$ otherwise its lateness $\boldsymbol{L_i = 0}$
- **Goal:** Find a schedule that minimize the Maximum lateness $\boldsymbol{L = \max_i L_i}$

	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.

	1	2
t_j	1	10
d_j	100	10

counterexample

- [Smallest slack]

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

	1	2
t_j	1	10
d_j	2	10

counterexample

- [Earliest deadline first]

  Consider jobs in ascending order of deadline d_j.

Greedy Algorithm: Earliest Deadline First

Sort deadlines in increasing order $(d_1 \leq d_2 \leq \cdots \leq d_n)$

$f \leftarrow 0$

for $i \leftarrow 1$ to n to

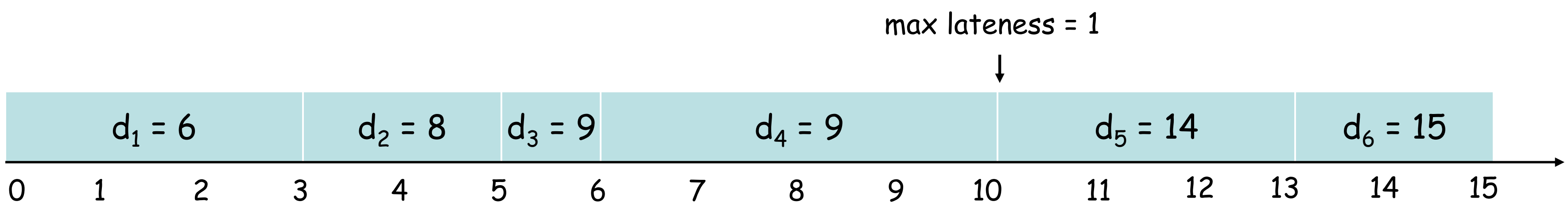
$\quad s_i \leftarrow f$

$\quad f_i \leftarrow s_i + t_i$

$\quad f \leftarrow f_i$

end for

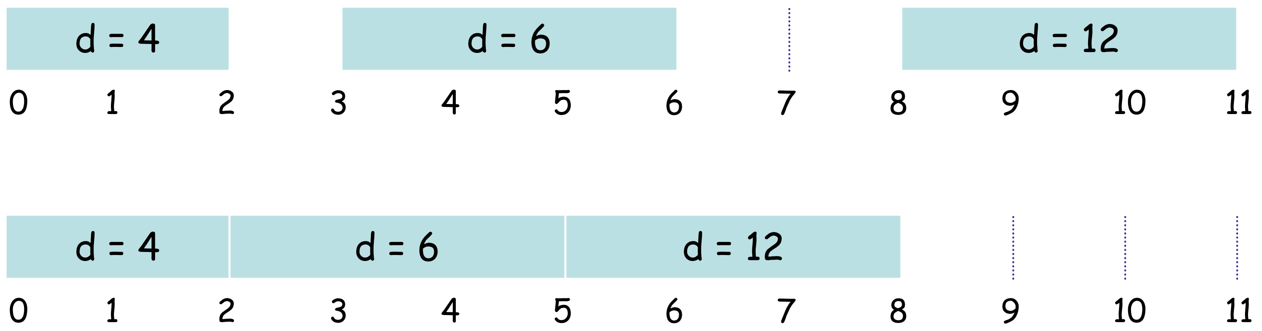
	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Minimizing Lateness: No Idle Time

Observation.

- There exists an optimal schedule with no idle time.



Observation.

- The greedy schedule has no idle time.

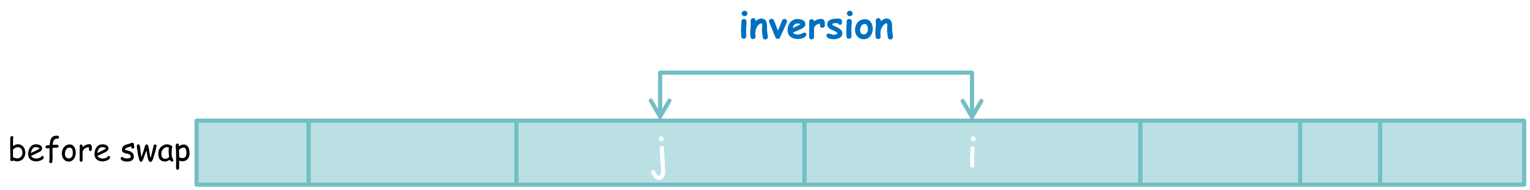
Proof for Greedy Algorithm: Exchange Argument

- We will show that if there is another schedule $\boldsymbol{O}$ (think optimal schedule) then we can gradually change $\boldsymbol{O}$ so that
    - at each step the maximum lateness in $\boldsymbol{O}$ never gets worse.
    - it eventually becomes the same cost as $\boldsymbol{A}$ (by greedy).

Minimizing Lateness: Inversions

Definition

- An adjacent inversion in schedule $\boldsymbol{S}$ is a pair of jobs $\boldsymbol{i}$ and $\boldsymbol{j}$ such that
  - $\boldsymbol{d_i < d_j}$
  - Job $\boldsymbol{i}$ is scheduled immediately after Job $\boldsymbol{j}$



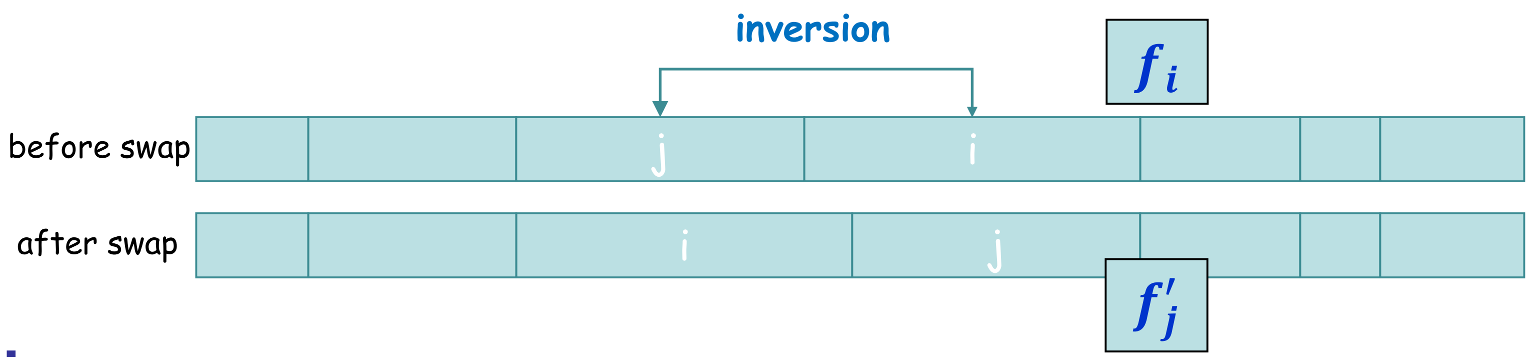
Observation

- Greedy schedule has no adjacent inversions.

Minimizing Lateness: Inversions

Definition

- An adjacent inversion in schedule S is a pair of jobs i and j such that
  - $d_i < d_j$
  - Job i is scheduled immediately after Job j

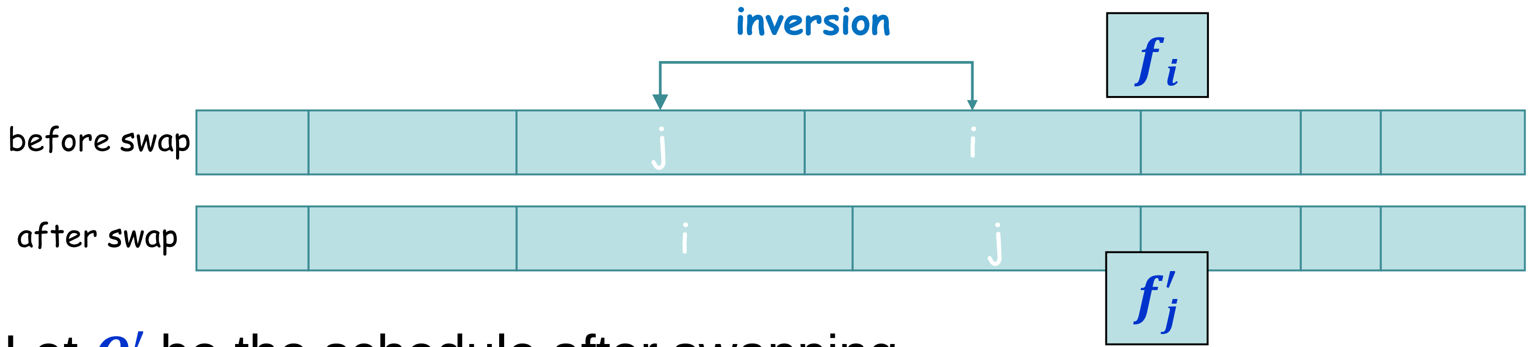


Claim

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

Minimizing Lateness: Inversions

Lemma: Swapping two adjacent, inverted jobs does not increase the maximum lateness.



Proof: Let O' be the schedule after swapping.

- Lateness $L_i' \leq L_i$ since i is scheduled earlier in O' than in O
- Requests i and j together occupy the same total time slot in both schedules
    - All other requests $k \neq i, j$ have $L_k' = L_k$
    - $f_j' = f_i$ so $L_j' = f_j' - d_j = f_i - d_j < f_i - d_i = L_i$
- Maximum lateness has not increased!

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 $\boldsymbol{O}$ with no idle time
- If $\boldsymbol{O}$ has an inversion then it has a **consecutive** pair of requests in its schedule that are inverted and can be swapped without increasing lateness
- Eventually these swaps will produce an optimal schedule with no inversions
  - Each swap decreases the number of inversions by **1**
  - There are at most $\boldsymbol{n(n-1)/2}$ inversions.
    (we only care that this is finite.)

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 the finish time of the last of these jobs but the set of these requests occupies the same time segment in both schedules
  - Last of these requests finishes at the same time in any such schedule.

Why Exchange Argument?

Greedy cannot handle problems with many local minimum.

Let S be any solution and A be the solution given by greedy.

Exchange argument gives a sequence

$$S \to S_1 \to S_2 \to S_3 \to \cdots \to A$$

such that

- each solution is “close to” the another solution
- the solution is improving.

It basically proves that there is no local min.

CSE 421

Greedy Algorithms / Caching Problem

Yin Tat Lee

Optimal Caching/Paging

Memory systems

- Many levels of storage with different access times
- Smaller storage has shorter access time
- To access an item it must be brought to the lowest level of the memory system

Consider the problem between 2 levels

- Main memory with n data items
- Cache can hold $k < n$ items
- Assume no restrictions about where items can be
- Suppose cache is full initially
  - ➢Holds k data items to start with

Optimal Offline Caching

Caching

- Cache with capacity to store $\boldsymbol{k}$ items.
- Sequence of $\boldsymbol{m}$ item requests $\boldsymbol{d_1, d_2, \cdots, d_m}$.
- Cache hit: item already in cache when requested.
- Cache miss: item not already in cache when requested: must bring requested item into cache, and evict some existing item, if full.

Goal

- Eviction schedule that minimizes number of evictions.

Example: $\boldsymbol{k = 2}$, initial cache $\boldsymbol{= a, b}$,
requests: $\boldsymbol{a, b, c, b, c, a, a, b}$.

Optimal eviction schedule: $\boldsymbol{2}$ cache misses.

requests	cache	
a	a	b
b	a	b
c	c	b
b	c	b
c	c	b
a	a	b
a	a	b
b	a	b

Why 2 is optimal?

Optimal Offline Caching: Farthest-In-Future

Which item we should evict?

Farthest-in-future

- Evict item in the cache that is not requested until farthest in the future.

current cache:

a	b	c	d	e	f

future queries: `g a b c e d a b b a c d e a f a d e f g h ...`

cache miss (at g)

eject this one (at f)

Theorem

- [Bellady, 1960s] FIF is an optimal eviction schedule.

Exchange Argument

- We can swap choices to convert other schedules to Farthest-In-Future without losing quality

Warm up ($n = k + 1$)

Farthest-in-future

- Evict item in the cache that is not requested until farthest in the future.

current cache:	a	b	c	d	e	f

future queries: **g a b c e d a b b a c d e a f a d e f g h . . .**

↑ cache miss (under g) ↑ eject this one (under f)

When $n = k + 1$,
between the cache miss and the farthest-item in the future,
"**g a b c e d a b b a c d e a f**"
contains all the item.
Hence, any algorithm must miss once.

Online Caching

- Online vs. offline algorithms.
  Offline: full sequence of requests is known a priori.
  Online (reality): requests are not known in advance.
  Caching is among most fundamental online problems in CS.

- LIFO. Evict page brought in most recently.
- LRU. Evict page whose most recent access was earliest.

FIF with direction of time reversed!

- Theorem. FIF is optimal offline eviction algorithm.
  Provides basis for understanding and analyzing online algorithms.
  LRU is k-competitive. [Section 13.8]
  LIFO is arbitrarily bad.