Online Algorithms

a topic in

DM573 – Introduction to Computer Science

Kim Skak Larsen

Department of Mathematics and Computer Science (IMADA)
University of Southern Denmark (SDU)

kslarsen@imada.sdu.dk
https://imada.sdu.dk/u/kslarsen/

November 28, 2024

Ski Rental – a simplest online problem

- A highly skilled, successful computer scientist is rewarded a ski vacation until her company desperately needs her again, at which point she is called home immediately, being notified when she wakes up in the morning.
- At the ski resort, skis cost 10 units in the local currency.
- One can rent skis for 1 unit per day.
- Of course, if she buys, she doesn't have to rent anymore.
- Which algorithm does she employ to minimize her spending?

Ski Rental – a simplest online problem

SDU

- A highly skilled, successful computer scientist is rewarded a ski vacation until her company desperately needs her again, at which point she is called home immediately, being notified when she wakes up in the morning.
- At the ski resort, skis cost 10 units in the local currency.
- One can rent skis for 1 unit per day.
- Of course, if she buys, she doesn't have to rent anymore.
- Which algorithm does she employ to minimize her spending?

- What is a good algorithm?

Ski Rental – a simplest online problem

- A highly skilled, successful computer scientist is rewarded a ski vacation until her company desperately needs her again, at which point she is called home immediately, being notified when she wakes up in the morning.
- At the ski resort, skis cost 10 units in the local currency.
- One can rent skis for 1 unit per day.
- Of course, if she buys, she doesn't have to rent anymore.
- Which algorithm does she employ to minimize her spending?

- What is a good algorithm?
- How do we measure if it's a good algorithm?

Competitive Analysis

Competitive analysis is one way to measure the quality of an online algorithm.

Idea:

Let OPT denote an *optimal offline algorithm*.

"Offline" means getting the entire input before having to compute.

This corresponds to knowing the future!

We calculate how well we perform compared to OPT.

Competitive Analysis

Competitive analysis is one way to measure the quality of an online algorithm.

Idea:

Let $\text{O}\small{\text{PT}}$ denote an *optimal offline algorithm*.

"Offline" means getting the entire input before having to compute.

This corresponds to knowing the future!

We calculate how well we perform compared to $\text{O}\small{\text{PT}}$.

Notation:

$\text{A}\small{\text{LG}}(I)$ denote the result of running an algorithm $\text{A}\small{\text{LG}}$ on the input sequence I.

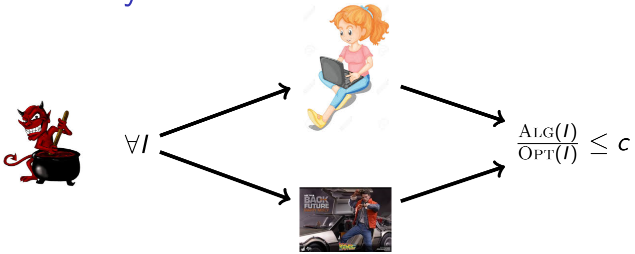
Thus, $\text{O}\small{\text{PT}}(I)$ is the result of running $\text{O}\small{\text{PT}}$ on I.

Competitive Analysis

$\forall I$

$$\frac{\text{ALG}(I)}{\text{OPT}(I)} \leq c$$

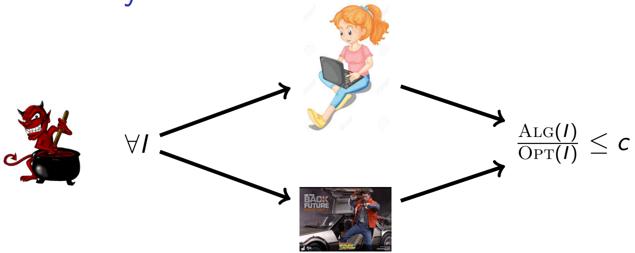
Competitive Analysis



An algorithm, $\textsc{Alg}$, is *c-competitive* if

$$\forall I: \frac{\textsc{Alg}(I)}{\textsc{Opt}(I)} \leq c.$$

Competitive Analysis



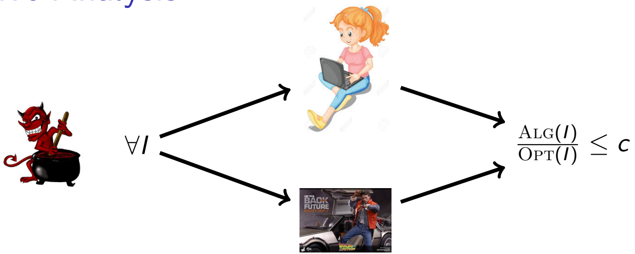
An algorithm, ALG, is *c-competitive* if

$$\forall I: \frac{\text{ALG}(I)}{\text{OPT}(I)} \leq c.$$

ALG has *competitive ratio c* if

c is the *best* (smallest) c for which ALG is c-competitive.

Competitive Analysis



An algorithm, $\textsc{Alg}$, is *c-competitive* if

$$\forall I : \frac{\textsc{Alg}(I)}{\textsc{Opt}(I)} \leq c.$$

$\textsc{Alg}$ has *competitive ratio c* if

c is the *best* (smallest) c for which $\textsc{Alg}$ is c-competitive.

Technically, the definition of being c-competitive is that $\exists b \, \forall I : \textsc{Alg}(I) \leq c \, \textsc{Opt}(I) + b$, but the additive term, b, does not become relevant for what we consider. Also not relevant today, a *best* c does not necessarily exist, so the competitive ratio is $\inf \{c \mid \textsc{Alg} \text{ is } c\text{-competitive}\}$.

Ski Rental – a simplest online problem

We have a very simple request sequence with very simple responses:

input sequence	decision
it's a new day	

Ski Rental – a simplest online problem

SDU

We have a very simple request sequence with very simple responses:

input sequence	decision
it's a new day	rent

Ski Rental – a simplest online problem

SDU

We have a very simple request sequence with very simple responses:

input sequence	decision
it's a new day	rent
it's a new day	

Ski Rental – a simplest online problem

We have a very simple request sequence with very simple responses:

input sequence	decision
it's a new day	rent
it's a new day	rent

Ski Rental – a simplest online problem

We have a very simple request sequence with very simple responses:

input sequence	decision
it's a new day	rent
it's a new day	rent
⋮	⋮
it's a new day	

Ski Rental – a simplest online problem

SDU

We have a very simple request sequence with very simple responses:

input sequence	decision
it's a new day	rent
it's a new day	rent
⋮	⋮
it's a new day	rent

Ski Rental – a simplest online problem

We have a very simple request sequence with very simple responses:

input sequence	decision
it's a new day	rent
it's a new day	rent
⋮	⋮
it's a new day	rent
it's a new day	

Ski Rental – a simplest online problem

We have a very simple request sequence with very simple responses:

input sequence	decision
it's a new day	rent
it's a new day	rent
⋮	⋮
it's a new day	rent
it's a new day	buy

Ski Rental – a simplest online problem

SDU

We have a very simple request sequence with very simple responses:

input sequence	decision
it's a new day	rent
it's a new day	rent
⋮	⋮
it's a new day	rent
it's a new day	buy
it's a new day	

Ski Rental – a simplest online problem

We have a very simple request sequence with very simple responses:

input sequence	decision
it's a new day	rent
it's a new day	rent
⋮	⋮
it's a new day	rent
it's a new day	buy
it's a new day	do nothing

Ski Rental – a simplest online problem

We have a very simple request sequence with very simple responses:

input sequence	decision
it's a new day	rent
it's a new day	rent
⋮	⋮
it's a new day	rent
it's a new day	buy
it's a new day	do nothing
it's a new day	do nothing
⋮	⋮

Ski Rental – a simplest online problem

SDU

We have a very simple request sequence with very simple responses:

input sequence	decision
it's a new day	rent
it's a new day	rent
⋮	⋮
it's a new day	rent
it's a new day	buy
it's a new day	do nothing
it's a new day	do nothing
⋮	⋮
come home	go home

Ski Rental – a simplest online problem

- So all reasonable algorithms are of the form **Buy on day X** (or never buy).

Ski Rental – a simplest online problem

- So all reasonable algorithms are of the form **Buy on day X** (or never buy).
- Reminder: It costs 10 to buy skis and 1 to rent.

Ski Rental – a simplest online problem

- So all reasonable algorithms are of the form **Buy on day X** (or never buy).
- Reminder: It costs 10 to buy skis and 1 to rent.
- We choose the algorithm: **Buy on day 10**.

Ski Rental – a simplest online problem

- So all reasonable algorithms are of the form **Buy on day X** (or never buy).
- Reminder: It costs 10 to buy skis and 1 to rent.
- We choose the algorithm: **Buy on day 10**.
- The optimal offline algorithm, OPT:

  d = the number of days we get to stay
  if $d < 10$:
  rent every day
  else: # $d \geq 10$
  buy on day 1

Ski Rental – a simplest online problem

- So all reasonable algorithms are of the form **Buy on day X** (or never buy).
- Reminder: It costs 10 to buy skis and 1 to rent.
- We choose the algorithm: **Buy on day 10**.
- The optimal offline algorithm, OPT:

  d = the number of days we get to stay
  if $d < 10$:
  rent every day
  else: # $d \geq 10$
  buy on day 1

- If we are called home *before* day 10, we (**Buy on day 10**) are optimal!
  - Our cost: $d < 10$.
  - OPT's cost: d.

Ski Rental – a simplest online problem

SDU

- So all reasonable algorithms are of the form **Buy on day X** (or never buy).
- Reminder: It costs 10 to buy skis and 1 to rent.
- We choose the algorithm: **Buy on day 10**.
- The optimal offline algorithm, OPT:

    d = the number of days we get to stay
    if $d < 10$:
        rent every day
    else: $\# \; d \geq 10$
        buy on day 1

- If we are called home *before* day 10, we (**Buy on day 10**) are optimal!
    - Our cost: $d < 10$.
    - OPT's cost: d.
- In the worst case, we are called home at (or any time after) day 10:
    - Our cost: $9 + 10$.
    - OPT's cost: 10.

Ski Rental – a simplest online problem

- So all reasonable algorithms are of the form **Buy on day X** (or never buy).
- Reminder: It costs 10 to buy skis and 1 to rent.
- We choose the algorithm: **Buy on day 10**.
- The optimal offline algorithm, $\textsc{Opt}$:

  d = the number of days we get to stay
  if $d < 10$:
  rent every day
  else: $\#\ d \geq 10$
  buy on day 1

- If we are called home *before* day 10, we (**Buy on day 10**) are optimal!
  - Our cost: $d < 10$.
  - $\textsc{Opt}$'s cost: d.
- In the worst case, we are called home at (or any time after) day 10:
  - Our cost: $9 + 10$.
  - $\textsc{Opt}$'s cost: 10.
- Result: we perform at most $\frac{19}{10} < 2$ times worse than $\textsc{Opt}$.

Competitive Analysis

In general, we want to *define* good algorithms for online problems and *prove* that they are good.

The ratio ($\frac{19}{10}$) from ski rental is a guarantee:

> *That algorithm never performs worse than* $\frac{19}{10}$ *times* $\textsc{Opt}$.

Competitive Analysis

In general, we want to *define* good algorithms for online problems and *prove* that they are good.

The ratio ($\frac{19}{10}$) from ski rental is a guarantee:

> *That algorithm never performs worse than* $\frac{19}{10}$ *times* $\textsc{Opt}$.

Is our algorithm $\frac{19}{10}$-competitive?

Competitive Analysis

In general, we want to *define* good algorithms for online problems and *prove* that they are good.

The ratio ($\frac{19}{10}$) from ski rental is a guarantee:

> *That algorithm never performs worse than* $\frac{19}{10}$ *times* $\textsc{Opt}$.

Is our algorithm $\frac{19}{10}$-competitive?	Yes

Competitive Analysis

In general, we want to *define* good algorithms for online problems and *prove* that they are good.

The ratio ($\frac{19}{10}$) from ski rental is a guarantee:

That algorithm never performs worse than $\frac{19}{10}$ *times* $\textsc{Opt}$.

Is our algorithm $\frac{19}{10}$-competitive? Yes

Is our algorithm also 2-competitive?

Competitive Analysis

In general, we want to *define* good algorithms for online problems and *prove* that they are good.

The ratio ($\frac{19}{10}$) from ski rental is a guarantee:

That algorithm never performs worse than $\frac{19}{10}$ *times* $\textsc{Opt}$.

Is our algorithm $\frac{19}{10}$-competitive?	Yes
Is our algorithm also 2-competitive?	Yes

Competitive Analysis

In general, we want to *define* good algorithms for online problems and *prove* that they are good.

The ratio ($\frac{19}{10}$) from ski rental is a guarantee:

That algorithm never performs worse than $\frac{19}{10}$ *times* $\textsc{Opt}$.

Is our algorithm $\frac{19}{10}$-competitive?	Yes
Is our algorithm also 2-competitive?	Yes
Is our algorithm also 42-competitive?	

Competitive Analysis

In general, we want to *define* good algorithms for online problems and *prove* that they are good.

The ratio ($\frac{19}{10}$) from ski rental is a guarantee:

That algorithm never performs worse than $\frac{19}{10}$ *times* $\textsc{Opt}$.

Is our algorithm $\frac{19}{10}$-competitive?	Yes
Is our algorithm also 2-competitive?	Yes
Is our algorithm also 42-competitive?	Yes

Competitive Analysis

In general, we want to *define* good algorithms for online problems and *prove* that they are good.

The ratio ($\frac{19}{10}$) from ski rental is a guarantee:

That algorithm never performs worse than $\frac{19}{10}$ *times* $\textsc{Opt}$.

Is our algorithm $\frac{19}{10}$-competitive?	Yes
Is our algorithm also 2-competitive?	Yes
Is our algorithm also 42-competitive?	Yes
Is our algorithm 1.5-competitive?	

Competitive Analysis

In general, we want to *define* good algorithms for online problems and *prove* that they are good.

The ratio ($\frac{19}{10}$) from ski rental is a guarantee:

That algorithm never performs worse than $\frac{19}{10}$ *times* $\textsc{Opt}$.

Is our algorithm $\frac{19}{10}$-competitive?	Yes
Is our algorithm also 2-competitive?	Yes
Is our algorithm also 42-competitive?	Yes
Is our algorithm 1.5-competitive?	No

Competitive Analysis

In general, we want to *define* good algorithms for online problems and *prove* that they are good.

The ratio ($\frac{19}{10}$) from ski rental is a guarantee:

> *That algorithm never performs worse than $\frac{19}{10}$ times* $\textsc{Opt}$.

Is our algorithm $\frac{19}{10}$-competitive?	Yes
Is our algorithm also 2-competitive?	Yes
Is our algorithm also 42-competitive?	Yes
Is our algorithm 1.5-competitive?	No
Does our algorithm have competitive ratio $\frac{19}{10}$?	

Competitive Analysis

In general, we want to *define* good algorithms for online problems and *prove* that they are good.

The ratio ($\frac{19}{10}$) from ski rental is a guarantee:

> *That algorithm never performs worse than* $\frac{19}{10}$ *times* $\textsc{Opt}$.

Is our algorithm $\frac{19}{10}$-competitive?	Yes
Is our algorithm also 2-competitive?	Yes
Is our algorithm also 42-competitive?	Yes
Is our algorithm 1.5-competitive?	No
Does our algorithm have competitive ratio $\frac{19}{10}$?	Yes

Competitive Analysis

In general, we want to *define* good algorithms for online problems and *prove* that they are good.

The ratio ($\frac{19}{10}$) from ski rental is a guarantee:

> *That algorithm never performs worse than* $\frac{19}{10}$ *times* $\textsc{Opt}$.

Is our algorithm $\frac{19}{10}$-competitive?	Yes
Is our algorithm also 2-competitive?	Yes
Is our algorithm also 42-competitive?	Yes
Is our algorithm 1.5-competitive?	No
Does our algorithm have competitive ratio $\frac{19}{10}$?	Yes
Does our algorithm have competitive ratio 2?	

Competitive Analysis

In general, we want to *define* good algorithms for online problems and *prove* that they are good.

The ratio ($\frac{19}{10}$) from ski rental is a guarantee:

That algorithm never performs worse than $\frac{19}{10}$ *times* $\textsc{Opt}$.

Is our algorithm $\frac{19}{10}$-competitive?	Yes
Is our algorithm also 2-competitive?	Yes
Is our algorithm also 42-competitive?	Yes
Is our algorithm 1.5-competitive?	No
Does our algorithm have competitive ratio $\frac{19}{10}$?	Yes
Does our algorithm have competitive ratio 2?	No

Competitive Analysis

In general, we want to *define* good algorithms for online problems and *prove* that they are good.

The ratio ($\frac{19}{10}$) from ski rental is a guarantee:

That algorithm never performs worse than $\frac{19}{10}$ *times* $\textsc{Opt}$.

Is our algorithm $\frac{19}{10}$-competitive?	Yes
Is our algorithm also 2-competitive?	Yes
Is our algorithm also 42-competitive?	Yes
Is our algorithm 1.5-competitive?	No
Does our algorithm have competitive ratio $\frac{19}{10}$?	Yes
Does our algorithm have competitive ratio 2?	No
Does our algorithm have competitive ratio 1.5?	

Competitive Analysis

In general, we want to *define* good algorithms for online problems and *prove* that they are good.

The ratio ($\frac{19}{10}$) from ski rental is a guarantee:

> *That algorithm never performs worse than $\frac{19}{10}$ times* $\textsc{Opt}$.

Is our algorithm $\frac{19}{10}$-competitive?	Yes
Is our algorithm also 2-competitive?	Yes
Is our algorithm also 42-competitive?	Yes
Is our algorithm 1.5-competitive?	No
Does our algorithm have competitive ratio $\frac{19}{10}$?	Yes
Does our algorithm have competitive ratio 2?	No
Does our algorithm have competitive ratio 1.5?	No

Online Problems

So, what characterizes an *online problem*?

- Input arrives *one request at a time*.
- For each request, we have to make an *irrevocable decision*.
- We want to *minimize cost*.

Online Problems

So, what characterizes an *online problem*?

- Input arrives *one request at a time*.
- For each request, we have to make an *irrevocable decision*.
- We want to *minimize cost*.

Sometimes we want to maximize a profit instead of minimizing a cost and there are some technicalities in adjusting definitions to accommodate that possibility.

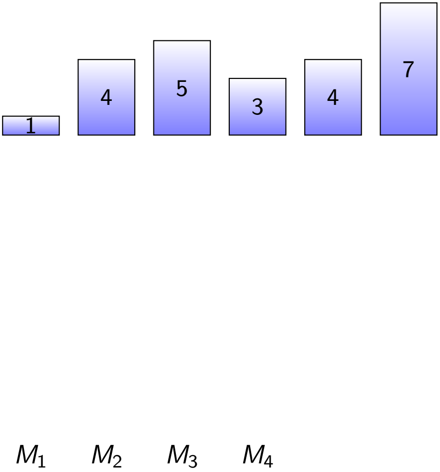
Machine Scheduling

- $m \geq 1$ machines.
- n jobs of varying sizes arriving one at a time to be assigned to a machine.
- The goal is to *minimize makespan*, i.e., finish all jobs as early as possible.

Machine Scheduling

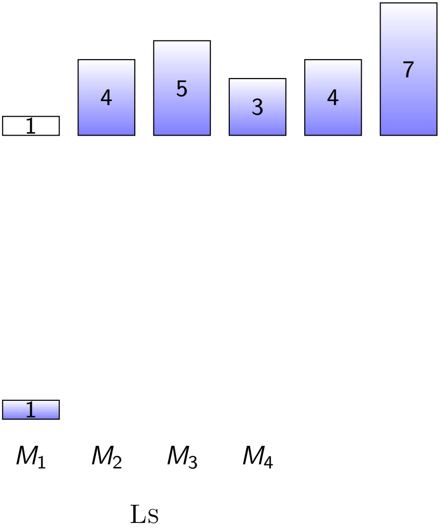
- $m \geq 1$ machines.
- n jobs of varying sizes arriving one at a time to be assigned to a machine.
- The goal is to *minimize makespan*, i.e., finish all jobs as early as possible.
- Algorithm List Scheduling (LS): place next job on the least loaded machine.

Machine Scheduling – List Scheduling example

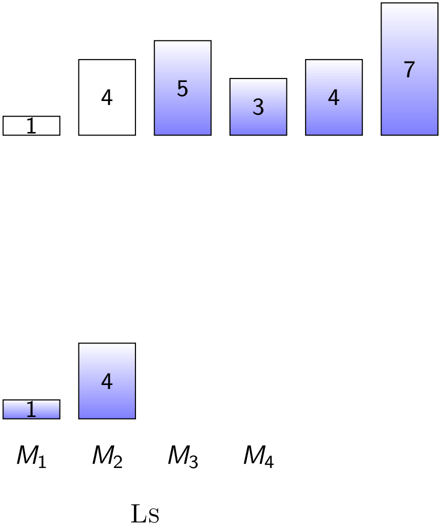


LS

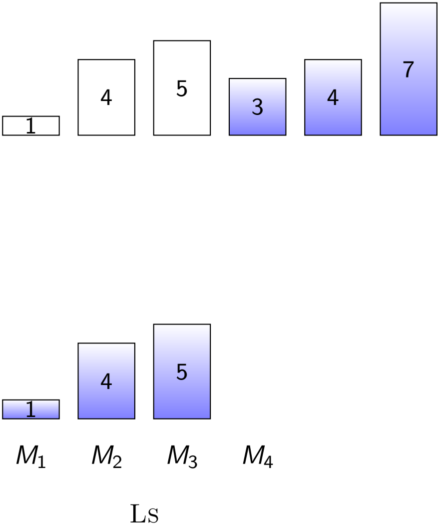
Machine Scheduling – List Scheduling example



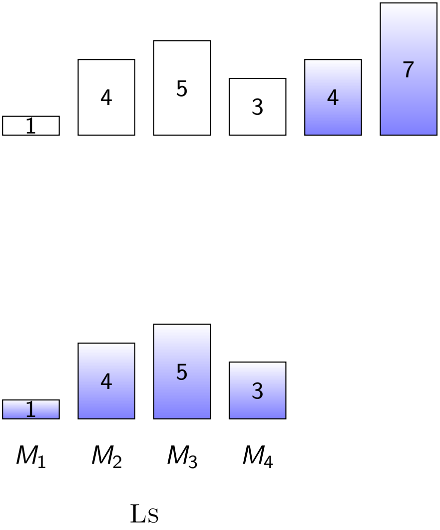
Machine Scheduling – List Scheduling example



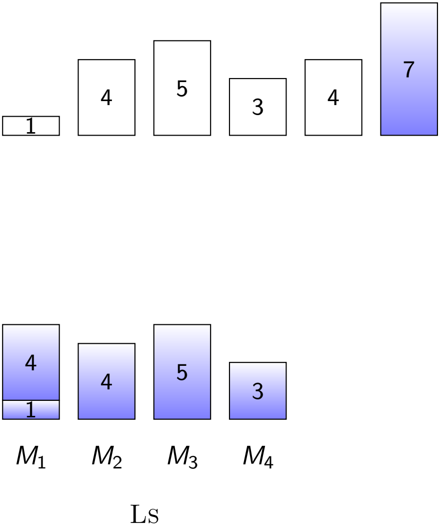
Machine Scheduling – List Scheduling example



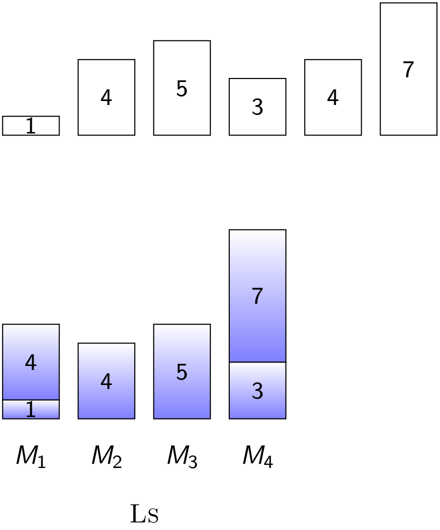
Machine Scheduling – List Scheduling example



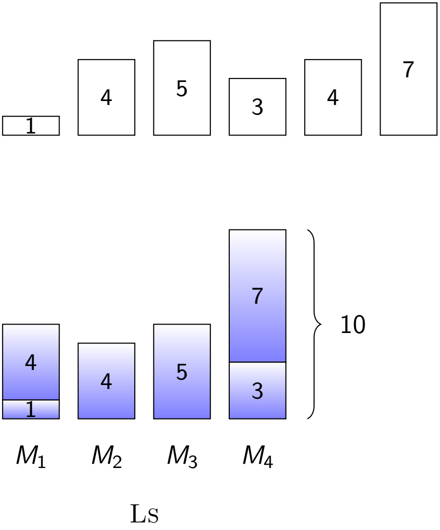
Machine Scheduling – List Scheduling example



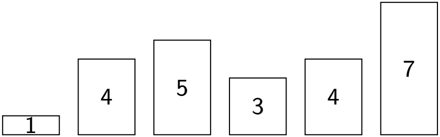
Machine Scheduling – List Scheduling example

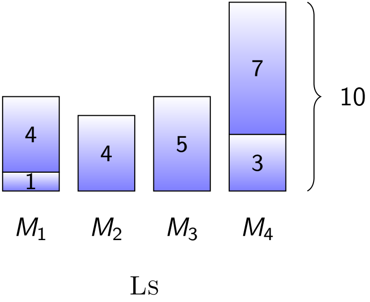


Machine Scheduling – List Scheduling example



Machine Scheduling – List Scheduling example





Is 10 a good result?

Machine Scheduling – List Scheduling example

On some sequences, 10 is a good result.

On some sequences, 1.000.000 is a good result.

Sometimes 1000 is a bad result.

It depends on what is possible, i.e., how large or “difficult” the input is.

Competitive Analysis – why $\textsc{Opt}$?

$\textsc{Opt}$ is not an online algorithm.

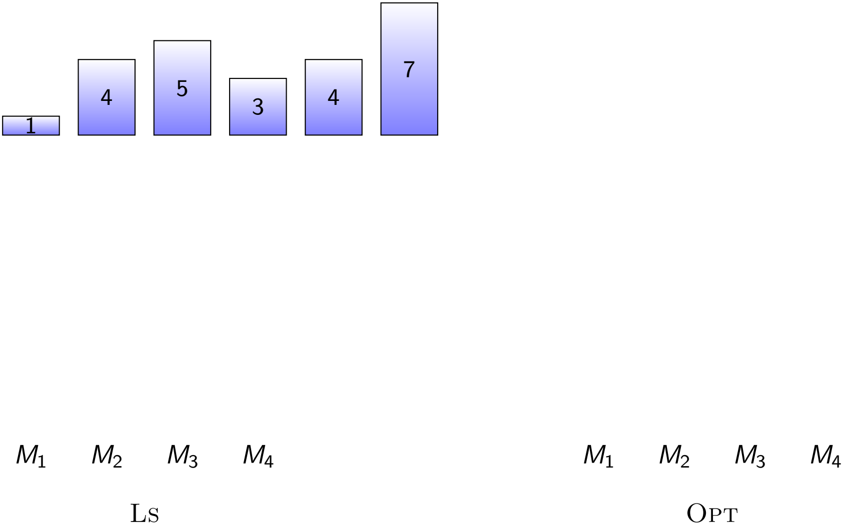
However, it is a natural *reference point*:

- Our online algorithms cannot perform better than $\textsc{Opt}$.
- As input sequences I get harder, $\textsc{Opt}(I)$ typically grows, so a comparison to $\textsc{Opt}$ remains somewhat reasonable.

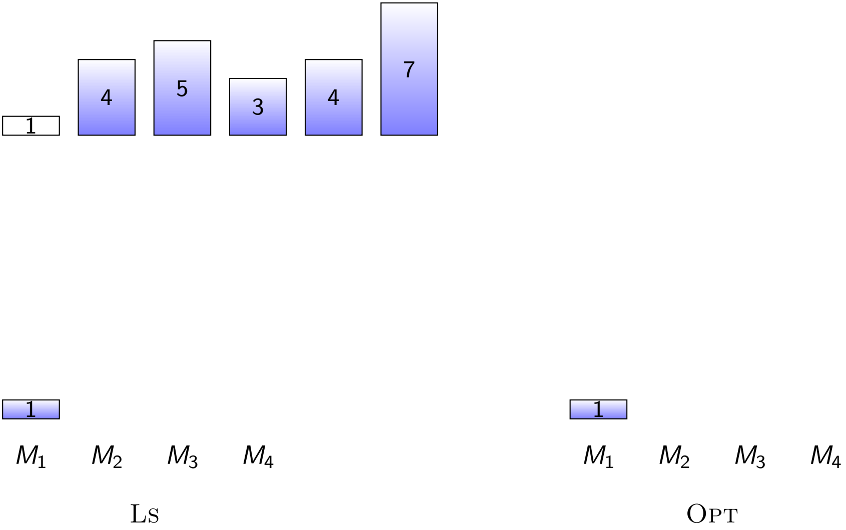
Formally:

- For any $\textsc{Alg}$ and any I, $\dfrac{\textsc{Alg}(I)}{\textsc{Opt}(I)} \geq 1$.
- We want to design algorithms with smallest possible competitive ratio.

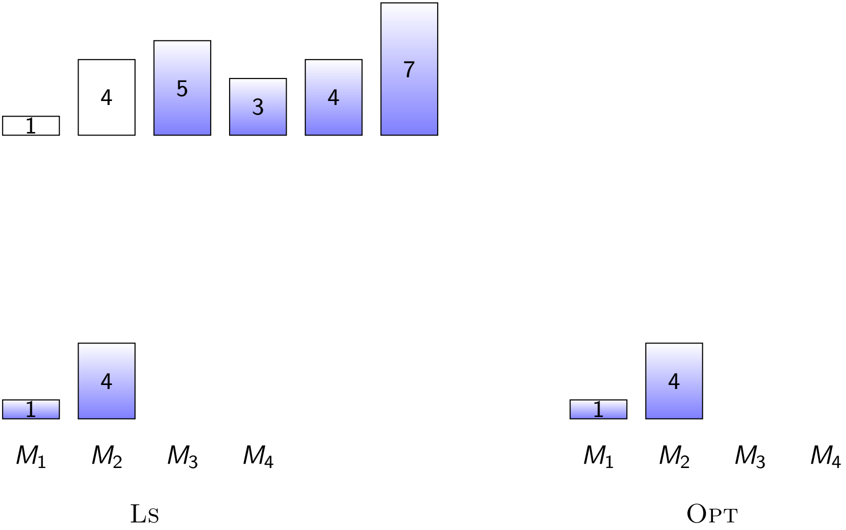
Machine Scheduling – List Scheduling example



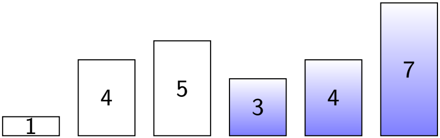
Machine Scheduling – List Scheduling example



Machine Scheduling – List Scheduling example

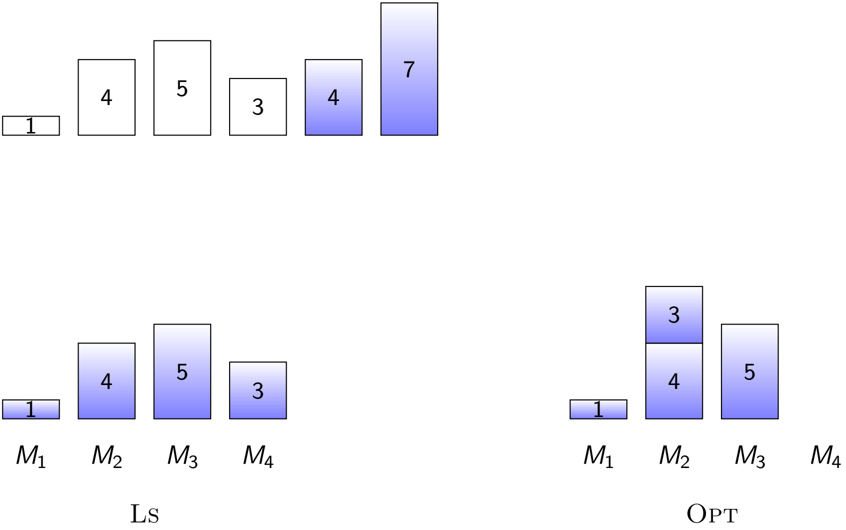


Machine Scheduling – List Scheduling example

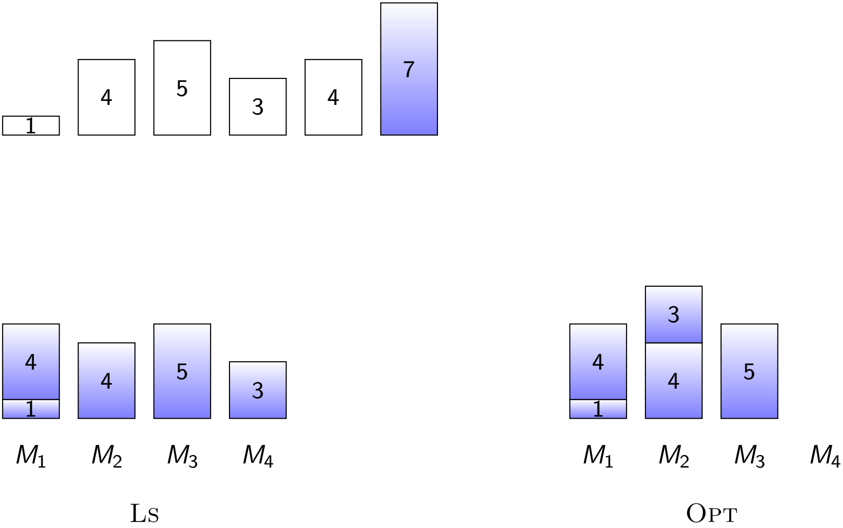


What does OPT do now?

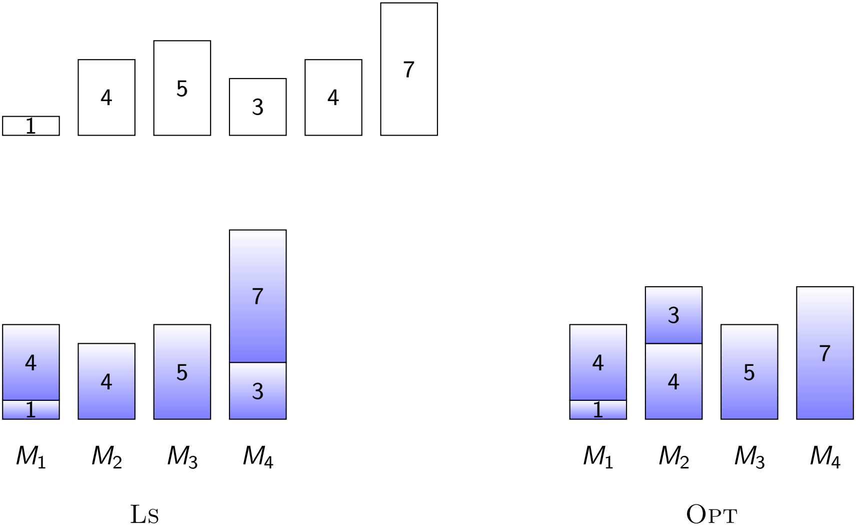
Machine Scheduling – List Scheduling example



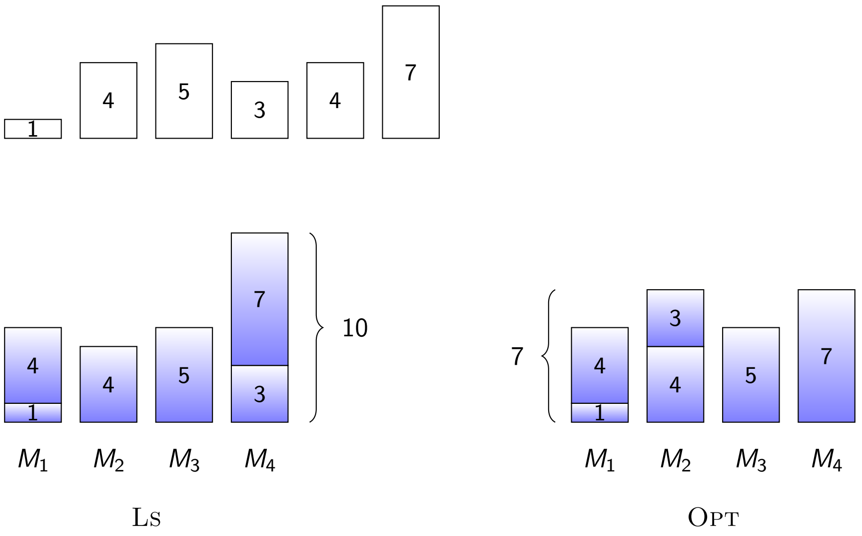
Machine Scheduling – List Scheduling example



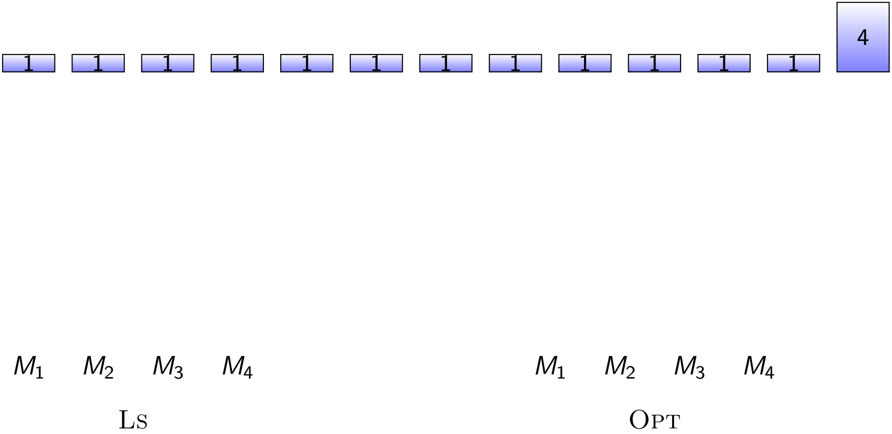
Machine Scheduling – List Scheduling example



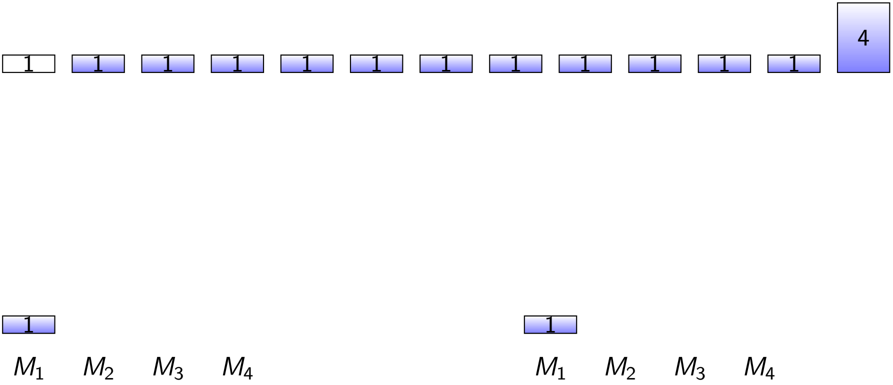
Machine Scheduling – List Scheduling example



Machine Scheduling – Ls is at best $(2 - \frac{1}{m})$-competitive



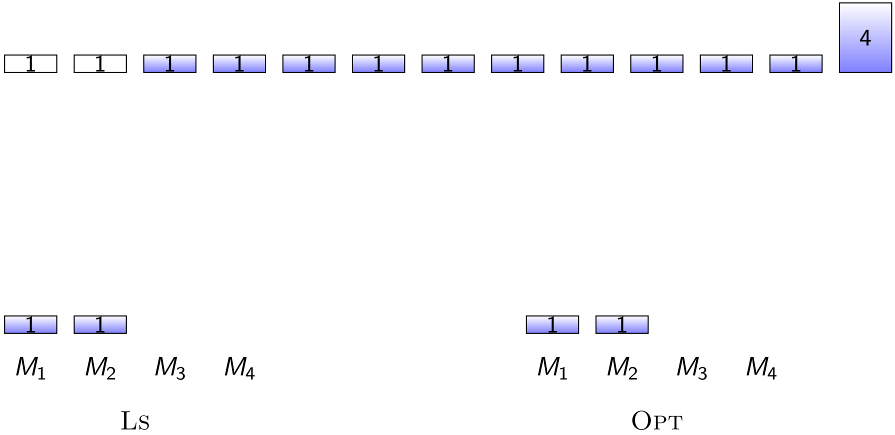
Machine Scheduling – Ls is at best $(2 - \frac{1}{m})$-competitive



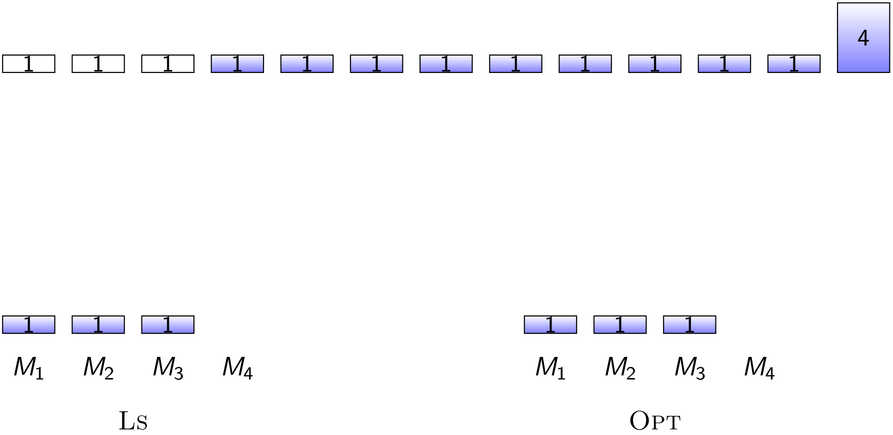
Ls

Opt

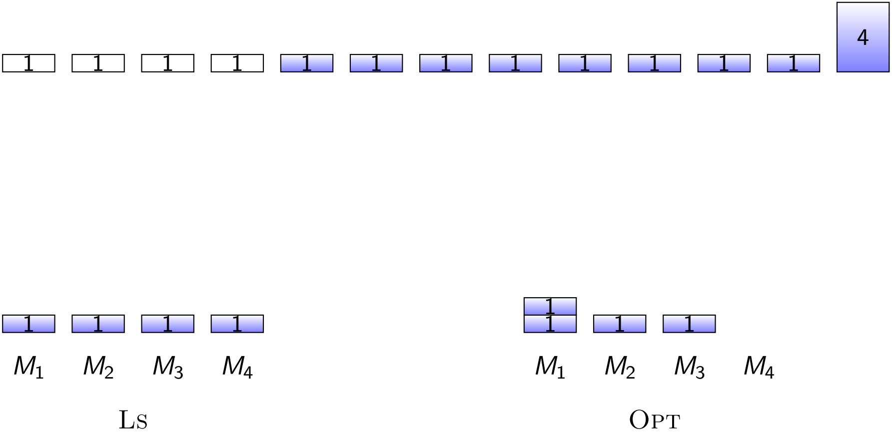
Machine Scheduling – LS is at best $(2 - \frac{1}{m})$-competitive



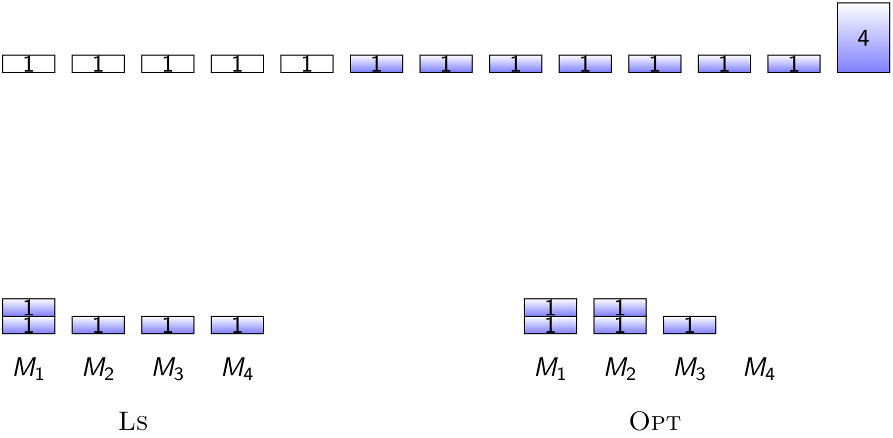
Machine Scheduling – Ls is at best $(2 - \frac{1}{m})$-competitive



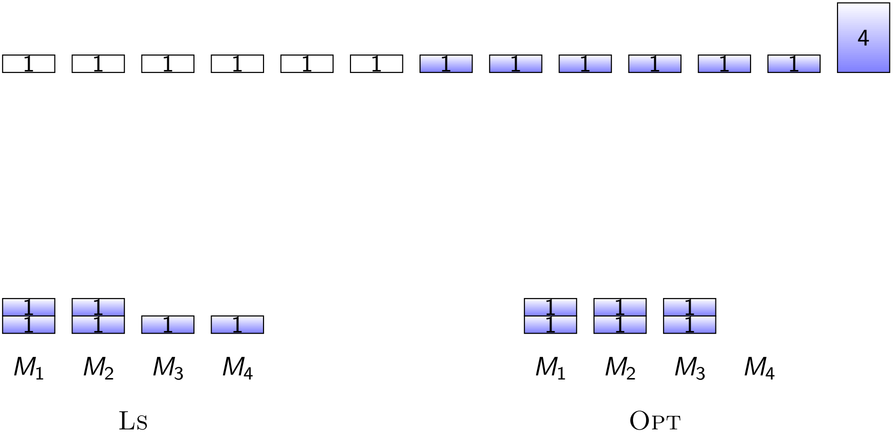
Machine Scheduling – Ls is at best $(2 - \frac{1}{m})$-competitive



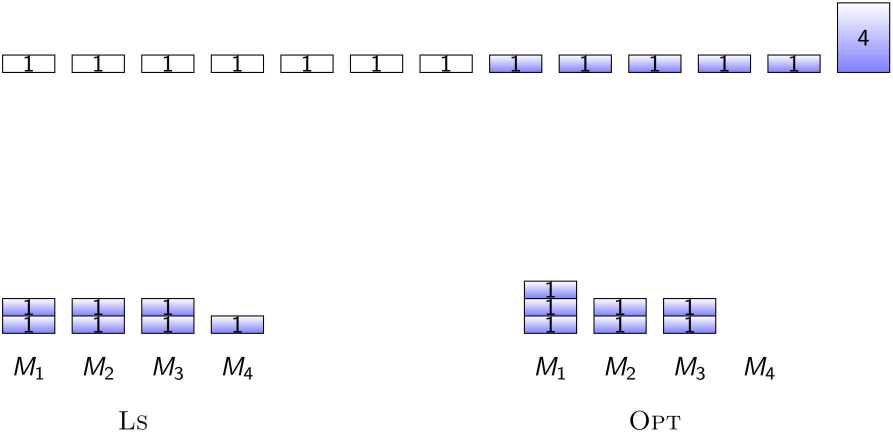
Machine Scheduling – Ls is at best $(2 - \frac{1}{m})$-competitive



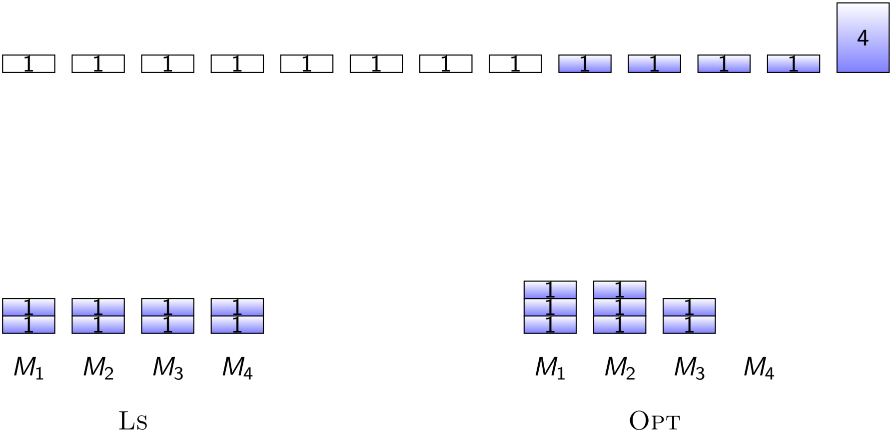
Machine Scheduling – Ls is at best $(2 - \frac{1}{m})$-competitive



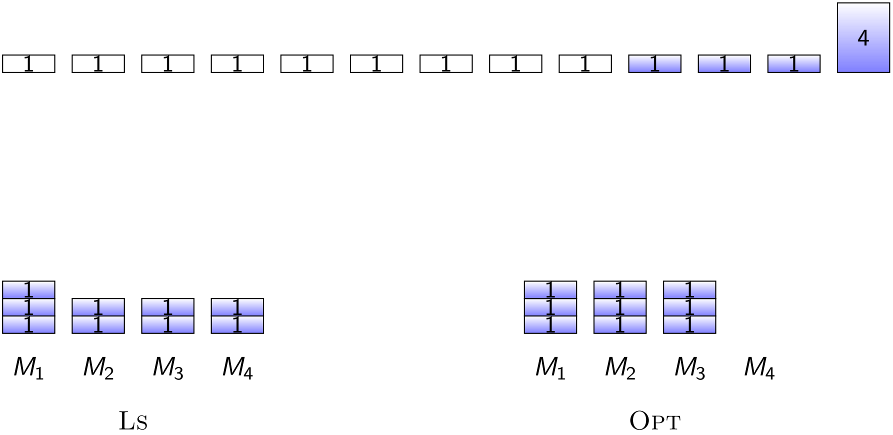
Machine Scheduling – LS is at best $(2 - \frac{1}{m})$-competitive



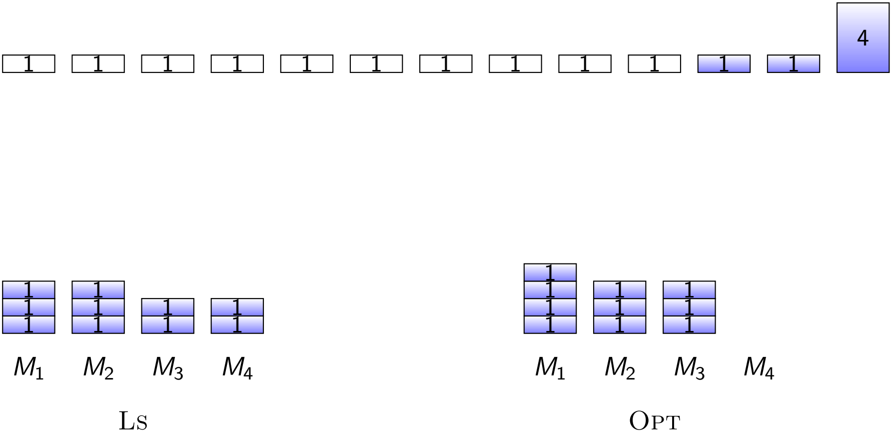
Machine Scheduling – Ls is at best $(2 - \frac{1}{m})$-competitive



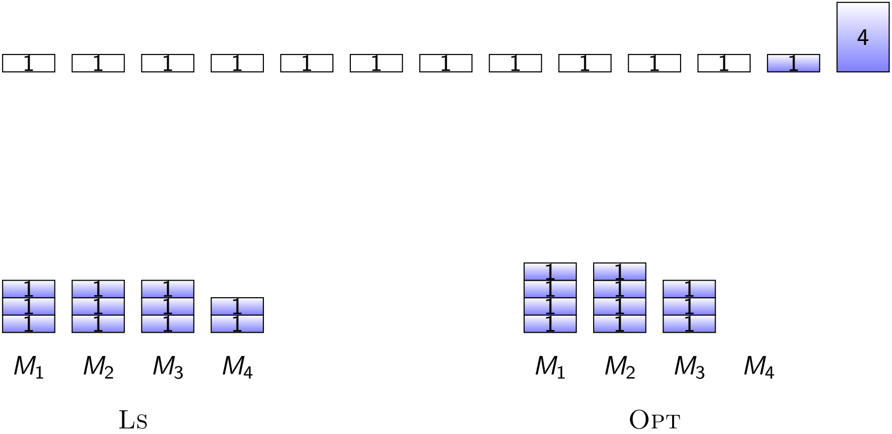
Machine Scheduling – Ls is at best $(2 - \frac{1}{m})$-competitive



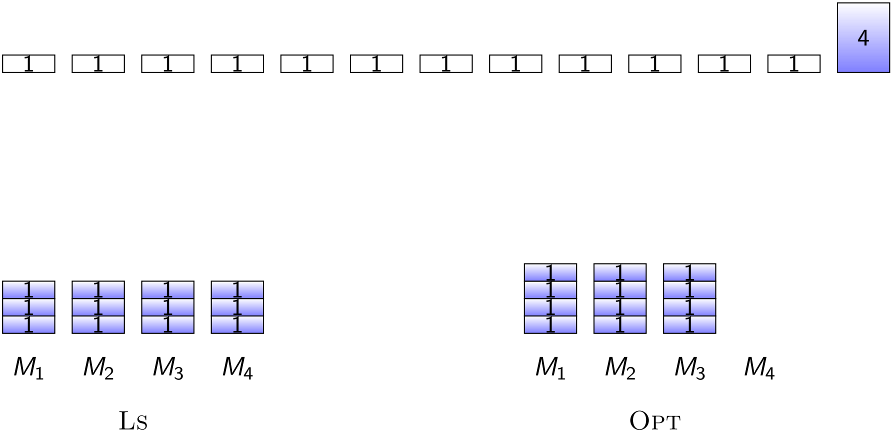
Machine Scheduling – Ls is at best $(2 - \frac{1}{m})$-competitive



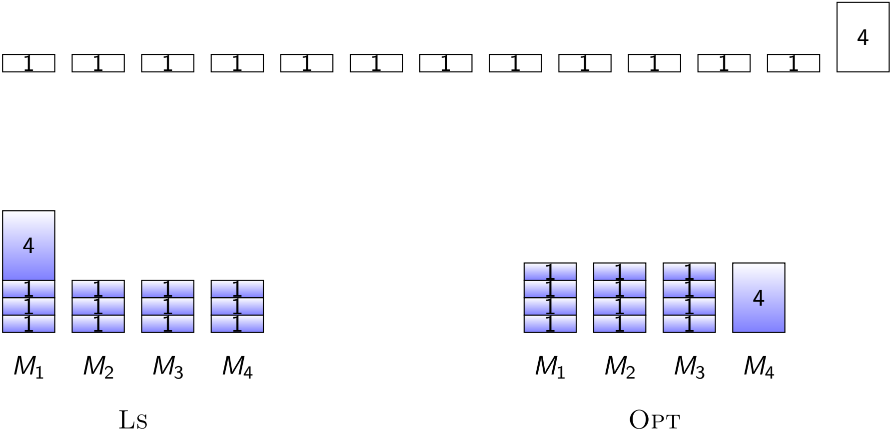
Machine Scheduling – Ls is at best $(2 - \frac{1}{m})$-competitive



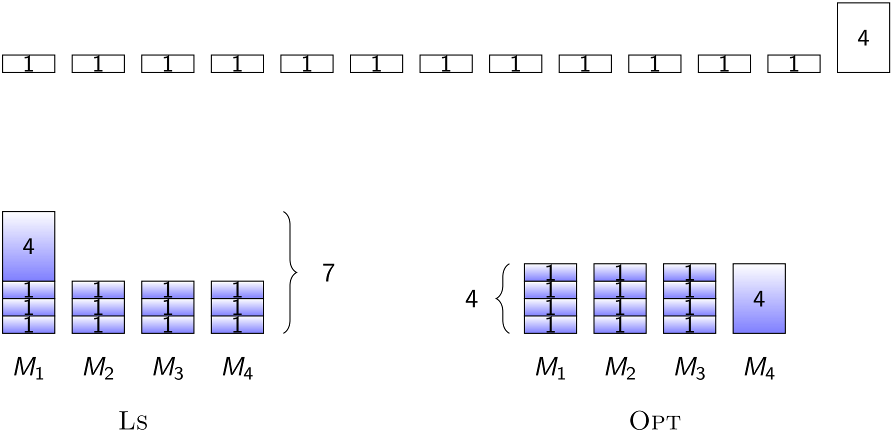
Machine Scheduling – LS is at best $(2 - \frac{1}{m})$-competitive



Machine Scheduling – Ls is at best $(2 - \frac{1}{m})$-competitive



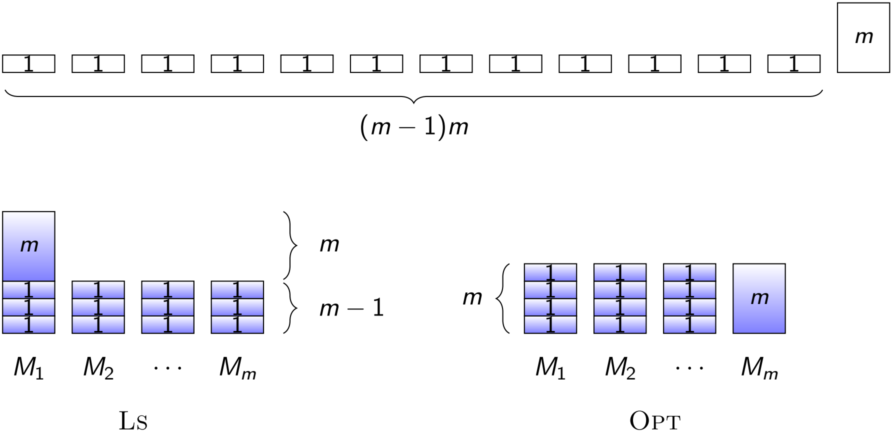
Machine Scheduling – Ls is at best $(2 - \frac{1}{m})$-competitive



Thus,

$$\frac{\textsc{Ls}(I)}{\textsc{Opt}(I)} = \frac{7}{4} = 2 - \frac{1}{4}$$

Machine Scheduling – LS is at best $(2 - \frac{1}{m})$-competitive



In general,

$$\frac{\textsc{Ls}(I)}{\textsc{Opt}(I)} \geq \frac{(m-1)+m}{m} = \frac{2m-1}{m} = 2 - \frac{1}{m}$$

Machine Scheduling - proof techniques

You have just seen a

lower bound proof

In our context, that is relatively easy: Just demonstrate an example (family of examples), where the algorithm performs worse than some ratio.

Machine Scheduling - proof techniques

You have just seen a

lower bound proof

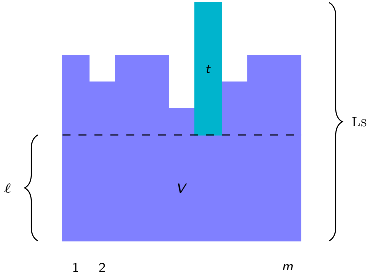
In our context, that is relatively easy: Just demonstrate an example (family of examples), where the algorithm performs worse than some ratio.

Now we want to show the much harder

upper bound proof

We must show that it is *never* worse than a given ratio for *any* of the (potentially infinitely many) possible input sequences.

Machine Scheduling – Ls is $(2 - \frac{1}{m})$-competitive



t is the length of the job starting at ℓ
and ending at the makespan

T is the total length of all jobs

Define $V = \ell \cdot m$

Now conclude:

$\textsc{Opt} \geq T/m$ and $\textsc{Opt} \geq t$

$T \geq V + t$, due to Ls's choice for t

$$
\begin{aligned}
\textsc{Ls} &= \ell + t \\
&\leq \tfrac{T-t}{m} + t, \text{ since } \ell = \tfrac{V}{m} \text{ and } V \leq T - t \\
&= \tfrac{T}{m} + (1 - \tfrac{1}{m})t \\
&\leq \textsc{Opt} + (1 - \tfrac{1}{m})\,\textsc{Opt}, \text{ from } \textsc{Opt} \text{ inequalities above} \\
&= (2 - \tfrac{1}{m})\,\textsc{Opt}
\end{aligned}
$$

Machine Scheduling

SDU

Since we have both an upper and lower bound, we have shown the following:

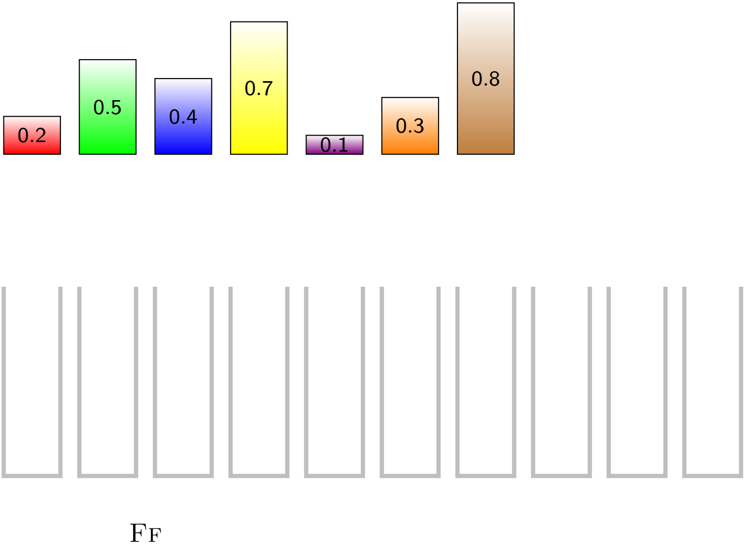
Theorem

The algorithm LS for minimizing makespan in machine scheduling with $m \geq 1$ machines has competitive ratio $2 - \frac{1}{m}$.

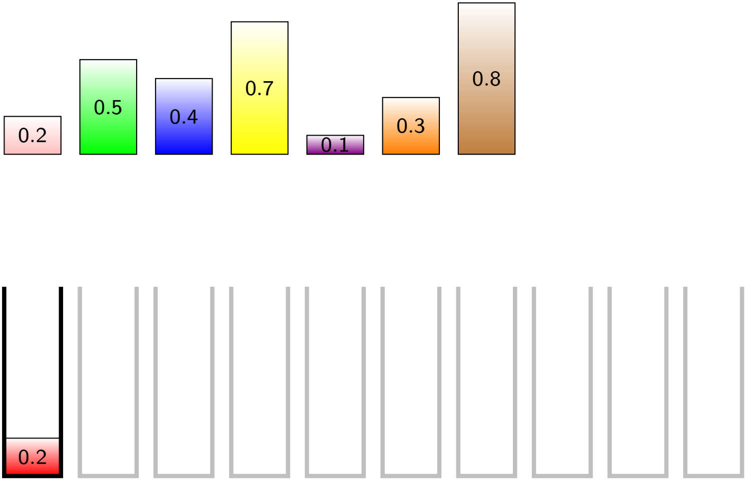
Bin Packing

- Unbounded supply of bins of size 1.
- n items, each of size strictly between zero and one, to be placed in a bin.
- Obviously, the items placed in any given bin cannot be larger than 1 in total.
- The goal is to *minimize the number of bins used*.
- Algorithm First-Fit ($\textsc{Ff}$): place next item in the first bin with enough space. The ordering of the bins is determined by the first time they receive an item.

Bin Packing – FF example

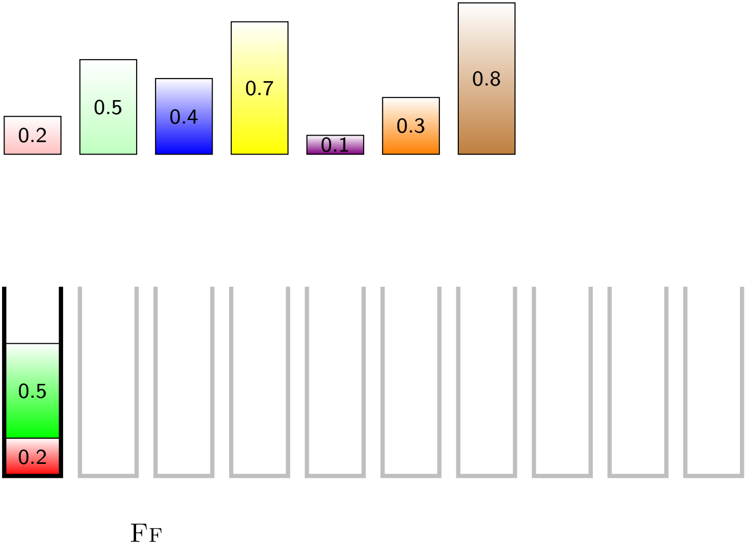


Bin Packing – FF example

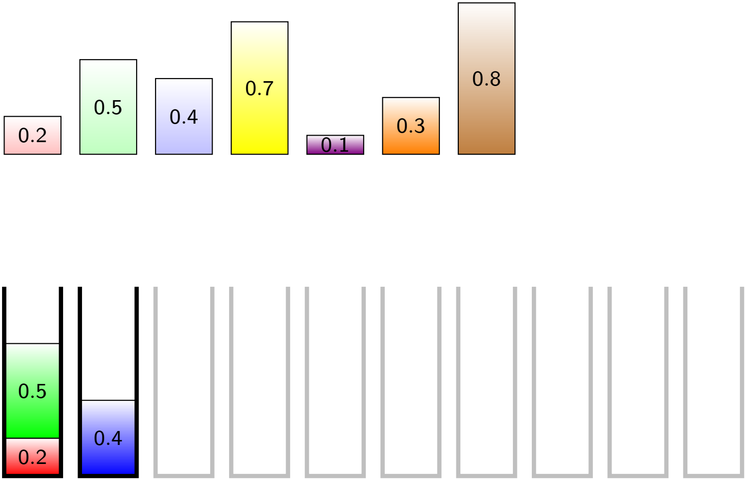


FF

Bin Packing – FF example

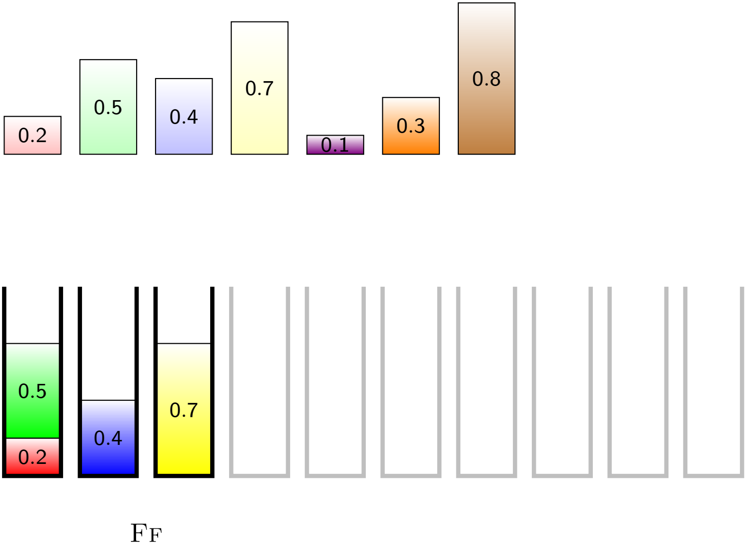


Bin Packing – FF example

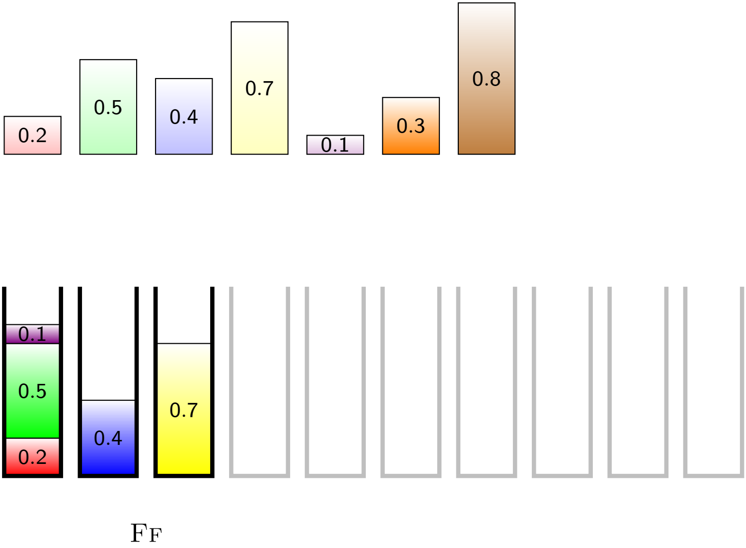


FF

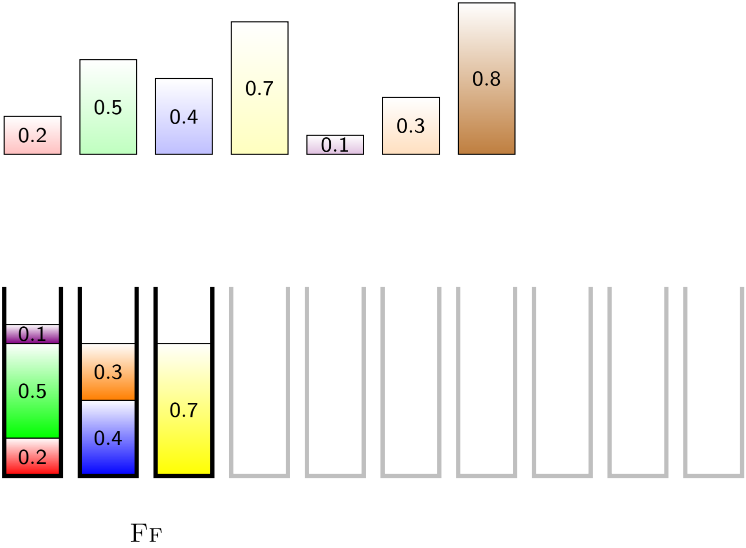
Bin Packing – FF example



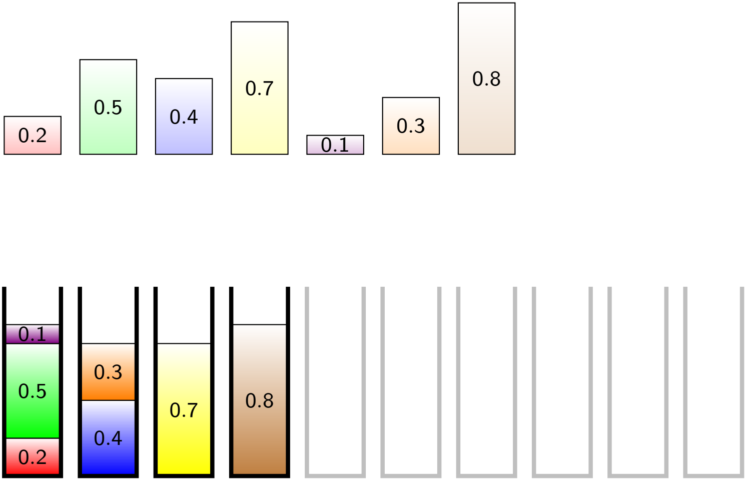
Bin Packing – FF example



Bin Packing – FF example

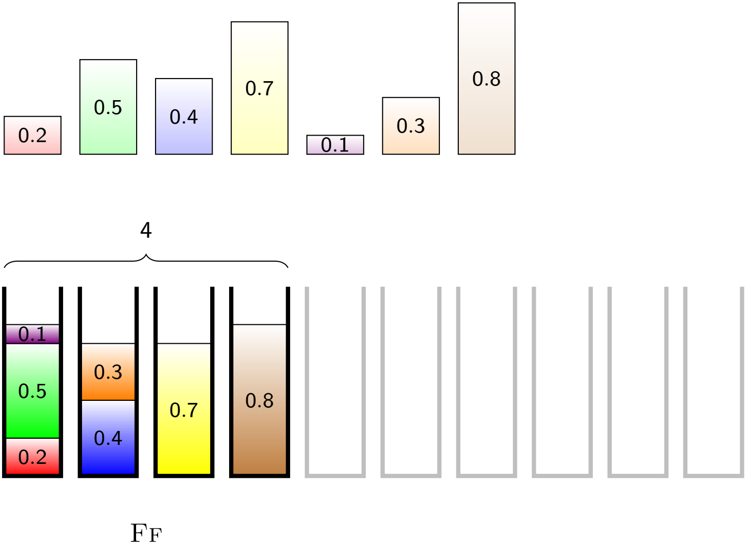


Bin Packing – FF example



FF

Bin Packing – FF example



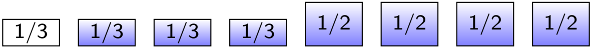
Bin Packing – simple lower bound against Ff

1/3 1/3 1/3 1/3 1/2 1/2 1/2 1/2

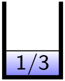
All items are slightly larger than the indicated fraction, e.g., $\frac{1}{3} + \frac{1}{1000}$.

Ff Opt

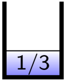
Bin Packing – simple lower bound against FF



All items are slightly larger than the indicated fraction, e.g., $\frac{1}{3} + \frac{1}{1000}$.

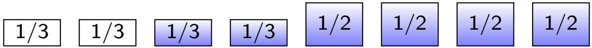


FF

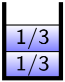


OPT

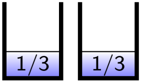
Bin Packing – simple lower bound against FF



All items are slightly larger than the indicated fraction, e.g., $\frac{1}{3} + \frac{1}{1000}$.

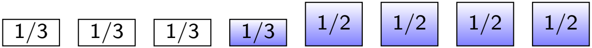


FF

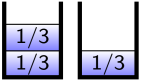


OPT

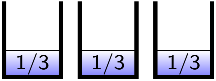
Bin Packing – simple lower bound against FF



All items are slightly larger than the indicated fraction, e.g., $\frac{1}{3} + \frac{1}{1000}$.

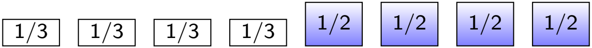


FF

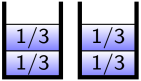


OPT

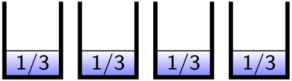
Bin Packing – simple lower bound against Ff



All items are slightly larger than the indicated fraction, e.g., $\frac{1}{3} + \frac{1}{1000}$.

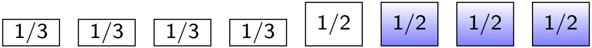


Ff

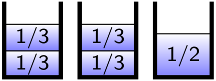


Opt

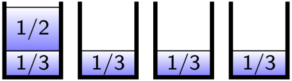
Bin Packing – simple lower bound against Ff



All items are slightly larger than the indicated fraction, e.g., $\frac{1}{3} + \frac{1}{1000}$.

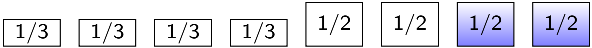


Ff

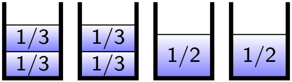


Opt

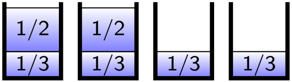
Bin Packing – simple lower bound against $\textsc{Ff}$



All items are slightly larger than the indicated fraction, e.g., $\frac{1}{3} + \frac{1}{1000}$.

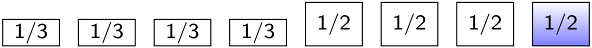


$\textsc{Ff}$

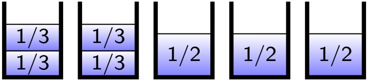


$\textsc{Opt}$

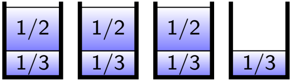
Bin Packing – simple lower bound against FF



All items are slightly larger than the indicated fraction, e.g., $\frac{1}{3} + \frac{1}{1000}$.

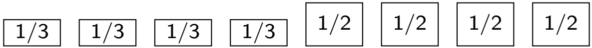


FF

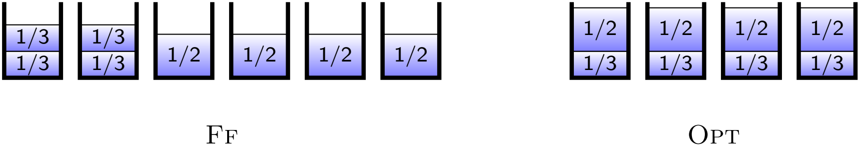


OPT

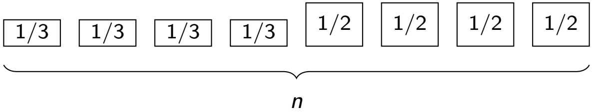
Bin Packing – simple lower bound against FF



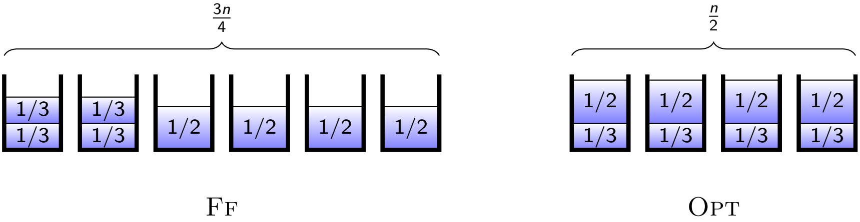
All items are slightly larger than the indicated fraction, e.g., $\frac{1}{3} + \frac{1}{1000}$.



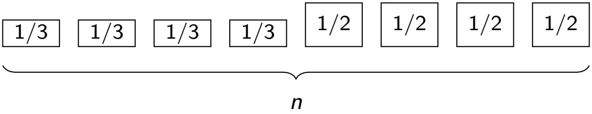
Bin Packing – simple lower bound against FF



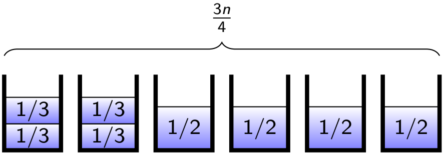
All items are slightly larger than the indicated fraction, e.g., $\frac{1}{3} + \frac{1}{1000}$.



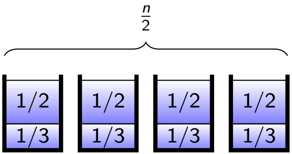
Bin Packing – simple lower bound against FF



All items are slightly larger than the indicated fraction, e.g., $\frac{1}{3} + \frac{1}{1000}$.



FF



OPT

Thus,

$$\frac{\text{FF}(I)}{\text{OPT}(I)} \geq \frac{\frac{n}{4} + \frac{n}{2}}{\frac{n}{2}} = \frac{3}{2}$$

Bin Packing – First-Fit

SDU

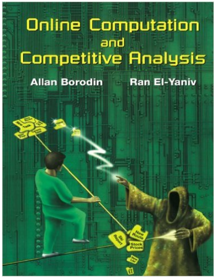
- FF has been shown to be 1.7-competitive [hard].
  Thus, the competitive ratio is *at most* 1.7.
- We have just shown that the competitive ratio is *at least* 1.5.
- So, the competitive ratio of FF is in the interval $[1.5, 1.7]$.
- We'll approach the precise value in the exercises.

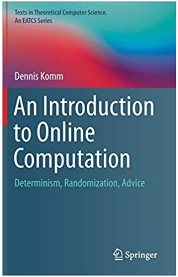
How To Learn More

You'll meet online algorithms in courses, without necessarily being told that they are *online* problems.
A formal treatment of this topic is somewhat abstract and heavy on proofs, and more appropriate for the MS level. At that point, you can take the course

DM860: Online Algorithms

or read one of





IMADA People in Online Algorithms

SDU

Online Algorithms		
	Joan Boyar	Online algorithms, combinatorial optimization, cryptology, computational complexity
	Lene Monrad Favrholdt	Online algorithms, graph algorithms
	Kim Skak Larsen	Online algorithms, algorithms and data structures, database systems, semantics
	Lars Rohwedder	Parameterized, approximation, and online algorithms, integer programming
	Kevin Schewior	Online algorithms, algorithms under uncertainty, approximation algorithms
	Magnus Berg	Online algorithms

Online Algorithms

a topic in

DM573 – Introduction to Computer Science

Kim Skak Larsen

Department of Mathematics and Computer Science (IMADA)
University of Southern Denmark (SDU)

kslarsen@imada.sdu.dk
https://imada.sdu.dk/u/kslarsen/

November 28, 2024