# Lecture 25: Online Algorithms

Michael Dinitz

December 3, 2024
601.433/633 Introduction to Algorithms

# Introduction

Class until now: difficulty was *computational power*

Today: difficulty is *lack of information*

Online:

- Input / data arrives *over time*
- Need to make decisions without knowing future

# Ski Rental Problem

Want to go skiing, but don't know how many times you'll be able to go this year.

Should you rent or buy?

- Renting skis: $50
- Buying skis: $500
- Every day you ski and haven't yet bought, need to decide: rent or buy?

# Ski Rental Problem

Want to go skiing, but don't know how many times you'll be able to go this year.

Should you rent or buy?

- Renting skis: \$50
- Buying skis: \$500
- Every day you ski and haven't yet bought, need to decide: rent or buy?

Buy right away:

# Ski Rental Problem

Want to go skiing, but don't know how many times you'll be able to go this year.

Should you rent or buy?

- Renting skis: \$50
- Buying skis: \$500
- Every day you ski and haven't yet bought, need to decide: rent or buy?

Buy right away:

- If you only ski once, should have rented (\$50), instead bought (\$500)

# Ski Rental Problem

Want to go skiing, but don't know how many times you'll be able to go this year.

Should you rent or buy?

- Renting skis: \$50
- Buying skis: \$500
- Every day you ski and haven't yet bought, need to decide: rent or buy?

Buy right away:

- If you only ski once, should have rented (\$50), instead bought (\$500)

Never buy:

# Ski Rental Problem

Want to go skiing, but don't know how many times you'll be able to go this year.

Should you rent or buy?

- Renting skis: \$50
- Buying skis: \$500
- Every day you ski and haven't yet bought, need to decide: rent or buy?

Buy right away:

- If you only ski once, should have rented (\$50), instead bought (\$500)

Never buy:

- What if you ski $M \approx \infty$ times?
- Should have bought (\$500), instead rented ($M \cdot \$50$)

# Ski Rental Problem

Want to go skiing, but don't know how many times you'll be able to go this year.

Should you rent or buy?

- Renting skis: \$50
- Buying skis: \$500
- Every day you ski and haven't yet bought, need to decide: rent or buy?

Buy right away:

- If you only ski once, should have rented (\$50), instead bought (\$500)

Never buy:

- What if you ski $M \approx \infty$ times?
- Should have bought (\$500), instead rented ($M \cdot \$50$)

What's the right strategy (for these costs)?

# Better Late Than Never

Rent until you realize you should have bought!

BLTN: Rent **9** times, buy on **10**'th.

# Better Late Than Never

Rent until you realize you should have bought!

BLTN: Rent $\mathbf{9}$ times, buy on $\mathbf{10}$'th.

If ski $\leq \mathbf{9}$ times:

# Better Late Than Never

Rent until you realize you should have bought!

BLTN: Rent $\mathbf{9}$ times, buy on $\mathbf{10}$'th.

If ski $\leq \mathbf{9}$ times: optimal

# Better Late Than Never

Rent until you realize you should have bought!

BLTN: Rent $\mathbf{9}$ times, buy on $\mathbf{10}$'th.

If ski $\leq \mathbf{9}$ times: optimal

If ski $\geq \mathbf{10}$ times:

# Better Late Than Never

Rent until you realize you should have bought!

BLTN: Rent $\mathbf{9}$ times, buy on $\mathbf{10}$'th.

If ski $\leq \mathbf{9}$ times: optimal

If ski $\geq \mathbf{10}$ times:

- $\boldsymbol{ALG = 450 + 500 = 950}$
- $\boldsymbol{OPT = 500}$

# Better Late Than Never

Rent until you realize you should have bought!

BLTN: Rent $\mathbf{9}$ times, buy on $\mathbf{10}$'th.

If ski $\leq \mathbf{9}$ times: optimal

If ski $\geq \mathbf{10}$ times:

- $\boldsymbol{ALG = 450 + 500 = 950}$
- $\boldsymbol{OPT = 500}$

Never more than twice (actually $\mathbf{\frac{19}{10}}$ times) what we should have done!

# Competitive Ratio

**Definition**

The *competitive ratio* of algorithm $\boldsymbol{ALG}$ is the maximum over all inputs/futures $\boldsymbol{\sigma}$ of

$$\frac{\boldsymbol{ALG(\sigma)}}{\boldsymbol{OPT(\sigma)}},$$

where $\boldsymbol{ALG(\sigma)}$ is the cost of $\boldsymbol{ALG}$ on $\boldsymbol{\sigma}$ and $\boldsymbol{OPT(\sigma)}$ is the optimal cost for $\boldsymbol{\sigma}$ (knowing the future).

# Competitive Ratio

## Definition

The *competitive ratio* of algorithm $\boldsymbol{ALG}$ is the maximum over all inputs/futures $\boldsymbol{\sigma}$ of

$$\frac{\boldsymbol{ALG(\sigma)}}{\boldsymbol{OPT(\sigma)}},$$

where $\boldsymbol{ALG(\sigma)}$ is the cost of $\boldsymbol{ALG}$ on $\boldsymbol{\sigma}$ and $\boldsymbol{OPT(\sigma)}$ is the optimal cost for $\boldsymbol{\sigma}$ (knowing the future).

So on ski rental problem with previous values, competitive ratio is $\frac{19}{10}$.

# Ski Rental: Generalized

$\$r$ to rent, $\$b$ to buy. Assume $r$ divides $b$ for simplicity.

# Ski Rental: Generalized

$\$r$ to rent, $\$b$ to buy. Assume $r$ divides $b$ for simplicity.

BLTN: Rent $\frac{b}{r} - 1$ times, then buy.

# Ski Rental: Generalized

$\$r$ to rent, $\$b$ to buy. Assume $r$ divides $b$ for simplicity.

BLTN: Rent $\frac{b}{r} - 1$ times, then buy.

**Theorem**

*BLTN has competitive ratio at most* $2 - \frac{r}{b}$.

# Ski Rental: Generalized

$\$r$ to rent, $\$b$ to buy. Assume $r$ divides $b$ for simplicity.

BLTN: Rent $\frac{b}{r} - 1$ times, then buy.

**Theorem**

*BLTN has competitive ratio at most* $2 - \frac{r}{b}$.

**Case 1:** Ski $z \leq \frac{b}{r} - 1$ times

# Ski Rental: Generalized

$\$r$ to rent, $\$b$ to buy. Assume $r$ divides $b$ for simplicity.

BLTN: Rent $\frac{b}{r} - 1$ times, then buy.

**Theorem**

*BLTN has competitive ratio at most* $2 - \frac{r}{b}$.

**Case 1:** Ski $z \leq \frac{b}{r} - 1$ times

- $ALG = z \cdot r$
- $OPT = \min(z \cdot r, b) = z \cdot r$

$\implies \frac{ALG}{OPT} = 1$

# Ski Rental: Generalized

$\$r$ to rent, $\$b$ to buy. Assume $r$ divides $b$ for simplicity.

BLTN: Rent $\frac{b}{r} - 1$ times, then buy.

**Theorem**

*BLTN has competitive ratio at most* $2 - \frac{r}{b}$.

**Case 1:** Ski $z \leq \frac{b}{r} - 1$ times

- $ALG = z \cdot r$
- $OPT = \min(z \cdot r, b) = z \cdot r$

$\implies \frac{ALG}{OPT} = 1$

**Case 2:** Ski $z \geq \frac{b}{r}$ times

# Ski Rental: Generalized

$\$r$ to rent, $\$b$ to buy. Assume $r$ divides $b$ for simplicity.

BLTN: Rent $\frac{b}{r} - 1$ times, then buy.

**Theorem**

*BLTN has competitive ratio at most* $2 - \frac{r}{b}$.

**Case 1:** Ski $z \leq \frac{b}{r} - 1$ times

- $ALG = z \cdot r$
- $OPT = \min(z \cdot r, b) = z \cdot r$

$\implies \frac{ALG}{OPT} = 1$

**Case 2:** Ski $z \geq \frac{b}{r}$ times

- $ALG = r \cdot \left(\frac{b}{r} - 1\right) + b = b - r + b = 2b - r$
- $OPT = \min(r \cdot z, b) = b$

$\implies \frac{ALG}{OPT} = \frac{2b-r}{b} = 2 - \frac{r}{b}$

# Ski Rental: Generalized

$\$r$ to rent, $\$b$ to buy. Assume $r$ divides $b$ for simplicity.

BLTN: Rent $\frac{b}{r} - 1$ times, then buy.

**Theorem**

*BLTN has competitive ratio at most* $2 - \frac{r}{b}$.

**Case 1:** Ski $z \leq \frac{b}{r} - 1$ times

- $ALG = z \cdot r$
- $OPT = \min(z \cdot r, b) = z \cdot r$

$\Longrightarrow \quad \frac{ALG}{OPT} = 1$

**Case 2:** Ski $z \geq \frac{b}{r}$ times

- $ALG = r \cdot \left(\frac{b}{r} - 1\right) + b = b - r + b = 2b - r$
- $OPT = \min(r \cdot z, b) = b$

$\Longrightarrow \quad \frac{ALG}{OPT} = \frac{2b-r}{b} = 2 - \frac{r}{b}$

So for all inputs / futures, $\frac{ALG}{OPT} \leq 2 - \frac{r}{b}$

# Lower Bound

**Theorem**

*No (deterministic) algorithm has competitive ratio better than BLTN.*

# Lower Bound

**Theorem**

*No (deterministic) algorithm has competitive ratio better than BLTN.*

Deterministic $\boldsymbol{ALG}$: "ski $\boldsymbol{x}$ times, then buy".

# Lower Bound

**Theorem**

*No (deterministic) algorithm has competitive ratio better than BLTN.*

Deterministic $ALG$: "ski $x$ times, then buy".
Input: ski $x + 1$ times.

# Lower Bound

**Theorem**

*No (deterministic) algorithm has competitive ratio better than BLTN.*

Deterministic $ALG$: "ski $x$ times, then buy".
Input: ski $x + 1$ times.

**Case 1:** $x \geq b/r$

# Lower Bound

**Theorem**

*No (deterministic) algorithm has competitive ratio better than BLTN.*

Deterministic $ALG$: "ski $x$ times, then buy".
Input: ski $x + 1$ times.

**Case 1:** $x \geq b/r$

- $OPT = \min(b, (x+1)r) = b$

# Lower Bound

**Theorem**

*No (deterministic) algorithm has competitive ratio better than BLTN.*

Deterministic $ALG$: "ski $x$ times, then buy".
Input: ski $x + 1$ times.

**Case 1:** $x \geq b/r$

- $OPT = \min(b, (x+1)r) = b$
- $ALG = xr + b \geq 2b$

# Lower Bound

**Theorem**

*No (deterministic) algorithm has competitive ratio better than BLTN.*

Deterministic $\boldsymbol{ALG}$: "ski $\boldsymbol{x}$ times, then buy".
Input: ski $\boldsymbol{x+1}$ times.

**Case 1:** $\boldsymbol{x \geq b/r}$

- $\boldsymbol{OPT = \min(b, (x+1)r) = b}$
- $\boldsymbol{ALG = xr + b \geq 2b}$

$\Longrightarrow \boldsymbol{\frac{ALG}{OPT} \geq 2 > 2 - \frac{r}{b}}$

# Lower Bound

**Theorem**

*No (deterministic) algorithm has competitive ratio better than BLTN.*

Deterministic $\boldsymbol{ALG}$: "ski $\boldsymbol{x}$ times, then buy".
Input: ski $\boldsymbol{x+1}$ times.

**Case 1:** $\boldsymbol{x \geq b/r}$

- $\boldsymbol{OPT = \min(b, (x+1)r) = b}$
- $\boldsymbol{ALG = xr + b \geq 2b}$

$\Longrightarrow \boldsymbol{\frac{ALG}{OPT} \geq 2 > 2 - \frac{r}{b}}$

**Case 2:** $\boldsymbol{x \leq \frac{b}{r} - 1}$

# Lower Bound

**Theorem**

*No (deterministic) algorithm has competitive ratio better than BLTN.*

Deterministic $\boldsymbol{ALG}$: "ski $\boldsymbol{x}$ times, then buy".
Input: ski $\boldsymbol{x+1}$ times.

**Case 1:** $\boldsymbol{x \geq b/r}$

- $\boldsymbol{OPT = \min(b, (x+1)r) = b}$
- $\boldsymbol{ALG = xr + b \geq 2b}$

$\implies \boldsymbol{\frac{ALG}{OPT} \geq 2 > 2 - \frac{r}{b}}$

**Case 2:** $\boldsymbol{x \leq \frac{b}{r} - 1}$

- $\boldsymbol{OPT = \min(b, (x+1)r) = (x+1)r}$

# Lower Bound

**Theorem**

*No (deterministic) algorithm has competitive ratio better than BLTN.*

Deterministic $ALG$: "ski $x$ times, then buy".
Input: ski $x + 1$ times.

**Case 1:** $x \geq b/r$

- $OPT = \min(b, (x+1)r) = b$
- $ALG = xr + b \geq 2b$

$\implies \frac{ALG}{OPT} \geq 2 > 2 - \frac{r}{b}$

**Case 2:** $x \leq \frac{b}{r} - 1$

- $OPT = \min(b, (x+1)r) = (x+1)r$
- $ALG = xr + b$

# Lower Bound

**Theorem**

*No (deterministic) algorithm has competitive ratio better than BLTN.*

Deterministic $ALG$: "ski $x$ times, then buy".

Input: ski $x+1$ times.

**Case 1:** $x \geq b/r$

- $OPT = \min(b, (x+1)r) = b$
- $ALG = xr + b \geq 2b$

$\Longrightarrow \frac{ALG}{OPT} \geq 2 > 2 - \frac{r}{b}$

**Case 2:** $x \leq \frac{b}{r} - 1$

- $OPT = \min(b, (x+1)r) = (x+1)r$
- $ALG = xr + b$

$$\frac{ALG}{OPT} = \frac{xr+b}{(x+1)r} = \frac{xr+b}{xr+r} = 1 + \frac{b-r}{xr+r}$$

$$\geq 1 + \frac{b-r}{(\frac{b}{r}-1)r + r} = 1 + \frac{b-r}{b} = 2 - \frac{r}{b}$$

# Elevator Problem

Trying to get up a building: takes $\boldsymbol{E}$ seconds by elevator, $\boldsymbol{S}$ seconds by stairs.

- How long should we wait for the elevator?
- Example: $\boldsymbol{E = 15, S = 45}$.

# Elevator Problem

Trying to get up a building: takes $\boldsymbol{E}$ seconds by elevator, $\boldsymbol{S}$ seconds by stairs.

- How long should we wait for the elevator?
- Example: $\boldsymbol{E} = \mathbf{15}, \boldsymbol{S} = \mathbf{45}$.

BLTN: Wait $\boldsymbol{S} - \boldsymbol{E}$ seconds, then give up and take stairs

# Elevator Problem

Trying to get up a building: takes $\boldsymbol{E}$ seconds by elevator, $\boldsymbol{S}$ seconds by stairs.

- How long should we wait for the elevator?
- Example: $\boldsymbol{E = 15, S = 45}$.

BLTN: Wait $\boldsymbol{S - E}$ seconds, then give up and take stairs

If elevator arrives at $\boldsymbol{x \leq S - E}$:

# Elevator Problem

Trying to get up a building: takes $E$ seconds by elevator, $S$ seconds by stairs.

- How long should we wait for the elevator?
- Example: $E = 15, S = 45$.

BLTN: Wait $S - E$ seconds, then give up and take stairs

If elevator arrives at $x \leq S - E$:

- $OPT = \min(S, x + E) = x + E$
- $ALG = x + E$

$\implies \frac{ALG}{OPT} = 1$

# Elevator Problem

Trying to get up a building: takes $E$ seconds by elevator, $S$ seconds by stairs.

- How long should we wait for the elevator?
- Example: $E = 15, S = 45$.

BLTN: Wait $S - E$ seconds, then give up and take stairs

If elevator arrives at $x \leq S - E$:

- $OPT = \min(S, x + E) = x + E$
- $ALG = x + E$

$\implies \frac{ALG}{OPT} = 1$

If elevator arrives at $x > S - E$:

# Elevator Problem

Trying to get up a building: takes $\boldsymbol{E}$ seconds by elevator, $\boldsymbol{S}$ seconds by stairs.

- How long should we wait for the elevator?
- Example: $\boldsymbol{E = 15, S = 45}$.

BLTN: Wait $\boldsymbol{S - E}$ seconds, then give up and take stairs

If elevator arrives at $\boldsymbol{x \leq S - E}$:

- $\boldsymbol{OPT = \min(S, x + E) = x + E}$
- $\boldsymbol{ALG = x + E}$

$\Longrightarrow \quad \boldsymbol{\frac{ALG}{OPT} = 1}$

If elevator arrives at $\boldsymbol{x > S - E}$:

- $\boldsymbol{OPT = \min(S, x + E) = S}$
- $\boldsymbol{ALG = (S - E) + S = 2S - E}$

$\Longrightarrow \quad \boldsymbol{\frac{ALG}{OPT} = \frac{2S-E}{S} = 2 - \frac{E}{S}}$

# Extensions: Randomization

What if our algorithm is allowed to be randomized?

- Choose our buying time from some distribution $\boldsymbol{p}$ over days (*purchase distribution*)
- Adversary knows $\boldsymbol{p}$, but not our random draw from $\boldsymbol{p}$ (*oblivious* adversary).
- Normalize so $\boldsymbol{r} = \mathbf{1}$

# Extensions: Randomization

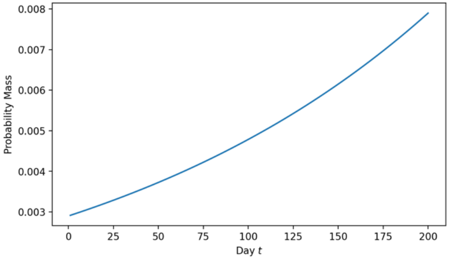
What if our algorithm is allowed to be randomized?

- Choose our buying time from some distribution $\boldsymbol{p}$ over days (*purchase distribution*)
- Adversary knows $\boldsymbol{p}$, but not our random draw from $\boldsymbol{p}$ (*oblivious* adversary).
- Normalize so $\boldsymbol{r} = \mathbf{1}$

**Theorem (Karlin, Manasse, McGeoch, Owicki (SODA '90))**

*If we set* $\boldsymbol{p_t} = \left(\frac{b-1}{b}\right)^{b-t} \frac{1}{b\left(1-\left(1-\frac{1}{b}\right)^{b}\right)}$ *for all* $\boldsymbol{t \le b}$*, then*

- $\frac{E[ALG(\sigma)]}{OPT(\sigma)} \le \frac{e}{e-1} \approx \mathbf{1.58}$ *for all* $\boldsymbol{\sigma}$*, and*
- *this is optimal.*



# Extensions: Randomization

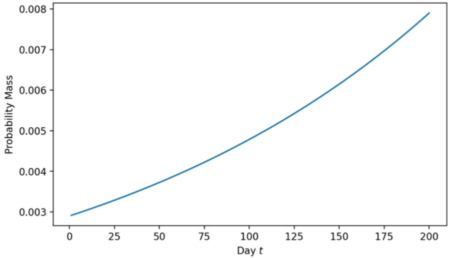
What if our algorithm is allowed to be randomized?

- Choose our buying time from some distribution $\boldsymbol{p}$ over days (*purchase distribution*)
- Adversary knows $\boldsymbol{p}$, but not our random draw from $\boldsymbol{p}$ (*oblivious* adversary).
- Normalize so $\boldsymbol{r} = \mathbf{1}$

**Theorem (Karlin, Manasse, McGeoch, Owicki (SODA '90))**

*If we set* $\boldsymbol{p_t} = \left(\frac{b-1}{b}\right)^{b-t} \frac{1}{b\left(1-\left(1-\frac{1}{b}\right)^{b}\right)}$ *for all* $\boldsymbol{t} \leq \boldsymbol{b}$, *then*

- $\frac{E[ALG(\sigma)]}{OPT(\sigma)} \leq \frac{e}{e-1} \approx \mathbf{1.58}$ *for all* $\boldsymbol{\sigma}$, *and*
- *this is optimal.*




Good in expectation, but what about probability of being bad (e.g., worse than BLTN)?

# Extensions: Randomization

$\boldsymbol{Pr}$[worse than BLTN] $\approx \mathbf{0.3775}$: quite large!

# Extensions: Randomization

$\boldsymbol{Pr}[\text{worse than BLTN}] \approx \mathbf{0.3775}$: quite large!

**Definition**

$(\gamma, \delta)$-tail bound: probability at most $\delta$ of having competitive ratio larger than $\gamma$

# Extensions: Randomization

$\boldsymbol{Pr}[\text{worse than BLTN}] \approx \mathbf{0.3775}$: quite large!

**Definition**

$(\gamma, \delta)$-tail bound: probability at most $\delta$ of having competitive ratio larger than $\gamma$

We introduced and studied recently: Dinitz, Im, Lavastida, Moseley, Vassilvitskii [SODA '24]

# Extensions: Randomization

$\boldsymbol{Pr}$[worse than BLTN] $\approx$ **0.3775**: quite large!

**Definition**

$(\gamma, \delta)$-tail bound: probability at most $\delta$ of having competitive ratio larger than $\gamma$

We introduced and studied recently: Dinitz, Im, Lavastida, Moseley, Vassilvitskii [SODA '24]

- Given collection of tail bounds, algorithm to compute optimal randomized algorithm satisfying them.
- They can look crazy!

# Extensions: Randomization

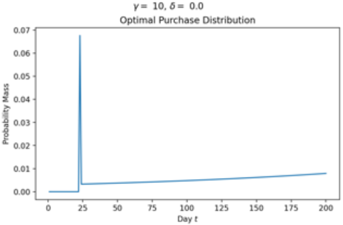
$\boldsymbol{Pr}$[worse than BLTN] $\approx$ **0.3775**: quite large!

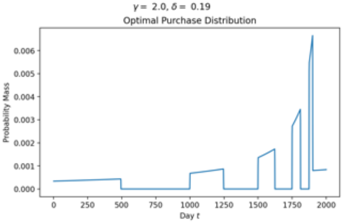
## Definition

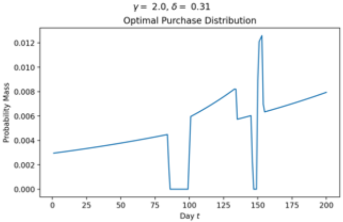
$(\gamma, \delta)$-tail bound: probability at most $\delta$ of having competitive ratio larger than $\gamma$

We introduced and studied recently: Dinitz, Im, Lavastida, Moseley, Vassilvitskii [SODA '24]

- Given collection of tail bounds, algorithm to compute optimal randomized algorithm satisfying them.
- They can look crazy!



# Paging

Classical problem in computer systems/theory

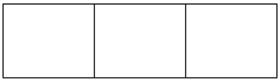
- Disk (slow) with $N$ pages
- Memory (fast) with room for $k < N$ pages
- If OS/application requests a page not in memory: "page fault"
  - Need to bring requested page into memory, *evict* a page from memory (if currently full)
- **Question:** What to evict?

# Paging

Classical problem in computer systems/theory

- Disk (slow) with $N$ pages
- Memory (fast) with room for $k < N$ pages
- If OS/application requests a page not in memory: “page fault”
  - Need to bring requested page into memory, *evict* a page from memory (if currently full)
- **Question:** What to evict?

**Example:** $k = 3$. Requests: $\mathbf{1, 2, 3, 2, 4, 3, 4, 1, 2, 3, 4}$



(Convention: initial page faults to fill table don’t count: only pay when we *evict* a page)

# LRU

Standard algorithm: "Least Recently Used" (LRU)

- Evict page from memory that hasn't been used in the longest time

# LRU

Standard algorithm: "Least Recently Used" (LRU)

- Evict page from memory that hasn't been used in the longest time
- Intuition:
  - Want to evict page that's next used furthest in the future. But don't know future!
  - Hope that since it hasn't been used for a long time, won't be requested again for a long time.

# LRU

Standard algorithm: "Least Recently Used" (LRU)

- Evict page from memory that hasn't been used in the longest time
- Intuition:
  - Want to evict page that's next used furthest in the future. But don't know future!
  - Hope that since it hasn't been used for a long time, won't be requested again for a long time.

Is this a good algorithm? What's the competitive ratio? Cost = # evictions.

# LRU

Standard algorithm: "Least Recently Used" (LRU)

- Evict page from memory that hasn't been used in the longest time
- Intuition:
  - Want to evict page that's next used furthest in the future. But don't know future!
  - Hope that since it hasn't been used for a long time, won't be requested again for a long time.

Is this a good algorithm? What's the competitive ratio? Cost = # evictions.

- $\boldsymbol{k = 3}$, $\boldsymbol{N = 4}$

# LRU

Standard algorithm: "Least Recently Used" (LRU)

- Evict page from memory that hasn't been used in the longest time
- Intuition:
  - Want to evict page that's next used furthest in the future. But don't know future!
  - Hope that since it hasn't been used for a long time, won't be requested again for a long time.

Is this a good algorithm? What's the competitive ratio? Cost = # evictions.

- $k = 3$, $N = 4$
- Requests: $\mathbf{1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, \ldots}$

# LRU

Standard algorithm: “Least Recently Used” (LRU)

- Evict page from memory that hasn’t been used in the longest time
- Intuition:
  - Want to evict page that’s next used furthest in the future. But don’t know future!
  - Hope that since it hasn’t been used for a long time, won’t be requested again for a long time.

Is this a good algorithm? What’s the competitive ratio? Cost $= \#$ evictions.

- $k = 3$, $N = 4$
- Requests: $1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, \dots$

So LRU has competitive ratio $\approx k$

- LRU evicts every time, $OPT$ evicts $1$ out of every $k$ times.

# Lower Bound

**Theorem**

*No deterministic algorithm has competitive ratio less than $k$.*

Let $ALG$ be some deterministic algorithm. Set $N = k + 1$

# Lower Bound

**Theorem**

*No deterministic algorithm has competitive ratio less than $k$.*

Let $ALG$ be some deterministic algorithm. Set $N = k + 1$

Request sequence:

# Lower Bound

**Theorem**

*No deterministic algorithm has competitive ratio less than $k$.*

Let $ALG$ be some deterministic algorithm. Set $N = k + 1$
Request sequence: Whatever is not in memory for $ALG$!

# Lower Bound

**Theorem**

*No deterministic algorithm has competitive ratio less than $k$.*

Let $ALG$ be some deterministic algorithm. Set $N = k + 1$
Request sequence: Whatever is not in memory for $ALG$!

$\implies$ $ALG$ has an eviction every time (after initialization)

# Lower Bound

**Theorem**

*No deterministic algorithm has competitive ratio less than $k$.*

Let $ALG$ be some deterministic algorithm. Set $N = k + 1$
Request sequence: Whatever is not in memory for $ALG$!

$\implies$ $ALG$ has an eviction every time (after initialization)

$OPT$: evict page whose next request is furthest in the future

- Every page in memory needs to be requested before next eviction. So next eviction is in at least $k$ steps.

# Lower Bound

**Theorem**

*No deterministic algorithm has competitive ratio less than $\boldsymbol{k}$.*

Let $\boldsymbol{ALG}$ be some deterministic algorithm. Set $\boldsymbol{N = k + 1}$
Request sequence: Whatever is not in memory for $\boldsymbol{ALG}$!

$\implies \boldsymbol{ALG}$ has an eviction every time (after initialization)

$\boldsymbol{OPT}$: evict page whose next request is furthest in the future

- Every page in memory needs to be requested before next eviction. So next eviction is in at least $\boldsymbol{k}$ steps.

$$\implies \frac{\boldsymbol{ALG}}{\boldsymbol{OPT}} \geq \boldsymbol{k}$$

# Marking Algorithm

Get around lower bound by using *randomization*

- Lower bound argument doesn't apply because can't set request sequence to ask for whatever's not in memory, since that involved randomness! (Oblivious adversary)

# Marking Algorithm

Get around lower bound by using *randomization*

- Lower bound argument doesn't apply because can't set request sequence to ask for whatever's not in memory, since that involved randomness! (Oblivious adversary)

Assume memory initially $1, 2, \ldots, k$.
Set all pages in memory to be "unmarked"

When page requested:

- If already in memory, "mark" it
- If not in memory:
  - If all pages in memory "marked", unmark all
  - Choose an *unmarked* page uniformly at random to evict
  - Bring in new page, mark it

# Marking Analysis

### Theorem

*Expected competitive ratio at most* $\boldsymbol{O(\log k)}$*:*

$$\frac{\boldsymbol{E[ALG(\sigma)]}}{\boldsymbol{OPT(\sigma)}} \leq \boldsymbol{O(\log k)} \text{ for all request sequences } \boldsymbol{\sigma}.$$

# Marking Analysis

**Theorem**

*Expected competitive ratio at most* $O(\log k)$*:*

$$\frac{E[ALG(\sigma)]}{OPT(\sigma)} \leq O(\log k) \text{ for all request sequences } \sigma.$$

Proof sketch for $N = k + 1$: full generality more complicated

# Marking Analysis

**Theorem**

*Expected competitive ratio at most $O(\log k)$:*

$$\frac{E[ALG(\sigma)]}{OPT(\sigma)} \leq O(\log k) \text{ for all request sequences } \sigma.$$

Proof sketch for $N = k + 1$: full generality more complicated

*Phase*: time between "unmark all" events.

# Marking Analysis

**Theorem**

*Expected competitive ratio at most* $O(\log k)$*:*

$$\frac{E[ALG(\sigma)]}{OPT(\sigma)} \leq O(\log k) \text{ for all request sequences } \sigma.$$

Proof sketch for $N = k + 1$: full generality more complicated

*Phase*: time between "unmark all" events.

In each phase:

- $OPT \geq 1$, since all $N$ pages requested

# *ALG* in each phase

Key point: the one page *not* in memory is *uniformly distributed* among all *unmarked* pages.

# *ALG* in each phase

Key point: the one page *not* in memory is *uniformly distributed* among all *unmarked* pages.

When page requested:

- If marked: in memory, no eviction
- If unmarked: if currently $i$ unmarked pages, then $Pr[\text{eviction}] = Pr[\text{requested page not in memory}] = 1/i$
  - Becomes marked

# *ALG* in each phase

Key point: the one page *not* in memory is *uniformly distributed* among all *unmarked* pages.

When page requested:

- If marked: in memory, no eviction
- If unmarked: if currently $i$ unmarked pages, then $Pr[\text{eviction}] = Pr[\text{requested page not in memory}] = 1/i$
  - Becomes marked

At beginning of phase $i = N$, at end of phase $i = 1$. Goes down by one every time page gets marked.

# $\boldsymbol{ALG}$ in each phase

Key point: the one page *not* in memory is *uniformly distributed* among all *unmarked* pages.

When page requested:

- If marked: in memory, no eviction
- If unmarked: if currently $\boldsymbol{i}$ unmarked pages, then $\boldsymbol{Pr}[\text{eviction}] = \boldsymbol{Pr}[\text{requested page not in memory}] = \mathbf{1}/\boldsymbol{i}$
  - Becomes marked

At beginning of phase $\boldsymbol{i} = \boldsymbol{N}$, at end of phase $\boldsymbol{i} = \mathbf{1}$. Goes down by one every time page gets marked.

$\Longrightarrow$ expected cost in phase at most $\frac{1}{N} + \frac{1}{N-1} + \frac{1}{N-2} + \cdots + \frac{1}{2} + 1 = O(\log N) = O(\log k)$