Online Algorithms

Algorithm Design and Analysis

Hello

Nabi Efe Çekirge, April 8th 2025

Apply to be a TA!

- You, yes you, should apply to be a 15-451/651 TA in the fall semester.
- Fill out the following form by **Thursday, April 17th**:
  - https://forms.gle/g8UurLt9TZGxmDE9A

Signposting

- What are online algorithms?
  - Analyzing online algorithms: Competitive ratio
- Example online problems
  - Rent or buy
  - List update
    - Potential function analysis
  - Page caching
    - Randomization

What are online algorithms?

- In the online framework, our input is presented **one by one**.
- At each time step, our algorithm must make a decision.
  - Each decision will have a cost.
- **Formally:**
  - The input is a sequence $\sigma_1, \sigma_2, \ldots, \sigma_n$. This is **invisible** to the algorithm.
  - For $i = 1, 2, \ldots, n$:
    - The algorithm is presented σ_i, and makes a decision.
    - It incurs a cost c_i based on this decision and the problem **cost model**.
- **Goal:**
  - Make **good** decisions with the current information.

The Rent or Buy Problem

- You want to ski for the upcoming days, as much as possible. However, you don't have skis and the season might end any day.

- Every day, you have the option of either:
  - **Renting** skis for $\$50$ for one day
  - **Buying** skis for $\$500$

- **Formally:**
  - **Input sequence:** $\text{good}, \ldots, \text{good}, \text{bad}.$
  - **Cost model:** $c_i = \begin{cases} 50 & \text{renting}, \\ 500 & \text{buying}, \\ 0 & \text{already bought}. \end{cases}$

Possible Algorithms

Always rent

Just buy

- Which one is **better**? We need to analyze them.

Analyzing Online Algorithms

- We define an optimal **omniscient** algorithm OPT.
  - This algorithm can see the whole input in advance and decide accordingly.
  - The cost of this algorithm on the input is $C_{\mathrm{OPT}}(\sigma)$.
- The **competitive ratio** of an algorithm ALG is

$$\max_{\sigma} \frac{C_{\mathrm{ALG}}(\sigma)}{C_{\mathrm{OPT}}(\sigma)} = c$$

over all inputs σ.

Omniscient Algorithm for Rent or Buy

- What is the optimal omniscient algorithm for the rent or buy problem if we know ski season is going to last n days?
  - Remember that renting costs $\$50$ and buying costs $\$500$.

if $n < 10$: just rent
else : buy

Competitive Analysis

- **Buy immediately:**
  - What is the worst case? $n = 1$
  - What is the competitive ratio?

$$\frac{C_{ALG}}{C_{OPT}} = \frac{500}{50} = 10$$

- **Rent forever:**
  - What is the worst case? $n \to \infty$
  - What is the competitive ratio?

$$\frac{C_{ALG}}{C_{OPT}} = \frac{50n}{500} \to \infty$$

A Better Algorithm Strategy

- **Observation:**
  - Every strategy can be characterized as “Buy on day k”.
- **Question:**
  - What is the worst case input if we buy on day k? $n = k$
- **Buy on day 5:**

$$\frac{C_{ALG}}{C_{OPT}} = \frac{4 \cdot 50 + 500}{250} = \frac{700}{250} > 2 \ldots$$

- **Buy on day 15:**

$$\frac{C_{ALG}}{C_{OPT}} = \frac{14 \cdot 50 + 500}{500} = \frac{1200}{500} = 2 \ldots$$

Better Late Than Never

- **Intuition:**
  - We shouldn't plan to rent for more than how much buying it would cost.
  - We also shouldn't plan on buying too quickly, in case the season ends early.
- **Algorithm**:
  - Buy on the day that renting costs catch up to buying costs (in our example, day 10).
- **Theorem**:
  - Better-late-than-never is 2-**competitive**.

if $n < 10$:

$C_{ALG} = C_{OPT}$

if $n \geq 10$: $\frac{C_{ALG}}{C_{OPT}} = \frac{9 \cdot 50 + 500}{500} = \frac{950}{500} = 1.9 < 2$

Generalized Problem

- **Generalized cost model:**
  - We will now say that renting costs $\$r$ and buying costs $\$b$.
- **Generalized Algorithm:**
  - We now buy on day $\lceil b/r \rceil$. We will assume for simplicity that r divides b.
- **Theorem:**
  - Generalized better-late-than-never is $(2 - r/b)$**-competitive**.

The List Update Problem

$\{2, 1, 3 \text{ -- } n\}$

- You have a list of n items $\{1, 2, \ldots, n\}$ and two operations:
  - Access(x): Access element x. The cost is the position of x in list.
  - Swap(x, y): Swap **adjacent** elements x and y. The cost is 1.
- The input is a sequence of t Access requests for $t \gg n$.
- The algorithm has a chance to do any number of Swaps **after** each request.
- **Goal:**
  - Have the minimum cost after $\cancel{n}$ t requests.

Possible Algorithms

Never swap

Put frequent requests up front

Do No Swaps

- What is the worst case? Acc(n), Acc(n), - - - -
- What is the competitive ratio?

$$C_{ALG} = \underline{n \cdot t}$$

$$C_{OPT} = n + (n-1) + \underline{(t-1)}$$

$$c = \frac{C_{ALG}}{C_{OPT}} = \Theta(n)$$

Do No Swaps

- What is the worst case? **Access(n)** t times.

- What is the competitive ratio?

  We have to access the last element each time, so

  $$C_{\text{ALG}} = n \cdot t\,.$$

  The optimal algorithm will move n to the front after the first access, so

  $$C_{\text{OPT}} = n + (n-1) + (t-1)\,.$$

  So, the competitive ratio is

  $$c = \frac{C_{\text{ALG}}}{C_{\text{OPT}}} = \frac{nt}{2n+t-2} \in \Theta(n)\,.$$

Single Exchange

$1, 2, \ldots, n-1, n \rightleftarrows 1, 2, \ldots, n, n-1$

- What is the worst case? $Acc(n), Acc(n-1), Acc(n), \ldots$
- What is the competitive ratio?

$$C_{ALG} = \underline{(n+1)} \cdot t$$

$$C_{OPT} = n + 2 \cdot (n-2) + \underline{(1+2)} \cdot \frac{t-1}{2}$$

$$c = \frac{C_{ALG}}{C_{OPT}} \in \Theta(n)$$

Single Exchange

- What is the worst case? **Access**(n), **Access**($n-1$), **Access**(n), **Access**($n-1$), ...

- What is the competitive ratio?

  We have to access the last element each time and move it forward, so

$$C_{\mathrm{ALG}} = (n+1) \cdot t\,.$$

  The optimal algorithm will move n and $n-1$ to the front as soon as possible, so

$$C_{\mathrm{OPT}} = n + 2 \cdot (n-2) + (1+2) \cdot \frac{t-1}{2}\,.$$

  So, the competitive ratio is

$$c = \frac{C_{\mathrm{ALG}}}{C_{\mathrm{OPT}}} \in \Theta(n)\,.$$

Frequency Count

- What is the worst case? Acc(1) t times, Acc(2) t times, - - - -
- What is the competitive ratio?

$$C_{ALG} = 1 \cdot t + 2 \cdot t + \dots + n \cdot t \in \Theta(n^2 t)$$

$$C_{OPT} = \underline{t} + 1 + \underline{t} + 2 + \underline{t} \dots (n-1) + \underline{t} \in \Theta(nt)$$

$$c = \frac{C_{ALG}}{C_{OPT}} \in \Theta(n)$$

Frequency Count

- What is the worst case? **Access**(1) t times, **Access**(2) t times, **Access**(3) t times, …
- What is the competitive ratio?

  We actually never make swaps, so

$$C_{\text{ALG}} = 1 \cdot t + 2 \cdot t + \cdots + n \cdot t \in \Theta(n^2 t)\,.$$

  The optimal algorithm will move each element to the front before it gets accessed, so

$$C_{\text{OPT}} = t + 1 + t + 2 + t + \cdots + n - 1 + t \in \Theta(nt)\,.$$

  So, the competitive ratio is

$$c = \frac{C_{\text{ALG}}}{C_{\text{OPT}}} \in \Theta(n)\,.$$

Move to Front

- **Algorithm:**
  - After receiving **Access**(x), move x to the front.
- **Theorem:**
  - Move-to-front is 4-**competitive**.

Analysis

- **Reminders:**
  - We are not interested in the worst case cost of Move-to-front over different inputs. We are interested in the **worst-case ratio** of its cost to **any** other algorithm.
  - If we do not now what the optimal omniscient algorithm is, we can instead bound the ratio against **any** algorithm.
- **Analysis**:
  - Let C_{MTF} be the cost of Move-to-front on $\boldsymbol{\sigma}$, and C_{B} be the cost of **any** algorithm $\boldsymbol{B}$ on $\boldsymbol{\sigma}$.
  - We want to show that $C_{\text{MTF}} \leq 4 \cdot C_{\text{B}}$.
- **Key point:**
  - Not all Access operations cost the same. However, we only care about the **total cost** over many operations.

Enter Potentials

Enter Potentials

- We can define a potential function Φ for our online algorithms!
  - This potential function depends on the state of our algorithm, **and** on the state of the competing algorithm B.

- We can now define AC_{MTF} as the **amortized cost** of Move-to-front.

- Now, we can instead prove $AC_{\text{MTF}} \leq 4 \cdot C_{\text{B}}$.
  - If we define Φ such that $\Phi \geq 0$ and $\Phi_i = 0$, this means that $C_{\text{MTF}} \leq AC_{\text{MTF}} \leq 4 \cdot C_{\text{B}}$.

- Generally, we will prove that our amortized cost of **each individual operation** is at most 4 times the cost of the same operation for B.

Picking a Good Potential

- What affects the relative cost of Move-to-front and B?
- **Answer:** How different the two lists currently are.
  - The potential Φ should be **high** when the lists are very **different**.
  - The potential Φ should be **low** when the lists are very **similar**.
- What determines the difference between lists?

inversion between the lists

a - - - b
b - - - a

- What should our potential function be?

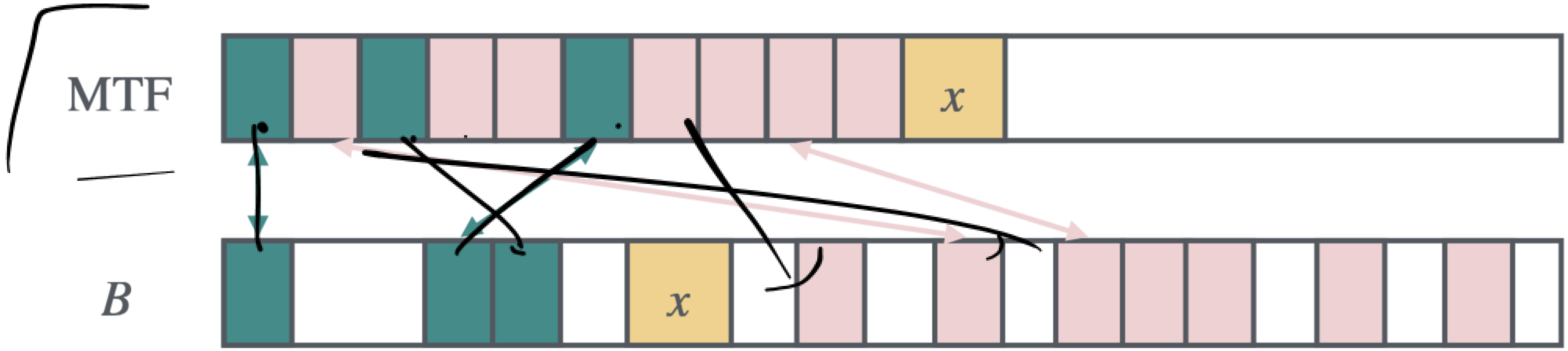
$$\Phi = \overset{2}{\cancel{k}}\,(\#\text{ of inversions})$$

Amortized Analysis

- **Observation:**
  - Each individual operation contains three distinct steps:
    1. Both Move-to-front and $\boldsymbol{B}$ perform **Access($\boldsymbol{x}$)**.
    2. Move-to-front performs its **Swaps**, moving $\boldsymbol{x}$ to the front.
    3. $\boldsymbol{B}$ performs its **Swaps**, which can be different than Move-to-front.
- We will actually analyze step 3 **separately**.
  - Analyzing the operation of the competing algorithm is a common strategy we will employ.

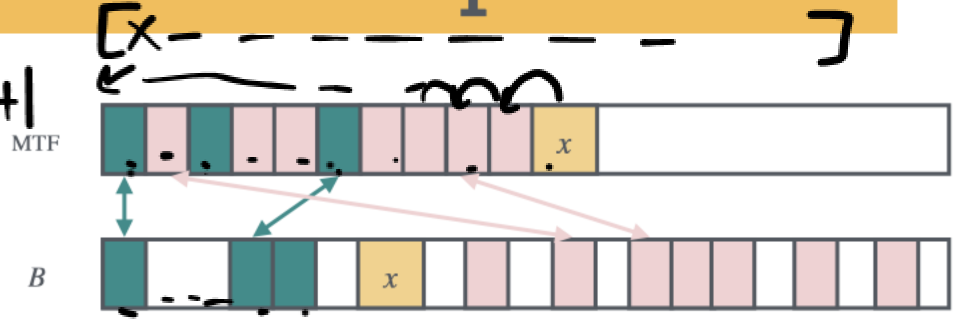
Analysis of **Access(x)** and **MTF**'s **Swaps**

- **States of Move-to-front and B:**



- We define two sets of items.
  - $S = \{$items before x in **MTF** and before x in $B\}$. These are dark/teal above.
  - $T = \{$items before x in **MTF** and after x in $B\}$. These are light/lilac above.
    - **Note:** Arrows are between elements in the same sets.

Analysis of **Access(x)** and **MTF**'s **Swaps**



$S = \{$items before x in MTF and before x in $B\}$.

$T = \{$items before x in MTF and after x in $B\}$.

$$C_{ALG} = |S| + |T| + 1 + |S| + |T| = 2|S| + 2|T| + 1$$

$$C_{OPT} \geq |S| + 1$$

$$\Delta \Phi = 2\left(|S| - |T|\right) = 2|S| - 2|T|$$

$$AC_{ALG} = 4|S| + 1 \leq 4|S| + 4 = 4\left(|S| + 1\right)$$

$$AC_{ALG} \leq 4 \cdot C_{OPT}$$

Analysis B's **Swaps**

$C_{ALG} = 0$

$C_B = 1$

$\Delta\Phi \leq 2$

$AC_{ALG} = 2 \leq 4 = 4 \cdot (1)$

Putting It Together

$\Phi_i = \emptyset$

$\Phi \geq 0$

$\underline{\sum C_{ALG}} \leq \sum AC_{ALG} \leq \underline{\sum 4 \cdot C_B}$

The Paging Problem

- You have:
  - N pages
  - A cache that can hold $k < N$ pages, initially holding $1, 2, \ldots, k$.
- The input is a sequence of page requests.
  - If a page is in the cache, the request if free.
  - If a page is not in the cache, called a **page fault**, we evict one of the pages in the cache, and replace it with this new page.
- **Goal:**
  - Have the minimum number of page faults.

Least Recently Used

- **Algorithm (LRU):**
  - Evict the least recently used page.
- What's a bad case? Say we have $k = 3$ and $N = 4$. $= \{1, 2, 3, 4\}$

1, 2, 3, 4, 1, 2, 3, 4 - - —

Competitive Ratio of LRU

- **Theorem:**
  - LRU is k-competitive.

- Define a **phase** as a group of requests with k **distinct** requests, where the request after the phase is **distinct from the phase**.

- **Example with ~~$k = 5$:~~**

$$\ldots, 2, 4, 3, 3, 2, 8, 8, 8, 8, 10, \ldots, 2, 451, \ldots$$

This sequence contains a phase with pages 2, 3, 4, 8, and 10. The **next** sequence begins with 451.

- How many page faults can LRU incur in a phase? $\leq k$

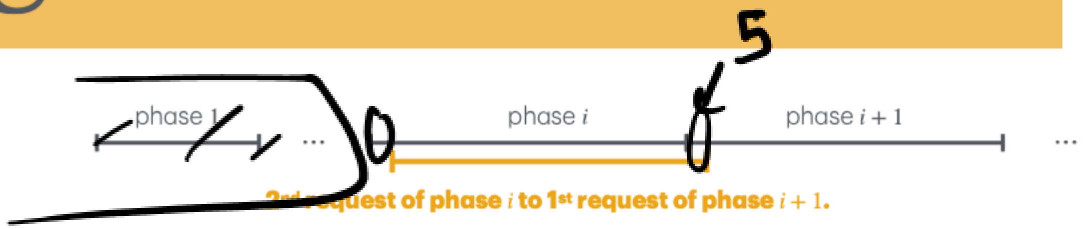
- Given m phases, we have $\leq m \cdot k$ page faults

Competitive Ratio of **LRU**

- Now we look at the cost for any algorithm B.
  - Remember that this will give us a bound on C_{OPT}.
- **Lemma:**
  - Any algorithm must incur at least $m - 1$ page faults over m phases.
- **Proof:**
  - For any phase $i \in \{1, 2, \ldots, m - 1\}$, let's look at **offset phases** from the 2nd request of phase i to the 1st request of phase $i + 1$.

2nd request of phase i to 1st request of phase $i + 1$.

Competitive Ratio of LRU



- **Case 1:** 1st request of phase $i+1$ is a page fault.

  ≥ 1

- **Case 2:** 1st request of phase $i+1$ is not a page fault.
  - Then, this page must be in the cache throughout the offset phase.
  - The 1st request of phase i must also be in the cache after the request.
  - There are still $k-1$ distinct requests in the offset phase, however there are only $k-2$ spots left in the cache. So, there must be a page fault.

Putting It Together

- **Conclusion:**
  - Each offset phase has at least 1 fault. So, there are $m - 1$ page faults over m phases.
  - Thus, LRU has a competitive ratio of k.

Limitation of Deterministic Algorithms

- **Theorem:**
  - Every deterministic algorithm must have a competitive ratio of at least k.

- **Proof:** We show this only for $N = k + 1$, meaning only one page will be out of the cache. The general case is similar.
  - What is the worst case for any algorithm?

every request is a page fault

Limitation of $\boxed{\text{Deterministic}}$ Algorithms

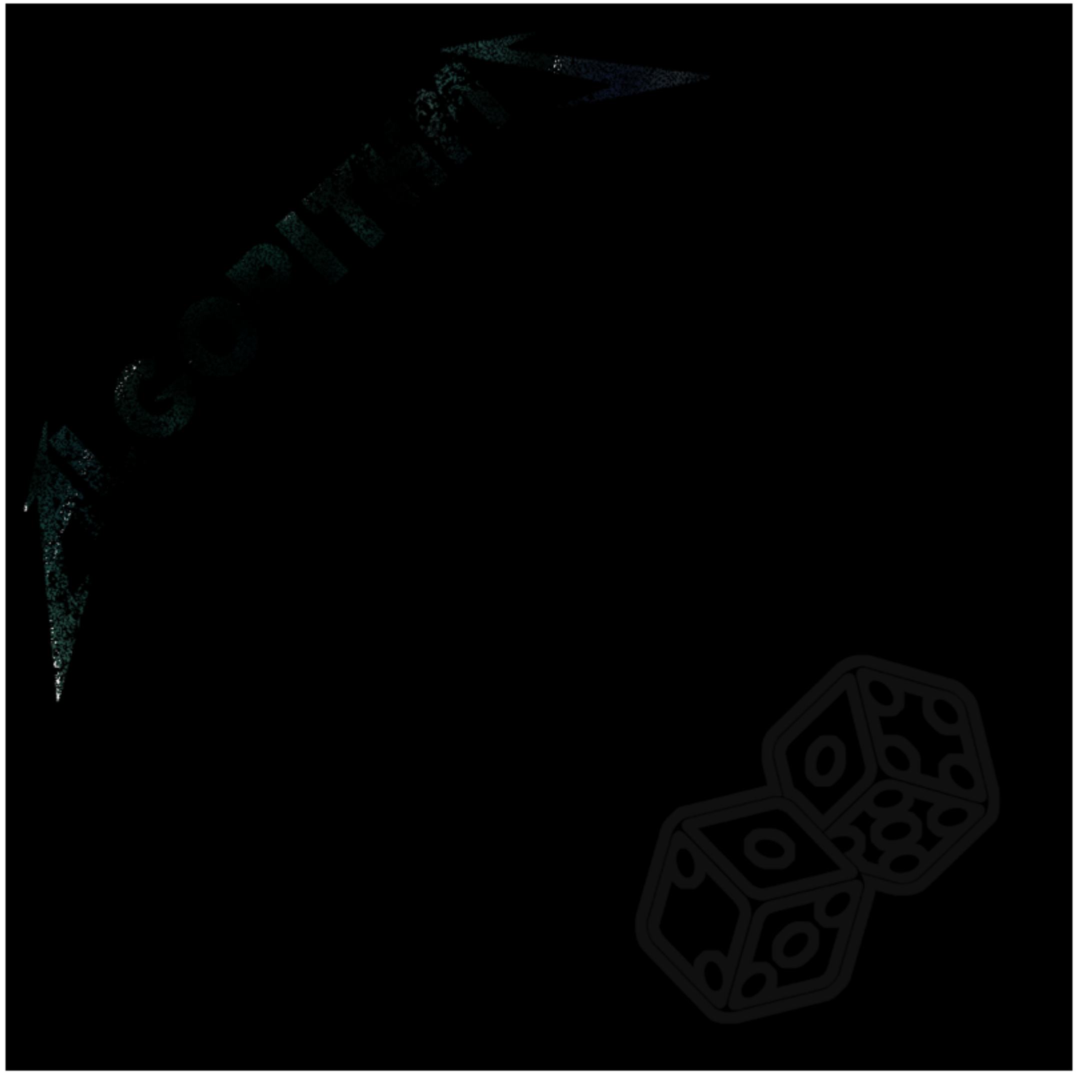
- However, for the same input, an optimal omniscient algorithm can always throw out the page that is requested **furthest in the future**.

  1, 2, 3, 4, 1, 2, 3 ----.

  - Since there are k pages in the cache, one of them has to be requested next at least k time-steps in the future.
  - So, the earliest the next page fault can occur is ~~in~~ after $k - 1$ steps.
  - So, this algorithm can have a page fault **at most once every k steps**.

- Thus, the competitive ratio of any deterministic algorithm is k.

Enter Randomization



$\log k$

Enter Randomization

- **Algorithm:**
  - Start with pages $1, 2, \ldots, k$ in the cache, all **marked**.
  - On a page request:
    - If it's in the cache, mark it and return.
    - Otherwise:
      - If every page is marked, unmark every page.
      - Evict a **random** unmarked page, and mark the newly requested page.

Example

- Let $k = 4$ and $N = 5$.
- Call the interval between unmarkings of all pages **phases**.
- Here's an example phase, with probabilities at each step of each page being in the cache.
- The marked pages are shaded orange.

	pages					
request	1	2	3	4	5	expected cost
	1	1	1	1	0	
5	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	1	1
2	$\frac{2}{3}$	1	$\frac{2}{3}$	$\frac{2}{3}$	1	$\frac{1}{4}$
1	1	1	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{3}$
4	1	1	0	1	1	$\frac{1}{2}$

Analysis of the Randomized Marking Algorithm

- Each phase contains k unique page requests.
  - The first time each page is requested in the phase, it is unmarked, and thus it incurs some **nonzero expected cost**. Then, It gets marked and any further requests to it incur **zero cost**.
    - The first of these requests is out of the cache by definition, so has a cost of 1.
    - For any other request $i \in \{2, 3, \dots, k\}$, at that point there are $i - 1$ marked pages in the cache. The request is equally likely to be any one of the $N - (i - 1) = N - i + 1$ remaining pages.
      - One of these pages is out of the cache, so the expected cost is

$$\frac{1}{N - i + 1}.$$

	pages					
request	1	2	3	4	5	expected cost
	1	1	1	1	0	
5	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	1	1
2	$\frac{2}{3}$	1	$\frac{2}{3}$	$\frac{2}{3}$	1	$\frac{1}{4}$
1	1	1	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{3}$
4	1	1	0	1	1	$\frac{1}{2}$

Analysis of the Randomized Marking Algorithm

- So, we have that the the first unique page has a cost of 1 through the phase, and any other unique page $i \in \{2, 3, \ldots, k\}$ has an expected cost of $\dfrac{1}{N - i + 1}$.

- Thus, in total, the expected cost of a phase is

pages

request	1	2	3	4	5	expected cost
	1	1	1	1	0	
5	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	1	1
2	$\frac{2}{3}$	1	$\frac{2}{3}$	$\frac{2}{3}$	1	$\frac{1}{4}$
1	1	1	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{3}$
4	1	1	0	1	1	$\frac{1}{2}$

Analysis of the Randomized Marking Algorithm

- So, we have that the the first unique page has a cost of 1 through the phase, and any other unique page $i \in \{2, 3, \ldots, k\}$ has an expected cost of $\dfrac{1}{N - i + 1}$.

- Thus, in total, the expected cost of a phase is

$$
\begin{aligned}
1 + \frac{1}{N-2+1} + \frac{1}{N-3+1} + \cdots + \frac{1}{N-k+1} &= 1 + \frac{1}{N-1} + \frac{1}{N-2} + \cdots + \frac{1}{N-1+1-k+1} \\
&= 1 + \frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{k+1-k+1} \\
&= 1 + \frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{2} \\
&= H_k .
\end{aligned}
$$

- **Note:** When $N > k + 1$, the expected cost can be bounded by $2H_k$.

pages

request	1	2	3	4	5	expected cost
	1	1	1	1	0	
5	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	1	1
2	$\frac{2}{3}$	1	$\frac{2}{3}$	$\frac{2}{3}$	1	$\frac{1}{4}$
1	1	1	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{3}$
4	1	1	0	1	1	$\frac{1}{2}$

Putting It Together

Postsigning: Learning Objectives

- Know the definition and goals of **online algorithms**.
- Know the definition of **competitive ratio** in relation to online algorithms.
- Know how to analyze the competitive ratio of an online algorithm.
- Know how to create and use **potential functions** for online algorithms.
- Know how to use **randomization** in order to achieve better (expected) competitive ratios for online algorithms.