Lecture 17: Online Algorithms & Paging

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

November 9, 2021

Overview

- Part I
  - Why Study Online Algorithms?
  - Competitive Analysis
  - Examples
- Paging & Caching
- Conclusion
- Acknowledgements

Why Study Online Algorithms?

- Online algorithms are important for many applications, when we need to make decisions right when we receive the information.

[1] "Hindsight is 20/20"

Why Study Online Algorithms?

- Online algorithms are important for many applications, when we need to make decisions right when we receive the information.
- Applications in
  - Stock Market
  - Dating
  - Skiing
  - Caching
  - Machine Learning (regret minimization)
  - many more...

[1] "Hindsight is 20/20"

Why Study Online Algorithms?

- Online algorithms are important for many applications, when we need to make decisions right when we receive the information.
- Applications in
  - Stock Market
  - Dating
  - Skiing
  - Caching
  - Machine Learning (regret minimization)
  - many more...
- *Competitive Analysis*: measures performance of our algorithm against best algorithm that could *see into the future* (that is, see the entire input beforehand)[1]
  1. Worst-case analysis

[1] "Hindsight is 20/20"

Different Online Models

We will see other online models in class:

Different Online Models

We will see other online models in class:

- Data Streaming (lecture 20): in this case, we not only receive the input in an online fashion, but we have also *memory constraints*
    1. Goal here was to get reasonable (approximate) answers while obeying memory constraints
    2. worst-case analysis

Different Online Models

We will see other online models in class:

- Data Streaming (lecture 20): in this case, we not only receive the input in an online fashion, but we have also *memory constraints*
  1. Goal here was to get reasonable (approximate) answers while obeying memory constraints
  2. worst-case analysis
- Today, we will only see algorithms which must deal with the input as it receives it, *no constraints in memory*.
  1. Goal here is to *be competitive* against *any offline algorithm* (that is, algorithms that could see the entire input beforehand)
  2. worst-case analysis

Competitive Analysis

- Input is given as a sequence $s = s_1, s_2, \ldots, s_n$ of events.

Competitive Analysis

- Input is given as a sequence $s = s_1, s_2, \ldots, s_n$ of events.
- Let $C_{opt}(s)$ be the *minimum cost* that *any algorithm* (even one that could look at the *entire input* beforehand) could achieve for input s

Competitive Analysis

- Input is given as a sequence $s = s_1, s_2, \ldots, s_n$ of events.
- Let $C_{opt}(s)$ be the *minimum cost* that *any algorithm* (even one that could look at the *entire input* beforehand) could achieve for input s
- Let $C_A(s)$ be the cost of your online algorithm on input s

Competitive Analysis

- Input is given as a sequence $s = s_1, s_2, \ldots, s_n$ of events.
- Let $C_{opt}(s)$ be the *minimum cost* that *any algorithm* (even one that could look at the *entire input* beforehand) could achieve for input s
- Let $C_A(s)$ be the cost of your online algorithm on input s

Definition (Deterministic Competitive Ratio)

A deterministic online algorithm A has *competitive ratio* k (aka k-competitive) if for all inputs s, we have:

$$C_A(s) \leq k \cdot C_{opt}(s) + O(1)$$

Competitive Analysis

- Input is given as a sequence $s = s_1, s_2, \ldots, s_n$ of events.
- Let $C_{opt}(s)$ be the *minimum cost* that *any algorithm* (even one that could look at the *entire input* beforehand) could achieve for input s
- Let $C_A(s)$ be the cost of your online algorithm on input s

Definition (Deterministic Competitive Ratio)

A deterministic online algorithm A has *competitive ratio* k (aka k-competitive) if for all inputs s, we have:

$$C_A(s) \leq k \cdot C_{opt}(s) + O(1)$$

Definition (Randomized Competitive Ratio)

A randomized online algorithm A has *competitive ratio* k (aka k-competitive) if for all inputs s, we have:

$$\mathbb{E}[C_A(s)] \leq k \cdot C_{opt}(s).$$

- Part I
  - Why Study Online Algorithms?
  - Competitive Analysis
  - Examples
- Paging & Caching
- Conclusion
- Acknowledgements

Ski Rental Problem

- In pandemic times, I am stuck in Canada.

[2] One can technically go, but if not Canadian or PR, not allowed to come back... And Brazil is not handling covid well... alas

Ski Rental Problem

- In pandemic times, I am stuck in Canada.
- U Waterloo gave us one extra week of vacation in January

[2] One can technically go, but if not Canadian or PR, not allowed to come back... And Brazil is not handling covid well... alas

Ski Rental Problem

- In pandemic times, I am stuck in Canada.
- U Waterloo gave us one extra week of vacation in January
- So we decided to go Skiing this past winter (since we could not go back to Brazil and enjoy the beach and the summer)[2]

[2] One can technically go, but if not Canadian or PR, not allowed to come back... And Brazil is not handling covid well... alas

Ski Rental Problem

- In pandemic times, I am stuck in Canada.
- U Waterloo gave us one extra week of vacation in January
- So we decided to go Skiing this past winter (since we could not go back to Brazil and enjoy the beach and the summer)[2]
- Winters in Canada are veeerrryy long... so we may go a bunch of times...

[2] One can technically go, but if not Canadian or PR, not allowed to come back... And Brazil is not handling covid well... alas

Ski Rental Problem

- In pandemic times, I am stuck in Canada.
- U Waterloo gave us one extra week of vacation in January
- So we decided to go Skiing this past winter (since we could not go back to Brazil and enjoy the beach and the summer)[2]
- Winters in Canada are veeerrryy long... so we may go a bunch of times...
- Having never done this before, we have to decide whether to buy all the equipment or to rent it at the resort.

[2] One can technically go, but if not Canadian or PR, not allowed to come back... And Brazil is not handling covid well... alas

Ski Rental Problem

- In pandemic times, I am stuck in Canada.
- U Waterloo gave us one extra week of vacation in January
- So we decided to go Skiing this past winter (since we could not go back to Brazil and enjoy the beach and the summer)[2]
- Winters in Canada are veeerrryy long... so we may go a bunch of times...
- Having never done this before, we have to decide whether to buy all the equipment or to rent it at the resort.
- Buying the equipment costs us 1k CAD. Renting at the resort costs 100 CAD per day.

[2] One can technically go, but if not Canadian or PR, not allowed to come back... And Brazil is not handling covid well... alas

Ski Rental Problem

- In pandemic times, I am stuck in Canada.
- U Waterloo gave us one extra week of vacation in January
- So we decided to go Skiing this past winter (since we could not go back to Brazil and enjoy the beach and the summer)[2]
- Winters in Canada are veeerrryy long... so we may go a bunch of times...
- Having never done this before, we have to decide whether to buy all the equipment or to rent it at the resort.
- Buying the equipment costs us 1k CAD. Renting at the resort costs 100 CAD per day.
- Should we buy or rent?

[2]One can technically go, but if not Canadian or PR, not allowed to come back... And Brazil is not handling covid well... alas

Ski Rental Problem

- In pandemic times, I am stuck in Canada.
- U Waterloo gave us one extra week of vacation in January
- So we decided to go Skiing this past winter (since we could not go back to Brazil and enjoy the beach and the summer)[2]
- Winters in Canada are veeerrryy long... so we may go a bunch of times...
- Having never done this before, we have to decide whether to buy all the equipment or to rent it at the resort.
- Buying the equipment costs us 1k CAD. Renting at the resort costs 100 CAD per day.
- Should we buy or rent?
- Depends on how many times we will go skiing...

[2] One can technically go, but if not Canadian or PR, not allowed to come back... And Brazil is not handling covid well... alas

Ski Rental Problem

- Buying the equipment costs us 1k CAD. Renting at the resort costs 100 CAD per day.
- Should we buy or rent?
- Depends on how many times we will go skiing...

Ski Rental Problem

- Buying the equipment costs us 1k CAD. Renting at the resort costs 100 CAD per day.
- Should we buy or rent?
- Depends on how many times we will go skiing...
  1. If we go skiing 9 times or less (and we see that we are made for beaches and tropical islands), then clearly *better to rent*

Ski Rental Problem

- Buying the equipment costs us 1k CAD. Renting at the resort costs 100 CAD per day.
- Should we buy or rent?
- Depends on how many times we will go skiing...
  1. If we go skiing 9 times or less (and we see that we are made for beaches and tropical islands), then clearly *better to rent*
  2. If we go skiing at least 11 times (and surprise ourselves that we can withstand the cold) then clearly *better to buy*

Ski Rental Problem

- Buying the equipment costs us 1k CAD. Renting at the resort costs 100 CAD per day.
- Should we buy or rent?
- Depends on how many times we will go skiing...
  1. If we go skiing 9 times or less (and we see that we are made for beaches and tropical islands), then clearly *better to rent*
  2. If we go skiing at least 11 times (and surprise ourselves that we can withstand the cold) then clearly *better to buy*
  3. If we go 10 times, it doesn't matter which way it goes...

Ski Rental Problem

- Buying the equipment costs us 1k CAD. Renting at the resort costs 100 CAD per day.
- Should we buy or rent?
- Depends on how many times we will go skiing...
  1. If we go skiing 9 times or less (and we see that we are made for beaches and tropical islands), then clearly *better to rent*
  2. If we go skiing at least 11 times (and surprise ourselves that we can withstand the cold) then clearly *better to buy*
  3. If we go 10 times, it doesn't matter which way it goes...
- How is this an online algorithm?

Ski Rental Problem

- Buying the equipment costs us 1k CAD. Renting at the resort costs 100 CAD per day.
- Should we buy or rent?
- Depends on how many times we will go skiing...
  1. If we go skiing 9 times or less (and we see that we are made for beaches and tropical islands), then clearly *better to rent*
  2. If we go skiing at least 11 times (and surprise ourselves that we can withstand the cold) then clearly *better to buy*
  3. If we go 10 times, it doesn't matter which way it goes...
- How is this an online algorithm?
- Each time we go skiing, we have to decide whether to buy or rent (unless we bought it beforehand)

Ski Rental Problem

- Buying the equipment costs us 1k CAD. Renting at the resort costs 100 CAD per day.
- Should we buy or rent?
- Depends on how many times we will go skiing...
  1. If we go skiing 9 times or less (and we see that we are made for beaches and tropical islands), then clearly *better to rent*
  2. If we go skiing at least 11 times (and surprise ourselves that we can withstand the cold) then clearly *better to buy*
  3. If we go 10 times, it doesn't matter which way it goes...
- How is this an online algorithm?
- Each time we go skiing, we have to decide whether to buy or rent (unless we bought it beforehand)
- Algorithm has to decide *when to buy*, knowing only that we have gone skiing t times

Ski Rental Problem

- Buying the equipment costs us 1k CAD. Renting at the resort costs 100 CAD per day.

Ski Rental Problem

- Buying the equipment costs us 1k CAD. Renting at the resort costs 100 CAD per day.
- A 1.9-competitive algorithm:
  - If $t \leq 9$, then rent
  - When $t = 10$, buy

t = # times we will go skiing

Ski Rental Problem

- Buying the equipment costs us 1k CAD. Renting at the resort costs 100 CAD per day.
- A 1.9-competitive algorithm:
  - If $t \leq 9$, then rent
  - When $t = 10$, buy
- Analysis:
  - If $t \leq 9$, then best strategy is to rent: so cost is:

$$\frac{C_A}{C_{opt}} = \frac{100 \cdot t}{100 \cdot t} = 1$$

Ski Rental Problem

- Buying the equipment costs us 1k CAD. Renting at the resort costs 100 CAD per day.
- A 1.9-competitive algorithm:
  - If $t \leq 9$, then rent
  - When $t = 10$, buy
- Analysis:
  - If $t \leq 9$, then best strategy is to rent: so cost is:

$$\frac{C_A}{C_{opt}} = \frac{100 \cdot t}{100 \cdot t} = 1$$

  - If $t \geq 10$, we buy at the 10^{th} time, so cost is:

$$\frac{C_A}{C_{opt}} = \frac{100 \cdot 9 + 1000}{1000} = 1.9$$

$C_A(t) \leq 1.9\, C_{opt}(t)$

$\forall$ input sequence t = # times we go skiing in entire lives

~~Secretary~~ Dating Problem

- In the high-tech life, you decide to join a dating site...

[3] Assumptions: people are comparable AND we know how to do it

[4] Go big or go ~~home~~ lonely!

[5] Also assuming they will all want to date us...

~~Secretary~~ Dating Problem

- In the high-tech life, you decide to join a dating site...
- There are n people that you are interested in dating, and you would like to date the best person[3] out there.[4]

[3] Assumptions: people are comparable AND we know how to do it
[4] Go big or go ~~home~~ lonely!
[5] Also assuming they will all want to date us...

~~Secretary~~ Dating Problem

- In the high-tech life, you decide to join a dating site...
- There are n people that you are interested in dating, and you would like to date the best person[3] out there.[4]
- But you don't know who is the best person in advance...

[3] Assumptions: people are comparable AND we know how to do it

[4] Go big or go ~~home~~ lonely!

[5] Also assuming they will all want to date us...

~~Secretary~~ Dating Problem

- In the high-tech life, you decide to join a dating site...
- There are n people that you are interested in dating, and you would like to date the best person[3] out there.[4]
- But you don't know who is the best person in advance...
- One way to do it: go out with all of them at the same time,[5] and figure out which one is the best!

[3] Assumptions: people are comparable AND we know how to do it

[4] Go big or go ~~home~~ lonely!

[5] Also assuming they will all want to date us...

~~Secretary~~ Dating Problem

- In the high-tech life, you decide to join a dating site...
- There are n people that you are interested in dating, and you would like to date the best person[3] out there.[4]
- But you don't know who is the best person in advance...
- One way to do it: go out with all of them at the same time,[5] and figure out which one is the best!
- Not possible, due to time constraints and society's value system

[3] Assumptions: people are comparable AND we know how to do it
[4] Go big or go ~~home~~ lonely!
[5] Also assuming they will all want to date us...

~~Secretary~~ Dating Problem

- In the high-tech life, you decide to join a dating site...
- There are n people that you are interested in dating, and you would like to date the best person[3] out there.[4]
- But you don't know who is the best person in advance...
- One way to do it: go out with all of them at the same time,[5] and figure out which one is the best!
- Not possible, due to time constraints and society's value system
- So we have to go out with one of them at a time, and decide whether we want to stay with them or date another person, in which case we must break up

---

[3] Assumptions: people are comparable AND we know how to do it
[4] Go big or go ~~home~~ lonely!
[5] Also assuming they will all want to date us...

~~Secretary~~ Dating Problem

- In the high-tech life, you decide to join a dating site...
- There are n people that you are interested in dating, and you would like to date the best person[3] out there.[4]
- But you don't know who is the best person in advance...
- One way to do it: go out with all of them at the same time,[5] and figure out which one is the best!
- Not possible, due to time constraints and society's value system
- So we have to go out with one of them at a time, and decide whether we want to stay with them or date another person, in which case we must break up
- Clearly *online setting* (pun intended)

[3] Assumptions: people are comparable AND we know how to do it

[4] Go big or go ~~home~~ lonely!

[5] Also assuming they will all want to date us...

~~Secretary~~ Dating Problem

- In the high-tech life, you decide to join a dating site...
- There are n people that you are interested in dating, and you would like to date the best person[3] out there.[4]
- But you don't know who is the best person in advance...
- One way to do it: go out with all of them at the same time,[5] and figure out which one is the best!
- Not possible, due to time constraints and society's value system
- So we have to go out with one of them at a time, and decide whether we want to stay with them or date another person, in which case we must break up
- Clearly *online setting* (pun intended)
- **Goal:** maximize probability of dating the best person

[3] Assumptions: people are comparable AND we know how to do it
[4] Go big or go ~~home~~ lonely!
[5] Also assuming they will all want to date us...

~~Secretary~~ Dating Problem

- Consider the following algorithm:

[6] It's not about them, it's about you... you haven't seen enough, too young to commit, etc.

~~Secretary~~ Dating Problem

- Consider the following algorithm:
  1. Let's assume that all people you want to date are ranked and associate them with their rank: $1, 2, \ldots, n$

---

[6] It's not about them, it's about you... you haven't seen enough, too young to commit, etc.

~~Secretary~~ Dating Problem

- Consider the following algorithm:
  1. Let's assume that all people you want to date are ranked and associate them with their rank: $1, 2, \ldots, n$
  2. Pick random order of the n people: call it π

[6] It's not about them, it's about you... you haven't seen enough, too young to commit, etc.

~~Secretary~~ Dating Problem

- Consider the following algorithm:
  1. Let's assume that all people you want to date are ranked and associate them with their rank: $1, 2, \ldots, n$
  2. Pick random order of the n people: call it π
  3. Go out with n/e of them and reject them[6]

[6] It's not about them, it's about you... you haven't seen enough, too young to commit, etc.

~~Secretary~~ Dating Problem

- Consider the following algorithm:
  1. Let's assume that all people you want to date are ranked and associate them with their rank: $1, 2, \ldots, n$
  2. Pick random order of the n people: call it π
  3. Go out with n/e of them and reject them[6]
  4. After first n/e dates, you will decide to settle if the person you found is *better* than *anyone else you have dated before*

[6] It's not about them, it's about you... you haven't seen enough, too young to commit, etc.

~~Secretary~~ Dating Problem

- Consider the following algorithm:
  1. Let's assume that all people you want to date are ranked and associate them with their rank: $1, 2, \ldots, n$
  2. Pick random order of the n people: call it π
  3. Go out with n/e of them and reject them[6]
  4. After first n/e dates, you will decide to settle if the person you found is *better* than *anyone else you have dated before*
- This algorithm picks the best person (i.e., the one ranked 1) with probability $\approx 1/e$

[6] It's not about them, it's about you... you haven't seen enough, too young to commit, etc.

~~Secretary~~ Dating Problem

- Consider the following algorithm:
  1. Let's assume that all people you want to date are ranked and associate them with their rank: $1, 2, \ldots, n$
  2. Pick random order of the n people: call it π
  3. Go out with n/e of them and reject them[6]
  4. After first n/e dates, you will decide to settle if the person you found is *better* than *anyone else you have dated before*
- This algorithm picks the best person (i.e., the one ranked 1) with probability $\approx 1/e$
- More general algorithm: given a time t, go on t dates and from date $t + 1$ onwards you decide to settle with a person who is better than the previous ones.

[6] It's not about them, it's about you... you haven't seen enough, too young to commit, etc.

~~Secretary~~ Dating Problem

- Consider the following algorithm:
  1. Let's assume that all people you want to date are ranked and associate them with their rank: $1, 2, \ldots, n$
  2. Pick random order of the n people: call it π
  3. Go out with n/e of them and reject them[6]
  4. After first n/e dates, you will decide to settle if the person you found is *better* than *anyone else you have dated before*
- This algorithm picks the best person (i.e., the one ranked 1) with probability $\approx 1/e$
- More general algorithm: given a time t, go on t dates and from date $t + 1$ onwards you decide to settle with a person who is better than the previous ones.
- What is the probability that we pick the number 1 in our list?

[6] It's not about them, it's about you... you haven't seen enough, too young to commit, etc.

~~Secretary~~ Dating Problem

- More general algorithm: given a time t, go on t dates and from date $t+1$ onwards you decide to settle with a person who is better than the previous ones.
- What is the probability that we pick the number 1 in our list?

~~Secretary~~ Dating Problem

- More general algorithm: given a time t, go on t dates and from date $t+1$ onwards you decide to settle with a person who is better than the previous ones.
- What is the probability that we pick the number 1 in our list?
- Suppose we pick a person at time k, then want to compute probability

$$P_k = \Pr[\pi(k) = 1 \text{ and we pick person at time } k]$$

1st person shows up at time k

~~Secretary~~ Dating Problem

- More general algorithm: given a time t, go on t dates and from date $t+1$ onwards you decide to settle with a person who is better than the previous ones.
- What is the probability that we pick the number 1 in our list?
- Suppose we pick a person at time k, then want to compute probability

$$P_k = \Pr[\pi(k) = 1 \text{ and we pick person at time } k]$$

- Then our final success probability will be $P = \sum_{k>t}^{n} P_k$

~~Secretary~~ Dating Problem

- More general algorithm: given a time t, go on t dates and from date $t+1$ onwards you decide to settle with a person who is better than the previous ones.
- What is the probability that we pick the number 1 in our list?
- Suppose we pick a person at time k, then want to compute probability

$$P_k = \Pr[\pi(k) = 1 \text{ and we pick person at time } k]$$

- Then our final success probability will be $P = \sum_{k>t}^{n} P_k$
- If $\pi(k) = 1$, then $1 - P_k$ is the probability that we picked a person between $[t+1, k-1]$, which means someone in this range better than the first t people.

ranks of all dates up to date k-1

$$P_k = \Pr[\pi(k) = 1 \text{ and } \min \pi(1), \ldots, \pi(k-1) \text{ is in } \{\pi(1), \ldots, \pi(t)\}]$$

we did not pick anyone before date k

~~Secretary~~ Dating Problem

- Final success probability will be $P = \sum_{k>t}^{n} P_k$

~~Secretary~~ Dating Problem

- Final success probability will be $P = \sum_{k>t}^{n} P_k$
- From previous slide

$$P_k = \Pr[\pi(k) = 1 \text{ and } \min \pi(1), \ldots, \pi(k-1) \text{ is in } \{\pi(1), \ldots, \pi(t)\}]$$
$$= \frac{1}{n} \cdot \frac{t}{k-1}$$

picking a permutation of $a_1 < a_2 < \cdots < a_{k-1}$ s.t. a_1 appears within first t places

$$\frac{t}{k-1}$$

~~Secretary~~ Dating Problem

- Final success probability will be $P = \sum_{k>t}^{n} P_k$
- From previous slide

$$\begin{aligned} P_k &= \Pr[\pi(k) = 1 \text{ and } \min \pi(1), \ldots, \pi(k-1) \text{ is in } \{\pi(1), \ldots, \pi(t)\}] \\ &= \frac{1}{n} \cdot \frac{t}{k-1} \end{aligned}$$

- We get

$$P = \sum_{k>t}^{n} \frac{1}{n} \cdot \frac{t}{k-1} = \frac{t}{n} \cdot \sum_{k>t}^{n} \frac{1}{k-1} \approx \frac{t}{n} \cdot (\ln n - \ln t) = \frac{t}{n} \cdot \ln(n/t)$$

~~Secretary~~ Dating Problem

- Final success probability will be $P = \sum_{k>t}^{n} P_k$
- From previous slide

$$
\begin{aligned}
P_k &= \Pr[\pi(k) = 1 \text{ and } \min \pi(1), \ldots, \pi(k-1) \text{ is in } \{\pi(1), \ldots, \pi(t)\}] \\
&= \frac{1}{n} \cdot \frac{t}{k-1}
\end{aligned}
$$

- We get

$$
P = \sum_{k>t}^{n} \frac{1}{n} \cdot \frac{t}{k-1} = \frac{t}{n} \cdot \sum_{k>t}^{n} \frac{1}{k-1} \approx \frac{t}{n} \cdot (\ln n - \ln t) = \frac{t}{n} \cdot \ln(n/t)
$$

- Optimizing we get that we should set $t = n/e$, which gives us $1/e$ probability.

~~Secretary~~ Dating Problem

- Final success probability will be $P = \sum_{k>t}^{n} P_k$
- From previous slide

$$P_k = \Pr[\pi(k) = 1 \text{ and } \min \pi(1), \ldots, \pi(k-1) \text{ is in } \{\pi(1), \ldots, \pi(t)\}]$$
$$= \frac{1}{n} \cdot \frac{t}{k-1}$$

- We get

$$P = \sum_{k>t}^{n} \frac{1}{n} \cdot \frac{t}{k-1} = \frac{t}{n} \cdot \sum_{k>t}^{n} \frac{1}{k-1} \approx \frac{t}{n} \cdot (\ln n - \ln t) = \frac{t}{n} \cdot \ln(n/t)$$

- Optimizing we get that we should set $t = n/e$, which gives us $1/e$ probability.
- Wait a second, where is the competitive analysis?

Making Dating Competitive

- To make the dating problem competitive, we would have to modify it a little bit.
  - We can simply minimize the rank.

Making Dating Competitive

- To make the dating problem competitive, we would have to modify it a little bit.
  - We can simply minimize the rank.
  - Say we always want to end up with someone (loneliness has a cost of $-\infty$, after all nobody wants to be alone)

Making Dating Competitive

- To make the dating problem competitive, we would have to modify it a little bit.
  - We can simply minimize the rank.
  - Say we always want to end up with someone (loneliness has a cost of $-\infty$, after all nobody wants to be alone)
  - Previous algorithm would then either pick the best person, or the last person in the order.

input : sequence of people

$p_1 \; p_2 \; \cdots \; p_n$

$\pi(p_1) \; \pi(p_2) \; \cdots \; \pi(p_n)$

$\pi : [n] \to [n]$ this exists but don't know

all we can do is $comp(p_i, p_j) \to \arg\min(\pi(p_i), \pi(p_j))$

Making Dating Competitive

- To make the dating problem competitive, we would have to modify it a little bit.
  - We can simply minimize the rank.
  - Say we always want to end up with someone (loneliness has a cost of $-\infty$, after all nobody wants to be alone)
  - Previous algorithm would then either pick the best person, or the last person in the order.
  - With constant probability, rank of the last person is $\Omega(n)$, so we either date the best, or we date someone in the "bottom percentile" of our list

$$\mathbb{E}[C_A(\sigma)] = \Omega(n)$$

$$\geq \underbrace{\Pr[\text{end up last person}]}_{\text{constant}} \cdot \underbrace{(\text{rank of last person})}_{\Omega(n)}$$

Making Dating Competitive

- To make the dating problem competitive, we would have to modify it a little bit.
  - We can simply minimize the rank.
  - Say we always want to end up with someone (loneliness has a cost of $-\infty$, after all nobody wants to be alone)
  - Previous algorithm would then either pick the best person, or the last person in the order.
  - With constant probability, rank of the last person is $\Omega(n)$, so we either date the best, or we date someone in the "bottom percentile" of our list
  - Expected rank of our life-long partner is $\Omega(n)$
- What is $C_{opt}(s) = 1$

  opt can always see in the future

  (in particular opt knows rank function)

Making Dating Competitive

- To make the dating problem competitive, we would have to modify it a little bit.
  - We can simply minimize the rank.
  - Say we always want to end up with someone (loneliness has a cost of $-\infty$, after all nobody wants to be alone)
  - Previous algorithm would then either pick the best person, or the last person in the order.
  - With constant probability, rank of the last person is $\Omega(n)$, so we either date the best, or we date someone in the "bottom percentile" of our list
  - Expected rank of our life-long partner is $\Omega(n)$
- Can we do better?

Making Dating Competitive

- To make the dating problem competitive, we would have to modify it a little bit.
  - We can simply minimize the rank.
  - Say we always want to end up with someone (loneliness has a cost of $-\infty$, after all nobody wants to be alone)
  - Previous algorithm would then either pick the best person, or the last person in the order.
  - With constant probability, rank of the last person is $\Omega(n)$, so we either date the best, or we date someone in the "bottom percentile" of our list
  - Expected rank of our life-long partner is $\Omega(n)$
- Can we do better?
- Yes! There is an algorithm that picks person of average rank $O(1)$, which is therefore $O(1)$-competitive.

Making Dating Competitive

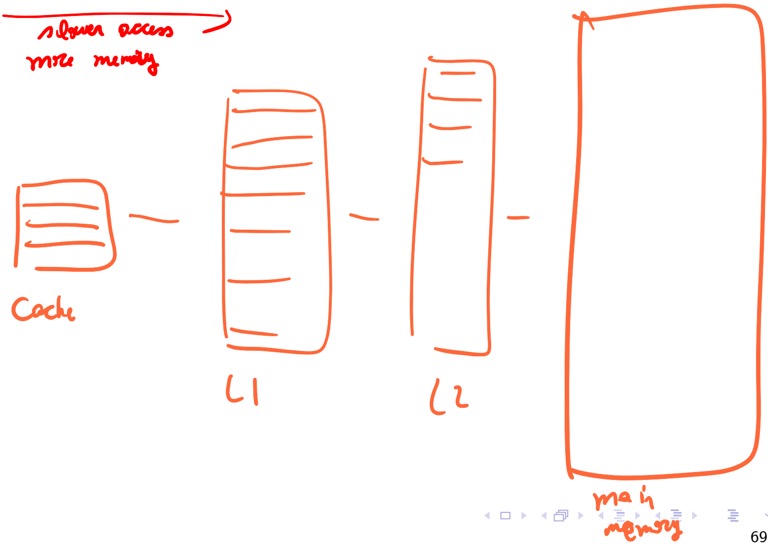
- To make the dating problem competitive, we would have to modify it a little bit.
  - We can simply minimize the rank.
  - Say we always want to end up with someone (loneliness has a cost of $-\infty$, after all nobody wants to be alone)
  - Previous algorithm would then either pick the best person, or the last person in the order.
  - With constant probability, rank of the last person is $\Omega(n)$, so we either date the best, or we date someone in the "bottom percentile" of our list
  - Expected rank of our life-long partner is $\Omega(n)$
- Can we do better?
- Yes! There is an algorithm that picks person of average rank $O(1)$, which is therefore $O(1)$-competitive.
- Complicated algorithm, based on computing time steps $t_0 \leq t_1 \leq \dots$ and between timesteps t_k and t_{k+1} we are willing to pick person who is $\leq k+1$ best from our current list.

Making Dating Competitive

- To make the dating problem competitive, we would have to modify it a little bit.
  - We can simply minimize the rank.
  - Say we always want to end up with someone (loneliness has a cost of $-\infty$, after all nobody wants to be alone)
  - Previous algorithm would then either pick the best person, or the last person in the order.
  - With constant probability, rank of the last person is $\Omega(n)$, so we either date the best, or we date someone in the "bottom percentile" of our list
  - Expected rank of our life-long partner is $\Omega(n)$
- Can we do better?
- Yes! There is an algorithm that picks person of average rank $O(1)$, which is therefore $O(1)$-competitive.
- Complicated algorithm, based on computing time steps $t_0 \leq t_1 \leq \ldots$ and between timesteps t_k and t_{k+1} we are willing to pick person who is $\leq k+1$ best from our current list.
- That is, as we get older, we become more desperate to find someone and lower our expectations...

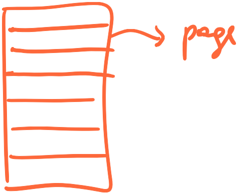
Online Paging Problem

- Computer memory is hierarchical: cache → L1 → L2 → main memory



Online Paging Problem

- Computer memory is hierarchical: cache $\rightarrow$ L1 $\rightarrow$ L2 $\rightarrow$ main memory
- Memory can be modelled in the following way:
  - Each layer of memory is an array with certain number of pages (hence the name)



Online Paging Problem

- Computer memory is hierarchical: cache $\rightarrow$ L1 $\rightarrow$ L2 $\rightarrow$ main memory
- Memory can be modelled in the following way:
  - Each layer of memory is an array with certain number of pages (hence the name)
  - Page stores the content of the item and its location in main memory

Online Paging Problem

- Computer memory is hierarchical: cache $\rightarrow$ L1 $\rightarrow$ L2 $\rightarrow$ main memory
- Memory can be modelled in the following way:
  - Each layer of memory is an array with certain number of pages (hence the name)
  - Page stores the content of the item and its location in main memory
  - When we get a request ($\Leftrightarrow$ event in online jargon), we first look up in cache, then L1, then L2, then main memory

Online Paging Problem

- Computer memory is hierarchical: cache $\rightarrow$ L1 $\rightarrow$ L2 $\rightarrow$ main memory
- Memory can be modelled in the following way:
  - Each layer of memory is an array with certain number of pages (hence the name)
  - Page stores the content of the item and its location in main memory
  - When we get a request ($\Leftrightarrow$ event in online jargon), we first look up in cache, then L1, then L2, then main memory
  - If request is in cache, we have a *hit* $\leftrightarrow$ request takes negligible time

Online Paging Problem

- Computer memory is hierarchical: cache $\rightarrow$ L1 $\rightarrow$ L2 $\rightarrow$ main memory
- Memory can be modelled in the following way:
  - Each layer of memory is an array with certain number of pages (hence the name)
  - Page stores the content of the item and its location in main memory
  - When we get a request ($\Leftrightarrow$ event in online jargon), we first look up in cache, then L1, then L2, then main memory
  - If request is in cache, we have a *hit* $\leftrightarrow$ request takes negligible time
  - Otherwise we have *miss* $\leftrightarrow$ need to fetch data from slower memory
  - In negligible extra time, can also copy new data & location to cache

thing that takes most time is fetching data from slower memory

→ also have to write the new datum in cache

Online Paging Problem

- Computer memory is hierarchical: cache $\rightarrow$ L1 $\rightarrow$ L2 $\rightarrow$ main memory
- Memory can be modelled in the following way:
  - Each layer of memory is an array with certain number of pages (hence the name)
  - Page stores the content of the item and its location in main memory
  - When we get a request ($\Leftrightarrow$ event in online jargon), we first look up in cache, then L1, then L2, then main memory
  - If request is in cache, we have a *hit* $\leftrightarrow$ request takes negligible time
  - Otherwise we have *miss* $\leftrightarrow$ need to fetch data from slower memory
  - In negligible extra time, can also copy new data & location to cache
  - If cache full, must delete an old entry before copying new data

Online Paging Problem

- Computer memory is hierarchical: cache $\rightarrow$ L1 $\rightarrow$ L2 $\rightarrow$ main memory
- Memory can be modelled in the following way:
  - Each layer of memory is an array with certain number of pages (hence the name)
  - Page stores the content of the item and its location in main memory
  - When we get a request ($\Leftrightarrow$ event in online jargon), we first look up in cache, then L1, then L2, then main memory
  - If request is in cache, we have a *hit* $\leftrightarrow$ request takes negligible time
  - Otherwise we have *miss* $\leftrightarrow$ need to fetch data from slower memory
  - In negligible extra time, can also copy new data & location to cache
  - If cache full, must delete an old entry before copying new data
- Main question: which entry of the cache to delete?

Online Paging Problem

- Computer memory is hierarchical: cache $\rightarrow$ L1 $\rightarrow$ L2 $\rightarrow$ main memory
- Memory can be modelled in the following way:
  - Each layer of memory is an array with certain number of pages (hence the name)
  - Page stores the content of the item and its location in main memory
  - When we get a request ($\Leftrightarrow$ event in online jargon), we first look up in cache, then L1, then L2, then main memory
  - If request is in cache, we have a *hit* $\leftrightarrow$ request takes negligible time
  - Otherwise we have *miss* $\leftrightarrow$ need to fetch data from slower memory
  - In negligible extra time, can also copy new data & location to cache
  - If cache full, must delete an old entry before copying new data
- Main question: which entry of the cache to delete?
- Cost function: *number of cache misses*

Online Paging Problem

- Computer memory is hierarchical: cache $\rightarrow$ L1 $\rightarrow$ L2 $\rightarrow$ main memory
- Memory can be modelled in the following way:
  - Each layer of memory is an array with certain number of pages (hence the name)
  - Page stores the content of the item and its location in main memory
  - When we get a request ($\Leftrightarrow$ event in online jargon), we first look up in cache, then L1, then L2, then main memory
  - If request is in cache, we have a *hit* $\leftrightarrow$ request takes negligible time
  - Otherwise we have *miss* $\leftrightarrow$ need to fetch data from slower memory
  - In negligible extra time, can also copy new data & location to cache
  - If cache full, must delete an old entry before copying new data
- Main question: which entry of the cache to delete?
- Cost function: *number of cache misses*
- Simplification: assume we only have cache and main memory.

Common Heuristics

1. **Least Recently Used (LRU)**: delete page in cache whose *most recent request* happened *furthest* in the past

Common Heuristics

1. **Least Recently Used (LRU)**: delete page in cache whose *most recent request* happened *furthest* in the past
2. **Random:** delete random page.

Common Heuristics

1. **Least Recently Used (LRU)**: delete page in cache whose *most recent request* happened *furthest* in the past
2. **Random:** delete random page.
3. **First-in, First-out (FIFO)**: delete page that has been in cache the *longest*

Common Heuristics

1. **Least Recently Used (LRU)**: delete page in cache whose *most recent request* happened *furthest* in the past
2. **Random:** delete random page.
3. **First-in, First-out (FIFO)**: delete page that has been in cache the *longest*
4. **Least Frequently Used (LFU)**: delete page in cache which has been requested *least often*

Common Heuristics

1. **Least Recently Used (LRU)**: delete page in cache whose *most recent request* happened *furthest* in the past
2. **Random:** delete random page.
3. **First-in, First-out (FIFO)**: delete page that has been in cache the *longest*
4. **Least Frequently Used (LFU)**: delete page in cache which has been requested *least often*

Today, we will analyze the **Least Recently Used** heuristic. We will assume that *the size of our cache is k pages*.

Common Heuristics

1. **Least Recently Used (LRU)**: delete page in cache whose *most recent request* happened *furthest* in the past
2. **Random:** delete random page.
3. **First-in, First-out (FIFO)**: delete page that has been in cache the *longest*
4. **Least Frequently Used (LFU)**: delete page in cache which has been requested *least often*

Today, we will analyze the **Least Recently Used** heuristic. We will assume that *the size of our cache is k pages*.

1. **Least Recently Used (LRU)**: k-competitive
2. **Random**: k-competitive
3. **First-in, First-out (FIFO)**: k-competitive
4. **Least Frequently Used (LFU)**: NOT competitive

LRU Analysis

Theorem

For cache of size k, LRU is k-competitive.

LRU Analysis

Theorem

For cache of size k, LRU is k-competitive.

1. Upper bound: divide input sequence into phases.
   - First phase starts immediately after our algorithm first faults, ends right after the algorithm faults k more times
   - Second phase starts at end of first phase, ends when algorithm faults for additional k times
   - and so on...

LRU Analysis

Theorem

For cache of size k, LRU is k-competitive.

1. Upper bound: divide input sequence into phases.
   - First phase starts immediately after our algorithm first faults, ends right after the algorithm faults k more times
   - Second phase starts at end of first phase, ends when algorithm faults for additional k times
   - and so on...
2. We will prove that OPT algorithm faults *at least* once per phase

in each phase: OPT faults ≥ 1

A faults $= k$

$$\frac{C_A(s)}{C_{OPT}(s)} \leq \frac{k}{1}$$

LRU Analysis

Theorem

For cache of size k, LRU is k-competitive.

1. Upper bound: divide input sequence into phases.
   - First phase starts immediately after our algorithm first faults, ends right after the algorithm faults k more times
   - Second phase starts at end of first phase, ends when algorithm faults for additional k times
   - and so on...
2. We will prove that OPT algorithm faults *at least* once per phase
3. This gives us that $C_A \leq k \cdot C_{opt}$, which is what we want.

LRU Analysis

Theorem

For cache of size k, LRU is k-competitive.

1. Upper bound: divide input sequence into phases.
   - First phase starts immediately after our algorithm first faults, ends right after the algorithm faults k more times
   - Second phase starts at end of first phase, ends when algorithm faults for additional k times
   - and so on...
2. We will prove that OPT algorithm faults *at least* once per phase
3. This gives us that $C_A \leq k \cdot C_{opt}$, which is what we want.
4. Examples of phases, for $k = 3$:

request → 1, 1, 2, 2, 1, 3, 4, 3, 2, 4, 5, 6, 15, 4, 4, 2, 3, 5, 6, 4,5

sequence for pages

LRU Analysis

assumption: we start with empty cache (any algorithm)

Theorem

For cache of size k, LRU is k-competitive.

1. Upper bound: divide input sequence into phases.
   - First phase starts immediately after our algorithm first faults, ends right after the algorithm faults k more times
   - Second phase starts at end of first phase, ends when algorithm faults for additional k times
   - and so on...
2. We will prove that OPT algorithm faults *at least* once per phase
3. This gives us that $C_A \leq k \cdot C_{opt}$, which is what we want.
4. Examples of phases, for $k = 3$:

1, 1, 2, 2, 1, 3, 4, 3, 2, 4, 5, 6, 15, 4, 4, 2, 3, 5, 6, 4,5

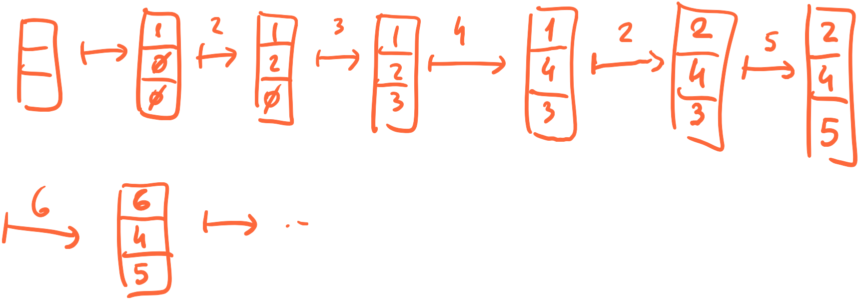
1 (1, 2, 2, 1, 3, 4) (3, 2, 4, 5, 6) (15, 4, 4, 2) (3, 5, 6) (4, 5

first miss — 3 cache misses

LRU Analysis - Example

Examples of phases, for $k = 3$:

1 (1, 2, 2, 1, 3, 4) (3, 2, 4, 5, 6) (15, 4, 4, 2) (3, 5, 6) (4, 5



LRU Analysis - Upper Bound

- Need to prove that OPT will fault at least once per phase.
- If the same page faulted twice in one phase:

page p faulted twice in one phase (we faulted when requesting p)

intermediate requests

r_1 r_2 r_3 -- r_{m-1} r_m within a phase

r_1: page p — first req. of p that faulted; had no page p in cache

r_m: page p and faulted again

⇒ must have queried k-1 different pages (∵ p LRU page) (must have evicted p) ← extra page

⇒ must have made k+1 distinct requests in this phase

(k-1 distinct pages after r_1 + page that evicted p + page p)

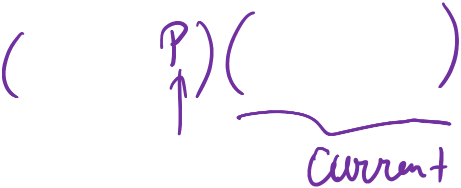
pigeon hole principle OPT has to fault once

LRU Analysis - Upper Bound

- If each page faulted once in a phase.

LRU Analysis - Upper Bound

- If each page faulted once in a phase.
- **Claim:** in the beginning of each phase, content of OPT and content of our algorithm A intersect in at least one page.
- Proof: Look at last fault page in previous phase.



A faulted at P so in current phase cache of A has page P.

if OPT did not fault at P OPT already had P in its cache

faulted at P then P will now be in OPT's cache

LRU Analysis - Upper Bound

- If each page faulted once in a phase.
- **Claim:** in the beginning of each phase, content of *OPT* and content of our algorithm *A* intersect in at least one page.

- Since *OPT* and *A* had a common page, then *OPT* must have faulted as well (since each page faulted in this phase)

⇒ have k distinct page faults

⇒ we will have k distinct page requests

OPT could only have already had k-1 out of the k pages that were requested

$P_1, \ldots P_k$ are the faulty requests $P_i \neq P_j$ $i \neq j$ $P \in$ Cache$_{OPT}$

OPT ⟷ $(P_1\ P_2 \ldots P_k)$ in beg. of phase but we know this doesn't happen because and $P \neq P_i$'s

Lower Bound - Deterministic Paging Algorithms

Theorem

Any deterministic algorithm for paging with k pages is at least k-competitive!

- Proof by trolling.[7] Let's use $k + 1$ pages, and let A be our paging algorithm.

[7] Common lower bound technique for online algorithms, also commonly used online as well :)

Lower Bound - Deterministic Paging Algorithms

Theorem

Any deterministic algorithm for paging with k pages is at least k-competitive!

- Proof by trolling.[7] Let's use $k + 1$ pages, and let A be our paging algorithm.
- **Input sequence:** at each step, request page that A *doesn't have*.

[7] Common lower bound technique for online algorithms, also commonly used online as well :)

Lower Bound - Deterministic Paging Algorithms

Theorem

Any deterministic algorithm for paging with k pages is at least k-competitive!

- Proof by trolling.[7] Let's use $k+1$ pages, and let A be our paging algorithm.
- **Input sequence:** at each step, request page that A *doesn't have*.
- A faults every single time.

[7] Common lower bound technique for online algorithms, also commonly used online as well :)

Lower Bound - Deterministic Paging Algorithms

Theorem

Any deterministic algorithm for paging with k pages is at least k-competitive!

- Proof by trolling.[7] Let's use $k+1$ pages, and let A be our paging algorithm.
- **Input sequence:** at each step, request page that A *doesn't have*.
- A faults every single time.
- **Offline Algorithm:** on cache miss, delete page which is requested *furthest in the future*.

[7] Common lower bound technique for online algorithms, also commonly used online as well :)

Lower Bound - Deterministic Paging Algorithms

Theorem

Any deterministic algorithm for paging with k pages is at least k-competitive!

- Proof by trolling.[7] Let's use $k+1$ pages, and let A be our paging algorithm.
- **Input sequence:** at each step, request page that A *doesn't have*.
- A faults every single time.
- **Offline Algorithm:** on cache miss, delete page which is requested *furthest in the future*.
- When offline algorithm deletes a page, it's next delete happens after at least k steps.

[7] Common lower bound technique for online algorithms, also commonly used online as well :)

Conclusion

- Online algorithms are important for many applications, when we need to make decisions right when we receive the information.
- Applications in
  - Stock Market
  - Dating
  - Skiing
  - Caching
  - Machine Learning (regret minimization)
  - many more...
- *Competitive Analysis*: measures performance of our algorithm against best algorithm that could *see into the future*

Acknowledgement

- Lecture based largely on:
  - Lecture 17 of Luca's Optimization class
  - Lectures 19 and 20 of Karger's 6.854 Fall 2004 algorithms course
  - [Motwani & Raghavan 2007, Chapter 13]
- See Luca's Lecture 17 notes at

  `https://lucatrevisan.github.io/teaching/cs261-11/lecture17.pdf`
- See Karger's Lecture 19 notes at

  `http://courses.csail.mit.edu/6.854/06/scribe/s22-online.pdf`
- See Karger's Lecture 20 notes at

  `http://courses.csail.mit.edu/6.854/06/scribe/s24-paging.pdf`

References I

Motwani, Rajeev and Raghavan, Prabhakar (2007)
Randomized Algorithms