LEC 9

CSE 122

Nested Collections

Talk to your neighbors:
Favorite warm weather drink?
Lemonade? Iced tea? Soda? Juice?

Instructor: Ido Avnon

TAs: Abby Williams
Chloë Mi Cartier
Connor Sun
Cynthia Pan
Katharine Zhang
Marcus Sanches
Rohini Arangam

Questions during Class?
Raise hand or send here

sli.do #cse122

Agenda

- **Announcements**
- Recap: Nested Collections
- Practice: Search Engine
- Practice: Generate Acronyms

Announcements

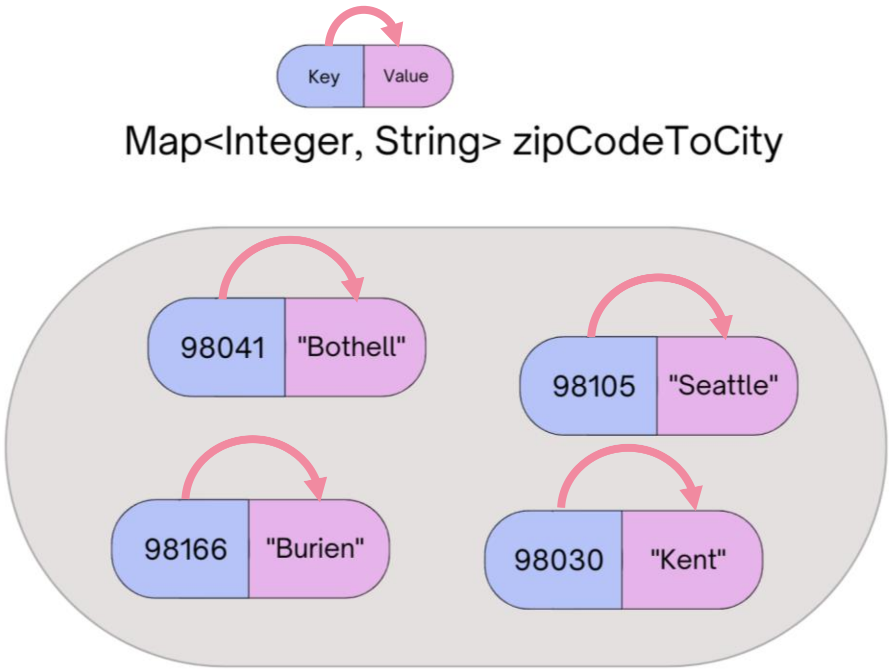
- Programming Assignment 2 (P2) **is now due Saturday July 27th!**
- Quiz 1 on **July 25th** in your registered Quiz Section
  - Topics: (Reference Semantics), Stacks and Queues, Sets, Maps
  - Practice Quiz 1 available, along with Extra Practice problems (by topic)
- Resubmission Cycle form out, due July 30th by 11:59

Agenda

- Announcements
- **Recap: Nested Collections**
- Practice: Search Engine
- Practice: Generate Acronyms

Map ADT

- Data structure to map keys to values
  - Keys can be any* type; Keys must be unique
  - Values can be any type
- Example: Mapping ticker to stock price in P0
- Operations
  - `put(key, value)`: Associate key to value
    - Overwrites duplicate keys
  - `get(key)`: Get value for key
  - `remove(key)`: Remove key/value pair

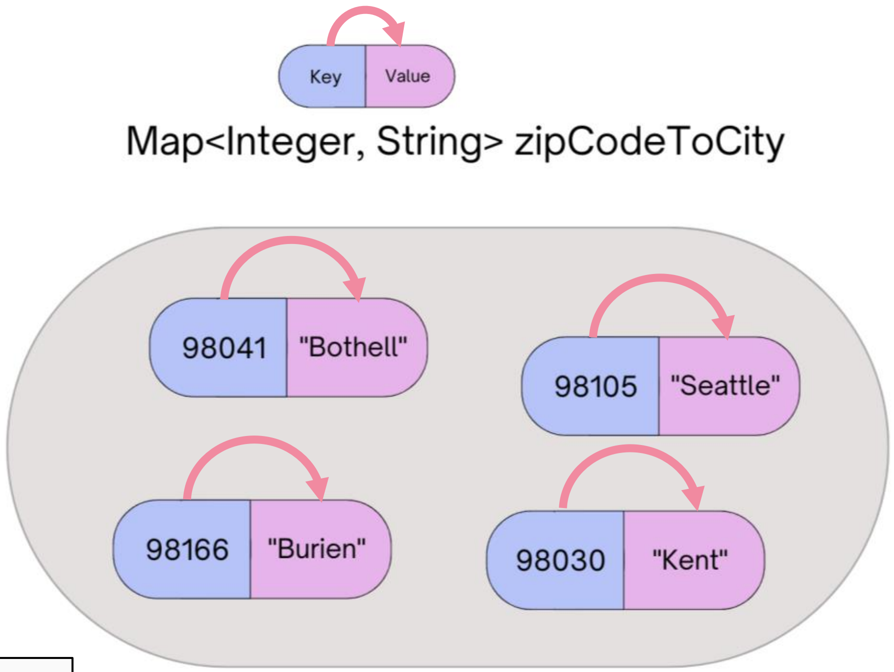


Same as Python's `dict`

Updating ‘Primitive Maps’

The “value” inside the Map is **NOT** reference to the data, it is the data itself!

You cannot “change” the data, you have to reassign the key to a different value!



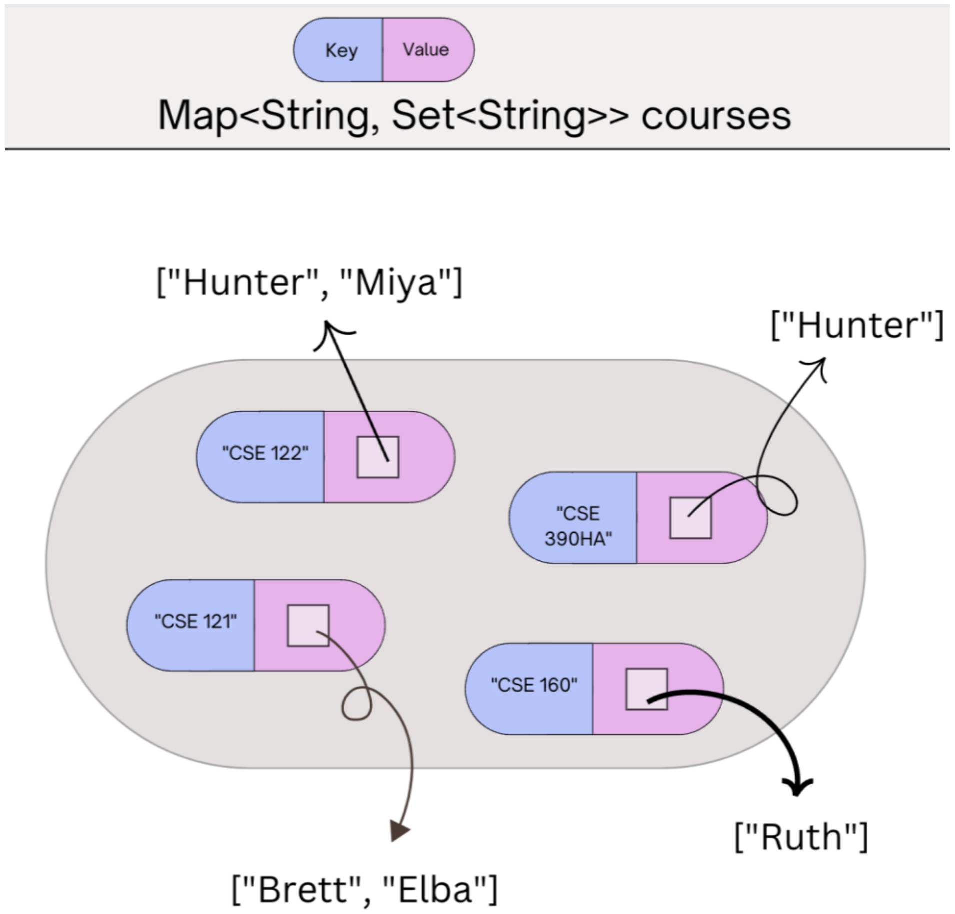

```
courses.put(98155, “Lake City?");

courses.get(98155) = “Lake Forest Park“;

courses.put(98155, “Lake Forest Park");
```

Nested Collections

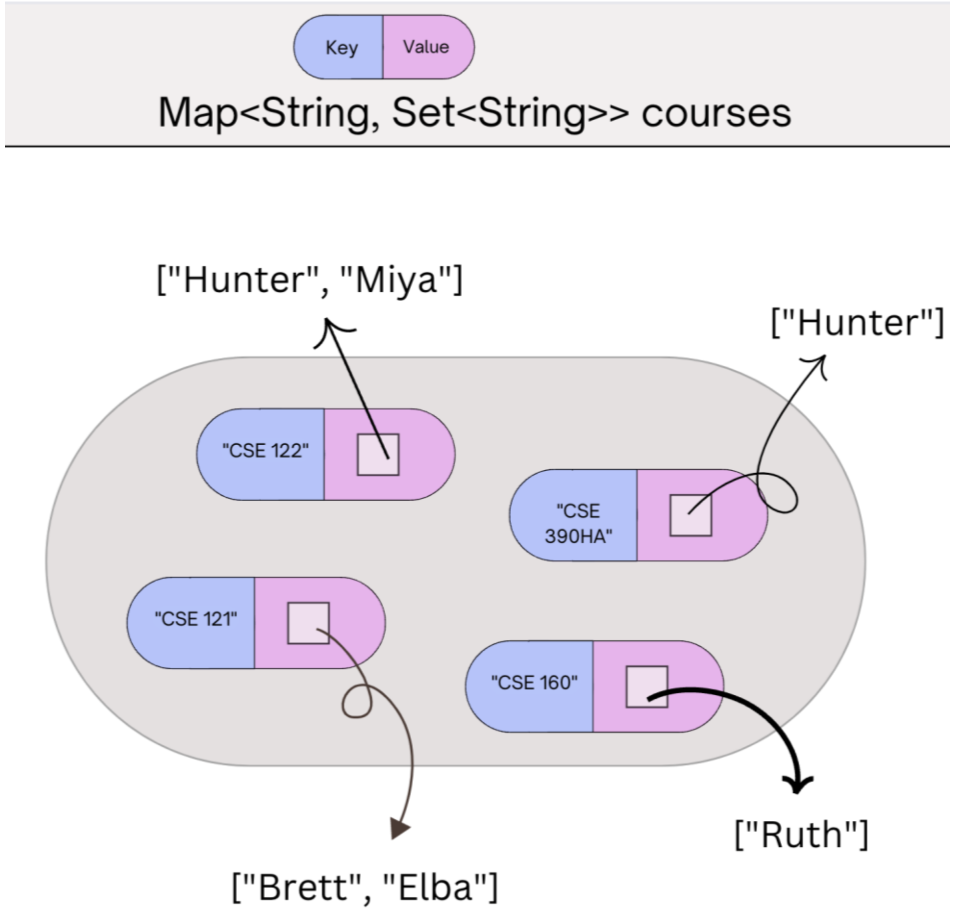
- The values inside a Map can be any type, including <u>data structures</u>
- Common examples:
  - Mapping: Section ➔ Set of students in that section
  - Mapping: Recipe ➔ Set of ingredients in that recipe
    - Or even Map<String, Map<String, Double>> for units!



Updating Nested Collections

The “value” inside the Map is a <u>reference</u> to the data structure!

- Think carefully about number of references to a particular object




```
courses.put("CSE 123", new HashSet<String>());
courses.get("CSE 123").add("Nathan");

Set<String> cse123 = courses.get("CSE 123");
cse123.add("Joe");
```

Practice : Think

sli.do #cse122

Suppose map had the following items. What would its items be after running this code?

```
map: {”KeyA"=[1, 2], ”KeyB"=[3], ”KeyC"=[4, 5, 6]}
```

```
Set<Integer> nums = map.get("KeyA");
nums.add(7);
map.put("KeyB", nums);
map.get("KeyA").add(8);
map.get("KeyB").add(9);
```

```
A. {"KeyA"=[1, 2],          "KeyB"=[1, 2, 7],          "KeyC"=[4, 5, 6]}
B. {"KeyA"=[1, 2, 8],       "KeyB"=[1, 2, 7, 9],       "KeyC"=[4, 5, 6]}
C. {"KeyA"=[1, 2, 7, 8],    "KeyB"=[1, 2, 7, 9],       "KeyC"=[4, 5, 6]}
D. {"KeyA"=[1, 2, 7, 8, 9], "KeyB"=[1, 2, 7, 8, 9],    "KeyC"=[4, 5, 6]}
```

Practice : Pair

sli.do #cse122

Suppose map had the following items. What would its items be after running this code?

```
map: {"KeyA"=[1, 2], "KeyB"=[3], "KeyC"=[4, 5, 6]}
```

```
Set<Integer> nums = map.get("KeyA");
nums.add(7);
map.put("KeyB", nums);
map.get("KeyA").add(8);
map.get("KeyB").add(9);
```

nums →

A: [1, 2, 7, 8, 9]

B: ~~[3]~~

C: [4, 5, 6]

```
A. {"KeyA"=[1, 2],          "KeyB"=[1, 2, 7],          "KeyC"=[4, 5, 6]}
B. {"KeyA"=[1, 2, 8],       "KeyB"=[1, 2, 7, 9],       "KeyC"=[4, 5, 6]}
C. {"KeyA"=[1, 2, 7, 8],    "KeyB"=[1, 2, 7, 9],       "KeyC"=[4, 5, 6]}
D. {"KeyA"=[1, 2, 7, 8, 9], "KeyB"=[1, 2, 7, 8, 9], "KeyC"=[4, 5, 6]}
```

Agenda

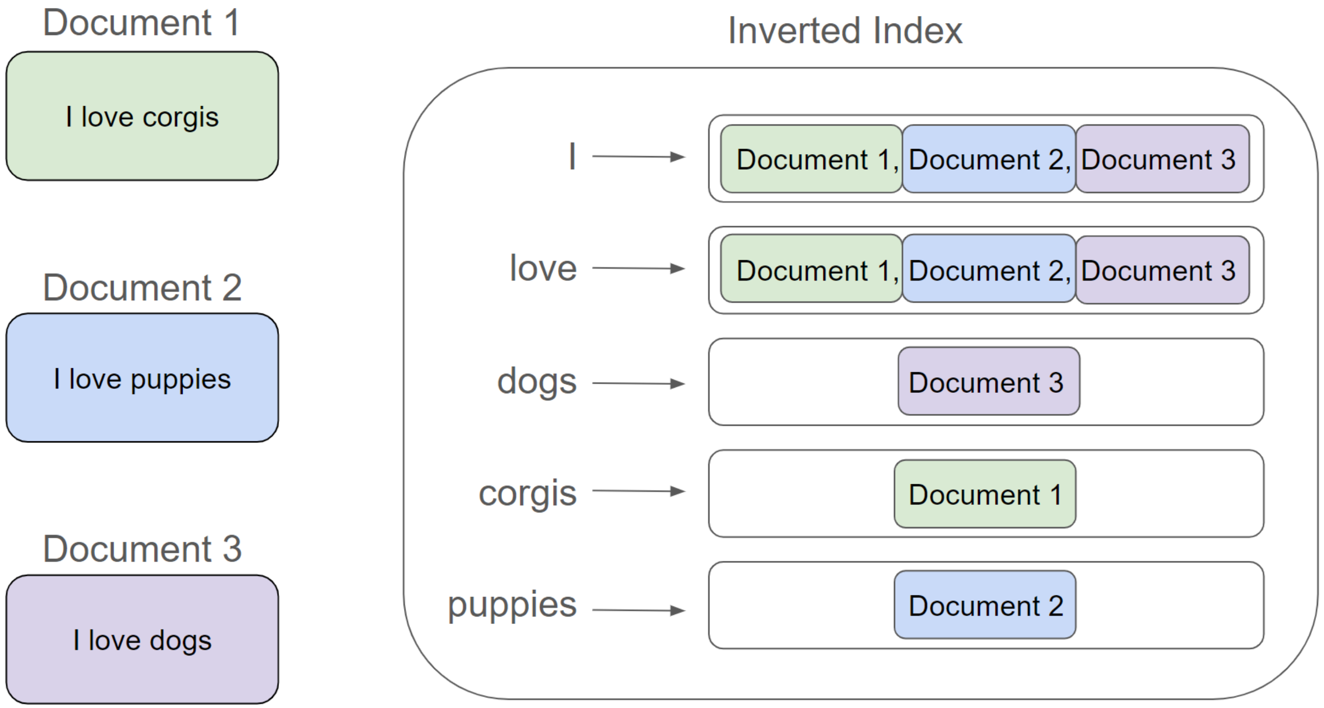
- Announcements
- Recap: Nested Collections
- Practice: Search Engine
- Practice: Generate Acronyms

Background: Search Engines

- A **search engine** receives a **query** and returns a set of relevant **documents.** Examples: Google.com, Mac Finder, more.
  - Queries often can have more
- A search engine involves two main components
  - An **index** to efficiently find the set of documents for a query
    - Will focus on “single word queries” for today’s example
  - A **ranking algorithm** to order the documents from most to least relevant
    - Not the focus of this example
- Goal: Precompute a data structure that helps find the relevant documents for a given query

Inverted Index

- An **inverted index** is a Mapping from possible query words to the set of documents that contain that word
  - Answers the question: “What documents contain the word ‘corgis’?”



slido

Please download and install the Slido app on all computers you use

What will be the type of the inverted index?

ⓘ Start presenting to display the poll results on this slide.

Ranking Results

- There is no one right way to define which documents are "most relevant" There are approximations, but make decisions about what relevance means
- Idea 1: Documents that have more hits of the query should come first
  - Pro: Simple
  - Con: Favors longer documents (query: "the dogs" will favor long documents with lots of "the"s)
- Idea 2: Weight query terms based on their "uniqueness". Often use some sort of score for "Term Frequency – Inverse Document Frequency (TF-IDF)
  - Pro: Doesn't put much weight on common words like "the"
  - Cons: Complex, many choices in how to compute that yield pretty different rankings
- Idea 3: Much more! Most companies keep their ranking algorithms very very secret ☺

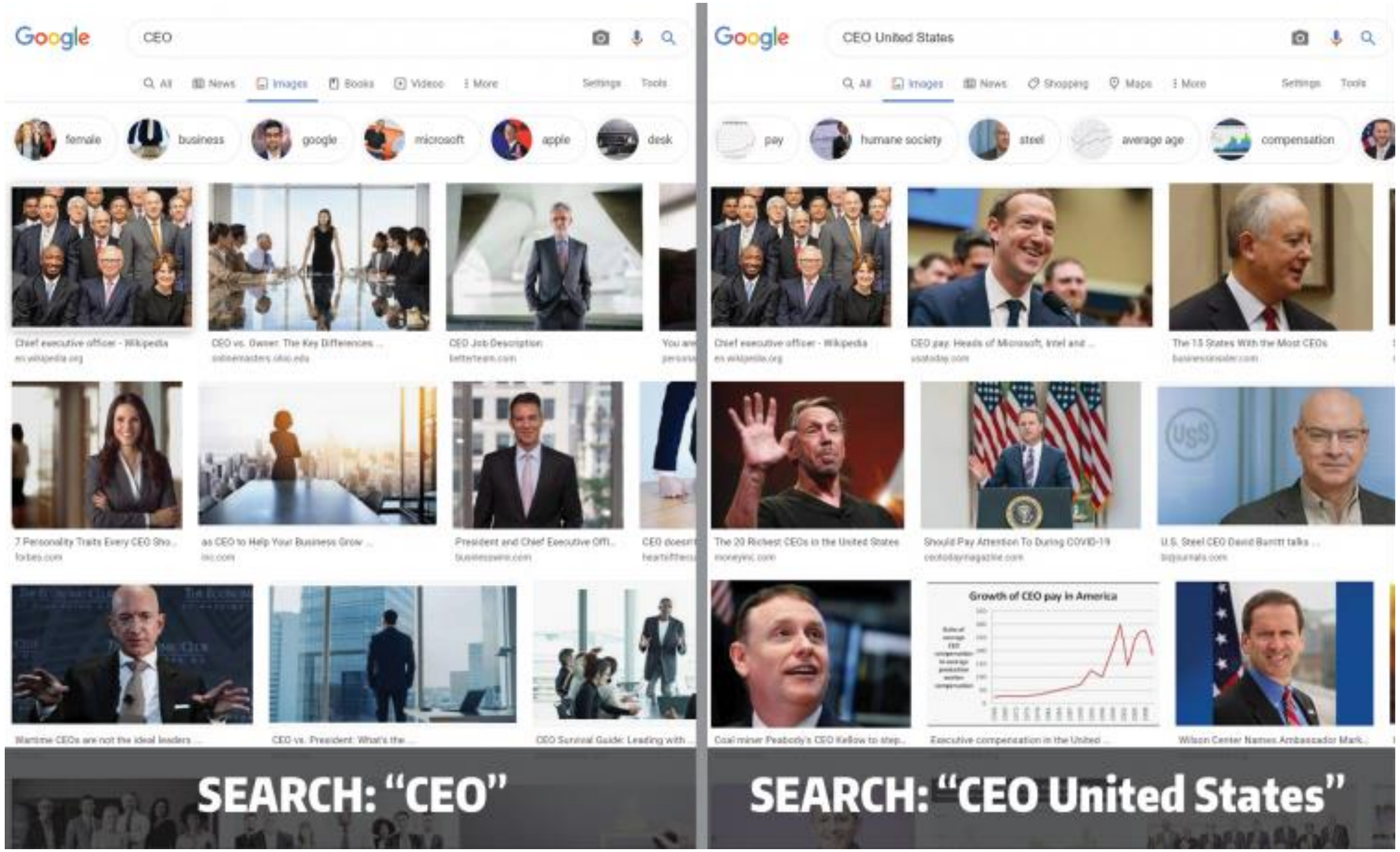
Data Bias

- Image results for searching the term “CEO” on Google (2015)
  - Notice anything about the results?

https://www.washington.edu/news/2015/04/09/whos-a-ceo-google-image-results-can-shift-gender-biases/

Data Bias

- Fix: Image results for searching “CEO” and “CEO United States” (2022)



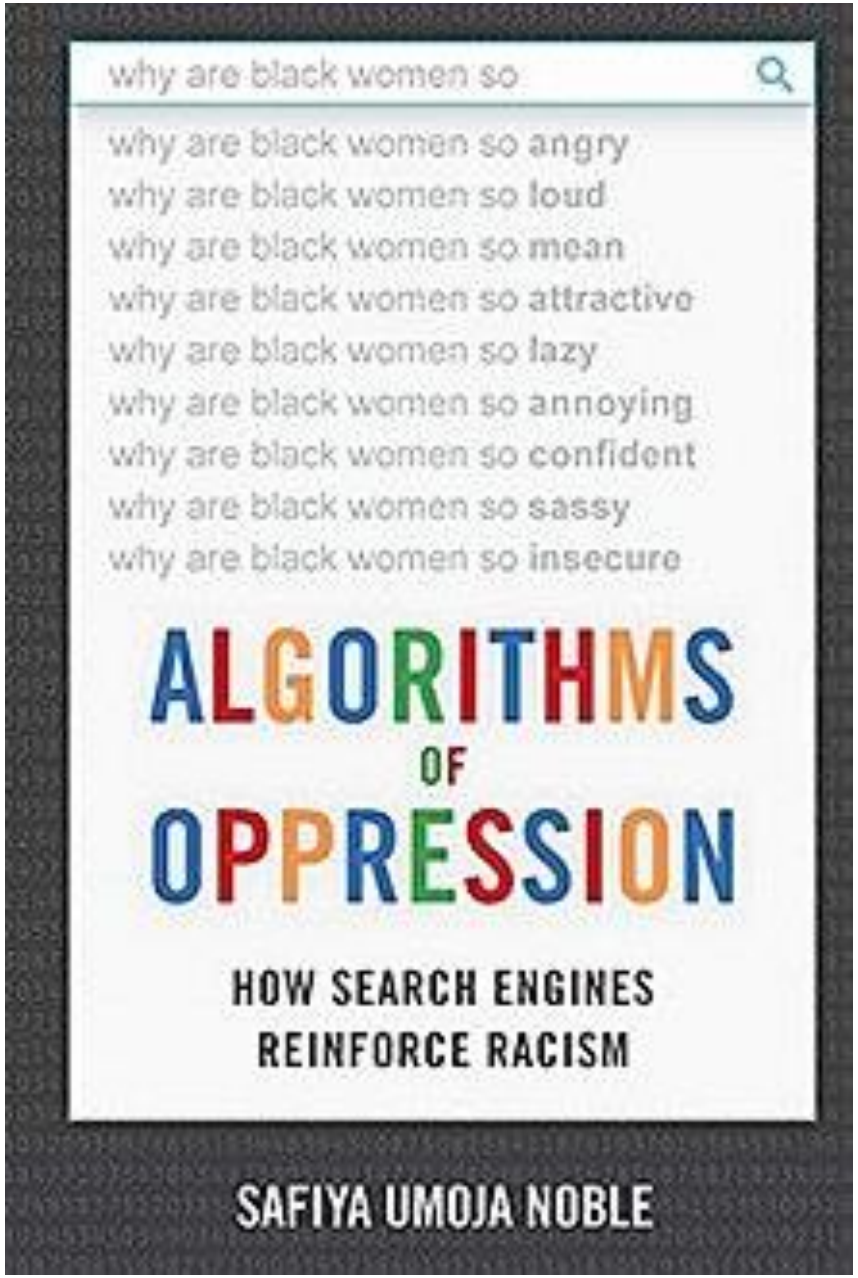
https://www.washington.edu/news/2022/02/16/googles-ceo-image-search-gender-bias-hasnt-really-been-fixed

Data Bias

- Google's autocomplete recommendations used to actually look like this
  - Fix: Don't display autocomplete results for phrases like "why are [group] ____"

<u>Are these changes fixing the right thing?</u>

Btw, this is a great book that you should check out if you're interested ->



Agenda

- Announcements
- Recap: Nested Collections
- Practice: Search Engine
- **Practice: Generate Acronyms**