Discussion 2

Homework 1, Part B

Welcome! Information Retrieval Discussion

Itinerary October 6, 2021

Review of Discussion policies

Homework 1

- Part B
  - Sorting
- Running your homework via the command line

Upcoming Deadlines

Friday: Course policies quiz

Ref: https://www.youtube.com/watch?v=rS00xWnqwvI

Discussion Sections **Logistics**

Interactive Guided Sessions

Led by Brooke

- 3pm, **in person** (SE2 1304)
- 4pm, **virtual & recorded**

Office Hours

Led by Rachel

- 5pm, **in person** (SE2 1304)
- 6pm, **in person** (SE2 1304)

What **3/4pm** Discussion **is**...

- A great place to go if you don't exactly know where to start or what to ask
- Participatory
  - I need your participation to continue on in the session
- Somewhere to get to know your fellow classmates
- Practice solving the **kinds** of problems that appear on homeworks and quizzes
- A safe space to ask questions and learn!

...and **is not**

- Passive lecture
- A great place to ask individually focused guidance (such as debugging your code)
  - Instead, go to office hours!
- Giving away "free" homework answers or doing your work

Questions?

Homework 1

Running on the Command Line

Requirements

- Need a terminal application that uses **<u>git bash</u>**
- If you are on a Mac/Linux, you can simply use the provided **terminal** application, or iTerm2 as suggested last week. (See Discussion 1 slides or recording for more info)
- If you are on Windows, you need to make sure that you install git bash. Follow the instructions provided here: https://gitforwindows.org/

Important Notes

- You **CANNOT** turn in a Jupyter Notebook
  - Follow instructions on Canvas about proper formatting for turn in
  - You will receive **zero** credit for turning in an ipynb file
- See Discussion 1 slides for recommended development setup of environment demonstrated.

Demo

Questions?

Homework 1

Part B

Part B Key Concepts

- Big-O Complexity
  - Don't have time in discussion to review this, but please see this or video for a review
- Leveraging Part A code for Part B
- **Pre-sorting** for improved efficiency in counting algorithm

Shared Tokens Algorithm - Naive Approach (7 min)

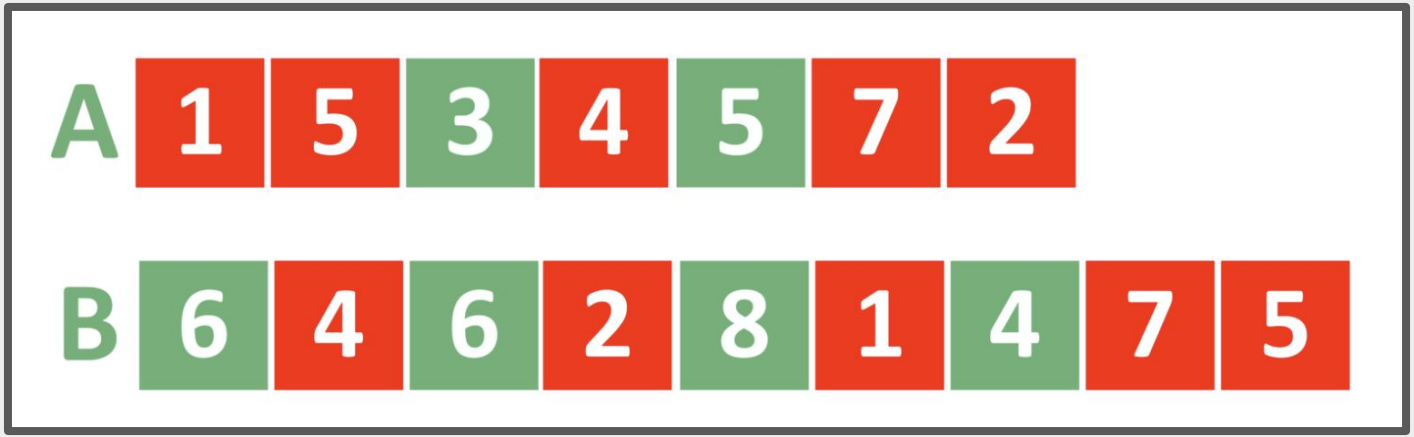
In a team of 2-3 people, write an algo. to find the common tokens **R** of **<u>unsorted</u>** lists **A** and **B**.

A	1	5	3	4	5	7	2		
B	6	4	6	2	8	1	4	7	5

Examples provided by https://www.baeldung.com/cs/list-intersection

Shared Tokens Algorithm - Naive Approach

Big O Complexity?



Examples provided by https://www.baeldung.com/cs/list-intersection

Shared Tokens Algorithm - Pre-Sorted Example (7 min)

In a team of 2-3 people, write an algo. to find the common tokens **R** of **<u>sorted</u>** lists **A** and **B**.

Hint: How can you leverage the sorted lists to make this algorithm faster?

A	1	2	3	4	5	5	7		
B	1	2	4	4	5	6	4	7	5

Examples provided by https://www.baeldung.com/cs/list-intersection

Shared Tokens Algorithm - Pre-Sorted Example

A

1	2	3	4	5	5	7

1 **?** 1

B

1	2	4	4	5	6	4	7	5

R

Examples provided by https://www.baeldung.com/cs/list-intersection

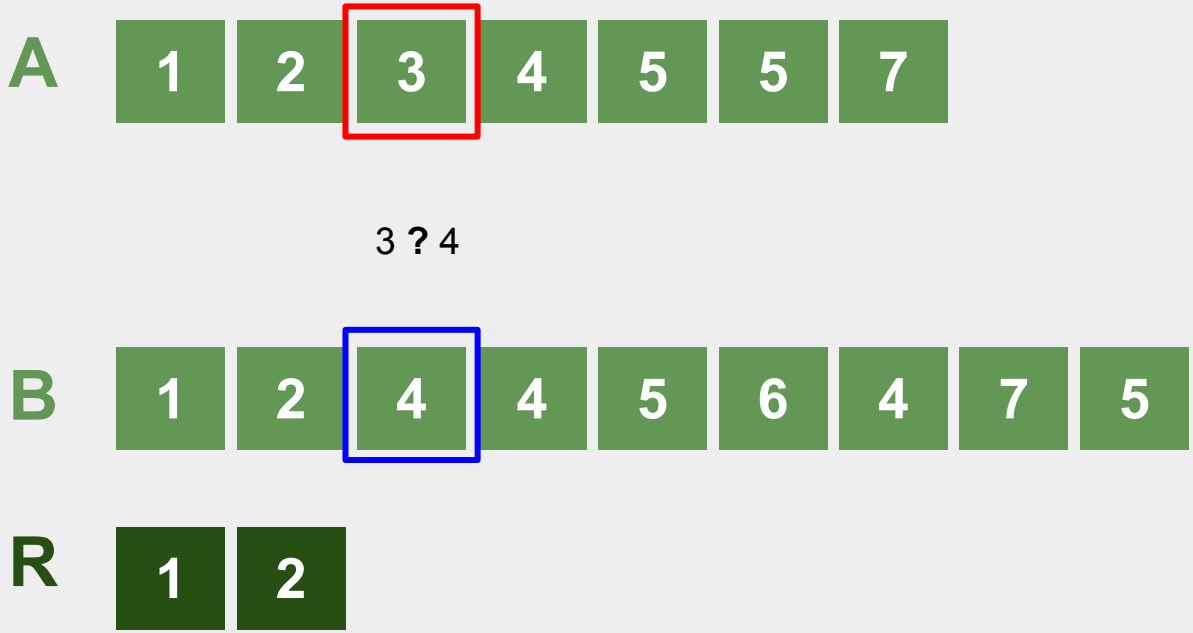
Shared Tokens Algorithm - Pre-Sorted Example

Examples provided by https://www.baeldung.com/cs/list-intersection

Shared Tokens Algorithm - Pre-Sorted Example

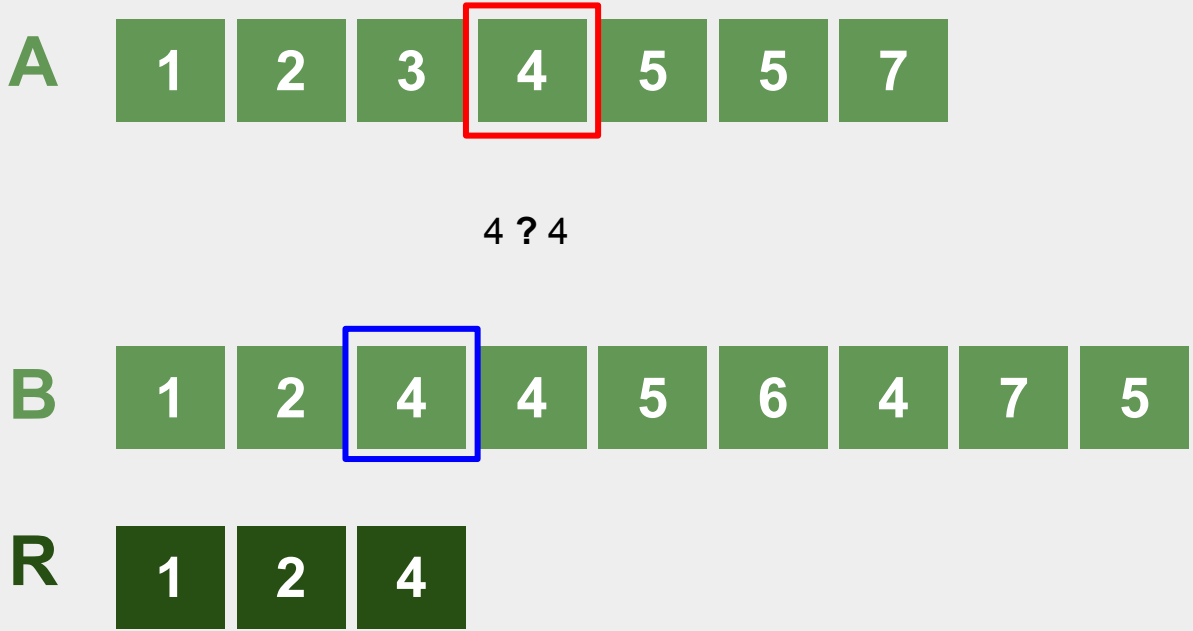
Examples provided by https://www.baeldung.com/cs/list-intersection

Shared Tokens Algorithm - Pre-Sorted Example



Examples provided by https://www.baeldung.com/cs/list-intersection

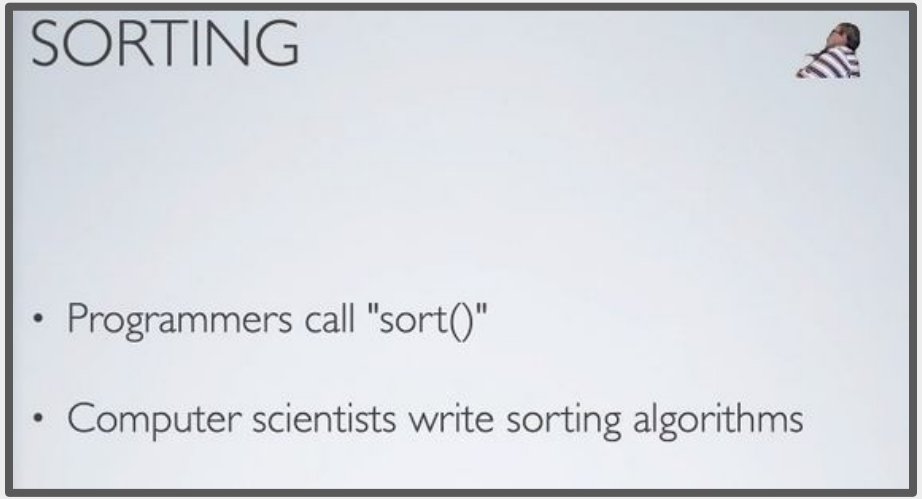
Shared Tokens Algorithm - Pre-Sorted Example



Examples provided by https://www.baeldung.com/cs/list-intersection

Pre-Sorting

- For pre-sorting the lists, it is sufficient to use built-in python functions
- Read the Python [documentation]() to determine the Big-O complexity of the functions you use
  - Be sure to write this up in the homework!



Big-O Complexity (7 min)

- Next steps: write out the pseudocode (or straight to python if you prefer) of the pre-sorted algorithm with the “pointers”

- What is the Big-O complexity of the pre-sorted approach?

- What is the **total** complexity of Part B, from start to finish?
  - Take into account the pre-sorting function + the token-counting function
  - Are there any alternative approaches or further improvements that can be made?

How to Use Part A Code for Part B?

PartA.py

```python
import regex as re
import sys
import os


def tokenize(text_file_path: str) -> list:
    """
    Runtime Complexity: TODO

    Reads in a text file and returns a list of the tokens in that file.
    For the purposes of this project, a token is a sequence of alphanumeric characters, independent of capitalization.

    :param text_file_path: Path to the text file to be read.
    :return: List of the tokens in that file.
    """


def compute_word_frequencies(tokens: list) -> dict:
    """
    Runtime Complexity: TODO

    Counts the number of occurrences of each token in the token list.
    Must write this from scratch.

    :param tokens: List of tokens.
    :return: dict, mapping each token to the number of occurrences.
    """


def print_frequencies(frequencies: dict) -> None:
    """
    Runtime Complexity: TODO

    TODO Choose one of the output formats to print the results:
    <token>\t<freq>
    <token> <freq>
    <token> - <freq>
    <token> = <freq>
    <token> > <freq>
    <token> -> <freq>
    <token> => <freq>

    Prints out the word frequency count, ordered by decreasing frequency (so, the highest frequency words first).

    :param frequencies: dict, mapping each token to the number of occurrences.
    """


if __name__ == '__main__':
    tokens = tokenize(sys.argv[1])
    freqs = compute_word_frequencies(tokens)
    print_frequencies(freqs)
```

PartB.py

```python
import PartA as A
import sys


if __name__ == '__main__':
    text_file_1 = sys.argv[1]
    text_file_2 = sys.argv[2]

    file1_tokens = A.tokenize(text_file_1)
    file2_tokens = A.tokenize(text_file_2)

    # ... etc.
```

File Structure

Next Week’s Session

- Quiz 1
- Come prepared with questions and ready to participate!

Recommended Homework

- Finish Homework 1 by next Tuesday so next week’s discussion we can focus on quiz (not due til October 15)