CODING RULES (VIOLATIONS OF THESE WILL LEAD TO DEDUCTIONS):

- Solutions should be submitted through Gradescope. Our canvas site has a link to it.
- In every file you submit, your name and net-id should be writen in a comment at the top.
- When naming class files, you should use camel-case that starts with an unpper-case letter.
- When naming variables and methods, you should use camel-case that starts with a lower-case letter.
- When defining constants, you should use all-capitals with multiple words connected by underscores.
- Your programs should be auto-formated; indentation errors will lead to deductions.
- Your methods should focus on one goal (this somewhat nebulous statement will become clearer throughout the session); methods that try to accomplish multiple things are often indicators of bad design.

OUR LATENESS POLICY:

- You have one slip day for **each** assignment you can use without having to notify us. **Work submitted beyond the slip day without an academically valid excuse will not earn any points.**
- The deadline to submit documents regarding academically valid excuses is the original deadline of the assignment (i.e. not that of the slip day).
- In order to reward complying with deadlines, anyone who uses only one slip day will have a 1% added to their final score and anyone who uses zero slip days will a 2% added to their final score.

REGARDING ACADEMIC INTEGRITY:

- You may collaborate with one or two of your peers in the course to brainstorm for solving the assigned problems. However, your solution must be written up completely on your own.
- If you collaborate with others during the brainstorming part of the assignment, you must list their names at the beginning of your submission.
- You cannot use any online resources, any solutions from past semesters, etc. You are not allowed to share digital or written notes or images of your work in any form.
- Even when a question feels overwhelming, you should not compromise your academic integrity. Since the exam questions will be similar to the assigned exercises/problems, not struggling with these questions (in the context of assignments) will be counter productive with respect to grades.
- We use software to help us detect plagiarism. Submissions that are marked by this software will be reviewed by the CS-2110 staff. In those case where plagiarism appears to have taken place, we will invoke the academic integrity rules of the institution. As unpleasent an event as that is for everyone, we have no choice.


er

- The package for this project should be called '`proj_2_similarity`'; if not, the autograder will fail.
- The main Java class for this project should be called '`Similarity`'; if not, the autograder will fail.
- If you work off of the Java files from the starter file, you won't have to worry about any of this.

1 Overview

Suppose we are comparing the way different people rank the movies we see in figure 1. In particular, suppose we are trying to find which two people are most similar to each other in their rankings. How would we do it? Who would be those two people, in particular, for the table we see in figure 2?

Figure 1: Four movies to compare/contrast the rankings of different viewers.

Name	0	1	2	4
Ali	Tangerines	Dead Poets Society	Contact	Darjeeling Limited
Donna	Tangerines	Dead Poets Society	Darjeeling Limited	Contact
Cheryl	Dead Poets Society	Tangerines	Contact	Darjeeling Limited
Michael	Tangerines	Contact	Dead Poets Society	Darjeeling Limited
John	Darjeeling Limited	Dead Poets Society	Contact	Tangerines
Elizabeth	Contact	Darjeeling Limited	Dead Poets Society	Tangerines
Steve	Darjeeling Limited	Contact	Dead Poets Society	Tangerines

Figure 2: Seven people's rankings of the movies where 0 indicates "most favorite" and 3 indicates "least favorite".

Name	Number (Ali)
Tangerines	0
Dead Poets Society	1
Contact	2
Darjeeling Limited	3

Name	1	2	3	4
Ali	0	1	2	3
Donna	0	1	3	2
Cheryl	1	0	2	3
Michael	0	2	1	3
John	3	1	2	0
Elizabeth	2	3	1	0
Steve	3	2	1	0

Figure 3: Numbering the movies based on a specific person's preferences (in this case, me) taken as the baseline.

Name	Number
Contact	1
Darjeeling Limited	2
Dead Poets Society	3
Tangerines	4

Name	1	2	3	4
Ali	4	3	1	2
Donna	4	3	2	1
Cheryl	3	4	1	2
Michael	4	1	3	2
John	2	3	1	4
Elizabeth	1	2	3	4
Steve	2	1	3	4

Figure 4: Numbering the movies based on an alphabetical ordering of their names.

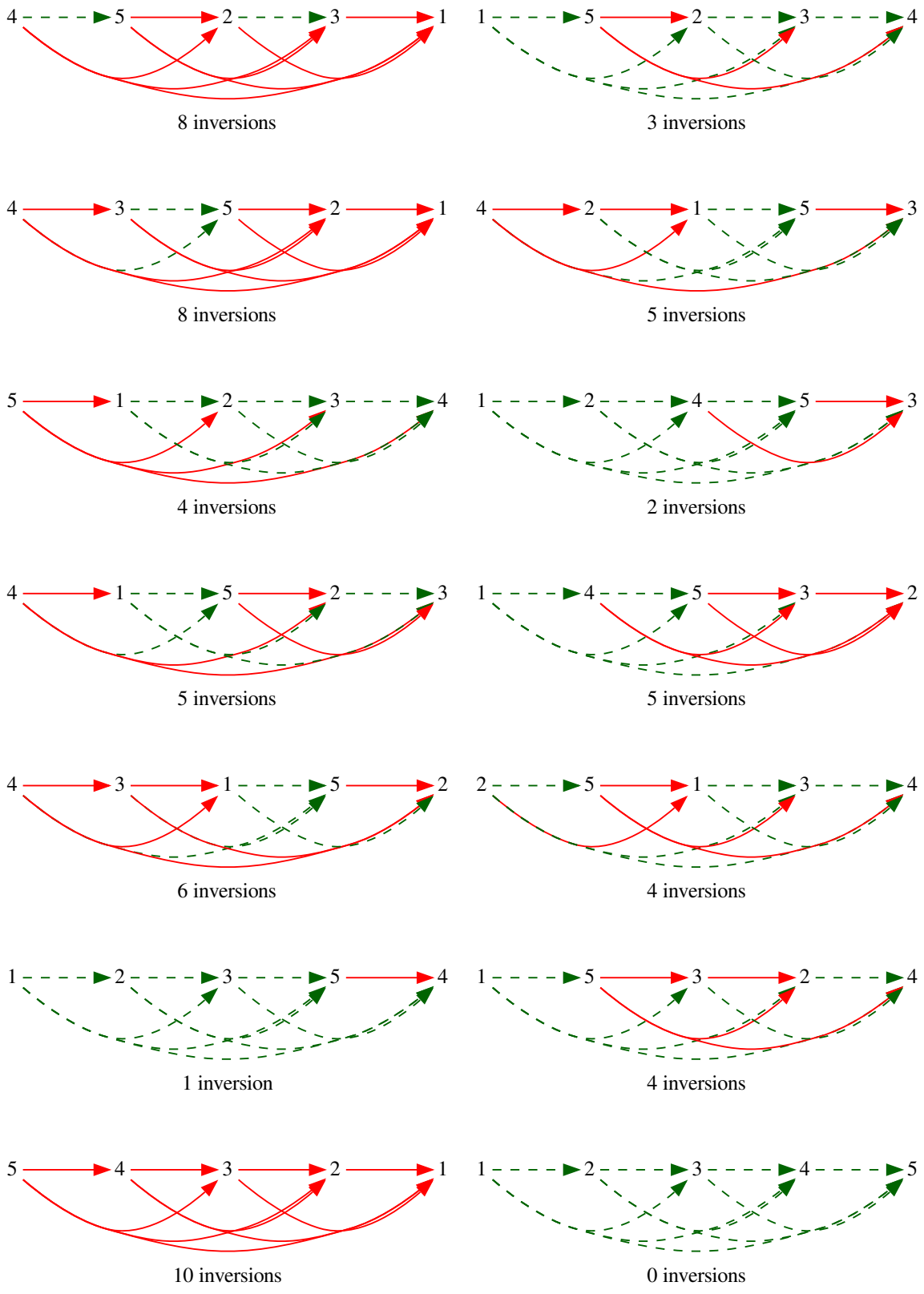

Figure 5: Several permutations of the numbers 1, 2, 3, 4, and 5 as well as the number of inversions each one contains with respect to an ascending order.

We could start by mapping movies to numbers. For example, while I love them all, I might say that my most favorite is first from left, my second favorite is second from the left, etc; the application of this map is shown in figure 3. Another map could associate numbers based on an alphabetical ordering of the names; this is shown in figure 4. Once we have our numbers, we can use MergeSort to discover the number of inversions it takes to convert one row of rankings into another (see figure 5 for details). The goal of this implementation exercise to do just that.

2 Learning Objectives

By completing this assignment, you will

- further understand the operation of MergeSort,
- understand a fundamental measure about order ("inversions"),
- get introduced to the notion of mapping (topic of chapter 3), and
- further understand recursion.

3 Implementation Goals

This program will operate in one of three modes. The first two are required; the third is optional only for people who would like to further sink their teeth in the exercise. In my mind, that includes all of you but you make the call:

- **First command line argument being '`implicit`'**: This mode of the program assumes that the numbering of the movies are according to the rankings of a person x who is not in the input file; movie 0 is x's most favorite, movie 1 is x's second favorite, etc. The goal is to consider all N rows of the input and decide which row is most consistent with the rankings of x. For example, if we run the following command

  ```
  java -cp algs4.jar:. proj_2_similarity.Similarity implicit moviesImplicit1.txt
  ```

  where the data file contains

  ```
  Cheryl 1 0 2 3
  Michael 0 2 1 3
  John 3 1 2 0
  Elizabeth 2 3 1 0
  Steve 3 2 1 0
  ```

  then the output should be '`Cheryl`' because the ordering of her choices is closest to an ascending ordering of the integers 0 through N. But if we run the following command

  ```
  java -cp algs4.jar:. proj_2_similarity.Similarity implicit moviesImplicit2.txt
  ```

  where the data file contains

  ```
  Donna 0 1 2 3
  Cheryl 1 0 2 3
  Michael 0 2 1 3
  John 3 1 2 0
  ```

```
Elizabeth 2 3 1 0
Steve 3 2 1 0
```

then the output should be '`Donna`'.

- **First command line argument being '`first`'**: This mode of the program assumes that person *x* is in position 0 (i.e. the first line of the input file) and that the numbering of the movies is **not** based on *x*. For example, if we run the following command

```
java -cp algs4.jar:. proj_2_similarity.Similarity first moviesFirst1.txt
```

where the data file contains

```
Mark 2 0 3 1
Donna 0 1 3 2
Cheryl 1 0 2 3
Michael 0 2 1 3
John 3 1 2 0
Elizabeth 2 3 0 1
Steve 3 2 1 0
```

then the output should be '`Elizabeth`'. This is because the number of inversions Mark has with everyone in the input file is as follows:

```
Donna: 4
Cheryl: 4
Michael: 2
John: 4
Elizabeth: 1
Steve: 3
```

- **First command line argument being '`any`'**: This mode of the program determines which two people are most consistent with each other. For example, if we run the following command

```
java -cp algs4.jar:. proj_2_similarity.Similarity any moviesAny1.txt
```

where the data file contains

```
Mark 2 0 3 1
Donna 0 1 3 2
Cheryl 1 3 0 2
Michael 0 2 1 3
Elizabeth 2 3 0 1
Steve 3 2 1 0
```

then the output should be '`Elizabeth Mark`' because the number of inversions between every pair of people is as follows:

```
Mark-Donna: 4
Mark-Cheryl: 6
Mark-Michael: 2
Mark-Elizabeth: 1
Mark-Steve: 3

Donna-Cheryl: 2
Donna-Michael: 2
Donna-Elizabeth: 5
Donna-Steve: 5

Cheryl-Michael: 4
Cheryl-Elizabeth: 5
Cheryl-Steve: 3

Michael-Elizabeth: 3
Michael-Steve: 5

Elizabeth-Steve: 2
```

But if we run the following command

```
java -cp algs4.jar:. proj_2_similarity.Similarity any moviesAny2.txt
```

where the data file contains

```
Mark 2 0 3 1
Donna 0 1 3 2
Cheryl 1 0 2 3
Michael 0 2 1 3
Elizabeth 2 3 0 1
Steve 0 1 3 2
```

then the output should be '`Donna Steve`' because the number of inversions between every pair of people is as follows:

```
Mark-Donna: 4
Mark-Cheryl: 4
Mark-Michael: 2
Mark-Elizabeth: 1
Mark-Steve: 4

Donna-Cheryl: 2
Donna-Michael: 2
Donna-Elizabeth: 5
Donna-Steve: 0

Cheryl-Michael: 2
Cheryl-Elizabeth: 5
```

```
Cheryl-Steve: 2

Michael-Elizabeth: 3
Michael-Steve: 2

Elizabeth-Steve: 5
```

Please note that the output of the program must "sort" the two names. That is, '`Mark Elizabeth`' for the first '`any`' example and '`Steve Donna`' for the second '`any`' example would have been considered wrong (by Gradescope).

4 Starter Code

Please start this implementation exercise by downloading the associated starter code from here; the file you need is called '`proj_2_similarity.zip`'. Please extract the zip file at the package. '`proj_2_similarity.zip`' also contains the sample text files used in this description.

5 Instructions

- Download the starter-code ZIP file '`proj_2_similarity.zip`' from our code distribution repository.
- Add your code to '`Similarity.java`'.
- After testing your program so that the correct answers are produced, submit all of the Java files (some modified, some not) in the '`proj_2_similarity`' folder to Gradescope.