Spring 2026 COP 3330 Program #6: Ducks and the Custom Sorting in Java

This assignment will introduce students to two things:

1. Writing a class that implements the Comparable interface so that Java's Arrays.sort can be utilized for a custom sort.

2. Getting a bit of an introduction to programming competitions and the standard format used by most competitions.

Programming Contest Format Background

Most programming contests consist of a set of very well-defined programming challenges that have to be solved by either an individual or team of students in a limited set of time. To make these challenges easy to judge, automatic judging systems run contestants' programs by piping data from a file to standard input and writing the contestant's program's standard output to an output file. Then, that output file is compared with the judge output. In order for the program to be judged correct, the two files have to be a perfect character by character match (ignoring whitespace at the beginning or end).

In practice, here's how that changes a contestant's solution compared to the usual class assignments:

1. There should be NO output prompting the user to enter values.

2. Input must be read precisely in the order specified in the assignment.

3. Output must be produced to standard output (System.out.print, System.out.println) without ANY changes to the specified output format.

The rest of the code (the actual solving of the problem) should be pretty much the same as one would do for a class assignment.

In class on Friday, March 6th, an example called Sorting Student Presentations, will be live-coded in class along with testing to show how students should approach this assignment.

Problem Format

The problem, as it originally appeared in its original contest is replicated on the next page. This format includes a description of the problem, a precise description of what input will be fed to your program, and a precise description of what output your program must produce. Your program will process one case, but we'll test your program on 20 separate input cases we've created.

Problem B: Ducksort

Filename: `ducksort`
Time Limit: `2 seconds`

Trevor the Duck lives in a big park with a pond in the center. All around the pond are ***n*** benches numbered from ***1*** to ***n***. Trevor usually hangs out in the center of the pond, but when he is hungry he likes to see if he can get some easy free food from one of the park goers. Trevor has lived in the pond for a long time, and recently he has started keeping track of how many other ducks visit each bench for food and the expected amount of food distributed from each bench in a day.

Trevor plans to visit each bench in search of food, in a specified order. In particular, he prioritizes benches with more expected food. If multiple benches have the same amount of expected food, he picks the one visited by the fewest amount of ducks first. If two benches have the same statistics exactly, he visits the one with the lower number first.

Given a description for each of ***n*** benches, output the order in which Trevor will visit benches looking for food.

Input

The first line contains an integer, ***n*** ($1 \le n \le 10^5$), the number of benches around the pond. Two lines follow. The first line of these lines has ***n*** integers, the i^{th} of which is how much food is expected at the i^{th} bench. The next line also has ***n*** integers, the i^{th} of which is how many ducks visit the i^{th} bench. All of the integers are positive and at most 10^9.

Output

Output a permutation of the integers 1 to ***n*** representing the order in which Trevor will visit the benches.

Samples

Input	Output
3 2 3 2 2 9 1	2 3 1
3 1 1 1 3 1 2	2 3 1

Implementation requirements

Please write two classes in a single file to solve this problem:

1. Bench class that implements Comparable<Bench>

2. Ducksort class that has a main method that will be executed.

Both files should be stored in the file Ducksort.java with the class Ducksort being public.

Please store the appropriate instance variables in a Bench object and write a compareTo method in this class that is consistent with the rules stated in the problem.

In your Ducksort class, please create an array of Bench of the appropriate size, fill that array with the appropriate Bench objects, and then sort that array using Arrays.sort. Finally, output the result to the problem using the sorted array.

Deliverables

Please submit a single file, **Ducksort.java** which contains two classes: Bench and Ducksort.

Make sure to include a header comment for each file you submit and appropriate internal comments.