CS-203: COMPUTING AND ALGORITHMS III
ASSIGNMENT 1 – 2017 SUMMER

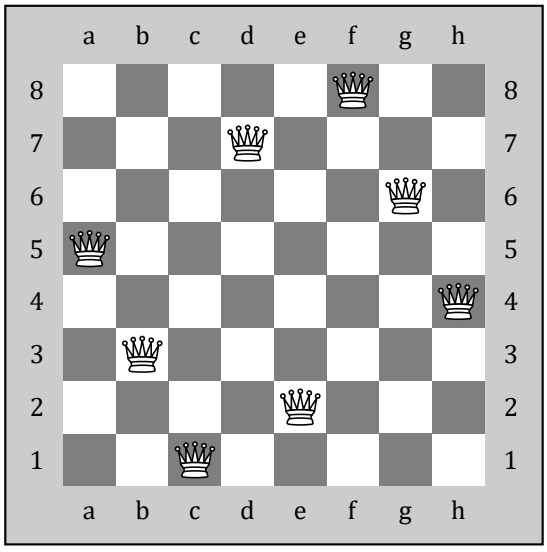
GIUSEPPE TURINI - KETTERING UNIVERSITY

BACKGROUND: THE N-QUEENS PROBLEM

The *eight queens problem* is a classic puzzle, originally proposed in 1848 by chess player Max Bezzel. The puzzle asks the player to place eight queens on a traditional 8×8 chessboard so that no two queens are attacking one another. That is, the objective is to select eight positions in an 8×8 square grid such that no two positions appear on the same row, column, or diagonal. [1]

The *n-queens problem* is a simple generalization of the eight queens problem. In the *n*-queens problem, given a positive integer *n*, the objective is to place *n* queens on an *n*×*n* square grid such that no two queens are attacking one another. The problem has solutions for every *n* except *n*=2.

The original eight queens problem (*n*=8) has 92 distinct solutions; the number of solutions drops to 12 if one disregards solutions that are rotations or reflections of the original board. As an example, below you can see the only symmetrical solution (excluding rotation and reflection) to the eight queens puzzle.



The *n*-queens problem can be straightforwardly solved by a *brute force* approach. Several simplifications can be made to the brute force algorithm to reduce its running time; observe, for example, that any correct solution has exactly one queen in each row, and exactly one queen in each column. However, most brute force approaches require exponential time (or worse).

One approach to solving the problem is the *"iterative repair" algorithm*. In this approach, the

[1] See: Wikipedia – Eight Queens Puzzle.

algorithm begins by placing one queen in each row, using random columns. If this placement yields a solution, the algorithm (obviously) halts. If not, the program then counts the number of conflicts (or attacks) for each queen, selects the queen with the largest number of conflicts, and moves it to the column on its row with the fewest number of conflicts. (Ties are broken arbitrarily.) [2]

THE ASSIGNMENT

The following is the detailed description of the assignment:

1. Write a Java program which, given in input *n* on the command line (where n is an integer with n≥4), finds a solution to the *n*-queens problem by *brute force*, and prints the result to standard output.
2. Write a Java program which, given in input *n* on the command line (where n is an integer with n≥4), finds a solution to the *n*-queens problem by the *"iterative repair" algorithm*, and prints the result to standard output.
3. Instrument both programs above with calls to `System.nanoTime`, which returns the current time of the available system timer in nanoseconds. [3] Both programs should measure the total time used in finding the solution, and print that time value to standard output in a *"readable" format*. Note that the total time computed should *exclude* the time used in printing the final results. (That is: the time computed should *only* count the time used to perform each algorithm.)
4. Create an ordinary text file called *"readme.txt"*, with instructions on how to run your programs.
5. Create an ordinary text file called *"writeup.txt"*, with answers to the following questions:
   - What is the theoretical *worst-case* running time of each of the two algorithms you implemented (i.e. in Θ-notation)? Justify your answer.
   - On average, how much time does each algorithm take to run on the original eight-queens problem (i.e. when n=8)? Obviously, you will need to run your programs several times to generate an average.

Remember: your program should use a *"good"* coding style, since a portion of the grade on this assignment will depend on the coding style you used (see the course syllabus).

[2] See: Wikipedia – Iterative Repair Algorithm.
[3] See: Java Standard Edition (SE) Online Documentation - System nanoTime.

SUBMITTING YOUR PROGRAM

Before the 11:59 p.m. of the 4th Friday, you must:

- send an email to the instructor (i.e. gturini@kettering.edu) using your Kettering email account, and setting the subject to *"CS-203: Assignment 1"*;
- attach to this email a zip archive (i.e. a file with extension *".zip"*), with a name in the format *"cs203_a1_gturini.zip"*, where: *"a1"* is the acronym for *"assignment 1"*, and *"gturini"* must be replaced by your Kettering login;
- store in this zip archive: (1) all and only the source code files of your assignment, and (2) the two text files named above (i.e. the file *"readme.txt"*, and the file *"writeup.txt"*). [4]

Please consider that any deviation from these submission rules will result in a loss of points when computing the grade for this assignment.

NOTES

1 *Start early!*

2 Please use an appropriate coding style (25% of your grade for this assignment is allocated for style).

3 For this and all successive programs, unless otherwise specified, you may use any classes from the Java Standard Edition (SE) framework that are useful to you. [5]

4 `System.nanoTime` returns a `long` value representing the current time, relative to some unknown starting value. [6] Successive calls to `System.nanoTime` allow the measurement of the elapsed time between calls.

5 The use of `System.currentTimeMills` (which returns the current time as the number of milliseconds since midnight 1/1/1970 UTC) for this assignment is not recommended, as the running time of your algorithm on small data sets might be less than a millisecond. [7]

[4] Please do not include *".class"* files or IDE configuration files in the zip archive you submit.
[5] See: Java Standard Edition (SE) Online Documentation.
[6] See: Java Standard Edition (SE) Online Documentation - Primitive Data Types.
[7] See: Wikipedia - Universal Time Coordinated (UTC).

CS-203: COMPUTING AND ALGORITHMS III
ASSIGNMENT 1 - 2017 SUMMER

GIUSEPPE TURINI - KETTERING UNIVERSITY

EVALUATION FORM

STUDENT NAME:

FUNCTIONALITY	_____ / 50
Brute force	_____ / 25
Correct answers (10)	
Correct brute force technique (10)	
Timing (5)	
Iterative repair	_____ / 25
Correct answers (10)	
Correct iterative repair technique (10)	
Timing (5)	
DESIGN AND ANALYSIS	_____ / 25
Proper use of OOP principles (5)	
Theoretical analysis (10)	
Empirical analysis (10)	
STYLE	_____ / 25
Comments and headers (inline, vars, files, methods) (10)	
Naming conventions (vars, funcs, classes, constants) (10)	
Files (one per class, proper submission) (5)	
TOTAL	_____ / 100