CS 4800: Algorithms and Data
Due **Thursday, October 4, 2012** at **9:00am**

Homework 2

Group Formation: This homework will be done in groups of 2-4. Each group must send one email to `cce@ccs.neu.edu`, copied to all other group members, announcing the homework group by **Wednesday, September 26, 2012** at **9:00am**.

Submission Instructions: This homework will be submitted online via email. Send your submissions to `cce@ccs.neu.edu` in the following format:

1. The subject of the email must be "`CS 4800 Homework 2 Submission`".
2. The recipients of the email must include all members of the team.
3. The homework solution must be included as an attachment.
   (a) The attachment must be in `.zip` or `.tar.gz` format.
   (b) The attachment's name must be the team members' last names in alphabetical order, lower case, separated by hyphens, plus the appropriate extension. For example, the team of Sam Raimi and John Doe would submit either `doe-raimi.zip` or `doe-raimi.tar.gz`.
   (c) The attachment must contain a single top-level directory of the same name, minus extension. This directory must contain the following:
      i. Solutions to prose exercises must be included as `solution.pdf`.
      ii. Solutions to programming exercises must be included as executable programs that run correctly on `login.ccs.neu.edu`; filenames are given in the exercises below.
      iii. The source code for each exercise must be included. The file `README.txt` must specify the name, subdirectory (if any), and language of each program's source code.
      iv. Any other files may be included in support of the assignment (*e.g.*, external programs or libraries necessary to run the executables), but their contents will be disregarded for purposes of grading.

Exercise 1 (10 points)

Problem: The Stable Matching Problem, as discussed in the text, assumes that all men and women have a fully ordered list of preferences. In this problem we will consider a version of the problem in which men and women can be *indifferent* between certain options. As before we have a set M of n men and a set W of n women. Assume each man and each woman ranks the members of the opposite gender, but now we allow ties in the ranking. For example (with $n = 4$), a woman could say that m_1 is ranked in first place; second place is a tie between m_2 and m_3 (she has no preference between them); and m_4 is in last place. We will say that *w prefers m to m'* if m is ranked higher than m' on her preference list (they are not tied).

With indifferences in the rankings, there could be two natural notions for stability. And for each, we can ask about the existence of stable matchings, as follows.

1. A *strong instability* in a perfect matching S consists of a man m and woman w, such that each of m and w prefers the other to their partner in S. Does there always exist a perfect matching with no strong instability? Either give an example of a set of men and women with preference lists for which every perfect matching has a strong instability; or give an algorithm that is guaranteed to find a perfect matching with no strong instability *and prove the guarantee*.
2. A *weak instability* in a perfect matching S consists of a man m and a woman w such that their partners in S are w' and m', respectively, and one of the following holds:
   - m prefers w to w', and either w prefers m to m' or is indifferent; or
   - w prefers m to m', and m either prefers w to w' or is indifferent.

   In other words, the pairing between m and w is either preferred by both, or preferred by one while the other is indifferent. Does there always exist a perfect matching with no weak instability? Either give an example of a set of men and women with preference lists for which every perfect matching has a weak instability; or give an algorithm that is guaranteed to find a perfect matching with no weak instability *and prove the guarantee*.

Exercise 2 (10 points)

Problem: Consider sorting a sequence S of n numbers by first finding the smallest element and using it as the first element of the result. Then find the second smallest element, and use it as the second element of the result. Continue in this manner for all n elements of S. This algorithm is known as *selection sort*.

1. Implement selection sort for lists. The executable file `selection-sort-list` must read a JSON sequence of integers from `stdin` and write the corresponding sorted JSON sequence of integers to `stdout`.
2. Give the best- and worst-case running times of selection sort for lists in Θ-notation.
3. Implement selection sort for arrays. The executable file `selection-sort-array` must read a JSON sequence of integers from `stdin` and write the corresponding sorted JSON sequence of integers to `stdout`. For full credit, the algorithm must be entirely *in-place*: you may not allocate any extra arrays.
4. Give the best- and worst-case running times of selection sort for arrays in Θ-notation.

Exercise 3 (10 points)

Problem: Consider an algorithm for sorting a prefix of a sequence. To sort a prefix of *at least* length n, follow these steps:

1. Find the longest forward- or backward-sorted prefix of the sequence.
2. Reverse the prefix if necessary.
3. If the sorted prefix's length k is at least n, return the prefix.

4. Otherwise, continue with the rest of the list. Recursively sort a prefix of the remainder, of at least length $\lfloor k/2 \rfloor$. (***Errata:*** the minimum length of the new prefix was previously given as k. Rounding down to half of k affects the complexity of the algorithm, though not its correctness.)
5. Merge the old and new prefixes together.
6. Repeat from step 3 using the merged prefix.

We can sort an entire sequence of length n by sorting a prefix of length at least n. This algorithm is called *Shivers sort*, after its creator's mother. Implement Shivers sort for lists and for arrays. The executable files `shivers-sort-list` and `shivers-sort-array` must read a JSON sequence of integers from `stdin` and write the corresponding sorted JSON sequence of integers to `stdout`.

Extra Credit: (2 points) What are the best-case and worst-case running times of Shivers sort in Θ-notation? You may choose the version for lists or for arrays, but you must provide both running times and an informal argument for each to get any extra credit.

Extra Credit: (2 points) What is the worst-case running time of Shivers sort when step 4 recursively sorts a prefix of length k rather than $\lfloor k/2 \rfloor$? Provide an informal argument and describe what kind of inputs cause this worst-case performance.

Exercise 4 (10 points)

Problem: Show that there is no comparison sort whose running time is $O(n)$ for at least half of the $n!$ inputs of length n. What about a fraction $1/n$ of the inputs of length n? What about a fraction $1/2^n$?