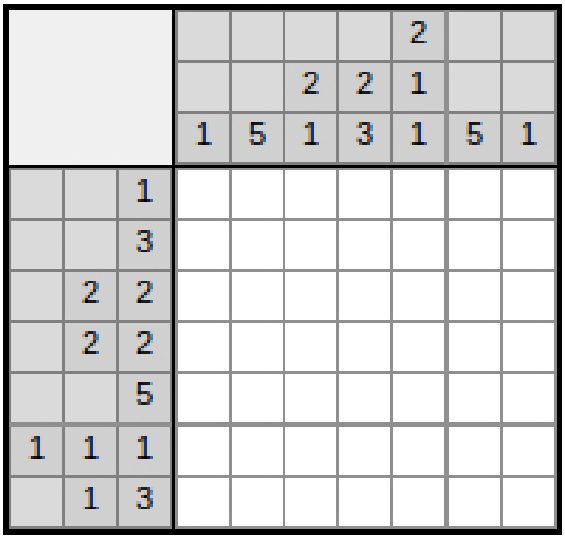
Algorithms and Computer Science: Nonograms

Modified from notes by Jeff Knerr

0. Partner's Info (name, year, etc): _____
(*remember this for Wednesday!*)

1. Try to start solving the nonogram (from `www.nonograms.org`) below using the following rules:

   - The numbers beside each row and column specify how many sets of "filled-in" squares need to be in each row. For example, if the row numbers are "2 3", then there needs to be a set of 2 consecutive filled-in squares, followed by a set of 3 consecutive filled-in squares.
   - Each filled-in set of squares must be separated from other filled-in sets by at least one non-filled-in square.
   - Once you know a square *cannot* be filled in, you can mark it with an ×.



Hint: start with the largest numbers. What constraints to they put on the filled-in squares?

2. Before thinking about an algorithm for solving an entire nonogram, what about an algorithm for checking or *verifying* a given candidate solution? For example, just considering this one row by itself, is it an example of a correct solution?

3. What about this solution? Is it correct?

4. We can more or less eyeball the difference between the two solutions above, but what about a computer? Say instead of an image the computer had a series of characters representing each row, with `F` for the filled-in squares and `E` for the empty squares. The examples above would be:

```
E E F E E F F F E E E F F E E F E
E E F E E F E F E F E F F E E F E
```

   Now it is more difficult to see if the solutions are correct or not. Define an series of steps that a computer could follow for determining whether or not this row is correct, given a target list of numbers ("1 3 2 1" in this case). Try to be as specific as possible about each step.

5. (Optional) Define an algorithm for solving nonograms (assuming a unique solution exists).