Section Handout 6

Based on a handout by Eric Roberts and Mehran Sahami

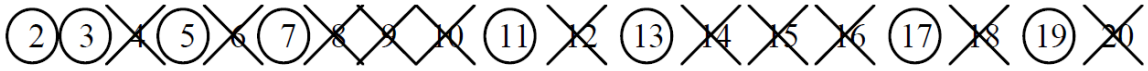
Problem One: The Sieve of Eratosthenes

In the third century B.C., the Greek astronomer Eratosthenes developed an algorithm for finding all the prime numbers up to some upper limit *N*. To apply the algorithm, you start by writing down a list of the integers between 2 and *N*. For example, if *N* were 20, you would begin by writing down the following list:

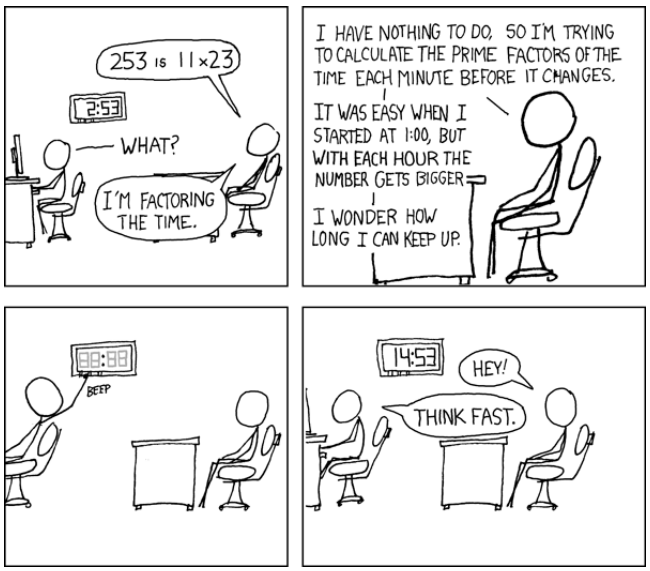
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

You then begin by circling the first number in the list, indicating that you have found a prime. You then go through the rest of the list and cross off every multiple of the value you have just circled, since none of those multiples can be prime. Thus, after executing the first step of the algorithm, you will have circled the number 2 and crossed off every multiple of two, as follows:

From here, you simply repeat the process by circling the first number in the list that is neither crossed off nor circled, and then crossing off its multiples. Eventually, every number in the list will either be circled or crossed out, as shown in this diagram:



The circled numbers are the primes; the crossed-out numbers are composites. This algorithm for generating a list of primes is called the sieve of Eratosthenes. Write a program that uses the sieve of Eratosthenes to generate a list of all prime numbers between 2 and 1000.



Courtesy: xkcd.com

Problem Two: Find the Median

The midterm has been graded, and it's your job to find the median score on the exam. The median score is defined as the score for which half of the scores on the exam are less than or equal to the median and half the exam scores are greater than or equal to the median. For example, if the exam scores are

```
84   86   108  96   115  110  67
```

then the median would be 96, because four of the numbers are no bigger than 96 and four of the numbers are no less than 96. You can assume that the number of scores on the exam is odd, so don't worry about the case where there are an even number of scores.

Your program should read the midterm scores from the file **`midterm-scores.txt`**, which contains the scores on the midterm exam (each of which is in the range 0 – 120). The scores are stored with one score per line of the file, as shown here:

File: **`midterm-scores.txt`**

```
113
86
108
96
115
110
67
98
109
74
120
```

There are many possible approaches to solving this problem... be creative and see what you can come up with!

Problem Three: Sudoku (Chapter 11, exercise 5, page 453)

In the last several years, a new logic puzzle called *Sudoku* has become quite popular throughout the world. In Sudoku, you start with a 9 × 9 grid of numbers in which some of the cells have been filled in with digits between 1 and 9. Your job in the puzzle is to fill in each of the empty spaces with a digit between 1 and 9 so that each digit appears exactly once in each row, each column, and each of the smaller 3 × 3 squares. Each Sudoku puzzle is constructed so that there is only one solution. For example, given the puzzle shown on the left of the following diagram, the unique solution is shown on the right:

		7		6		1		
					3		5	2
3			1		5	9		7
6		5		3		8		9
	1						2	
8		2		1		5		4
1		3	2		7			8
5	7		4					
		4		8		7		

2	5	7	9	6	4	1	8	3
4	9	1	8	7	3	6	5	2
3	8	6	1	2	5	9	4	7
6	4	5	7	3	2	8	1	9
7	1	9	5	4	8	3	2	6
8	3	2	6	1	9	5	7	4
1	6	3	2	5	7	4	9	8
5	7	8	4	9	6	2	3	1
9	2	4	3	8	1	7	6	5

Although the algorithmic strategies necessary to generate or solve Sudoku puzzles are beyond the scope of CS106A*, you can easily write a method that checks to see whether a proposed solution follows the Sudoku rules against duplicating values in a row, column, or outlined 3 × 3 square. Write a method

```
private boolean checkSudokuSolution(int[][] grid)
```

that performs this check and returns `true` if the grid is a valid solution.

---

* If you take CS106B, you'll see how to attack these puzzles. Interesting, solving generalized version of Sudoku that works with $n \times n$ grids, not just 3 × 3 grids, is a task that computers seem to have a lot of trouble with. They can solve them, but it might take a very, very long time. If you take CS103, you'll learn about a class of problems called the *NP-hard problems* that are conjectured to be fundamentally hard to solve. Solving generalized Sudokus is one of these NP-hard problems.