Reading: CORRECTION!
- This week: Sorting and Searching
  - 14.1 and 14.3 in 3rd edition
  - (19.1 and 19.3 in 2nd edition)
  - so weekly question is indeed required!
- course web page has also been updated

Midterm 2
- Midterm 2: Mon Mar 22, 6:30pm
- FSC 1005 again
- hour-long exam, reserve 6:30-8 time slot
- for buffer in case of fire alarms etc
- coverage: through arrays (Chap 8)
  - includes/builds on material covered in previous midterm
- study tips - same as before!
  - write and test programs, not just read book
  - try programming exercises from book

Reading through Midterm, 3rd edition
- 1.1-1.8
- 2.1-2.10
- 3.1-3.8
- 4
- 5.1-5.4
- 6.1-6.5
- 7.1-7.5-7.7
- 14.1,14.3
- see course page for 2nd edition list

Recap: Arrays of Arrays = 2D Arrays
- 2D array often easier to think about
- Internally, 2D arrays implemented as arrays of arrays in Java
  - they're equivalent

Recap: 2D Array Access Patterns
- Print average score for each student
  - for each row of scores
    - add up scores
    - divide by number of quizzes
    - length of row
- Print average score for each quiz
  - for each column of scores
    - add up scores
    - divide by number of students
    - length of column

Recap: Per-Student Averages
- public class ArrayEx4
  - public static void main(String[] args) {
    double scores[][] = {{95, 82, 13, 96},
      {73, 71, 84, 78},
      {51, 68, 63, 57},
      {91, 82, 12, 95}};
    double average;
    for (int col = 0; col < scores[0].length; col++) {
      int total = 0;
      for (int row = 0; row < scores.length; row++) {
        average = average + scores[row][col];
        total += scores[row][col];
      }
      average = average / scores[0].length;
      System.out.print("average for student " + (col + 1) + ": ");
      System.out.println(average);
    }
  }

Recap: Selection Sort
- Start at beginning
- Consider unsorted array elements: beyond current spot
  - Find smallest element
  - Swap with current spot
  - Move down by one

Estimating time required to sort
- We can go back to the selection sort example and count
  the comparisons. The first pass through the array of 5
  elements started with 16 being compared to 3, then 3
  was compared to 19, 8, and 12. There were 4
  comparisons. The value 3 was moved into the location
  at index 0.

Estimating time required to sort
- We can go back to the selection sort example and count
  the comparisons. The first pass through the array of 5
  elements started with 16 being compared to 3, then 3
  was compared to 19, 8, and 12. There were 4
  comparisons. The value 3 was moved into the location
  at index 0.

Estimating time required to sort
- We can go back to the selection sort example and count
  the comparisons. The first pass through the array of 5
  elements started with 16 being compared to 3, then 3
  was compared to 19, 8, and 12. There were 4
  comparisons. The value 3 was moved into the location
  at index 0.

Estimating time required to sort
- We can go back to the selection sort example and count
  the comparisons. The first pass through the array of 5
  elements started with 16 being compared to 3, then 3
  was compared to 19, 8, and 12. There were 4
  comparisons. The value 3 was moved into the location
  at index 0.

Estimating time required to sort
- We can go back to the selection sort example and count
  the comparisons. The first pass through the array of 5
  elements started with 16 being compared to 3, then 3
  was compared to 19, 8, and 12. There were 4
  comparisons. The value 3 was moved into the location
  at index 0.

Estimating time required to sort
- We can go back to the selection sort example and count
  the comparisons. The first pass through the array of 5
  elements started with 16 being compared to 3, then 3
  was compared to 19, 8, and 12. There were 4
  comparisons. The value 3 was moved into the location
  at index 0.

Estimating time required to sort
- We can go back to the selection sort example and count
  the comparisons. The first pass through the array of 5
  elements started with 16 being compared to 3, then 3
  was compared to 19, 8, and 12. There were 4
  comparisons. The value 3 was moved into the location
  at index 0.

Estimating time required to sort
- We can go back to the selection sort example and count
  the comparisons. The first pass through the array of 5
  elements started with 16 being compared to 3, then 3
  was compared to 19, 8, and 12. There were 4
  comparisons. The value 3 was moved into the location
  at index 0.

Estimating time required to sort
- We can go back to the selection sort example and count
  the comparisons. The first pass through the array of 5
  elements started with 16 being compared to 3, then 3
  was compared to 19, 8, and 12. There were 4
  comparisons. The value 3 was moved into the location
  at index 0.

Estimating time required to sort
- We can go back to the selection sort example and count
  the comparisons. The first pass through the array of 5
  elements started with 16 being compared to 3, then 3
  was compared to 19, 8, and 12. There were 4
  comparisons. The value 3 was moved into the location
  at index 0.

Estimating time required to sort
- We can go back to the selection sort example and count
  the comparisons. The first pass through the array of 5
  elements started with 16 being compared to 3, then 3
  was compared to 19, 8, and 12. There were 4
  comparisons. The value 3 was moved into the location
  at index 0.
Let's assume that your computer could make 1 billion (1,000,000,000) comparisons per second. That's a lot of comparisons in a second. And let's say your computer was using selection sort to sort the names of the people in the following hypothetical telephone books. Here's some mathematical food for thought.

<table>
<thead>
<tr>
<th>Phone Book</th>
<th>Number of People (N)</th>
<th>( N^2 )</th>
<th>Number of Comparisons Needed</th>
<th>Time Needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vancouver</td>
<td>544,320</td>
<td>296,284,262,400</td>
<td>296,000,000,000</td>
<td>296 seconds or 5 minutes</td>
</tr>
<tr>
<td>Canada</td>
<td>30,000,000</td>
<td>900,000,000,000</td>
<td>900,000,000,000</td>
<td>900,000 seconds or 10.4 days</td>
</tr>
<tr>
<td>People's Republic of China</td>
<td>1,000,000,000</td>
<td>1,000,000,000,000</td>
<td>1,000,000,000,000</td>
<td>1,000,000,000 seconds or 31.7 years</td>
</tr>
<tr>
<td>World</td>
<td>6,000,000,000</td>
<td>36,000,000,000,000</td>
<td>36,000,000,000,000</td>
<td>36,000,000,000 seconds or 1142 years</td>
</tr>
</tbody>
</table>

**Favorite Colors**
- Record everybody's favorite color
- How can we do "averages" per row?