Midterm 2
- coverage: through arrays (Chap 8)
  - includes/builds on material covered in previous midterm
- reading
  - Chap 1, 2, 3, 4, 6, 7 (not 5)
  - Chap 8.1, 8.5-8.7
- study tips
  - write and test programs, not just read book
  - try programming exercises from book!

Reading
- This week: no new reading
- Next week: 9.3-9.4, 9.6-9.8

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
  - for (int row = 0; row < scores.length; row++) {
    - double average = 0;
    - for (int col = 0; col < scores[row].length; col++) {
      - average += scores[row][col];
    - }
    - average = average / scores.length;
  - }
- Print average score for each quiz
  - for each column of scores
    - add up scores
    - divide by number of students
      - length of column
  - for (int col = 0; col < scores[0].length; col++) {
    - double average = 0;
    - for (int row = 0; row < scores.length; row++) {
      - average += scores[row][col];
    - }
    - average = average / scores.length;
  - }
Recap: Per-Student Averages

```java
public class ArrayEx4 {
    public static void main(String[] args) {
        double[][] scores = {{95, 82, 13, 96},
                             {51, 68, 63, 57},
                             {73, 71, 84, 78},
                             {50, 50, 50, 50},
                             {99, 70, 32, 12}};
        double average;
        // here's where we control looping row by row (student by student)
        for (int row = 0; row < scores.length; row++) {
            average = 0;
            // and here's where we control looping through the columns
            // (i.e., quiz scores) within each row
            for (int col = 0; col < scores[row].length; col++)
                average = average + scores[row][col];
            average = average / scores[row].length;
            System.out.println("average of row "+ row + " is "+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

The smallest value so far is 12
Its index is 4

Multidimensional Arrays

- now that we know 2D, we can do nD!
- any number of dimensions: 3D, 4D…
- up to 12D, actually
- example: student quiz scores over multiple terms
  - row: students
  - col: quiz scores
  - stack: term

Now for the good stuff

Computer science folks don't spend all their time writing programs. They're also concerned with the efficiency of those programs and their underlying algorithms. Efficiency can be expressed in terms of either time or memory needed to complete the task. In the case of sorting algorithms, we're typically interested in how much time it takes to sort.

So let's try to get some sense of the time requirements of selection sort. We don't use a stopwatch…instead, we use mathematics. The fundamental operation in sorting is the comparison to see if one value is less than the other, and the time required to sort corresponds to the number of comparisons that must be made to complete the sorting.

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. Then the second pass through the array began, starting with index 1. 16 was compared to 19, then 16 was compared to 8, which became the new minimum and was compared to 12. So among 4 elements in the array, there were 3 comparisons.
Estimating time required to sort

It takes 4 passes through the array to get it completely sorted. There are 4 comparisons on the first pass, 3 comparisons on the second pass, 2 comparisons on the third pass, and 1 comparison on the last pass. That is, it takes $4 + 3 + 2 + 1 = 10$ comparisons to sort an array of five values.

If you do this same computation on an array with six values, you’ll find it takes $5 + 4 + 3 + 2 + 1 = 15$ comparisons to sort the array. Do you see a pattern?

With a little math, you can figure out that the number of comparisons required to perform selection sort on an array of $N$ values is given by the expression: $N(N-1)/2$ or $(N^2-N)/2$

Estimating time required to sort

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 seconds needed to sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vancouver</td>
<td>544,320</td>
<td>296,284,262,400</td>
<td>296 or 5 minutes</td>
</tr>
<tr>
<td>Canada</td>
<td>30,000,000</td>
<td>900,000,000,000</td>
<td>900,000 or 15.1 days</td>
</tr>
</tbody>
</table>

Estimating time required to sort

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 seconds needed to sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vancouver</td>
<td>544,320</td>
<td>296,284,262,400</td>
<td>296 or 5 minutes</td>
</tr>
<tr>
<td>Canada</td>
<td>30,000,000</td>
<td>900,000,000,000</td>
<td>900,000 or 15.1 days</td>
</tr>
</tbody>
</table>

Estimating time required to sort

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.
Estimating time required to sort

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 Seconds Needed to Sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vancouver</td>
<td>544,820</td>
<td>(544,820^2)</td>
<td>(296,284,262,400) or 5 minutes</td>
</tr>
<tr>
<td>Canada</td>
<td>30,000,000</td>
<td>(300,000,000^2)</td>
<td>(900,000,000,000) or 10.4 days</td>
</tr>
<tr>
<td>People's Republic of China</td>
<td>1,000,000,000</td>
<td>(1,000,000,000^2)</td>
<td>(1,000,000,000,000) or 31.7 years</td>
</tr>
<tr>
<td>World</td>
<td>6,000,000,000</td>
<td>(6,000,000,000^2)</td>
<td>(36,000,000,000,000) or 1142 years</td>
</tr>
</tbody>
</table>

Favorite Colors

- record everybody's favorite color
- how can we do "averages" per row?