What’s the Point?

In this section, we’ll be implementing Bubble Sort, a simple (but notoriously slow) sorting algorithm.

Pair up!

Remember, we’ll be doing pair programming this semester. So choose a partner (somebody you’ve never worked with in section), and find a computer.

What is Bubble Sort?

Bubble Sort is a terrible way to sort your data. It’s really, really slow! But it has one great virtue: it’s very easy to write.

Bubble Sort works by comparing adjacent elements in an array. That is, it compares an element to the one just to its right - and if the two elements are out of order, it swaps them. It does this at all positions in the array, over and over, until we’re sure that the array has been sorted.

To be precise, the algorithm is as follows (where \( n \) is the number of elements in the array):

repeat \( n \) times
    for each element (except the last)
        compare the element to the one after it
        if they are out of order
            swap them

Bubble Sort is known as an "\( n^2 \)" algorithm because it has nested loops, and each loop involves (roughly) \( n \) steps - thus, the amount of work done by Bubble Sort is (roughly) proportional to the square of the number of elements in the array.

Program 1: Sorting 16 numbers

For Program 1 in this section, you will implement Bubble Sort.

This program will require three methods. In your `main()` method, allocate an array of numbers (integers or floating point, I don’t care which) with 16 elements. Fill these with random numbers which vary from 0 to 100. (Do you remember `Math.random()`? If not, you can review the class slides.) After
generating the random data, pass it to `printArray()` (the second method that you’ll write), and then to `bubbleSort()` (the third).

Q: Why do you call `printArray()` before `bubbleSort()`?
A: So we can see the initial state of the array, before any changes.

The second method, as mentioned, should be `printArray()`. It should take an array (the same type as you used in `main()`). This method should print out all of the elements in the array, all on one line. Make sure that you write it so that it uses `array.length` for its loop - don’t hard-code the length of the array. (Otherwise, this method will not work in Program 2.)

Finally, write the `bubbleSort()` method itself. Implement the pseudocode I gave above; each time that you swap two elements, pass the array to `printArray()` to print out a new version. Your output should look roughly like this:

Do you see how, on each line, two values are swapped? Look at the number 92, and see how it is moving gradually through the middle of the array.

✓ CHECKPOINT 1
Raise your hand. Your SL will come over and verify that you completed this step correctly.

Program 2: Predictable Numbers

In this program, we’ll be sorting again - but this time, we want the program to use a special set of numbers: we want it to be filled up with 1,2,3,4, ... (but all shuffled randomly).

We’ll also double the size of the array, to 32 values.

To do this, I’d like you to first fill up the array with the “right” data, sorted and in order. Then write a loop which runs 1000 times (or more). In each pass of the loop, randomly choose two locations in the array, and swap them. After 1000 iterations, your data should be well shuffled.

✓ CHECKPOINT 2
Raise your hand. Your SL will come over and verify that you completed this step correctly.
Program 3: Bar Chart

In this program, we’ll add a bar chart to our program. Write a new method (I called mine `drawChart()`) which takes the same array type as `printArray()`. In the new method, clear the screen, and then draw the values in the array as a bar chart - that is, as a series of rectangles. Call `StdDraw.show(int)` at the end of the method, to show the value to the screen.

Here’s what my bar chart looked like, partway through the sort. (You don’t have to color any of the bars green yet - that’s part of the Extra Features section.)

Remember: `StdDraw.filledRectangle(double,double,double,double)` requires that you specify the **middle** of the figure - and then its half-width and half-height.

In this program, you’ll want to set the `StdDraw` scale to make it easy to draw the bar graphs. What would be a useful scale, if you have an array that goes from index 0 to index 31, and value 1 to value 32?

Then update your code in `main()` and `bubbleSort()` so that it will call `drawChart()` each time that the array changes (in addition to calling `printArray()`). Or, if you prefer, you can just call `drawChart()` directly from `printArray()` itself.

☑ CHECKPOINT 3

Raise your hand. Your SL will come over and verify that you completed this step correctly.
Extra Features

If you happen to finish all three programs above, try adding the following features:

- A number is in the “correct” location if its value is one more than its index (that is, element \([0]\) is supposed to hold the value 1).
  
  Update `drawChart()` so that all of the bars that are in the correct place are green (including those which are only temporarily in the right place).

- Test your program with a larger dataset size, such as 256 or 1000 values. Do all of your methods work properly? Does Bubble Sort get a lot slower as the dataset grows, or not?
  
  (When I ran my own solution with 1000 values, it took literally hours to complete - and the action was slow slow, I often couldn’t see the picture changing at all.)

If you still have free time, then let’s experiment with other sorts. Try implementing one or both of the following sorts. (These are also \(n^2\) sorts, but they are typically faster than Bubble Sort.)

**Selection Sort**

\[
\text{for } i = 0 \text{ to } n-1 \text{ (inclusive)} \\
\quad \text{scan the array, starting at element } i, \text{ to find the minimum} \\
\quad \text{swap elements } i \text{ and the minimum}
\]

**Insertion Sort**

\[
\text{for } i = 1 \text{ to } n-1 \text{ (inclusive)} \\
\quad \text{pos} = i \\
\quad \text{while } \text{pos} > 0 \text{ and } \text{array[pos]} > \text{array[pos-1]} \\
\quad \quad \text{swap pos,pos-1} \\
\quad \quad \text{pos}--
\]

Please do not leave Section until either (a) the time ends; or (b) you finish all of the extra code listed above.

The End: Clean Up

The G/S 930 lab is host to sections from several of our programming classes, and is available the rest of the time for any CS student to use to work on their programming assignments. To help you get in the habit of leaving your work space neat and clean, we’ll end each section activity with this checkpoint, which also serves as confirmation that you attended section - even if you get nothing else done, you can clean up before you go, thereby earning one checkpoint and proving you attended.
• Log out of your computer.
• Pick up your papers, writing implements, cell phones, etc.
• Push in your chairs.

✓ CHECKPOINT 4

Raise your hand. Your SL will come over and verify that you completed this step correctly.

As with so much in this course, thanks to Dr. McCann for some of the ideas - and even some of the exact text - for this Section Activity!