What we will cover:
Selection Sort
Insertion Sort

Selection Sort
Selection sort is a slightly smarter sort than bubble sort. The selection sort algorithm starts by finding the smallest number in the list and placing that element first in the array. That first element is longer considered, as it has been sorted. The algorithm then compares the next element to all remaining elements, swapping positions with the smallest element. This procedure repeats until a single element remains.

Let's look at the implementation of this code (`SelectionSort.java`):

```java
public static void selectionSort(int[] array) {
    for (int i = 0; i < array.length; i++) {
        // find the min value in the list
        int currentMin = array[i];
        int currentMinIndex = i;

        for (int j = i + 1; j < array.length; j++) {
            if (currentMin > array[j]) {
                currentMin = array[j];
                currentMinIndex = j;
            }
        }

        // swap the current minimum with the first
        // value in our subarray, notice that this
        // is a simple swap since we have already
        // stored our "temp" value
        if (currentMinIndex != i) {
            array[currentMinIndex] = array[i];
            array[i] = currentMin;
        }
    }
}
```

Figure 1 - A sample run of selection sort
The outer loop walks us through the remaining unsorted subarray that we have not examined yet. The inner for loop will look at each value after the first element in this subarray and find the smallest value remaining. After the inner loop is done, the first value is swapped with the first element in this remaining subarray.

Note that this could be done by finding the maximum value and swapping that element to the last element in the remaining subarray that is being examined.

Class Exercise 32

Insertion Sort
The insertion sort works by examining elements and inserting them into a sorted subarray that the algorithm creates at the front of the original array. The algorithm looks at every element to the left, shifts existing values, then “drops” the current element into its sorted position in the array. As each element is “dropped” into its new position, the sorted subarray gets longer by one.

Specifically, the algorithm starts at index 1 and looks at the item to the left (index-1). It then determines if that value is larger than the current element. If it is, shift that value to the right (into the current element’s position) and compare the next item to the left. Continue to do this compare and shift until you reach the front of the array or find a value smaller than the current element. Once you have a found a value that is smaller, insert the current item one position to the right (there will be room since you have shifted everything to the right. Each time the outer loop executes, you have one more item in the sorted subarray.

As an example, look at the internal operation, and hand-tracing, for an algorithm iteration when it reaches the value 4 (index 3) in the array:
Figure 3 - Internal iteration of insertion sort

View the implementation of this algorithm (`InsertionSort.java`):

```java
public static void insertionSort(int[] array) {
    // start at second element (index 1) of the array
    for (int i = 1; i < array.length; i++) {
        int currentElement = array[i];
        int k;

        // Compare the current element to the element before it.
        // If the element is larger than the current, shift it
        // to the right and look at the next element to the left.
        // Keep doing this until you find a value smaller than
        // the current element.
        for (k = i - 1; k >= 0 && array[k] > currentElement; k--) {
            array[k + 1] = array[k];
        }

        // move the current element into the position to keep
        // the list sorted
        array[k + 1] = currentElement;
    }
}
```

**Class Exercise 33**

View interactive sorting demos at:
- [http://www.bluffton.edu/~nesterd/java/SortingDemo.html](http://www.bluffton.edu/~nesterd/java/SortingDemo.html)