More Iteration: Sort an Array

Problem. Print out the command-line arguments in order:

```java
% java sort the quick brown fox jumped over the lazy dog brown fox jumped lazy over quick the
```

Plan.
```java
class Sort {
   /** Sort and print WORDS lexicographically. */
   public static void main (String[] words) {
      sort (words, 0, words.length-1);
      print (words);
   }

   /** Sort items A[L..U], with all others unchanged. */
   static void sort (String[] A, int L, int U) {
      // TOMORROW
   }

   /** Print A on one line, separated by blanks. */
   static void print (String[] A) {
      // TOMORROW
   }
}
```

Selection Sort

```java
/** Sort items A[L..U], with all others unchanged. */
static void sort (String[] A, int L, int U) {
   if (L < U) {
      int k = indexOfLargest (A, L, U);
      sort (A, L, U-1); // Sort items L to U-1 of A
   }
}
```

Iterative version:
```
int i, k;
for (i = i1; i >= i0; i -= 1)
   k = (V[i].compareTo (V[k]) > 0) ? i : k;
```

And we're done! Well, OK, not quite.

Really Find Largest

```java
/** Value k, i0<=k<=i1, such that V[k] is largest element among
 * V[i0], ... V[i1]. Requires i0<=i1. */
static int indexOfLargest (String[] V, int i0, int i1) {
   if (i0 >= i1)
      return i1
   else /* if (i0 < i1) */ {
      int k = indexOfLargest (V, i0+1, i1);
      return (V[i0].compareTo (V[k]) > 0) ? i0 : k
      // or if (V[i0].compareTo (V[k]) > 0) return i0; else return k;
   }
}
```

Iterative:
```
int i, k;
k = i1; // Deepest iteration
for (i = i1-1; i >= i0; i -= 1)
   k = (V[i].compareTo (V[k]) > 0) ? i : k;
return k;
```
Finally, Printing

```java
/** Print A on one line, separated by blanks. */
static void print (String[] A) {
    for (int i = 0; i < A.length; i += 1)
        System.out.print (A[i] + " ");
    System.out.println ();
}
/* Looking ahead: There’s a brand-new syntax for the for
 * loop here (as of J2SE 5): */
for (String s : A)
    System.out.print (s + " ");
/* Use it if you like, but let’s not stress over it yet! */
```

Another Problem

Given an array of integers, A, move its last element, A[A.length-1], to just after nearest previous item that is ≤ to it (shoving other elements to the right). For example, if A starts out as

{ 1, 9, 4, 3, 0, 12, 11, 9, 15, 22, 12 }

then it ends up as

{ 1, 9, 4, 3, 0, 12, 11, 9, 12, 15, 22 }

If there is no such previous item, move A[A.length-1] to the beginning of A (i.e., to A[0]). So

{ 1, 9, 4, 3, 0, 12, 11, 9, 15, 22, -2 }

would become

{ -2, 1, 9, 4, 3, 0, 12, 11, 9, 15, 22 }

(Preliminary question: How can I state this without making this last case special?)

A Solution (from class)

```java
class Shove {
    /** Move A[A.length-1] so that all items after it are greater than
     * it is, displacing those items to the right (towards
     * higher indices). */
    static void moveOver(int[] A) {
        for (int i = A.length - 2; i >= 0; i -= 1) {
            if (A[i] <= A[i+1])
                break;
        }
    }
}
```