

CS61B Lecture #3

- **Reading:** Please read Chapter 4 of the reader *A Java Reference* for Friday (on Values, Types, and Containers).
- **Labs:** We are forgiving during the first week, but try to get your lab1 submitted properly by Friday night. *DBC: Let us know if you can't get something to work!*
- **Homework:** Please see Homework #1 on the lab page.

More Iteration: Sort an Array

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

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

Plan.

```
public 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

```
/** Sort items A[L..U], with all others unchanged. */
static void sort (String[] A, int L, int U) {
    if (L < U) {
        int k = /*( Index s.t. A[k] is largest in A[L] , ..., A[U] )*/;
        /*{ swap A[k] with A[U] }*/;
        /*{ Sort items L to U-1 of A. }*/;
    }
}
```

Selection Sort

```
/** 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);
        /*{ swap A[k] with A[U] }*/;
        /*{ Sort items L to U-1 of A. }*/;
    }
}
```

Selection Sort

```
/** 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);
        /*{ swap A[k] with A[U] }*/;
        sort (A, L, U-1);      // Sort items L to U-1 of A
    }
}
```

Selection Sort

```
/** 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);
        String tmp = A[k];  A[k] = A[U]; A[U] = tmp;
        sort (A, L, U-1);    // Sort items L to U-1 of A
    }
}
```

Selection Sort

```
/** 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);
        String tmp = A[k];  A[k] = A[U]; A[U] = tmp;
        sort (A, L, U-1);    // Sort items L to U-1 of A
    }
}
```

Iterative version:

```
while (L < U) {
    int k = indexOfLargest (A, L, U);
    String tmp = A[k];  A[k] = A[U]; A[U] = tmp;
    U -= 1;
}
```

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

Really Find Largest

```
/** 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

```
/** 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 \leq 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?)

Your turn

```
public class Shove {  
  
    /** Move A[A.length-1] so that it is just after the nearest  
     * previous item that is <= A[A.length-1], or to A[0] if  
     * there isn't such an item. Move all succeeding items  
     * to the right (i.e., up one index). */  
    // BETTER DESCRIPTION?  
    static void moveOver(int[] A) {  
        // FILL IN  
    }  
  
}
```