Please make sure you have obtained an account and used our "Account Administration" page to register and create keys by the end of today, no matter what TeleBEARS thinks.

Finish lab stuff (the survey and day1 hand-in) as soon as possible, but definitely before the next lab.

Reading: Please read Chapter 4 of the reader A Java Reference for Friday (on Values, Types, and Containers).

Homework: Please see Homework #1 on the lab page.

Public Service Announcement: HKN is offering free drop-in tutoring 11AM–5PM in 345 Soda and 290 Cory.

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);
        sort (A, L, U-1); // Sort items L to U-1 of A
    }
}

Iterative version:

while (L < U) {
    int k = indexOfLargest (A, L, U);
    U -= 1;
}

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

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.

class sort {
    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 */
    }
}

/** 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:

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

Really Find Largest

/** Value k, 0<=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 --)
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], so that 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, 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)

/** Move A[A.length-1] to the first position, k, in A such that there are no smaller elements after it, moving all elements * A[k .. A.length-2] over to A[k+1 .. A.length-1]. */
static void moveOver (int A[]) {
  moveOver (A, A.length-1);
}

/** Move A[U] to the first position, k<=U, in A such that there are no smaller elements after it, moving all elements * A[k .. U-1] over to A[k+1 .. U]. */
static void moveOver (int A[], int U) {
  if (U > 0) {
      moveOver (A, U-1);
    }
  }
}