**CS61B, Fall 2011** **HW #8** **P. N. Hilfinger**

**Due:** Mon., 31 October 2011

**Homework Exercises.** You'll find a skeleton for your answers in the `hw8` staff directory.

**1.** Suppose that we have an array, $D$, of $N$ records. Without modifying this array, I would like to compute an $N$-element array, $P$, containing a permutation of the integers 0 to $N-1$ such that the sequence $D[P[0]],\ D[P[1]], \ldots, D[P[N-1]]$ is sorted *stably*. Give a general method that works with *any* sorting algorithm (stable or not) and doesn't require any additional storage (other than that normally used by the sorting algorithm). Fill in the template file `hw8/StableSort.java` to get this effect.

**2.** I am given a list of ranges of numbers, $[x_i, x_i']$, each with $0 \le x_i \le x_i' \le MAX$. I want to know all the ranges of values between 0 and $MAX$ that are *not* covered by one of these ranges of numbers. So, if the only inputs are [2,3] and [12,1000], and the maximum value is 2000, then the output would be [0,1], [4,11], and [1001,2000]. See the template `hw8/Ranges.java"`.

**3.** [Goodrich&Tamassia] Given a sequence of $n$ distinct integers, each one of which is in the range $[0, n^2-1]$, develop an $O(n)$ algorithm for sorting them. See the skeleton file `hw8/SortInts.java`. You can't use ordinary distribution sort for this, because that would require initializing and traversing arrays of size $n^2$, which would take too long.

**4.** Find an algorithm that runs in $O(n \lg n)$ time for computing the number of inversions in a list of $n$ items. See the skeleton file `hw8/Inversions.java`. We will test this by giving it a rather large list.

**5.** [Goodrich&Tamassia] Given two sequences of integers, $A$ and $B$, find an algorithm that runs in $O(n \lg n)$ time (where $n$ is the total number of integers in $A$ and $B$) that determines, for a given parameter $m$, whether there is an integer $a$ in $A$ and an integer $b$ in $B$ such that $m = a + b$. See the skeleton file `hw8/Sum.java`. We will test this by giving it rather large sequences. Feel free to use any of the methods in `java.util.Arrays`.