CS61B, Fall 2004 **HW #7** **P. N. Hilfinger**

Due: Fri., 5 November 2004 at midnight

Create a directory to hold your answers to this homework set. Copy files from `$master/hw/hw7` into this directory. Put non-program answers into file `hw7.txt`. Use the command `submit hw7` to submit your solutions to the problems below.

1. Assume that we have a heap that is stored with the largest element at the root. To print all elements of this heap that are greater than or equal to some key X, we *could* perform the `removeFirst` operation repeatedly until we get something less than X, but this would presumably take worst-case time $\Theta(k \lg N)$, where N is the number of items in the heap and k is the number of items greater than or equal to X. Furthermore, of course, it changes the heap. Show how to perform this operation in $\Theta(k)$ time *without* modifying the heap. See `~cs61b/hw/hw7/HeapStuff.java`

2. 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).

3. I am given a list of ranges of numbers, $[x_i, x'_i]$, each with $0 \leq x_i < x'_i \leq 1$. I want to know all the ranges of values between 0 and 1 that are *not* covered by one of these ranges of numbers. So, if the only input is $[0.25, 0.5]$, then the output would be $[0.0, 0.25]$ and $[0.5, 1.0]$ (never mind the end points). See the template `~cs61b/hw/hw7/Ranges.java`.

4. [Goodrich&Tamassia] Given a sequence of n distinct integers, each one of which is in the range $[0, n^2 - 1]$, describe an $O(n)$ algorithm for sorting them.

5. Find an algorithm that runs in $O(n \log n)$ time for finding the number of inversions in a list of n items. See the skeleton file `~cs61b/hw/hw7/Inversions.java`. We will test this by giving it a rather large list.

6. [Goodrich&Tamassia] Given two sequences of integers, A and B, find an algorithm that runs in $O(n \log n)$ 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 `~cs61b/hw/hw7/Sum.java`. We will test this by giving it rather large sequences. Feel free to use any of the methods in `java.util.Arrays`.