CS 350 Spring 2011 HW 4—Shrinkage and Remolding

1. Consider the following version of insertion sort of a (zero-based) array a of size n.

   Buggy Reverse Insertion Sort

   for $i \leftarrow n-2$ **to** 0
   $j \leftarrow i-1$
   $v \leftarrow a[j]$
   while $j < n-1$ and $v \leq a[j]$
   $a[j] \leftarrow a[j+1]$
   $j \leftarrow j+1$
   $a[j] \leftarrow v$

   (a) Find the bug in this sort and suggest a simple correction.

   (b) On what inputs will the corrected algorithm have minimum running time?

   (c) Earlier we saw that selection sort always does at most n swaps, so at most $3n$ copies of elements. What is an upper bound on the number of element copies for this corrected insertion sort algorithm?

2. Textbook problem 5.2.8(a). Give the worst-case big-O running time of your algorithm. Implement it, and test it on the two graphs from the text.

3. Textbook problem 5.6.11. No need to give full pseudocode here, just describe a basic decrease-by-one step and sketch how the algorithm will work. This should give you an algorithm with number of flips worst case $O(n)$ where n is the number of pancakes.

4. Consider the following problem from a recent Google Code Jam. (Thanks to K Wilson for introducing me to this one.)

   Two-Price Total

   Given: An array a of n product prices in cents, with each price greater than zero. A target price t.

   Find: A pair of prices p_1, p_2 from different elements of a such that $p_1 + p_2 = t$.

   (a) Give a pseudocode for a brute-force algorithm for this problem, and find its worst-case big-O running time.

   (b) Give an asymptotically more efficient transform-and-conquer algorithm based on presorting a, and find its worst-case big-O running time.

   (c) Implement both algorithms and test them against arrays of random values with randomly chosen pairs as the target price. Which one is faster in practice, and by how much?

5. Implement a heap-based priority queue. Measure its running time on random inputs.