CISC621 Algorithm Design and Analysis, Fall 2015
Homework set 1, due Wednesday September 30, 5:30pm

Check the "homework sheet" from the syllabus for general homework details. Submit in 201 Smith Hall directly to Chunbo Song or place on shelf marked CISC 621.

1. Do problem 1.1, but instead of microseconds, use nanoseconds. Fill out the table on the assumption the algorithm uses $f(n)$ nanoseconds. After all, our cpu's can do basic instructions at the gigahertz rate.

2. Argue that the solutions to the following recurrences are as indicated by using recursion trees. Note that you must attend to both upper and lower bound. The c is a constant. [For reference, problems of this type are in section 4.4 and it's exercises.]

   (a) For $T(n) = T(n/4) + T(2n/3) + cn$, show $T(n) = \Theta(n)$.

   (b) For $T(n) = T(n/4) + 2T(3n/8) + cn$, show $T(n) = \Theta(n \log(n))$.

   (c) For $T(n) = T(n/2) + T(2n/3) + cn$, show $T(n) = \Omega(n^{\log_2(7/6)})$. and $T(n) = O(n^{\log_{3/2}(7/6)})$.

3. Consider a function `m = median(A,n)`, that takes an array A of n distinct numbers and returns index m such that $A[m]$ is the median of A. Assume that `median` does not rearrange A and runs in linear time, i.e. $T(n) = \Theta(n)$, where $T(n)$ is the runtime of `median` on an array of length n. Use this "blackbox" function for median to build a linear time algorithm for `select`, where `x = select(A,n,k)`, for the array A of size n returns the rank k entry. You may assume the numbers in A are distinct (no repeats).

   It is permitted that `select` rearranges the elements A as you wish (but, of course, moving elements around is part of the cost). Also note that this permission to rearrange is not necessary. In linear time you could copy the array A to another array and do your rearranging in that second array.

   Recall that the rank of an element in array A is the index position that the element would occupy if A were in sorted order. Thus, for example, the minimal element in A has rank 1 and the (lower) median has rank $\lceil n/2 \rceil$.

4. Old tableaux: An $m \times n$ *Old tableau* is an $m \times n$ matrix of nonnegative numbers such that the entries of each row are in nonincreasing order from left to right and the entries of each column are in nonincreasing order from top to bottom. Note that the largest entry is in the top left position, if any entry is zero, then the bottom right entry is zero. If the top left entry is zero, the entire matrix is zero. We may use an Old tableau to implement the max-priority queue data structure.

   Give an algorithm to implement the function `Extract-Max` on an Old tableau. Your algorithm should use a recursive subroutine that solves an $m \times n$ problem by recursively solving either an $m \times n-1$ subproblem or a $m-1 \times n$ subproblem. Define $T(p)$, where $p = m + n$, to be the maximum running time of `Extract-Max` on any $m \times n$ Old tableau. Give and solve a recurrence for $T(p)$ that yields the $O(m + n)$ time bound. Note that extracting the maximum means to replace the largest entry (top left one) with a zero and then rearrange entries so as to restore the Old tableau property of being nonincreasing in each row and column. [For reference, the material in chapter 6 on heaps and priority queues is relevant.]