Assignment 1. CS341, Winter 2011
Distributed Thursday, Jan. 6, due Jan. 20, 2011. Hand in to the assignment boxes on the 3rd floor of MC.

1. (25 marks) Given an array of length n, containing n integers. A *majority element* is an integer that appears more than $n/2$ times in the array.

   (a) (5 marks) Develop an algorithm to find the majority element, in $O(n \log n)$ time in the worst case. (Reduce the problem to a known problem)

   (b) (10 marks) Develop an algorithm, using only equality tests (no "<" or ">" tests), to find the majority element in $O(n \log n)$ time in the worst case. (Divide and conquer).

   (c) (10 marks) Develop an algorithm to find the majority element in $O(n)$ time, in the worst case. (Do not use hashing or radix sort, you can assume the integers are large.)

2. (10 marks) Let f and g be two functions with $f(n) = \Theta(g(n))$.

   (a) Must $e^{f(n)}$ be $O(e^{g(n)})$? Prove or give a counterexample.

   (b) Must $\ln(f(n)$ be $O(\ln(g(n)))$? Prove or give a counterexample.

3. (10 marks) Rank the following functions by order of growth from slowest to fastest; that is, find an arrangment $g_1, g_2, \ldots, g_{21}$ of functions satisfying $g_i = O(g_{i+1})$, for $1 = 1, \ldots, 20$. Partition your list into equivalence classes such that $f(n)$ and $g(n)$ are in the same class if and only if $f(n) = \Theta(g(n))$:

   n^2, $n^2 \ln n$, $100 * e^{\sqrt{n}}$, $\ln n$, $\lg n$, $\ln \ln n$, $n^{0.0000001}$, $n^{\ln n}$, $n^{\lg n}$, $n!$, 1000,000,000, n, 2^n, $50 * n^3$, $(n+5)!$, $\sqrt{n}$, $(\ln n)^{\ln n}$, e^n, $(4/3)^n$, $(4/3)^{\log n}$, 2^{2^n}.

   ln is logarithm with base e and lg is logarithm with base 10. You do not need to give any justification for your ordering.

4. (10 marks) Construct two strictly increasing functions (from natural numbers to natural numbers) $f(n)$ and $g(n)$ such that $f(n) \neq O(g(n))$ and $g(n) \neq O(f(n))$. Prove that your functions have the desired properties. (For the definition of 'strictly increasing', see page 51 of the textbook.)

5. (20 marks) Consider the Insertion-Sort algorithm defined in Lecture 1. For each value j in line 1, let $f(j)$ be the number of steps the element $A[j]$ moved in that round. Show that from the $f(j)$'s ($j = 2, \ldots, n$), one can infer the permutation π, by simulating the Insertion-Sort algorithm reversely. That is, given $f(j)$'s and the Insertion-Sort program only, one can reconstruct the input. (Thus, roughly $\sum_{j=i}^{n} \log f(j) + O(1)$ bits are sufficient to encode the permutation π, where the $O(1)$ bits are needed to encode the Insertion-Sort program whose size is independent of n.)