1. Lower Bounds for Coin Algorithm.

In the first lecture, we considered two versions of an algorithm to make change using the fewest possible number of coins (see the course Web page for the programs and proof). We proved that the version with memoization ran in $O(n)$ time, where n is the input amount of money for which change is computed. But the original recursive version was very slow, and we speculated that it required exponential time.
a) Prove that it does. Specifically, show that the number of calls to function `change()` is $\Omega(c^n)$, where c is a constant. You need not worry about getting the best possible value of c, and it is OK to express c as an expression using only constants, rather than a decimal number.
b) We proved that the memoization version runs in $O(n)$ time. Is this optimal? That is, is there an obvious lower bound of $\Omega(n)$ for this problem?

2. Asymptotic growth rates.

Do problem 3-3, part a.

3. Find the duplicate.

Your are given an array A containing all of the integers from 0 to $2^b - 1$, not necessarily in sorted order, with one appearing twice. A contains $n + 1$ integers, each represented using b bits, where $n = 2^b$. It is easy to figure which is the duplicate in time $O(n)$. But now consider the situation in which we can only read the numbers one bit at a time, so that reading one bit costs $O(1)$ (constant) time. Prove that you can figure out which is the duplicate using $O(n)$ bit reads.

4. Inversions.

Read Sections 5.1-5.2, and then do problem 5.2-5.