1. (1.7) Without losing generality, let’s assume $x$ has length $n$, $y$ has length $m$, and $n > m$. Based on the multiplication algorithm, the computation terminates in $m$ recursive calls. The recursion tree is a single chain. In half of these levels, we do single bit left shift which is $O(1)$ and in the other half we add two numbers that are at most $n + m$ bits long with complexity $O(n + m)$. Overall, the complexity is $O(nm + m^2)$ or $O(nm)$.

2. (1.8) We have done this in class. Check your notes.

3. (1.12) 
\[ 2^{2^{2006}} \mod 3 = 4^{2^{2005}} \mod 3 = (4 \mod 3)^{2^{2005}} = 1 \]

4. (1.14) The idea (as shown in exercise 0.4) is to use the matrix computation to get Fibonacci numbers. The problem turns into how $\text{pow}([0 \ 1; \ 1 \ 1], n) \mod N$ can be computed efficiently. The modexp algorithm in page 19 can be modified to efficiently compute the matrix power by replacing the scalar multiplications into matrix multiplications. The recursion terminations in $\log(n)$ calls and the complexity of each recursion is $O(\log^2(N))$; the overall complexity is $\log(n) \log^2(N)$.

5. (1.33) We use the equation 
\[ \text{LCM}(x, y) = \frac{xy}{\text{GCD}(x, y)}. \]

We assume that $x$ and $y$ are both $n$ bit long. The complexity of GCD with Euclid’s algorithm is $O(n^3)$. The rest of the computation is $O(n^2)$. Therefore the algorithm is $O(n^3)$.

6. (1.39) We use Fermat’s little theorem to simplify the problem. The idea is that since $p$ is prime, $a^{k(p-1)} \mod p = 1$. To turn $b^c$ into the form of $k(p - 1) + i$, we need to compute $i = b^c \mod (p - 1)$. We know we can efficiently compute $i$ with a $O(n^3)$ algorithm, where $n$ is the length of $b$ and $c$. In the next step, we now have $a^{b^c} \mod p = a^{k(p-1)+i} \mod p = a^i \mod p$. $a^n \mod p$ can also be computed efficiently with a cubic complexity. We are all set.

7. (1.42) Merge into assignment 3. The solution appears latter.