CSE 311: Foundations of Computing (Spring, 2015)

Homework 5 Out: Wed, 6-May. Due: Friday, 15-May, **before class (1 pm)** on **Gradescope**

Additional directions: You should write down carefully argued solutions to the following problems. Your first goal is to be complete and correct. A secondary goal is to keep your answers simple and succinct. The idea is that your solution should be easy to understand for someone who has just seen the problem for the first time. (Re-read your answers with this standard in mind!) You may use any results proved in lecture (without proof). Anything else must be argued rigorously. Make sure you indicate the specific steps of your proofs by induction.

[The extra credit problems are forthcoming (after Friday, May 8).]

1. An induction with equality (10 points)

Prove that for every positive integer n, it holds that

$$\sum_{k=1}^{n} k\, 2^k = (n-1)2^{n+1} + 2$$

2. In induction with inequality (10 points)

Prove that $3^n < n!$ if $n > 6$ is an integer.

3. A harder inequality (12 points)

Prove that if x and y are real numbers with $0 < y < x$, then for any positive integer n,

$$x^n - y^n \leq n\, x^{n-1}\,(x - y)$$

4. An induction with divisibility (12 points)

Use induction to show that if $S \subseteq \{1,2,\ldots,2n\}$ and $|S| \geq n+1$, then there are distinct elements $a, b \in S$ such that $a \equiv 0 \pmod{b}$.

5. Modular exponentiation (8 points)

Compute $5^{97} \bmod 100$ by hand using the modular exponentiation algorithm, showing your intermediate results. How many multiplications does the algorithm use for this computation?

6. Extended Euclidean algorithm (12 points)

Use the extended Euclidean algorithm to find $\gcd(1001, 100001)$, then express it as a linear combination of 1001 and 100001.

7. Modular congruences (15 points)

(a) Prove that there are no solutions to the modular equation $9x \equiv 2 \pmod{15}$

(b) Find an x such that $200\,x \equiv 13 \pmod{1001}$. Show your work.

8. Extra credit: RSA and modular exponentiation

We know that we can reduce the base of exponent modulo n: $a^k \equiv (a \bmod n)^k \pmod{n}$.

But the same is not true of the exponent itself! We cannot write $a^k \equiv a^{k \bmod n} \pmod{n}$. This is easily seen to be false in general. Consider, for instance, $2^{10} \bmod 3 = 1$, but $2^{10 \bmod 3} \bmod 3 = 2^1 \bmod 3 = 2$.

The correct law for the exponent is more subtle. Define

$$\varphi(n) = |\{\, m \in \mathbb{N} : 1 \le m \le n-1, \gcd(m,n) = 1 \,\}|$$

For instance, if p is a prime, you should check that $\varphi(p) = p - 1$. In this problem, you will prove the following.

Theorem: For all integers $n \ge 1$ and $a > 0$ with $\gcd(a,n) = 1$, it holds that $a^{\varphi(n)} \equiv 1 \pmod{n}$.

Your proof should proceed as follows. Let $R = \{\, m : 1 \le m \le n-1,\ \gcd(m,n) = 1 \,\}$.

(a) Define the set $aR = \{\, ax \bmod n : x \in R\}$. Prove that $aR = R$ for every integer $a > 0$ with $\gcd(a,n) = 1$.

(b) Consider the product of all the elements in R modulo n and the product of all the elements in aR modulo n. By comparing those two expressions, prove carefully that the theorem is true.

(c) Using the theorem, prove that under the assumptions, for any $b \ge 0$, we have

$$a^b \equiv a^{b \bmod \varphi(n)} \pmod{n}.$$

This theorem is the fundamental fact underlying the RSA cryptosystem.

Let e be such that $\gcd\big(e, \varphi(n)\big) = 1$. Recall that in this case (by Bezout's theorem), we can find a multiplicative inverse to d to e modulo $\varphi(n)$, i.e. such that $ed \equiv 1 \big(\bmod \varphi(n)\big)$.

(d) Suppose you know d. Let $m \in \{0,1,\dots,n-1\}$ be a message. Given the value $c = m^e \bmod n$, state an efficient algorithm for recovering the initial message m, and argue why it's correct.

Now we'll see how RSA works.

(e) Prove that if $\gcd(a,b) = 1$ then $\varphi(ab) = \varphi(a)\varphi(b)$ for all positive integers a, b. This shows that if p and q are prime then $\varphi(pq) = (p-1)(q-1)$.

Choose two large distinct prime numbers p and q. We'll see in the next homework extra credit how you can generate large primes efficiently. Compute the product $n = pq$ and the value $\varphi(n) = (p-1)(q-1)$. Also choose an integer e such that $\gcd\big(e, (p-1)(q-1)\big) = 1$, and find its multiplicative inverse d modulo $(p-1)(q-1)$.

Your **public key** is the pair (n, e) and your **private key** is the pair (n, d). You release (n, e) to the public and keep (n, d) secret. The idea is now that someone can take a message m and encrypt it to $c = m^e \pmod{n}$ using the modular exponentiation algorithm and your public key. You, knowing d, can use part (c) to decrypt the message.

This is basically how SSL works: To send your password to your bank, you take their public key and use it to encrypt your password. Then the bank can decrypt your password (to see if you can login), but no eavesdropper sniffing your wifi can do the same. The benefit of this **public key** cryptosystem is that you don't need a secure channel in order to communicate securely with your bank. (Classical **private key** cryptosystems would require that you meet ahead of time and exchange a secret key before being able to communicate securely.)

(f) Prove that if an attacker knew the factorization $n = pq$ then they would have an efficient algorithm to decrypt the message (and thereby break the cryptosystem).

Presently, the most efficient known way to break RSA is to factor n. Using any known factoring algorithm, this takes an extremely long time (if the primes p, q are chosen large enough initially), and that's what makes RSA secure in practice. But it is not known (i) whether factoring is actually a difficult computational problem, and (ii) whether breaking RSA requires factoring---there could be an easier way to recover the plaintext message.