CS 70 Discrete Mathematics for CS
Fall 2003 Wagner HW 5

Due Thursday October 9

1. **(5 pts.) Any questions?**
Is there anything you'd like to see explained better in lecture or discussion sections?

2. **(15 pts.) Modulo arithmetic**
Solve the following equations for x and y modulo the indicated modulus, or prove that no solution exists. Show your work.

   (a) $8x \equiv 1 \pmod{13}$.

   (b) $3x + 23 \equiv 24 \pmod{51}$.

   (c) $3x + 25 \equiv 16 \pmod{31}$.

   (d) The system of simultaneous equations $3x + 2y \equiv 1 \pmod{7}$ and $2x + y \equiv 4 \pmod{7}$.

3. **(20 pts.) Binary gcd**

   (a) Prove that the following statements are true for all $m, n \in \mathbf{N}$.

$$\begin{aligned} \text{If } m \text{ is even and } n \text{ is even,} \quad & \gcd(m,n) = 2\gcd(m/2, n/2). \\ \text{If } m \text{ is even and } n \text{ is odd,} \quad & \gcd(m,n) = \gcd(m/2, n). \\ \text{If } m, n \text{ are both odd and } m \geq n, \quad & \gcd(m,n) = \gcd((m-n)/2, n). \end{aligned}$$

   (b) Give an algorithm that computes $\gcd(m,n)$ using at most $O(\lg m + \lg n)$ subtractions, halvings, doublings, and odd/even tests.

4. **(20 pts.) Big-O notation**
The purpose of this problem is to teach you Big-O notation in a careful way. First, study the following. (If you'd like additional reading on this subject, you may refer to Rosen, Chapter 2.2.)

Formally: If $f(n), g(n)$ are two non-negative functions of a single integer variable, the statement $f(n) \in O(g(n))$ means that

$$\exists N_0 \in \mathbf{N}. \quad \exists C \in \mathbf{N}. \quad \forall x \in \mathbf{N}. \quad x \geq N_0 \implies f(x) \leq C \cdot g(x).$$

In other words, $O(g(n))$ is the set of functions $\{f_i(n) : \exists N_0 \in \mathbf{N}. \exists C \in \mathbf{N}. \forall x \in \mathbf{N}. x \geq N_0 \implies f_i(x) \leq C \cdot g(x)\}$. This is the definition of Big-O notation.

Informally: $f(n) \in O(g(n))$ means, roughly, that $f(n)$ grows "no faster than" $g(n)$ (except possibly for a constant factor), as n gets large. For instance, $n^2 \in O(n^2)$, $n(n+1)/2 \in O(n^2)$, and $10000n^2 \in O(n^2)$, because these functions all grow at asymptotically the same rate (ignoring constant factors). Also, $n^2 \in O(n^3)$, because n^2 grows more slowly than n^3 does, as n gets large.

Some basic facts: If $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$, then $f(n) \in O(h(n))$. If $f(n) \in O(g(n))$ and $f'(n) \in O(g'(n))$, then $f(n) + f'(n) \in O(g(n) + g'(n))$. If $f(n) \in O(g(n))$ and $f'(n) \in O(g'(n))$, then $f(n) \times f'(n) \in O(g(n) \times g'(n))$.

Common notation: Instead of writing $f(n) \in O(g(n))$, almost everyone instead writes $f(n) = O(g(n))$. Strictly speaking, this is a sloppy abuse of notation, but this practice is widespread; you are guaranteed to see it throughout your studies of computer science, so be prepared. Also, we often write something like n^2 as a shorthand for the function $f(n) = n^2$, just to make our life easier.

Now, with that background established, do the following problems:

(a) Prove that $n^2 + 2003 \in O(n^3)$.
Hint: One possible approach is to give an example of constants N_0, C that satisfy the definition.

(b) Prove that $100n^2 \lg n \in O(n^3)$.

(c) True or false: There exists $e \in \mathbf{N}$ such that $2^n \in O(n^e)$. (You do not need to justify your answer.)

(d) Critique the following argument. Is the reasoning valid? If not, why not?

> We have $n^2 = O(n^4)$.
> Also, we have $n^2 = O(n^3)$.
> By transitivity, it follows that $O(n^4) = O(n^3)$.
> This means that $n^4 = O(n^3)$.

5. (20 pts.) Program checking: "Casting out p's"

You've bought a fast integer calculation library from Goofle, Inc. ("featuring patented addition technology!"), and holy cow, their code is *fast*!

However, you're a little suspicious about whether Goofle's code is always returning the correct answer. They don't supply the source code to their library[1], so you have no way to check their algorithms directly.

Instead, you decide to sanity-check every result you get back from their library at run-time to give yourself a good chance of detecting any erroneous results. To this end, you're going to write a wrapper (e.g., `CheckedAdd()`) around their API (e.g., `GoofleAdd()`) that checks Goofle's result—hopefully without incurring too much performance penalty.

(a) As a warm-up, prove that, for every positive $D \in \mathbf{N}$, the number of distinct prime factors of D is at most $\lg D$.

(b) Now, back to writing `CheckedAdd()`. Let's suppose you use the following algorithm:

> `CheckedAdd`(m,n):
> // Given $m,n \in \mathbf{N}$, computes $m+n$ using Goofle's library and checks for errors.
> 1. Set $k :=$ `GoofleAdd`(m,n).
> 2. Pick a random 64-bit prime p, i.e., choose uniformly at random among all primes satisfying $2^{63} < p < 2^{64}$.
> 3. Compute $m' := m \bmod p$, $n' := n \bmod p$, $k' := k \bmod p$.
> 4. If $k' \neq m' + n' \bmod p$, then signal an error and abort.
> 5. Return k.

Assume that Steps 2–4 can be performed using a trusted library that is known to work correctly. Prove that if `GoofleAdd`(m,n) correctly computes $m+n$, then no error will be signaled.

(c) Next, let's check whether errors in Goofle's library will be detected. Let k denote the result returned by `GoofleAdd`(m,n) in Step 2. Prove the following: If `GoofleAdd()` returns the wrong result, then with probability at least $1 - \frac{\lg D}{2^{57}}$, an error will be signaled, where $D = \max(k, m+n)$.

[1] Bad dog, no bone for them.

Hint: Even if Goofle is trying to fool us maliciously, they still have to pick k before they have any idea what prime we'll choose. What is their best strategy for picking k erroneously to maximize the probability of avoiding detection?

Hint: You may use, without proof, the fact that there are at least 2^{57} different 64-bit primes.

(d) Goofle's library also includes a function `GoofleMultiply`(m,n) that claims to efficiently compute $m \times n$, when given inputs $m,n \in \mathbf{N}$.

Specify an algorithm `CheckedMultiply`(m,n) such that: (1) When m,n are sufficiently large numbers, `CheckedMultiply`(m,n) runs almost as quickly as `GoofleMultiply`(m,n); (2) Similarly to `CheckedAdd()`, you have a good chance of detecting erroneous results from any multiplication computation; and (3) Similarly to `CheckedAdd()`, you never signal an error without justification.

You may assume that you have a trusted library that can be used to pick a random 64-bit prime p, add modulo p, and multiply modulo p, correctly and in $O(1)$ time. Also, you may assume that your trusted library can compute $n \bmod p$ correctly, in $O(\lg n)$ time.

6. (25 pts.) Polynomial interpolation

(a) Prove the following: If p is a prime and $y_1, \dots, y_n \in \mathbf{N}$ are all different from 0 modulo p, then $y_1 \times \cdots \times y_n$ is also different from 0 modulo p.

(b) Prove the following: Given a prime p and two integers a, b, it is always possible to find a polynomial $f(x)$ of degree at most one such that $f(0) \equiv a \pmod{p}$ and $f(1) \equiv b \pmod{p}$.

(c) You are given a prime p and a positive number $n < p$. Show how to find a polynomial $f(x)$ of degree at most n satisfying $f(0) \equiv f(1) \equiv \cdots \equiv f(n-1) \equiv 0 \pmod{p}$ and $f(n) \equiv 1 \pmod{p}$. In other words, the polynomial f should be congruent to zero at the points $x = 0, \dots, n-1$; at $x = n$ the polynomial should be 1 mod p.

Hint: Consider $F(x) = (x-0)(x-1)(x-2)\cdots(x-(n-1))$; what can you say about it?

(d) You are given p and n as before, but now you are also given an index j with $0 \le j \le n$. Show how to find a polynomial $g_j(x)$ of degree at most n satisfying

$$g_j(i) \equiv \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases} \pmod{p} \qquad\qquad \text{for each } i = 0, 1, \dots, n.$$

In other words, the polynomial g_j should be congruent to zero at the points $x = 0, \dots, n$, except that at $x = j$ it should be congruent to 1 mod p.

(e) You are given a prime p, a number n with $0 < n < p$, and a sequence of values $a_0, a_1, \dots, a_n \pmod{p}$. Describe an efficient algorithm to find a polynomial $h(x)$ of degree at most n satisfying $h(0) \equiv a_0 \pmod{p}$, $h(1) \equiv a_1 \pmod{p}$, ..., $h(n) \equiv a_n \pmod{p}$.

Hint: What can you say about the polynomial $3g_0(x) + 7g_1(x)$, where $g_0(x), g_1(x)$ are as defined in part (d)? Does this give you any ideas?