Prof. Sergio A. Alvarez
Fulton Hall 410–B
Computer Science Department
Boston College
Chestnut Hill, MA 02467 USA

http://www.cs.bc.edu/∼alvarez/
alvarez@cs.bc.edu
voice: (617) 552-4333
fax: (617) 552-6790

CS383, Algorithms
Spring 2009
HW3 Solutions

1. Consider the following algorithm for computing modular remainders:

   Algorithm 1: recMod
   Input: Two integers $a \geq 0,\ b \geq 1$.
   Output: The modular reminder $a \pmod b$.
   RECMOD(a, b)
   (1) **if** $a = 0$ **then return** 0
   (2) rem $= 2$ RECMOD($\lfloor a/2 \rfloor$, b) $+ a \pmod 2$
   (3) **if** rem $< b$ **then return** rem
   (4) **else return** rem $- b$

   (a) Trace the execution of Algorithm 1 on input $(15, 4)$ by drawing the recursion tree for recMod$(15, 4)$. Include the value returned by each invocation (node).

   Solution

   The hierarchy of recursive calls is summarized below. Each level (x, y) corresponds to the invocation recMod(x, y), and generates the call in the level immediately below it. Values are returned starting at the bottom level and are propagated up the ladder one level at a time.

   ```
   (15,4) returns 2*3 + 1 - 4 = 3
   (7, 4) returns 2*3 + 1 - 4 = 3
   (3, 4) returns 2*1 + 1 = 3
   (1, 4) returns 2*0 + 1 = 1
   (0, 4) returns 0
   ```

   (b) Give a general argument to show that recMod(a, b) correctly computes the modular remainder $a \pmod b$ for any integers $a \geq 0$ and $b \geq 1$.

   Solution

   I'll show that recMod(a, b) correctly computes $a \bmod b$, by induction in a. This approach is natural because it is patterned after the definition of recMod by recursion in a.

   - (Basis) recMod$(0, b) = 0 = 0 \bmod$ b. Check.

- (Induction Hypothesis) recMod(x, b) returns x mod b for all non-negative integers x less than a.
- (Inductive Step) Assume that the Induction Hypothesis holds. Show that recMod(a, b) returns a mod b. We do this in two cases, depending on whether a is even or odd. In either case, we let r be the value returned by the recursive invocation recMod$(a/2, b)$. By the Induction Hypothesis, $r = a/2 \bmod b$. Thus, $a/2 = q*b+r$, with q an integer; also, $0 \leq r < b$.
  i. Assume first that a is even. Then the remainder a mod 2 is 0, so recMod(a, b) returns the value $2r$ if $2r < b$, and $2r - b$ if $2r \geq b$. Since a is even, there is no rounding in the quotient $a/2$, so a itself may be written as $(2q) * b + 2r$. Since $0 \leq r < b$, we know that $0 \leq 2r < 2b$. Thus, the remainder a mod b is either $2r$ itself if $2r < b$, or $2r - b$ if $b \leq 2r < 2b$. This proves that recMod$(1/2, b)$ returns a mod b if a is even.
  ii. Now assume that a is odd. Then a mod 2 is 1, so recMod(a, b) returns the value $2r + 1$ if $2r + 1 < b$, and $2r + 1 - b$ if $2r + 1 \geq b$. We must show that this return value equals a mod b. This time the integer quotient $a/2$ involves rounding down by $1/2$, so 1 must be added to the result of doubling $a/2$ in order to recover the original value of a, that is, $a = (2q) * b + (2r + 1)$. Thus, the remainder a mod 2 equals either $2r + 1$ or $2r + 1 - b$ depending on whether $2r + 1 < b$ or not. This is exactly the value returned by recMod(a, b), so the proof is complete.

(c) Notice that each of the arithmetic operations in the recursive call in Algorithm 1 involves the number 2 as one of the operands. On computers that use binary arithmetic at the machine level, these operations can be performed quickly: multiplication and integer division by 2 reduce to left and right shifts by one bit, and the remainder modulo 2 just involves testing the least significant bit. Therefore, we will assume that these three operations may be performed in time $O(d)$. Addition and subtraction of d-digit numbers are also assumed to take time $O(d)$. Analyze the asymptotic running time of Algorithm 1 on d-digit inputs a and b, keeping these comments in mind. Explain in detail.

Solution

Each recursive call divides a by 2 (in integer arithmetic). The base case occurs when a hits the value 0. It takes $\Theta(d)$ steps for this to occur, assuming that d is the number of digits in a (this is because the number of digits is of the order $\log a$, which is of the same order as the number of divisions by 2 required to bring a down to less than 1). Thus, the recursion depth is $\Theta(d)$. At each level of the recursion, the total time required for addition, subtraction, multiplication by 2, division by 2, boolean tests, assignments, and the return statement, is $O(d)$ (addition and subtraction times dominate). Adding up all of the levels, we see that the total running time is $O(d^2)$.

2. Consider the following "beefed up" RSA encryption scheme (my apologies to any vegetarians). The recipient picks *three* large primes p, q, r and a number e between 1 and $(p-1)(q-1)(r-1)$ that is relatively prime to $(p-1)(q-1)(r-1)$, and publishes the pair $(N = pqr,\ e)$ as his public key. The encryption of a message m (number between 0 and $N-1$) is the quantity

$$\text{encrypt}(m) = m^e \pmod{N}$$

(a) Show that the above encryption function is invertible by explicitly computing its inverse function. Proceed by analogy with the discussion of RSA from class and the textbook. Include a step-by-step justification of your answer. You'll need Fermat's little theorem.

Solution

I claim that the inverse of the above encryption function is:

$$\text{decrypt}(m) = m^d \pmod{N},$$

where d is the multiplicative inverse of $e \pmod{(p-1)(q-1)(r-1)}$.
First, notice that the multiplicative inverse d exists because e is stated to be relatively prime to $\phi = (p-1)(q-1)(r-1)$. One can therefore use the extended Euclid gcd algorithm to compute d as a byproduct of the computation of $\gcd(\phi, e) = 1$ (the algorithm yields (x, y, d), where $1 = d = \gcd(\phi, e) = x * \phi + y * e$, which shows that $1 \equiv y * e \bmod \phi$, which means that y is the multiplicative inverse of $e \bmod \phi$).
Now construct the composite function:

$$\text{decrypt}(\ \text{encrypt}(m)) = m^{ed} \pmod{N}$$

I will show that

$$m^{ed} \equiv m \pmod{N},$$

which implies that decrypt is the inverse of encrypt. Equivalently, we just need to show that the difference $m^{ed} - m$ is divisble by N. Since N is the product of the three primes p, q, r, it is enough to show that the difference $m^{ed} - m$ is divisible by each of p, q, r.
Here goes. We know that

$$ed = 1 + k\phi$$

for some integer k. Therefore,

$$m^{ed} - m = m(m^{k\phi} - 1) = m((m^{\phi})^k - 1)$$

But by Fermat's little theorem,

$$m^{p-1} = 1 \bmod p$$

Raising the quantity on the left to the power $k(q-1)(r-1)$, it follows that

$$m^{k\phi} = 1 \bmod p,$$

so that

$$m^{ed} - m = 0\ (\bmod p)$$

This proves that $m^{ed} - m$ is divisible by p. The same argument shows that $m^{ed} - m$ is divisible by q and by r as well. In light of our opening arguments, this proves that the proposed inverse actually recovers the original message from the encrypted version.

(b) Is the proposed scheme cryptographically secure? That is, if someone were to intercept the transmitted encrypted message encrypt(m), would it still be very difficult for them to recover the original message m based only on encrypt(m) and the public key (N, e)? Discuss, paying particular attention to the time complexity of computing the inverse function in the preceding subtask.

Solution

An eavesdropper may intercept an encrypted message $m' = \text{encrypt}(m)$ when it is being transmitted. In order to recover the original message m, he/she would need to compute decrypt(m'). The issue is what the time complexity of computing the inverse function here is *given only the encrypted message m' and the public key* (N, e). The crucial step is recovering the decryption exponent d, since computation of the modular exponential function given d takes only polynomial time in the number of digits. However, recovering d requires first finding $\phi = (p-1)(q-1)(r-1)$, and this in turn requires first knowing the prime factors p, q, r. No efficient algorithms (that run in polynomial time in the number of digits) for factoring are known. Hence, if p, q, r are large, the attacker will be stuck. Conclusion: this *is* a cryptographically secure scheme.