# Homework 1 Solutions

CS 344 - Design and Analysis of Algorithms

February 22, 2013

Typesetted answers receive 10% extra credit. Typewritten answers that are not typesetted receive 5% extra credit.

On any assignment (homework or exam), you can either attempt to answer the question, in which case you will receive between 0 and 100% credit for that question, or you can write "I don't know", in which case you receive 25% credit for that question. Leaving the question blank is the same as writing "I don't know."

## 1 Problem 1

### 1.1 A [10 points]

**Algorithm 1:** `MR_Function`(odd integer $N$, integer $s$)

```
1  express $N-1$ as $2^t \cdot u$, where $u$ is odd ;                        // $O(log(N))$
2  for $i \leftarrow 1$ to $s$ do // Loop runs $O(log(N))$ times.
3      $\alpha \leftarrow$ Random_Integer$(2, N-2)$ ;                          // $O(log(N))$
4      if EuclidGCD$(\alpha, N) \neq 1$ then // $O(log^3(N)$
5          return FALSE ;                                                      // $O(1)$
6      $x_0 \leftarrow \alpha^u$ mod $N$ ;                                     // $O(log^3(N))$
7      for $j \leftarrow 1$ to $t$ do // Loop runs $O(log(N))$ times.
8          $x_j \leftarrow x_{j-1}^2$ mod $N$ ;                                // $O(log^2(N))$
9          if $x_j = 1$ and $x_{j-1} \neq 1$ and $x_{j-1} \neq N-1$ then // $O(log(N))$
10             return FALSE ;                                                  // $O(1)$
11     if $x_t \neq 1$ then // $O(1)$
12         return FALSE ;                                                      // $O(1)$
13     return TRUE ;                                                           // $O(1)$
```

For this analysis, it is assumed that quadratic-time solutions in the number of bits of number $N$ are used for the arithmetic operations, as described during the lectures.

Line 1 uses input parameter $N$ and expresses it using two numbers, $t$ and $u$. Both of these parameters can be computed in $O(log(N))$ time as we just need to scan the bits in the even number $N-1$. Number $t$ is actually the number of 0's at the end of the binary representation of $N-1$, while $u$ is the remaining number after right shifting $t$ times number $N-1$.

Line 2 is the outer loop of the algorithm, and runs $s$ iterations.

Line 3 performs random number generation, which is assumed to be independent of the input size; however, one of the arguments to the function requires an subtraction, which runs in $O(log(N))$ time.

The `EuclidGCD` algorithm at line 4 runs in $O(log^3(N))$ time under the assumption of quadratic running time aritmetic operations.

Line 6 requires computing $a^u \bmod N$, which requires $O(log(N))$ multiplication operations, which each take $O(log^2(N))$ operations, for a total time of $O(log^3(N))$.

Line 7 is the inner loop, which runs for $t$ iterations, where $t$ was computed in line 1.

Line 8 computes $x_{j-1}^2$, which is just multiplication with itself, incurring a cost of $O(log(N))$.

Line 9 is mostly comparisons, which are assumed to run at most $O(log(N))$ time, while the line certainly runs in that much time due to the subtraction being performed.

All assignments and return operations are assumed to be constant time ($O(1)$). Begin aggregating the cost by examining the cost of the inner loop. The expensive operation inside this loop is the squaring that takes $O(log^2(N))$ time, and since the inner loop runs a total of $t$ times, the overall cost of the inner loop is $O(t \cdot log^2(N))$ time. What is the value of $t$ though? $t$ is $O(log(N))$, as is the number of 0 bits at the end of $N-1$, which could be at most $\lceil log(N) \rceil - 1 = O(log(N))$ bits. This means the inner loop's actual running time is $O(log^3(N))$. Aggregating all the terms for the outer loop, we see the cost of the operations inside the loop is $O(log^3(N)) + O(log^3(N)) = O(log^3(N))$. Thus, the total running time of the algorithm is $O(s \cdot log^3(N))$. Because we set $s = O(log(N))$, the total running time of the algorithm is $O(log^4(N))$.

## 1.2 B [10 points]

There are two possible ways to interpret the result from part 1.A. as far as the answer to this question is concerned: the running time is in term of the original input number, $N$, of $O(log^4(N))$, used as $f_{10} = O(log^4(n))$ below, or in terms of the number of bits, in which case we can use $f_{11} = O(n^4)$. Either interpretation is valid, and credit will be given as long as one of the two cases is examined correctly. Then, we can order the running times as follows:
$f_4(n) < f_{10}(n) < f_5(n) < f_3(n) \equiv f_6(n) < f_8(n) < f_1(n) < f_{11}(n) < f_9(n) < f_2(n) < f_7(n)$

The following inexact explanations are sufficient for full credit. First, however, it is prudent to reduce the form of $f_8$ first. It is shown that $f_8 = O(n^2)$, given the following expressions:

$$4^{log_2 n} = n^2$$

$$log_2 4^{log_2 n} = log_2 n^2$$

$$log_2 n \ \cdot \ log_2 4 = 2 log_2 n$$

$f_4$ is clearly the smallest function, as it does not depend on $n$, and is constant.

$f_{10} = O(log^4(n)) < f_5 = O(n)$ because, although both functions do increase to infinity, the rate of increase of any logarithmic function is much smaller than a linear function.

$f_5 = O(n) < f_3 = O(n\ log_2(n))$ as $f_3$ is just $f_5$ multiplied by a non-constant term, $log(n)$.

$f_3 = \Theta(f_6)$. Consider the value of $\frac{f_3}{f_6}$. From change of base, we know that $\frac{log_a(x)}{log_b(x)} = \frac{log(b)}{log(a)}$, which is just a constant term. So in this case, we get $\frac{log(10)}{10 \cdot log(2)}$, which is some constant.

$f_6 = O(n\ log(n)) < f_8 = O(n^2)$. From here, it follows that $n \cdot n > n \cdot log(n)$ for $n \to \infty$.

$f_8 = O(n^2) < f_1 = O(n^3)$, as all polynomial terms of higher order are asymptotically greater than lower terms.

$f_1 = O(n^3) < f_{11} = O(n^4)$ follows from the above argument.

$f_{11} = O(n^4) < f_9 = O(n^{log(log(n))})$ as the function $log(log(n))$ grows to infinity. This means for some finite value of $n$, $log(log(n)) > 4$, thus it follows that $f_9$ has a larger exponent and asymptotically grows larger.

$f_9 = O(n^{log(log(n))} < f_2 = O(n!)$. We can test this using Stirling's Approximation for the value of $ln(n!)$. Begin by taking the natural log of both sides, yielding $(log(log(n))(ln(n)) < ln(n!) \cong n(ln(n)) - n + O(ln(n))$. Dividing both sides by $ln(n)$ yields $log(log(n)) < O(n)$, which holds given prior arguments.

$f_2 = O(n!) < f_7 = O((n+1)!)$. We know this by simply examining $\frac{f_7}{f_2} = n + 1$. Therefore, $f_7$ is larger than $f_2$ by a non-constant factor of $n + 1$.

## 2 Problem 2

### 2.1 A [5 points]

The number of bits needed to represent $N$ is $\lfloor log_2(N) \rfloor + 1$, and the number of digits needed is $\lfloor log_{10}(N) \rfloor + 1$. The ratio of these two values is thus $\frac{\lfloor log_2(N) \rfloor + 1}{\lfloor log_{10}(N) \rfloor + 1} \cong log_2(10)$.

### 2.2 B [5 points]

The method is a recursive function call. Every iteration will call the method with input $(x, \lfloor \frac{y}{2} \rfloor)$, meaning each call removes the lowest bit from $y$ at every call. Because $y$ has $n$ bits, the function will recurse at most $n$ times. Consider now the rule when $y$ is odd. This rule incurs additional cost due to the addition. Note that multiplication and division by 2 is a constant time operation. The addition, however, depends on the number of bits in the numbers being added. This means that in the worst case, this addition could always be called, and will incur a running time cost of $O(m + n)$, as the number of bits of the resulting number from $x \cdot \lfloor \frac{y}{2} \rfloor$ is at most $m + n$. Thus, the overall running time is $n \cdot O(m + n) = O(mn + n^2)$.

# 3 Problem 3

## 3.1 A

**[5 points]** $2^{2013}$ $mod$ $3 = 2$. We can write $2^{2013}$ as $2 \cdot 2^{2 \cdot 1006} = 2 \cdot 4^{1006}$. Now, we take the mod 3 of this expression, and we get $(4^{1006} + 4^{1006})$ mod 3, where $4^{1006}$ mod $3 = 1$, as any power of 4, mod 3, will be 1. This leaves $1 + 1 = 2$.

**[5 points]**

$MMI($ 20 $mod$ 79 $) = 4$

$MMI($ 3 $mod$ 62 $) = 21$

$MMI($ 21 $mod$ 91 $)$ Does not exist. The reason this is so is because 21 and 91 are not relatively prime. That is that $gcd(21, 91) = 7 \neq 1$. If $gcd(a, n) > 1$, for $a, n \in \mathbb{Z}$, then $a \cdot x \not\equiv 1$ mod $n$ for any $x$; therefore, $a$ cannot have an MMI modulo $n$ (see Section 1.2.5 in the DPV book).

$MMI($ 5 $mod$ 23 $) = 14$

## 3.2 B [5 points]

Since $m$ has an inverse modulo $y$, the congruence $mx \equiv 1(\text{mod } y)$ has a solution for $x$ (for some integer $n$). Thus:

$$y \text{ is divisible by } mx - 1$$

$$\text{This implies that } \exists\ n : yn = mx - 1 \text{ or}$$

$$mx = yn + 1$$

$$\text{and then: } m \text{ is divisible by } yn + 1 \text{ or}$$

$$yn \equiv -1(mod\ m) \text{ and finally}$$

$$y(-n) \equiv 1(mod\ m)$$

Thus, $-n$ is an multiplicative inverse of $y$ modulo $m$.

# 4 Problem 4

## 4.1 A [5 points]

For all $x, y \in \mathbb{R} \mid x < y$, then $gcd(x, y) = gcd(x + y, x + 2y)$. Repeated application of Euclid's Rule produces the desired result.

$$gcd(x + y, x + 2y) = gcd(x + y, x + 2y\ mod\ x + y)$$

$$\begin{aligned} &= gcd(x+y, y) \\ &= gcd(x, x+y \ mod \ x) \\ &= gcd(x, y \ mod \ x) \\ &= gcd(x, y) \end{aligned}$$

### 4.2 B [5 points]

**Proof by Induction:** Base case is for the pairs $2, 1$, and $1, 3$. For two numbers to be relatively prime, it must be that $gcd(a, b) = 1$. In this case, it is easy to see that both $gcd(2, 1) = 1$ and $gcd(1, 3) = 1$. For the inductive step, we assume $L_{n-2}$ and $L_{n-1}$ are relatively prime, that is that $gcd(L_{n-2}, L_{n-1}) = 1$. According to the given rule, $L_n = L_{n-2} + L_{n-1}$ for the general case. From Euclid's rule, it follows that $gcd(L_{n-1}, L_{n-2} + L_{n-1}) = gcd(L_{n-1}, L_{n-2} + L_{n-1} \ mod \ L_{n-1}) = gcd(L_{n-2}, L_{n-1}) = 1$.

## 5 Problem 5

### 5.1 A [5 points]

| value | mod 5 | mod 7 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 3 | 3 |
| 4 | 4 | 4 |
| 5 | 0 | 5 |
| 6 | 1 | 6 |
| 7 | 2 | 0 |
| 8 | 3 | 1 |
| 9 | 4 | 2 |
| 10 | 0 | 3 |
| 11 | 1 | 4 |
| 12 | 2 | 5 |
| 13 | 3 | 6 |
| 14 | 4 | 0 |
| 15 | 0 | 1 |
| 16 | 1 | 2 |

### 5.2 B [5 points]

Numbers $x$, $y$ are prime, and furthermore, $q \equiv m \bmod x$ and $q \equiv n \bmod y$, where $0 \leq m < x$ and $0 \leq n < y$. To show that $q$ is unique, assume that there exists $r$ which satisfies $r \equiv m \bmod x$ and $r \equiv n \bmod y$. Furthermore, let $q = kx + m$ and $r = k'x + m$, since we know the mod remainder is $m$. Additionally, $q = ly + n$ and $r = l'y + n$. From this, we compute $q - r = (k - k')x$, or equivalently $q - r = (l - l')y$. It then follows that $q - r \equiv 0 \bmod x$ and $q - r \equiv 0 \bmod y$.

This implies that $q - r$ is a multiple of $xy$: $q - r = nxy, n \in \mathbb{Z}$. The only value of $q - r$ which is valid is $q - r = 0$, since $q, r < xy.$, implying that $r = q$ and there is thus a unique $q$ with this property.

## 5.3 C [5 points]

Using the terms provided in the hint, let $q = (m \cdot c_x + n \cdot c_y)$. Note that ($n$ mod $xy$) mod $x = n$ mod $x$ for arbitrary $n$. Compute the values of $c_x, c_y$ under mod $x$ and mod $y$. Then, we will have $c_x \equiv 1 mod x$, since $c_x$ corresponds to the product of $y$ with its multiplicative inverse under modulo $x$. Furthermore, $c_x \equiv 0 mod y$, since $c_x$ is a multiple of $y$. Similarly for $c_y$, which results in the following table:

| | $c_x$ | $c_y$ |
|---|---|---|
| mod $x$ | 1 | 0 |
| mod $y$ | 0 | 1 |

Then, $q$ mod $x = (m \cdot c_x + n \cdot c_y)$ mod $x = ((m \cdot c_x \text{ mod} x) + (n \cdot c_y \text{ mod } x)) = (m \cdot 1 + n \cdot 0) = m$. By a symmetric argument, $q$ mod $y = n$. This implies $q \equiv m(\text{mod } x)$, and $q \equiv n(\text{mod } y)$ and from part 5.B, we know that this is the unique value that can correspond to $m$ and $n$ modulo $x$ and $y$ correspondingly.

## 5.4 D [5 points]

The given properties do extend to three primes $x, y, z$ (actually extends to $n$ primes). In this case, they look like the following:

$$q \equiv m \; mod \; x \; and \; q \equiv n \; mod \; y \; and \; q \equiv p \; mod \; z$$

And we can write $q$ as:

$$q = (m \cdot c_x + n \cdot c_y + p \cdot c_z)$$

where:

$$c_x = yz((yz)^{-1} \; mod \; x)$$
$$c_y = xz((xz)^{-1} \; mod \; y)$$
$$c_z = xy((xy)^{-1} \; mod \; z)$$

# 6 Problem 6 [15 points]

First, consider how messages are generated given the same exponent $e = 3$. The encoding of the message satisfies: $c_i \equiv m^3 \; mod \; N_i$. Recall that $N_i$ is a component of individual $i$'s public key. An assumption is that the message is smaller than $N_i$, so it can transmitted in one communication attempt (otherwise it has to be split into smaller parts).

There are two cases for $N_i$. The $N_i$ may not be relatively prime. Then there is a method to generate the three recipient's private keys. Namely, if two of these $N_i$ are not relatively prime, then $gcd(N_i, N_j) \neq 1$. This means by computing this $gcd$, one of the values $p_i$ or $q_j$, which will be common between the two agents, will be revealed and it will become possible to factor both $N$'s. By factoring each $N_i$ to the values $p_i \cdot q_i = N_i$, it is possible to decode message $m$ by computing the decoding exponent: $d = 3^-1 mod(p_i - 1)(q_i - 1)$.

The $N_i$'s, however, may not be relatively prime. In this case, consider the three equalities that are true:

$$m^3 = c_1 \ mod \ N_1$$

$$m^3 = c_2 \ mod \ N_2$$

$$m^3 = c_3 \ mod \ N_3$$

By the definition of problem 5C, however, we can then compute the value $m^3 \ mod \ N_1N_2N_3$. Given that $m < N_i$, it has to be that $m^3 < N_1N_2N_3$, so we can actually directly compute the message raised to the power of 3. Thus, to compute $m$, we need to simply take the cubic root of $m^3$ to recover $m$. This is known as the Coppersmith/Håstad attack.

## 7 Problem 7 [20 points]

Consider the number of elements, $n$, in any given hashing location, $y \in Y$. To compute the lower bound of $p$, we must reason about the case which causes the least number of collisions. Because $|X| > |Y|$, there will be collisions. Consider an element $x \in X$ mapped to a location $y \in Y$. The probability that other elements in $X$ will also be mapped to the same $y$ is at most $p$. Therefore, the expected number of elements in this location will be at most $1 + p(|X| - 1)$. To minimize collisions, all of the $|Y|$ cells need to be populated with elements resulting in an expected number of elements assigned to cells computed according to: $|Y|(1 + p(|X| - 1))$. But we know that there are $|X|$ elements and thus $|Y|(1+p(|X|-1)) = |X|$, which simplifies to $p|Y|(|X|-1) = |X| - |Y|$. Solving for p, we get $p = \frac{|X|-|Y|}{|Y||X|-|Y|}$, which is an lower bound for the probability $p$.