CS 344: Homework 1

Mark Labrador, Janelle Barcia,Conrado Uraga

February 18, 2014

1 Big Oh Comparison

1. $\mathbf{f(n) = \sqrt{2^{7n}}}$, $\mathbf{g(n) = lg(7^{2n})}$
$\sqrt{128^n}$ = lg(49^n)
$128^{n/2}$ = lg(49) * n
lg($128^{n/2}$) = lg(lg(49) * n)
$\frac{lg(128)}{2}$ * n = lg(lg(49)) + lg(n)
Any polynomial dominates any logarithm.

Ω(g(n)) = {f(n): there exists positive constants c and n_0, such that 0 ≤ c * g(n) ≤ f(n) for all n ≥ n_0
$\frac{lg(128)}{2}$ * n ≥ lg(lg(49)) + lg(n) ≥ 0 for all n ≥ 0
with multiplicative terms omitted: n ≥ lg(n) ≥ 0 for all n ≥ 0
ANSWER: f(n) = Ω(g(n))

2. $\mathbf{f(n) = 2^{n*ln(n)}}$, g(n) = n!
By Stirling's Approximation, n! = $\sqrt{2\pi n}$ * $(\frac{n}{e})^n$
$2^{n*ln(n)}$ = $\sqrt{2\pi n}$ * $(\frac{n}{e})^n$
lg($2^{n*ln(n)}$) = lg($\sqrt{2\pi n}$ * $(\frac{n}{e})^n$)
n*ln(n)*lg(2) = n*lg($\sqrt{2\pi n}$ * $(\frac{n}{e})$)
n*ln(n) = n*lg($\sqrt{2\pi n}$ * $(\frac{n}{e})$)

$\frac{f(n)}{g(n)}$ is bounded:
$\frac{n*ln(n)}{n*lg(\sqrt{2\pi n}*(\frac{n}{e}))}$ < 1 as n approaches ∞

$\frac{g(n)}{f(n)}$ is bounded:
$\frac{n*lg(\sqrt{2\pi n}*(\frac{n}{e}))}{n*ln(n)}$ < 2 as n approaches ∞

As both functions are bounded as n grows to ∞, then **f(n) = Θg(n).**
ANSWER: f(n) = Θg(n).

3. f(n) = lg(lg * n), g(n) = lg*(lgn)
We can use the identity: lg*n = 1+lglg(n)
lg(1 + lglg(n) = 1 + lglglg(n)
$2^{lg(1+lglg(n)} = 2^{1+lglglg(n)}$
1 + lglg(n) = 2lglg(n)
1 + lglg(n) = lglg(n) + lglg(n)
2lglg(n) > lglg(n)
Therefore: O(g(n)) = {f(n): there exists positive constants c and n_0, such that 0 ≤ f(n) ≤ c * g(n) for all n ≥ n_0
0 ≤ f(n) ≤ g(n) for all n > 4
ANSWER: f(n) = O(g(n))

4. $\mathbf{f(n) = \frac{lgn^2}{n}}$, g(n) = lg*n
As we are concerned with large n as input grows when examining run time, we can make the substitution lg*n = 1 + lg(lgn) when n > 2.
$\frac{lgn^2}{n}$ = 1 + lg(lgn)
$\frac{2lgn}{n}$ = 1 + lg(lgn)
if we substitute k for lg(n), we can represent this function as:
$\frac{2k}{n}$, where k <n for all n
$\frac{2k}{n}$ = 1 + lg(k)
lg(n) ≥ k * $\frac{1}{n}$ for all n ≥ 1, as $\frac{1}{n}$ is a limit that approaches 0 when taken to infinity and is not increasing.

Therefore: O(g(n)) = {f(n): there exists positive constants c and n_0, such that 0 ≤ f(n) ≤ c * g(n) for all n ≥ n_0
0 ≤ f(n) ≤ g(n) for all n > 2
ANSWER: f(n) = O(g(n))

5. f(n) = 2^n, g(n) = n^{lgn}
2^n = n^{lgn}
lg(2^n) = lg(n^{lgn})
n * lg(2) = lgn * lgn
n = lg^{2}n
Any polynomial dominates any logarithm.
Ω(g(n)) = {f(n): there exists positive constants c and n_0, such that 0 ≤ c * g(n) ≤ f(n) for all n ≥ n_0
n ≥ lg^{2}n ≥ 0 for all n > 1
ANSWER: f(n) = Ω(g(n))

6. f(n) = 2$^{\sqrt{lgn}}$, g(n) = n(lgn)^3
$2^{\sqrt{lgn}}$ = n(lgn)^3
lg($2^{\sqrt{lgn}}$) = lg(n*(lgn)^3)
$\sqrt{lgn}$ * lg(2) = 3 * lg(n*(lgn))
$\sqrt{lgn}$ = 3 * lg(n*(lgn))
Linearithmic functions grow faster than linear functions (n < n*lgn), because linearithmic means the log function performed n times, therefore 3 * lg(n*(lgn)) ≥ $\sqrt{lgn}$ ≥ 1.
ANSWER: f(n) = O(g(n))

7. f(n) = e^{cosx}, g(n) = lgn
The function e^{cosx} always oscillates between the values $e^{cos(0)} \approx 2.71$ and $\frac{1}{e} \approx 0.367$, therefore it does not continuously increase. g(n) increases, which means the function will continuously grow for greater values of n.
O(g(n)) = {f(n): there exists positive constants c and n_0, such that 0 ≤ f(n) ≤ c * g(n) for all n ≥ n_0
lgn≥ e^{cosx} ≥ 0 for all n ≥ 0.
ANSWER: f(n) = O(g(n))

8. f(n) = lgn^2, g(n) = (lgn)^2
lgn^2 = (lgn)^2
2*lgn = lgn*lgn

$\frac{2*lgn}{lgn*lgn}$ < 1 for all n ≥ 8 (roughly e^2)
$\frac{lgn*lgn}{2*lgn}$ is unbounded and approaches ∞
The e O(g(n)) = {f(n): there exists positive constants c and n_0, such that 0 ≤ f(n) ≤ c * g(n) for all n ≥ n_0
lgn*lgn ≥ 2*lgn ≥ 0 for all n≥0
ANSWER: f(n) = O(g(n)).

9. f(n) = $\sqrt{4n^2 - 12n + 9}$, g(n) = n$^{\frac{3}{2}}$
$\sqrt{4n^2 - 12n + 9} = \sqrt{n^3}$
$\sqrt{4n^2 - 12n + 9}^2 = \sqrt{n^3}^2$
$4n^2 - 12n + 9 = n^3$
The exponent with the greatest value always dominates.

O(g(n)) = {f(n): there exists positive constants c and n_0, such that 0 ≤ f(n) ≤ c * g(n) for all n ≥ n_0
$n^3 \geq 4n^2 - 12n + 9$ ≥0 for all n≥ 0
ANSWER: f(n) = O(g(n))

10. f(n) = $\sum_{k=1}^{n} k$, g(n) = (n + 2)^2
$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$ for all natural numbers n shown by mathematical induction.
$\frac{n(n+1)}{2}$ = (n + 2)^2
$\frac{1}{2} * (n^2$ + n) = n^2 + 2n + 4
n^2 + n = 2n^2 + 4n + 8
$\frac{f(n)}{g(n)}$ is bounded:
$\frac{n^2+n}{2n^2+4n+8} \leq \frac{1}{2}$ as n approaches ∞

$\frac{g(n)}{f(n)}$ is bounded:
$\frac{2n^2+4n+8}{n^2+n} \leq 2$ as n approaches ∞

As both functions are bounded as n grows to ∞, then **f(n) = Θg(n).**
ANSWER: f(n) = Θg(n).

2 Runtime of Number Theoretic Algorithm

Algorithm 1: Number_Theoretic_Algorithm(integer n)
line 1: **N $\leftarrow$ Random_Sample(0,2$^n - 1$);** This runs at O(n) from bit shift 2 exponent
line 2: **if N is even then** O(1)
line 3: **N $\leftarrow$ N + 1 ;** O(1)
line 4: **m $\leftarrow$ N mod n;** O(n^2) because modular operaton
line 5: **for j $\leftarrow$ 0 to m do** linear loop that will execute from 0 to N-1, O(n)
line 6: **if Greatest_Common_Divisor(j,N) $\neq$ 1 then** GCD is O(n^3) according to DPV
line 7: **return FALSE;** O(1)
line 8: **Compute x,z so that N-1 = 2^z * x and x is odd;** this takes O(log n), shift until the last binary number is 1(this shows it's odd), then the number of shifts is z. N-1 is known previously and the value of x is the new binary number value. The number of bits is N-1 so O(log N-1), which goes to O(log n)
line 9: $y_0 \leftarrow$ **(N - 1 - j)^x mod N;** modular exponentiation O(n^3)
line 10: **for i $\leftarrow$ 1 to m do** linear loop that will execute from 0 to N-1, O(n)
line 11: $y_i \leftarrow {y_{i-1}}^2$ **mod N;** O(n^2) because modular operation costs O(n^2) and y*y costs O(n^2), O(n^2) + O(n^2)
line 12: $y_i \leftarrow y_i + y_{i-1}$ **mod N;** O(n^2) + O(n)
line 13: **if Low_Error_Testy_m == FALSE** prime test is O(n^3) according to DPV
line 14: **return FALSE;** O(1)
line 15: **return TRUE;** O(1)
The running time for lines 1-4 are: O(n) + O(1) + O (1) + O(n^2)
Lines 5-14 are contained within a loop that runs at most (n-1) times. The run time of these combined lines is: O(n) + O(n^3) + O(1) + O(log n) + O(n^3) + O(n) + O(n^2) + O(n^2) + O(n^3) + O(1)
These lines all run n times so the runtime here is n*(O(n) + O(n^3) + O(1) + O(log n) + O(n^3) + O(n) + O(n^2) + O(n^2) + O(n^3) + O(1))
giving a combination of: O(n) + O(1) + O (1) + O(n^2) + n*(O(n) + O(n^3) + O(1) + O(log n) + O(n^3) + O(n) + O(n^2) + O(n^2) + O(n^3) + O(1))
The most expensive operation is n*O(n^3), giving a total runtime of O(n^4).

3 Asymptotic Tree Analysis

Problem 3a -
The lower bound of the height of a tree data structure T_m^N, where every node has at most m children and the tree has at most N nodes, occurs when a complete tree is formed (which is when every level has m children, and the last level has at most m children). The height of the lower bound can be shown by comparing it to the total number of nodes at every level: 1 + m + m^2 + m^3 + ... + m^{h-1} = $\frac{m^h-1}{m-1}$ = N
$\frac{m^h-1}{m-1}$ = N
m^h = N * (m - 1) + 1
m^h = N * (m - 1 + $\frac{1}{N}$)
$\log_m(m^h) = \log_m$(N * (m - 1 + $\frac{1}{N}$))
h = $\log_m$(N) + $\log_m$(m - 1 + $\frac{1}{N}$))
Therefore the total height can be computed by $\lceil \log_m N(m-1) \rceil$.

Problem 3b -
To show the asymptotic behavior of two functions, we can take a limit of both of the functions to see what they are approaching when tending to ∞.
h = $\log_m$(N) + $\log_m$(m - 1) = $\log_m$(N(m - 1))
f(n) = $\log_m$(N(m - 1))
g(n) = $\log_{m'}$(N(m$'$ - 1))
$\lim_{n\to\infty} \frac{f(n)}{g(n)}$
$\lim_{n\to\infty} \frac{\log_m(N(m-1))}{\log_{m'}(N(m'-1))}$
$\lim_{n\to\infty} \frac{\log_m(Nm-N))}{\log_{m'}(Nm'-N))}$
$\lim_{n\to\infty} \frac{\log_m(m)}{\log_{m'}(m')}$
It must be noted that this only holds true when, m > 1 because (1-1) is 0 and the log(0) does not exist.

Problem 3c -
We are given a similar recursive algorithm for Modular exponentiation in DPV 1.2.2. We are shown:
x * 2^i comes from repeated doubling for exponentiation the corresponding terms x^{2^i} are generated by repeated squaring. We square the value repeated modulo N for a more efficient runtime, as there are only log y multiplications when doing it this way.Therefore the n is the size in bits of the max(x, y, N). the algorithm runs for n recursive calls and it multiplies n-bit numbers which gives a running time of O(n^3). The difference in the given problem is that N is in the order of 2^o, y is in the order of 2^n, x is in the order of 2^m.
If y is odd or even then the algorithm starts off with bit shifts shown by $\lfloor \frac{y}{2} \rfloor$, O(n)
To get the value squared for both odd and even, we use two multiplications (using modulo N) with o-bits that cost O(o^2).

Problem 4a - Multiplicative Inverses

$$2^{902} \equiv\ 2^{6\cdot150+2} \equiv\ 2^2 2^{6\cdot150} \equiv\ 2^2(2^6)^{150} \equiv\ 2^2(1)^{150} \equiv\ 2^2 \equiv\ 4 \bmod 7, \text{ by Fermat's Little Theorem.}$$

Problem 4b - Multiplicative Inverses

- $11y \equiv\ 1 \bmod 120,\ y = 11$
- $13y \equiv\ 1 \bmod 45,\ y = 7$
- $35y \equiv\ 1 \bmod 77$, y, does not exist because 35 and 77 are not relatively prime.
- $9y \equiv\ 1 \bmod 11,\ y = 5$
- $11 \equiv\ 1 \bmod 1111$, y does not exist because 11 and 1111 are not relatively prime.

Problem 4c - NO ANSWER

Problem 5a - Greatest Common Divisor True.

$$\begin{aligned}\gcd(x,y) &= \gcd(x, x+y)\\ &= \gcd(x+y, x+x+y)\\ &= \gcd(2x+y+x+y, 2x+y)\\ &= \gcd(3x+2y, 2x+y+3x+2y)\\ &= \gcd(3x+2y, 5x+3y)\end{aligned}$$

Problem 5b - Greatest Common Divisor This will be proved using the property that $\gcd(a,b) = \gcd(b, amodb)$.

Assume that $1 \le i, j \le n$, $i \ne j$, and $i < j$. Observe the following:

$$s_j \ \equiv\ 1 \text{mod } s_i$$

This is because $s_j = 1 + \prod_{l=0}^{j-1} s_l$, which means s_i is contained in the term $\prod_{l=0}^{j-1} s_l$. So applying the mod operator with s_i will cause this term to disappear, and leave 1 as the remaining term. This implies, $\gcd(s_j, s_i) = \gcd(s_i, s_j \bmod s_i) = \gcd(s_i, 1) = 1$. Therefore, all s_k are relatively prime.

Problem 6a - Universal Hashing Suppose $h \in H$, where H is the family of hashing functions, and $m \in M$, where M is the set of all 8 x 32 binary matrices. If a 32-bit integer is selected and converted to a 32 x 1 matrix called, y, then the following operation is performed,

$$h(y) = m \cdot y \text{ mod } 2$$

Let $s_i = \sum_{j=0}^{31} m_{i,j} y_j$ mod 2, where $m_{i,j}$ is the entry of the i$^{\text{th}}$ row and j$^{\text{th}}$ column of the matrix M and y_j is the j$^{\text{th}}$ row of the y matrix. After $h(m, y)$ is performed, the resulting 8-bit vector call, H has the entries s_i for $i = 0, \ldots, 7$. To determine the probability of hashing to any one slot of the 256 possible slots, the probability of hashing to any one 8-bit number is what needs to be determined, bit-by-bit.

Suppose two distinct integers are chosen, y_1 and y_2 such that their last bit differs. So to compute the probability of picking a row like this, the following relationship is established.

Let E be the event where the last bit of each of column of m is chosen such that the relationship below holds.

$$\sum_{j=0}^{30} m_{i,j} \left(y_{2j} - y_{1j}\right) \;\equiv\; m_{i,31}(y_{2(31)} - y_{1(31)}) \text{ mod } 2$$

$$Pr\{h(m, y_1) = h(m, y_2)\} = Pr\{\text{E}\}$$

Since 2 is prime and $y_{2j} \neq y_{1j}$, there is an unique inverse for $y_{2(31)} - y_{1(31)}$ that is either 0 or 1. So $Pr\{E\} = \frac{1}{2}$.

This occurs for every row of the matrix H. So the probability of getting an 8-bit matrix H is the product of its parts. This means, $Pr\{\text{Hashing to 1 out of 256 slots}\} = Pr\{E\} = (\frac{1}{2})^8 = \frac{1}{256}$. Therefore, the family of functions, H is universal.

Problem 6b - Random Bits This family required 256 random bits.

Problem 7a
Finding the integers that are their own inverses is the same as asking, $x^2 \equiv\ 1 \bmod n$. This gives the following,

$$\begin{aligned}
x^2 &\equiv\ 1 \bmod n\\
x^2 - 1 &\equiv\ 0 \bmod n\\
(x+1)(x-1) &\equiv\ 0 \bmod n\\
x+1 &\equiv\ 0 \bmod n \rightarrow x \equiv\ -1\ \equiv\ n-1 \bmod n\\
x-1 &\equiv\ 0 \bmod n \rightarrow x \equiv\ 1 \bmod n
\end{aligned}$$

So the integers that are their own inverses are $n-1$ and 1 modulo n for x in the range of 0 to $n-1$.

Problem 7b
For $p = 2$, $(p-1)! \equiv\ (2-1)\ \equiv\ 1\ \equiv\ -1 \bmod 2$.

Suppose $p > 2$ and p is prime. Then $b \in B = \{0, 1, 2, \ldots, p-1\}$ has a multiplicative inverse modulo p because $(\forall\, b \in B)\, (\gcd{(b, p)} = 1)$, which will be called b^{-1}. These inverses lie in the set B. So there will be $\frac{p-3}{2}$ pairs of inverses because $p-1$ and 1 are their own inverses from part a of this problem. This implies the following,

$$\begin{aligned}
(p-2)! &\equiv\ 1 \bmod p\\
(p-1)(p-2)! &\equiv\ p-1 \bmod p\\
(p-1)! &\equiv\ -1 \bmod p
\end{aligned}$$

Problem 7c
Suppose n is a composite number. So there are integers a and b such that $n = ab$. This implies that $a < n$ and $b < n$, which means a and b will be in the product $(n-1)!$. So $(n-1)! \equiv\ 0 \bmod n$, and not $-1 \bmod n$.

Problem 7d
This primality test requires $n-2$ multiplications to compute $(n-1)!$. This requires, $O(n(log_2 n)^2)$ bit operations.

Problem 8a - Chinese Remainder Theorem

Number	modulo 5	modulo 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
17	2	3
18	3	4
19	4	5
20	0	6
21	1	0
22	2	1
23	3	2
24	4	3
25	0	4
26	1	5
27	2	6
28	3	0
29	4	1
30	0	2
31	1	3
32	2	4
33	3	5
34	4	6
35	0	0
36	1	1

Problem 8b - Chinese Remainder Theorem
Suppose x and y are two different prime numbers, and for every pair of integers m and n, $0 \leq m < x$ and $0 \leq n < y$.

Let A $= \{0, 1, \ldots, xy - 1\}$. This is the range of xy.

Since $0 \leq m < x$ and $0 \leq n < y$, it is known that $0 \leq my < xy$ and $0 \leq nx < xy$, which implies the following, $my \in A$ and $nx \in A$. If q is selected to be the following:

$$q = myy^{-1} + nxx^{-1}, \text{ where } y^{-1} \text{ and } x^{-1} \text{ are inverses of } y \bmod x \text{ and } x \bmod y \text{, respectively.}$$

The inverses of x and y are defined because they are two different primes, making them relatively prime. Then $q \pmod{xy} \in A$ by definition of the modulus operator, which allows the following, $0 \leq q < 2xy \rightarrow 0 \leq q \pmod{xy} < xy$.

The next step is to show the following:

$$q \equiv m(\text{mod } x)$$
$$q \equiv n(\text{mod } y)$$

Using the selection of q as the starting point, the integers m and n will be derived, mod x and mod y, respectively.

$$q \text{ mod } x \equiv myy^{-1} + nxx^{-1} \equiv mmodx, \text{ because } yy^{-1} \text{ is 1 mod } x \text{ since they're inverses of each other, and } nxx^{-1} \text{ disappears.}$$
$$q \text{ mod } y \equiv myy^{-1} + nxx^{-1} \equiv nmody, \text{ because } xx^{-1} \text{ is 1 mod } y \text{ since they're inverses of each other, and } myy^{-1} \text{ disappears.}$$

Now to prove the uniqueness of q. Suppose there are two choices that satisfy the system above, $q_1, q_2 \in A$. Then the following is true,

$$q_1 \equiv m \text{ mod } x$$
$$q_1 \equiv n \text{ mod } y$$
$$q_2 \equiv m \text{ mod } x$$
$$q_2 \equiv n \text{ mod } y$$

$$q_1 - q_2 \equiv 0 \text{ mod } x \rightarrow x \mid q_1 - q_2$$
$$q_1 - q_2 \equiv 0 \text{ mod } y \rightarrow y \mid q_1 - q_2$$

So $xy \mid q_1 - q_2$, which means $q_1 \equiv q_2 \text{ mod } xy \rightarrow q_1 = q_2$ because $q_1, q_2 \in A$.

Therefore, q is unique.

Problem 8c - Chinese Remainder Theorem
Suppose x and y are different prime numbers such that,

$$q \equiv m \text{ mod } x$$
$$q \equiv n \text{ mod } y$$

Let $M_x = y$, $M_y = x$

a_x, be the inverse of M_x mod x.
a_y, be the inverse of M_y mod y.

So the follow equation for q is derived,

$$q = mM_x a_x + nM_y a_y \text{ mod } xy$$

When q is mod-ed with x, the second term $nM_y a_y$ disappears because $M_y = x$, and the part of the first term, $M_x a_x \equiv 1$ mod x because they are inverses of each other mod x. So $q \equiv m$ mod x.

When q is mod-ed with y, the first term $mM_x a_x$ disappears because $M_x = y$, and the part of the second term, $M_y a_y \equiv 1$ mod y because they are inverses of each other mod y. So $q \equiv n$ mod y.

Problem 8d - Chinese Remainder Theorem
In the case of three primes, x, y, and z, the property still holds. When it is three primes the equation for q changes to the following:

$$q \equiv a_x M_x I_x + a_y M_y I_z + a_z M_z I_z \bmod M$$

where the parts of the equation are defined as followed: Let $M = xyz$.

a_x, a_y, a_z, be the residues when mod-ed x, y, and z, respectively.
$M_x = \dfrac{M}{x} = yz, M_y = \dfrac{M}{y} = xz, M_z = \dfrac{M}{z} = xy$
I_x, I_y, I_z, be the inverses of M_x mod x, M_y mod y, and M_z mod z, respectively.

Problem 9 - RSA Cryptography

Let N_b, N_c, N_d be Bob, Charlie, and David's public key, respectively.
$M = N_b N_c N_d$, $M_b = \dfrac{M}{N_b}$, $M_c = \dfrac{M}{N_c}$, $M_d = \dfrac{M}{N_d}$
e, be the encryption key for Bob, Charlie, and David.
m_a, be the message sent by Alice.

With the given information, the Chinese Remainder Theorem is applicable to find m_a:

$$\begin{aligned}
e &= 3 \\
M &= 674 \cdot 36 \cdot 948 = 23002272 \\
M_b &= 34128,\ M_c = 638952,\ M_d = 24264 \\
(m_a)^e \equiv\ (m_a)^3 &\equiv\ 674 \bmod N_b\ \equiv\ 674 \bmod 3337 \\
&\equiv\ 36 \bmod N_c\ \equiv\ 36 \bmod 187 \\
&\equiv\ 948 \bmod N_d\ \equiv\ 948 \bmod 1219
\end{aligned}$$

From the theorem, the equation we are looking for is:

$$(m_a)^3\ \equiv\ (674)M_b y_b + (36)M_c y_c + (948)M_d y_d\ (modM) \tag{1}$$

The next step is to determine the inverses, y_b, y_c, and y_d.

- $M_b y_b$ mod N_b, $y_b = 2593$.
- $M_c y_c$ mod N_c, $y_c = 90$.
- $M_d y_d$ mod N_d, $y_d = 620$.

Going back to equation (1), insert the terms determined here:

$$\begin{aligned}
(m_a)^e &\equiv\ (674)M_b y_b + (36)M_c y_c + (948)M_d y_d \text{ (mod } M) \\
(m_a)^3 &\equiv\ (674)(34128)(2593) + (36)(638952)(90) + (948)(24264)(620) \text{ (mod 23002272)} \\
&\equiv\ 75976504416 \text{ (mod 23002272)} \\
&\equiv\ 0 \text{ (mod 23002272)} \\
m_a &\equiv\ 0 \text{ (mod 23002272)}
\end{aligned}$$

Therefore, the original message was $m_a = 0$.