CS 3510: Design and Analysis of Algorithms
Section A, Spring 2017

Homework 3
Due: Wednesday 11:55PM, March 1st, 2017

Name: Alan Chiang

Notes:

1. Please don't write a program when you describe an algorithm. You should give an idea of what the algorithm does and then discuss the details of how your algorithm does specific tasks. If you give pseudo-code, you should explain your code.
2. For every algorithm that you are asked to describe, please clearly list the steps of your algorithm and explain its correctness and runtime behavior. We recommend separating the algorithm, correctness argument, and runtime analysis into three paragraphs.
3. You can assume that basic operations on integers which are not described to have a variable length (addition, multiplatication, subtraction, floor, ceiling, absolute value, comparison of less than, greater than, or equals, and anything you can form out of a constant number of these) can be done in constant time. You can also assume that, given i, you can read a_i from a list $a_1, ...a_n$ in constant time.

1. Consider the following recursive multiplication algorithm.

```
Function multiply(x, y)
    Data: Two $n$-bit integers $x$ and $y$, where $y \geq 0$
    Result: Their product $xy$

    if $y = 0$ then
        return 0
    end
    $z = multiply(x, \lfloor y/2 \rfloor)$;
    if $y$ is even then
        return $2z$;
    else
        return $x + 2z$;
    end
```

(a) How long does this algorithm take to multiply two n-bit numbers? Justify your answer.

(b) Give an algorithm to compute $N! = 1 \times 2 \times 3 \times ... \times N$, and analyze its running time.

Answer: a. The algorithm runs in $O(n^2)$ time because it makes n recursive calls, once every time it divides y in half. It divides y until $y = 0$, which must be done $log(n)$ times because y is an n-bit number and dividing by two can be done by rightshifting one bit. In each of the n recursive calls, the algorithm must perform conditional checks $y = 0$ and y is even, which can both be done in $O(1)$ time using bitmasks to check all bits and the rightmost bit, respectively. It also performs one of the two operations $2 * z$ or $2 * z + x$. Since $2 * z$ can be rewritten as $z + z$, and addition is an $O(n)$ operation, the algorithm takes $O(n)$ time during each of the n recursive calls. Therefore, the total runtime of this algorithm is $O(n * n$ which can be written $O(n^2)$.

b. The naive algorithm to compute $N!$ goes as follows: int sum = 1 for each number num in $1, 2, 3, ...N$ sum = multiply(sum, num) return sum

If we define s as the length of sum in bits (worst case $s = N * log(N)$ bits), and n as the length of num in bits ($n = log(N)$ bits), and N as the magnitude of the input, then this algorithm runs in $O(N * s * n)$ time. Because it iterates through all numbers from $1....N$ and at each iteration it performs the $O(s * n)$ operation of multiplying sum and num.

2. Calculate $2^{125} \mod 127$. Explain your method. (Hint: 127 is prime).

Answer: By Fermat's little theorem, we know that $a^{p-1} \equiv 1 \pmod{p}$. In this case, $a = 2$ and $p = 127$. We then can see that $2^{126} \equiv 1 \mod 127$. $2^{126} \mod 127$ is equivalent to $2 \mod 127 * 2^{125} \mod 127$ by the rules of modular multiplication. Since we know $2^{126} \mod 127 = 1$, we can simplify the problem to $(2 \mod 127) * (2^{125} \mod 127) = 1$. Multiplying both sides of the equation by $2^{-1} \mod 127$ will cancel out $2 \mod 127$ on the left side and on the right side, $2^{-1} \mod 127 * 1 = 2^{-1} \mod 127$. So our answer is $2^{-1} \mod 127 = 64$.

3. (a) Compute $gcd(6, 21)$ two different ways: by finding the factorization of each number, and by using Euclid's algorithm.
   (b) Find the inverse of: 20 mod 79, 21 mod 91.
   (c) Prove or disprove: If a has an inverse modulo b, then b has an inverse modulo a.

Answer: a. 1. Euclid's algorithm: $gcd(6, 21) = gcd(21, 6) = gcd(6, 3) = gcd(3, 0)$. So $gcd(6, 21) = 3$. 2. Facorization: Factors of six are $1, 2, 3, 6$. Factors of twenty-one are $1, 3, 7, 21$. The greatest common factor is obviously 3.

b. 1. Modular inverse of $20 \mod 79$ is the number a which, multiplied by 20, equals one $20 * a \mod 79 = 1$. $4 * 20 = 80$. $80 \mod 79 = 1$. So the modular inverse of $20 \mod 79$ is 4. i. Using Euclid's extended algorithm, we find:

$79|1|0$

$20|0|1$

$19|1|-3$

$1|-1|4$

So the modular inverse of $20 \mod 79$ is 4.

2. Because 21 and 91 share the common factor of 7, the modular inverse of $21 \mod 91$ does not exist. i. Using Euclid's extended algorithm does not change the common factor of 7, so we cannot find a modular inverse.

c. This is true. To prove this, we begin with the definition of modular inverse. If a has an inverse modulo b, this means $a * x - 1$ is divisible by b for some arbitrary x. This in turn means that $b * y = a * x - 1$ for some arbitrary y, by the definition of division. Moving the one, we get $b * y + 1 = a * x$. By the definition of division, this means that $b * y + 1$ is divisible by a. By the definition of modulus, we get $b * y + 1 = 0 \mod a$, which equals $b * y = -1 \mod a$, which equals $b * (-y) = 1 \mod a$. At this point, we can see that the definition of inverse modulo means that $(-y)$ is the inverse of $b \mod a$, and since $(-y)$ clearly exists, this means that b has an inverse modulo a.

4. In an RSA cryptosystem, $p = 7$ and $q = 11$.

   (a) Find appropriate exponents d and e.

   (b) Encrypt the secret message "GT" by encoding each letter individually (assume A=1).

Answer: a. $N = p * q = 7 * 11 = 77$. $M = (7-1)*(11-1) = 6*10 = 60$. $e = 73, gcd(73, 60) = gcd(60, 13) = gcd(13, 8) = gcd(8, 5) = gcd(5, 3) = gcd(3, 2) = gcd(1, 0)$, so 73 and 60 are coprimes. $d = e^{-1} \mod M = 73^{-1} \mod 60 = 37$, because $37 * 73 = 2701$ and $2701 \mod 60 = 1$.

So $d = 37$ and $e = 73$

b. Given $A = 1$, $G = 7$ and $T = 20$:

G, the encoded value of G $= X^e \mod N = (7^{73}) \mod 77 = 35$. T, the encoded value of T $= X^e \mod N = (20^{73}) \mod 77 = 69$

So the encoded message is the concatenation of $G + T = 3569$.