Assignment 01

Paul Jones and Matthew Klein
Professor Professor Kostas Bekris
Design and Analysis of Computer Algorithms (01.198.344)

February 18, 2014

Part A

Problem 1

In each of the following situations indicate whether $f = O(g)$ or $f = \Omega(g)$ or $f = \Theta(g)$:

1. $f(n) = \sqrt{2^{7x}}, g(n) = \lg(7^{2x})$

$$f(n) = \sqrt{2^{7x}} = \sqrt{128^x}$$
$$g(n) = \lg(7^{2x}) = \lg(49^x)$$
$$lg(49^1) \approx 5.6$$
$$\sqrt{128^1} \approx 11.3$$

   Notice that both of these functions only grow relative to x.

$$f = \Omega(g)$$

2. $f(n) = 2^{nln(n)}, g(n) = n!$

   The factorial, that is $n!$, function grows much, much faster than 2^n.

$$f = \Omega(g)$$

3. $f(n) = \lg(\lg^*(n)), g(n) = \lg^*(\lg(n))$

$$f = \Theta(g)$$

4. $f(n) = \frac{lg(n^2)}{n}, g(n) = lg^*(n)$

$$f(n) = \frac{lg(n^2)}{n} = \frac{2lg(n)}{n}$$

$$f = \Theta(g)$$

5. $f(n) = 2^n, g(n) = n^{\lg(n)}$

   This is comparing the exponential function to a function that is less than n^2.

$$f = \Omega(g)$$

6. $f(n) = 2^{\sqrt{\ln(n)}}, g(n) = n(\lg(n)^3)$

$$f(n) = 2^{\sqrt{n}}, g(n) = (2^n)(n^3)$$

$$f = \Omega g$$

7. $f(n) = e^{\cos(x)}, g(n) = \lg(x)$

$$f = \Omega(g)$$

8. $f(n) = \lg(n^2), g(n) = (\lg(n))^2$

$$f = \Theta(g)$$

9. $f(n) = \sqrt{4n^2 - 12n + 9}, g(n) = n^{\frac{3}{2}}$

$$f = \Theta(g)$$

10. $f(n) = \sum_{k=1}^{n} k, g(n) = (n+2)^2$

$$f = \Omega(g)$$

Problem 2

Algorithm 1: `Number_Theoretic_Algorithm` (integer n)

```
N ← Random_Sample(0, 2^n − 1);
if N is even then
    N ← N + 1 /* Worse case, N is odd, 2 ** N - 1. */ ;
m ← N mod n /* worse case same as n */ ;
for j ← 0 to m do
    if Greatest_Common_Divisor( j, N) ≠ 1 then
        return FALSE; /* GCD is O(n) */
    Compute x, z so that N − 1 = 2^z · x and x is odd;
    y_0 ← (N − 1 − j)^x mod N;
    for i ← 1 to m do
        y_i ← y_{i−1}^2 mod N ;
        y_i ← y_i + y_{i−1} mod N;
    if Low_Error_Primality_Test(y_m) == FALSE then
        return FALSE /* Naive primality test is O(sqrt(n)) */ ;
return TRUE;
```

Compute the asymptotic running time of the above algorithm as a function of its input parameter, given:

- **The running times of integer arithmetic operations (e.g., multiplication of two large n-bit numbers is $O(n^2)$).**
- **Assume that sampling a number N is an operation linear to the number of bits needed to represent this number.**

- Worse case running n operations with times $O(n)$, $O(n)$, and $O(\sqrt{n})$ (refer to marked-up algorithm above, in comments).
- That's a run time of $O(2n^2 + n^{\frac{3}{2}})$, resulting in big-O of $O(n^2)$.

Part B

Problem 3

- **Consider that we have a tree data structure T_m^N, where every node can have at most m children and the tree has at most N nodes total. Compute a lower bound for the height of the tree.**

  A tree with m children is $\log_m(N+1) - 1$.

- **Consider two such trees T_m^N and $T_{m'}^N$ that are "perfect", i.e., every node has exactly m and m' children correspondingly. Now, consider the functions $h_m(N)$ and $h_{m'}(N)$ that express the heights of these perfect trees for different values of N. What is the asymptotic behavior of h_m relative to $h_{m'}$ and under what conditions?**

  A perfect tree will only be changing based on the m, m' values. Whichever value is larger will run faster.

- **Consider the following rule for modular exponentiation, where x is in the order of 2^m and y is in the order of 2^n. What is the running time of computing the result according to this rule?**

$$x^y = \begin{cases} (x^{\lfloor \frac{y}{2} \rfloor})^2, & \textbf{if y is even} \\ x \cdot (x^{\lfloor \frac{y}{2} \rfloor})^2, & \textbf{if y is odd} \end{cases}$$

  - On the top level, just like multiplication, this algorithm will have at most n recursive calls.
  - During each call it multiplies n-bit numbers, which is in the order of $O(n^2)$.
  - The resulting $O(n \cdot n^2)$ is $O(n^3)$.

Problem 4

- **Compute the following: 2^{902} mod 7.**

  I found out how to do this using a website, since I didn't understand how to from lecture ? 2^{902} mod 7 We can find the original, 2 mod 7 = 2 because 7 doesn't go into 2 at all. We can next square, finding 4 mod 7 = 4. Divide exponent in half, 2^{451} mod 7. Next we can do 4 mod 7 = 4 again, and square. 16 mod 7 = 2. Once again we cut our exponent, 2^{225} mod 7. Now we have $4 \cdot 2$ mod 7 $\rightarrow$ 8 mod 7 = 1. Next we square our other value, 4 mod 7 = 4. We divide exponent again, 2^{112} mod 7, and we do 16 mod 7 = 2. Another cut, 2^{56} mod 7. We can check 2^2 mod 7 = 4. Another time we cut, 2^{28} mod 7. We need to use previous value again, 16 mod 7 = 2. 2^{14} mod 7 from another cut, and we use 4 mod 7 = 4. We can cut again,

$2^7 \bmod 7$ and we use $4 \bmod 7 = 4$. We are almost done and use $2^3 \bmod 7$. We must check $8 \bmod 7 = 1$, and now we are on the final step. $2^1 \bmod 7 = 4$

- **Find the modulo multiplicative inverse of 11 mod 120, 13 mod 45, 35 mod 77, 9 mod 11, 11 mod 1111.**

  $11 \bmod 120 = 121, 13 \bmod 45 = 91, 9 \bmod 11 = 45$. For the last one and third one I used Extended Euclidean Algorithm discussed in class. I also used $p_i = p_{i-2} - p_{i-1}q_{i-2} \bmod n$.

  Third one: $35 \bmod 77 \to 77 = 2(35) + 7$ and $p_0 = 0$. Next, $35 = 5(7) + 0$ and $p_1 = 1$. However, this can't be solved.

  Last one: $11 \bmod 1111 \to 1111 = 101(11) + 0$. This one can't be solved either because we were unable to get past the step, like the third one.

- **Assume that for a number x the following property is true: $\forall\ y \in [1, x-1]$: $gcd(x, y) = 1$. Compute the running time of an efficient algorithm for finding all the inverses modulo x^m from the set $\{0, 1, \ldots, x^m - 1\}$ that exist.**

  $\forall y \in [1, x-1] : \gcd(x, y) = 1$. If we want to find all of the modulo x^m between $0, 1, ..., x^m - 1$ then we can assume there are m total modulo inverses to compute. An example is that there $x = 2, m = 2$ to keep it simple. This means that every number from $1 \to 1 : \gcd(1, 1) = 1$ which is correct. Now we need to find $0, ..., 2^2 - 1$ which becomes $0, ..., 3$. We have a total of 4 numbers to modulo inverse. The running time to find is the amount multiplied by the time it takes to run the euclidean algorithm. There's a total of x^m to find and the Extended Euclidean algorithm takes $\log(m^2)$. Our total runtime is $x^m \log(m^2)$.

Problem 5

- **Assume two positive integers $x < y$. Then the pairs $(5x+3y, 3x+2y)$ and (x, y) have the same greater common divisor. True or False, explain.**

  - The way we can solve this is using the following lemma, where for all integers and numbers greater than zero:

    $$\gcd(x, y) = \frac{xy}{\text{lcm}(x, y)}$$

  - Notice that the left half of the equation will serve as an answer to the the second pair in the given problem.
  - Witness that the lowest common multiple of x and y is xy.
  - For the first pair, witness that by the same rule:

    $$\gcd((5x+3y), (3x+2y)) = \frac{(5x+3y)(3x+2y)}{\text{lcm}((5x+3y), (3x+2y))}$$

  - Witness that the lowest common multiple of $(5x+3y)$ and $(3x+2y)$ is $(5x+3y)(3x+2y)$.
  - Multiply

    $$(5x+3y)(3x+2y) = 15x^2 + 19xy + 6y^2$$

– Notice that the claim being tested by the problem can be symbolized by the following:

$$\forall x \forall y((x < y \wedge x, y \neq 0 \wedge x, y \in \mathbb{Z}) \rightarrow \gcd((5x+3y),(3x+2y)) = \gcd(x,y))$$

– Just consider the consequent:

$$\gcd((5x+3y),(3x+2y)) = \gcd(x,y)$$

– Substitute the equalities we found earlier:

$$\frac{(5x+3y)(3x+2y)}{\operatorname{lcm}((5x+3y),(3x+2y))} = \frac{xy}{\operatorname{lcm}(x,y)}$$

– Substitute the lowest common multiples we found earlier:

$$\frac{(5x+3y)(3x+2y)}{15x^2+19xy+6y^2} = \frac{xy}{xy}$$

– Multiply

$$\frac{15x^2+19xy+6y^2}{15x^2+19xy+6y^2} = \frac{xy}{xy}$$

– Simplify

$$1 = 1$$

– It's true.

- **Consider the following sequence of numbers: $s_n = 1 + \Pi_{i=0}^{n-1} s_i$, where $s_0 = 2$. Prove that any two numbers in this sequence are relatively prime.**

  – I don't know.

Part C

Problem 6

- **A proof that the hash function family $\mathcal{M}$ is universal.**

  The hash function for the family is definitely consistent because each item is only either $0, 1$, and we are modding by the total amount of choices, but I'm not sure how to prove this other than by what was stated in class: $Pr = h_\alpha(x) = h_\alpha(y) = Pr\sum_{i=1}^{4} \alpha_i \cdot x_i = \sum_{i=1}^{4} \alpha_i \cdot y_i \bmod N =$ $Pr \underbrace{\sum_{i=1}^{3} \alpha_i(x_i - y_i) = \alpha_4(y_4 - x_4 \bmod N)}_{\text{given x, y and randomly picked a1,a2,a3: c is constant}}$. However, in this case this would be different because we have $0, 1$ not $1, ..., 4$: $Pr = h_\alpha(x) = h_\alpha(y) = Pr\sum_{i=1}^{2} \alpha_i \cdot x_i = \sum_{i=1}^{2} \alpha_i \cdot y_i \bmod N =$

$$Pr \underbrace{\sum_{i=1}^{1} \alpha_i(x_i - y_i) = \alpha_2(y_2 - x_2 \bmod N)}_{\text{given x, y and randomly picked a1: c is constant}} .$$

- **A comparison to the universal hash function family described in DPV chapter 1.5.2. How many random bits are needed here?**

Problem 7

- **Assume that number n is prime, then all numbers $1 \leq x < n$ are invertible modulo n. Which of these numbers are their own inverse modulo n?**

  - x is its own inverse mod$n$ if $x^2 = 1 \bmod n$ which means:

    $$x^2 - 1 = 0 \bmod n$$

  - Were interested in the numbers less than n but greater than or equal to 1, so observe that for all n:

    $$1^2 - 1 = 0 \bmod n$$
    $$0 = 0 \bmod n$$

  - Therefore 1 is a value of x that will be its own inverse modulo n.

- **Show that** $(n-1)! \equiv -1 \bmod n$ **for prime** n**.**

  - Expand left-hand side:

    $$(n-1)! = (n-1) \times (n-2) \times \cdots \times 3 \times 2 \times 1$$

  - What we want to do is pair each number with its inverse.
  - But some number are there own inverses! Which ones are these?

    $$x^2 \equiv 1(\text{mod} n)$$
    $$x^2 - 1 \equiv 0(\text{mod} n)$$
    $$(x-1)(x+1) \equiv 0(\text{mod} n)$$

  - Now solve just with algebra

    $$x - 1 \equiv 0(\text{mod} n) \to x \equiv 1(\text{mod} n)$$
    $$x + 1 \equiv 0(\text{mod} n) \to x \equiv -1(\text{mod} n)$$

  - So 1 and $n-1$ are there own inverses.

– Now group all others with their inverses,

$$(n-1)! \equiv (2 \times 2^{-1})(3 \times 3^{-1}) \cdots (n)(n-1)$$
$$(n-1)! \equiv (1)(1) \cdots (n)(n-1)$$
$$(n-1)! \equiv (n)(n-1)$$
$$(n-1)! \equiv -1 \bmod n$$

- **Show that if n is not prime, then $(n-1)! \neq -1 \bmod n$.**

  – Proof by counterexample for $n = 4$ because 4 is not prime.

$$(4-1)! = 6$$
$$-1 \bmod 4 = -1$$
$$6 \neq -1$$
$$(n-1)! \neq -1 \bmod n \,\forall\, n$$

- **The above process can be used as a primality test instead of Fermat's Little theorem as it is an if-and-only-if condition for primality. Why can't we immediately base a primality test on this rule?**

  – It's obviously possible *in principle* but why both *in practice*? Computing $(n-1)! \mod n$ for large n is computationally very expensive.
  – Why? It'll take *factorial time*, which is among the worst.
  – Especially when Fermat's Little Theorem gets us a much better time complexity, and even manual checking is more efficient than factorial time.

Part D

Problem 8

Part A

Make a table with three columns. The first column is all numbers from 0 to 36. The second is the residues of these numbers modulo 5; the third column is the residues modulo 7.

- See following table.

Part B

Consider two different prime numbers x and y. Show that the following is true: for every pair of numbers m and n so that: $0 \leq m < x$ and $0 \leq n < y$, there is a unique integer q, where $0 \leq q < xy$, so that:

Table 1: Problem 8, Part A

1 through 36	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

$$q \equiv m \bmod x$$

$$q \equiv n \bmod y$$

- I don't know.

Part C

The previous problem asks to go from q to (m, n). It is also possible to go the other way. In particular, show the following:

$$q = (m \cdot y \cdot (y^{-1} \bmod x) + n \cdot x \cdot (x^{-1} \bmod y)) \bmod xy$$

- I don't know.

Part D

What happens in the case of three primes x, y and z? Do the above properties still hold? If they do, how do they look like in this case?

- I don't know.

Problem 9

- **How did Mallory do this?**
  - Represent encrypted messages with C, original messages with M, and the multiple of two primes p and q with N. The exponent of M by e.
  - Mallory intercepted the C_1, C_2, and C_3 which belong to Bob, Charlie, and David respectively. She knows these equal the following:

$$C_i \equiv M^e \bmod N_i$$

  - Observe the following from the table,

$$C_1 = 153, C_2 = 196, C_3 = 27$$

  - Also from the table, observe the following,

$$e = 3$$

  - When Mallory looked up their public keys, she gained the following information:

$$N_1 = 155, N_2 = 203, N_3 = 117$$

– She was thus able to construct the following series of equations,

$$153 = M^3 \bmod 155$$
$$196 = M^3 \bmod 203$$
$$27 = M^3 \bmod 117$$

– Now Mallory uses the Chinese Remainder Theorem to compute:

$$C = M^3 \bmod 155 \times 203 \times 117$$

– This message will be smaller than $155 \times 203 \times 117$ because $e = 3$.

– Now the attacker can get the original message by taking the cube root.

- **What is the original message?**

42