Group members: Matt Demusz, Frank Porco, Joyce Wang

due 02/18/14

CS 344: DESIGN AND ANALYSIS OF COMPUTER ALGORITHMS SPRING 2014

Homework 1

Asymptotics and Number Theoretic Algorithms

Part A (20 points)

Problem 1: In each of the following situations indicate whether $f = O(g)$ or $f = \Omega(g)$ or $f = \Theta(g)$:

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

   $f(n) = \sqrt{2^{7n}} = 2^{7n/2}$ and $g(n) = 2nlg(7)$. $f(n)$ is an exponential function, and $g(n)$ is a linear function. Because an exponential grows faster than a linear function, $f(n) = \Omega(g)$.

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

   Since $g(n) = n!$, we can use Stirling's approximation: $ln(n!) = nln(n) - n + O(lnn)$. To get it in the same form as Stirling's we take ln of $f(n)$ and ln of $g(n)$ to be consistent so that we can compare the two functions later.

$$\begin{aligned} ln(f(n)) &= ln(2^{nln(n)}) \\ &= ln(2)nln(n) \end{aligned}$$

$$\begin{aligned} ln(g(n)) &= ln(n!) \\ &= nln(n) - n + O(ln(n)) \end{aligned}$$

   $ln(f(n))$ is $O(nlnn)$, which grows faster than $ln(g(n))$ because it has more terms appended to it in addition to $O(nlnn)$, so $f(n) = \Omega(g)$.

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

   Since $lg^* n = 1 + lg(lgn)$ if $n > 1$, $f(n) = lg(lg^* n) = lg(1 + lg(lgn))$ and $g(n) = lg^*(lgn) = 1 + lg(lg(lgn))$. Let $x = lg(lgn)$. Then, $f(n) = lg(1 + x)$ and $g(n) = 1 + lg(x)$. As the limit of x approaches infinity, $lg(1 + x) - lg(x) = 0$, so $lg(1 + x) - lg(x) = 0 = 1$.

$$\frac{f(n)}{g(n)} = \frac{lg(1+x)}{1+lg(x)} \leq lg2$$

$$\frac{g(n)}{f(n)} = \frac{1+lg(x)}{lg(1+x)} \leq 2$$

   for all $x \geq 1$ $f(n) = \Theta(g)$.

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

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

   To compare the two functions and to keep them consistent, we can take the lg of both of them. Therefore, we have $lg(f(n)) = lg(2^n) = nlg(2)$ and $lg(g(n)) = lg(n^{lgn}) = lgnlgn = lg^2 n$. The linear function, $f(n)$ grows faster, so $f(n) = \Omega(g)$.

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

   To compare the two funcitons and to keep them consistent, we can take the lg of both of them. Therefore, we have $lg(f(n)) = lg(2^{\sqrt{lgn}}) = \sqrt{lgn}lg(2)$ and $lg(g(n)) = lg(n(lgn)^3) = 3lg(n(lgn))$.

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

   To compare the two funcitons and to keep them consistent, we can take the lg of both of them. Therefore, we have $ln(f(n)) = ln(e^{cos(n)}) = cos(n)$ and $ln(g(n)) = ln(lgn)$. The cosine curve ranges from -1 to 1 and is infinitely repetitive.

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

   $f(n) = lg(n^2) = 2lgn$ and $g(n) = (lgn)^2 = (lgn)^2$. The exponential function grows faster, so $f(n) = O(g)$.

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

   $f(n) = \sqrt{4n^2 - 12n + 9} = (4n^2 - 12n + 9)^{1/2}$ Since the n^2 term dominates, we can drop all the other terms to n^2, giving us $(4n^2)^{1/2}$, or, simply, a linear function since $\sqrt{n^2} = n$. $g(n) = n^{3/2}$ is an exponential function, which grows faster than a linear function, so $f(n) = O(g)$.

10. $f(n) = \sum_{k=1}^{n} k, g(n) = (n+2)^2$ $f(n) = \sum_{k=1}^{n} k = \frac{n(n+1)}{2} = \frac{n^2+n}{2}$, and n^2 is the term that dominates, so $f(n)$ is order n^2. $g(n) = (n+2)^2 = n^2 + 4n + 4$. Similarly, n^2 is the term that dominates, so $g(n)$ is order n^2 as well. Therefore, $f(n) = \Theta(g)$.

*N*ote: The lg^*n function is the number of times the logarithm function must be iteratively applied before the result is less than or equal to 1, i.e., $lg^*n = 0$ if $n \leq 1$ and $lg^*n = 1 + lg^(lgn)$ if $n > 1$. For instance, $lg^*4 = 2$, $lg^*16 = 3$, $lg^*65536 = 4$, etc.

Algorithm 1: `Number_Theoretic_Algorithm` (integer n)

```
1  N ← Random_Sample(0, 2^n − 1);
2  if N is even then
3      N ← N + 1;
4  m ← N mod n;
5  for j ← 0 to m do
6      if Greatest_Common_Divisor( j, N) ≠ 1 then
7          return FALSE;
8      Compute x, z so that N − 1 = 2^z · x and x is odd;
9      y_0 ← (N − 1 − j)^x mod N;
10     for i ← 1 to m do
11         y_i ← y_{i−1}^2 mod N;
12         y_i ← y_i + y_{i−1} mod N;
13     if Low_Error_Primality_Test(y_m) == FALSE then
14         return FALSE;
15 return TRUE;
```

Problem 2: 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.

Do not just present the final result. For each line of pseudo-code indicate the best running time for the corresponding operation given current knowledge from lectures and recitations and then

show how the overall running time emerges.

1. $O(n)$: from notes
2. $O(1)$: check the last bit
3. $O(n)$: worst case for addition is having to flip 1s through the entire number
4. $O(n^2)$
5. $O(n)$: for loop runs linearly; everything inside gets multiplied by n
6. $O(n^3)$: Euclid's algorithm from notes
7. $O(1)$: returning a boolean doesn't depend on input size
8. $O(n)$
9. $O(n^2)$: problem specifies multiplication (the "power x") as $O(n^2)$; number in front is dropped
10. $O(n)$
11. $O(n^2)$: problem specifies multiplication (the "power 2") as $O(n^2)$
12. $O(n)$:a mod b
13. $O(nlog^3n)$: primality test
14. $O(1)$: returning a boolean doesn't depend on input size
15. $O(1)$: returning a boolean doesn't depend on input size

Part B (30 points)

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.

  For a binary tree, the number of nodes is $2^{h+1}-1$, so the number of nodes in a m-ary tree is $m^{h+1}-1$. To find the lower bound for the height of the tree, we can solve for h, where N is the number of nodes, m is the number of children each node has, and h is the height:

$$\begin{aligned} N &= m^{h+1}-1 \\ N+1 &= m^{h+1} \\ ln(N+1) &= ln(m^{h+1}) \\ ln(N+1) &= (h+1)(lnm) \\ \frac{ln(N+1)}{lnm} &= h+1 \\ ln(N+1-m) &= h+1 \\ h &= ln(N+1-m)-1 \end{aligned}$$

  Therefore, the lower bound forthe height of the tree is $h = ln(N+1-m)-1$.

- 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?

  The asympototic behavior is the $O(h)$. $ln(N+1-m)-1$ is $O(ln(N-m))$, dropping all constants, where N is the number of nodes and m is the number of children each node has. We made the assumption that the tree was perfect.

- 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?

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

Since we're computing the result according to the rule above, we need to calculate the run time of the rule. We are given that x is in the order of 2^m (m+1 bits) and y is in the order of 2^n (n+1 bits) as our input. The recursion ends after $n+1$ calls since at each recursive step, y is halved, so the number of bits for y is decreased by 1. Then, in every call, we have a right shift, finding the last bit, a left shift, and sometimes addition, which gives us $O(n)$ bit operations. Therefore, the total running time is $(n+1)*(m+1)$, which gives us $O(nm)$ as the run time.

Problem 4:

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

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

- 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.

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.

- 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.

Part C (30 points)

Problem 6: Our goal is to assign $(2^{31}-1)$ integers into 256 slots $\{0, 1, \ldots, 255\}$ and achieving the properties of universal hashing. Notice that 256 is not a prime number. Assume the following approach towards this objective.

Consider a 32-bit integer y and the following set $\mathcal{M}$ of hash functions, so that:

- each $m \in \mathcal{M}$ corresponds to a unique 8×32 matrix M having elements only 0 or 1.
- and $m(y) = M \cdot y$ mod 2, i.e., multiply the matrix with the 32-bit vector corresponding to number y and then apply the modulo operation on each bit of the resulting vector.

Such functions map a 32-bit vector into an 8-bit vector, which can then be interpreted as an 8-bit number that indicates the original number's slot in the range $0 \sim 255$. Notice that there are $2^{8\times 32} = 2^{256}$ different $\{0, 1\}$ matrices in the set $\mathcal{H}$.

Provide the following:

- A proof that the hash function family $\mathcal{M}$ is universal.
- A comparison to the universal hash function family described in DPV chapter 1.5.2. How many random bits are needed here?

Problem 7: Answer the following sequence of problems:

- 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 module n?

  For a number (x) to be its own inverse, the following must be true:

  $$x^2 \equiv 1 \pmod{n}$$

  So $x^2 \equiv 1 \pmod{n}$ –> $x^2 - 1 \equiv 0 \pmod{n}$ –> $(x-1)(x+1) \equiv 0 \pmod{n}$
  Since n is prime, we must have either $(x-1) \equiv 0 \pmod{n}$ or $(x+1) \equiv 0 \pmod{n}$. So the only numbers that will satisfy these two equations for x are 1 ($((1)-1) \equiv 0 \pmod{n}$ and n-1 ($((n-1)+1) \equiv 0 \pmod{n}$. This means that the only numbers that are greater than or equal to 1 and less than n that are their own inverse are 1 and n-1.

- Show that $(n-1)! \equiv -1 \mod n$) for prime n. [Hint: Compute the value of $(n-1) \mod n$ by considering how it arises by pairing two numbers smaller than n that are multiplicative inverses modulo n.]

  Since n is prime, each number $a = 1, \ldots, n-1$ has an inverse $\pmod{n}$. We can pair up the numbers in between 2 and n-2 so that a number in the pair is the modular inverse of the other. 1 and n-1 are their own modular inverse, as proven above. Multiplying these pairs, 1 , and n-1 is the same as (n-1)!. Since each of these pairs are a number grouped with their modular inverse, they each only contribute a value of 1, since for each pair:

  $$pair \equiv 1 \pmod{n}$$

  So that leaves the 1 and the n-1, meaning that:

  $$(n-1)! \equiv 1 * (n-1) \equiv (n-1) \equiv -1 \pmod{n} \quad => \quad n \equiv 0 \pmod{n}$$

  Showing that the original equation is true for a prime n.

- Show that if n is not prime, then $(n-1)! \neq -1 \mod n$. [Hint: What does it mean that n is not prime in terms of the numbers $1 \leq x < n$?]

  Since n is not a prime number, it has a divisor (a) such that 1 < a < n. So a must be one of the numbers that makes up (n-1)!. This tells us that a divides (n-1)! Looking at the origanal equation, $(n-1)! = -1 \pmod{n}$ –> $(n-1)! + 1 = 0 \pmod{n}$. This tells us that n divides ((n-1)! + 1), and since a is a divisor of n, a must also divide ((n-1)! + 1). But if that were the case, then that would mean that a divides both (n-1)! and ((n-1)! + 1). This would mean a would have to equal 1, which contradicts 1 < a < n.
  Therefore, if n is not prime, then $(n-1)! \neq -1 \pmod{n}$.

- 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? [Tip: Even if you are not able to answer the previous two questions, you should be able to argue about this question.]

Part D (30 points)

Problem 8:
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.

#	%5	%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

#	%5	%7
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
34	4	3

#	%5	%7
35	0	4
36	1	5
37	2	6
38	3	0
39	4	1
30	0	2
31	1	3
32	2	4
33	3	5
34	4	6
35	0	0
36	1	1

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:

$$q \equiv m \pmod{x}$$
$$q \equiv n \pmod{y}$$

Suppose there exists q_1, q_2 and $q_1 ! = q_2$ such that $q_1 = (m, n)$ and $q_2 = (m, n)$

1. $q_1 = k_1 x + m$, $q_2 = l_1 x + m$
2. $q_1 = k_2 y + n$, $q_2 = l_2 y + n$
3. $q_1 - q_2 = (k_1 x + m) - (l_1 x + m) = (k_1 - l_1)$ from 1
4. $q_1 - q_2 = (k_2 - l_2)y$ from 2
5. $(k_1 - l_1)x = (k_2 - l_2)y$
6. $x = \dfrac{(k_2 - l_2)}{(k_1 - l_1)} y$
7. $\dfrac{(k_2 - l_2)}{(k_1 - l_1)} = 1$ because prime number x can't be the product of two numbers
8. $x = y$ contradicts hypothesis
9. x and y have to be equal

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:

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

[Hint: Ensure that if the above is true then the expressions in section B are also true. Consider the values of the following terms: $c_x = y \cdot (y^{-1} \bmod x)$ and $c_y = x \cdot (x^{-1} \bmod y)$ and their values $\bmod x$ and $\bmod y$.]

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?

Problem 9: There is an office, in which every member uses RSA to conduct secure communication with others. For example, when Alice wants to send Bob a message, she will first use Bob's public key to encrypt the message, then sends it to Bob, Bob then uses his private key to decrypt the message. In order to make things easy, the office maintains a directory listing every member's public key, everybody in the office has access to it. A public key looks like (N, e) (as defined in DPV-1.4.2).
One day, Alice sent a message (smaller than any N) to Bob, Charlie, and David via the RSA based secure communication, but the encrypted messages were intercepted by the webmaster Mallory. Mallory used some network tricks and found out who the receivers were, then he checked their public keys in the directory. This is what Mallory got:

Receiver	Encrypted message	Public key
Bob	674	(3337, 3)
Charlie	36	(187, 3)
David	948	(1219, 3)

Finally, Mallory recovered the original message. How did Mallory do this? What is the original message?

How Malory did it: Malory could have used a combination of the chinese remainder theorem and Gauss's algorithm. Malory can use Gauss's algorithm to find the message (x) using the enrypted messges (c_1,c_2,c_3), and the N values (n_1,n_2,n_3) respectivley.

$$x \equiv c_1 \pmod{n_1}$$
$$x \equiv c_2 \pmod{n_2}$$
$$x \equiv c_3 \pmod{n_3}$$

From the chinese remainder theorem, we know that the above congruences have a solution (x) which is unique modulo n_1,n_2,n_3. Now we can use Guass's algorithm to solve for x.

$n_1 = 3337$ $c_1 = 674$
$n_2 = 187$ $c_2 = 36$ $e = 3$
$n_3 = 1219$ $c_3 = 948$
$N = n_1 n_2 n_3 = 3337 * 187 * 1219 = 760679161$
$N_1 = N/n_1 = 760679161/3337 = 227953$
$N_2 = N/n_2 = 760679161/187 = 4067803$
$N_3 = N/n_3 = 760679161/1219 = 624019$
$d_1 = N_1^{-1} \pmod{n_1} = 227953^{-1} \pmod{3337} = 251$
$d_2 = N_2^{-1} \pmod{n_2} = 4067803^{-1} \pmod{187} = 70$
$d_3 = N_3^{-1} \pmod{n_3} = 624019^{-1} \pmod{1219} = 671$
$x^3 = c_1 N_1 d_1 + c_2 N_2 d_2 + c_3 N_3 d_3 \pmod{N}$
$x^3 = ((674 * 227953 * 251) + (36 * 4067803 * 70) + (948 * 624019 * 671)) \pmod{760679161}$
$x^3 = 74088$
$x = 42$

Malory, knowing the encrypted messages and the respective public keys could have applied Gauss's theorem like this and have come up with the original message (x), which is 42.