CS 344 Assignment 1

Craig Perkins, Alex Tang, Steve Grzenda

Due February 18, 2014

Problem 1

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

   $f(n) = \sqrt{2^{7n}} = (2^{7n})^{\frac{1}{2}} = 2^{\frac{7}{2}n} = (2^{\frac{7}{2}})^n = a^n \implies f(n)$ is exponential
   $g(n) = \lg(7^{2n}) = 2n\lg(7) = (2\lg 7)n = cn \implies g(n)$ is linear
   Since f(n) is exponential and g(n) is linear $\implies \boxed{f(n) = \Omega(g(n))}$

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

   $\lg(f(n)) = \lg 2^{n \ln n} = n \ln n \implies \lg(f(n))$ is linearithmic
   $\lg(g(n)) = \lg(n!) \approx n \lg n \implies \lg(g(n))$ is linearithmic. Note that the approximation of $\lg(n!)$ is done using Stirling's approximation.
   Since $\lg(f(n))$ and $\lg(g(n))$ are both linearithmic $\implies \boxed{f(n) = \Theta(g(n))}$

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

   $f(n) = \lg(\lg^* n) = \lg(1 + \lg(\lg n))$. In the preceding function we can ignore the 1 because it is constant and does not grow with n, so we can say that $f(n) = O(\lg(\lg(\lg n)))$.
   $g(n) = \lg^*(\lg n) = 1 + \lg(\lg(\lg(n)))$. In the preceding function we can ignore the 1 because it is constant and does not grow with n, so we can say that $g(n) = O(\lg(\lg(\lg n)))$.
   Since f(n) and g(n) grow at the same rate as a function of the input $\implies \boxed{f(n) = \Theta(g(n))}$

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

   $f(n) = \frac{\lg n^2}{n} = \frac{2\lg(n)}{n}$
   Since n dominates $\lg n$ the preceding function will approach 0 as the input approaches ∞. (Possibly the only program in the world that gets faster as the input gets larger)
   $g(n) = \lg^*(n)$. g(n) is a very slow growing function, but still grows as a function of its input unlike g(n).
   Since f(n) takes almost no time to execute for large n and g(n) grows with n, g(n) will dominate f(n) $\implies \boxed{f(n) = O(g(n))}$

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

   $f(n) = 2^n \implies \lg(f(n)) = \lg(2^n) = n \cdot \lg(2)$ is linear with respect to the input.
   $g(n) = n^{\lg n} \implies \lg(g(n)) = \lg(n^{\lg n}) = (\lg n)^2$ is polylogarithmic with respect to the input.
   Since polynomial (linear is a polynomial) functions dominate polylogarithmic functions $\implies \boxed{f(n) = \Omega(g(n))}$

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

   $\lg(f(n)) = \lg(2^{\sqrt{\lg n}}) = \sqrt{\lg n} \cdot \lg 2 = \sqrt{\lg n}$ is polylogarithmic with respect to input
   $\lg(g(n)) = \lg(n(\lg n)^3) = \lg n + 3\lg(\lg n)$ is logarithmic with respect to input because $\lg(n)$ dominates the expression.
   Since $\lg(f(n))$ and $\lg(g(n))$ are both polylogarithmic, but the power of $\lg(g(n))$ is greater than that of $\lg(f(n)) \implies \boxed{f(n) = O(g(n))}$

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

   This problem is a bit confusing because you need to consider the range of the $\cos n$ function. Since the cos function has a range of $-1 \leq \cos n \leq 1$, that implies that the range of $e^{\cos n}$ is $\frac{1}{e} \leq e^{\cos n} \leq e$ and that f(n) is actually constant time with respect to the input.
   $g(n) = \lg n \implies g(n)$ is logarithmic
   Since logarithmic dominates constant time $\implies \boxed{f(n) = O(g(n))}$

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

   $f(n) = \lg(n^2) = 2\lg(n)$ is logarithmic with respect to input.
   $g(n) = (\lg(n))^2$ is polylogarithmic with respect to input.
   Since g(n) is polylogarithmic and the power is greater than 1, g(n) dominates f(n) $\implies \boxed{f(n) = O(g(n))}$

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

   $f(n) = \sqrt{4n^2 - 12n + 9} = \sqrt{(2n-3)^2} = 2n - 3 \implies f(n)$ is linear
   $g(n) = n^{\frac{3}{2}} \implies g(n)$ is more than linear, but less than quadratic.
   Since f(n) and g(n) are both polynomial, but g(n) has a higher power than f(n) $\implies \boxed{f(n) = O(g(n))}$

10. $f(n) = \sum\limits_{k=1}^{n} k$, $g(n) = (n+2)^2$
    $f(n) = \sum\limits_{k=1}^{n} k = \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n \implies f(n)$ is quadratic
    $g(n) = (n+2)^2 = n^2 + 4n + 4 \implies g(n)$ is quadratic
    Since f(n) is quadratic and g(n) is also quadratic $\implies \boxed{f(n) = \Theta(g(n))}$

Problem 2

```
1. Random Sample: $O(n)$
2. Even Check: $O(1)$
3.     Add 1: $O(n)$
4. Mod: $O(n^2)$
5. For loop: $O(n)$
6.     GCD $O(n^3)$
7.         return False
8.     Compute x,z $O(n)$
9.     Modular Exponentiation $O(n^3)$
10.     For loop $O(n)$
11.         Modular Exponentiation $O(n^3)$
```

```
12.             Mod $O(n^2)$
13.       Low Error Primality Test $O(n^3)$
14.             return FALSE
15. return TRUE
```

Total: $O(n^5)$. This was calculated by multiplying the big O times of the outer loop, the inner loop and the dominating modular exponentiation, i.e. $O(n) \cdot O(n) \cdot O(n^3)$.

Problem 3

- Let a tree with a single node have height 0 and define this row as row 0. This means that at row k, there will be at most m^k nodes. Therefore, a tree of height h will have at most N nodes where

$$N = \sum_{k=0}^{h} m^k = \frac{m^{h+1} - 1}{m - 1}.$$

  Solving for h gives us the lower bound for the height of the tree:

$$\boxed{h = \lceil \log_m [N(m-1)+1] \rceil - 1}.$$

- $h_m = \lceil \log_m [N(m-1)+1] \rceil - 1$ and $h_{m'} = \lceil \log_{m'} [N(m'-1)+1] \rceil - 1$. h_m and $h_{m'}$ will have the same asymptotic running time, i.e. $h_m = \Theta(h_{m'})$, because they will both be logarithmic as a function of input and the change of base formula tells us that logarithms of different bases only differ by a constant. The only exception to this rule is if it is a 1-ary tree, in which case the height grows linear and the heights will have different rates of growth.

- Using this definition of modular exponentiation, there will be $O(\log y) = O(n)$ recursive calls. Each call involves the multiplication of two m-bit numbers, which will take $O(m^2)$ time with the result being at most $2m$ bits. After the multiplication, the $2m$ bit result modulo N, which is an o-bit number, will take $O(mo + \max(o, m)) = O(mo)$ time. In total, this algorithm will have a running time of $\boxed{O(n \cdot (m^2 + mo))}$.

Problem 4

- $2^{902} = (2^3)^{300} \cdot 2^2 \mod 7 \equiv 1^{300} \cdot 2^2 \mod 7 \equiv \boxed{4 \mod 7}$

- $11^{-1} \mod 120 = 11$
  $13^{-1} \mod 45 = 7$
  $35^{-1} \mod 77$ does not exist because $gcd(35, 77) > 1$
  $9^{-1} \mod 11 = 5$
  $11^{-1} \mod 1111$ does not exist because $gcd(11, 1111) > 1$

  To find the modulo multiplicative inverse in the previous problems we used the extended Euclid algorithm. The algorithm can be demonstrated with $9^{-1} \mod 11$ by first finding the $gcd(9, 11)$ using the Euclid algorithm:

  $$\text{step } 0 : 11 = 1(9) + 2$$
  $$\text{step } 1 : 9 = 4(2) + 1$$
  $$\text{step } 2 : 1 = 1(1) + 0$$

  So $gcd(9, 11) = 1$. This means that there is a linear combination of 9 and 11 equaling 1. Working backwards using the extended Euclid algorithm:

  $$1 = 9 - 4(2)$$
  $$1 = 9 - 4(11 - 1(9)) = 5(9) - 4(11)$$

  So $9^{-1} \mod 11 = 5$.

- For a number x, if $\forall y \epsilon [1, x-1] : gcd(x, y) = 1$, then x must be prime. The extended Euclid algorithm gives us the multiplicative inverse of a mod n if and only if $gcd(a, n) = 1$. This means that the only numbers in the set $S = \{0, 1, \ldots, x^m - 1\}$ that do not have multiplicative inverses modulo x^m are the numbers in the set

  $$U = \{a \mid a \in S, gcd(a, x^m) = x\}$$

  where
  $$|U| = \frac{x^m}{x}.$$

  The $gcd(a, x^m) = x$ because x is prime. Therefore, there are $x^m - \frac{x^m}{x} = x^{m-1}(x-1)$ numbers in S that have multiplicative inverses modulo x^m. Since finding the multiplicative inverse using the extended Euclid algorithm takes $O(n^3)$, the total running time takes

  $$\boxed{O(n^3) \cdot [x^{m-1}(x-1)]}.$$

Problem 5

- True. Let $g = gcd(x, y)$. Then, $x = a_x g$ and $y = a_y g$ such that a_x and a_y do not share any common factors, i.e. a_x and a_y are coprime. Rewrite
$$(5x + 3y, 3x + 2y)$$
as
$$(5a_x g + 3a_y g, 3a_x g + 2a_y g).$$
Factoring out g gives
$$(g \cdot (5a_x + 3a_y), g \cdot (3a_x + 2a_y)).$$
Here, you can see that (x, y) and $(5x + 3y, 3x + 2y)$ have a common divisor, g. The greatest common divisor of (x, y) and $(5x+3y, 3x+2y)$ cannot be $> g$ because of the following: if a_y shared a common factor with 5, then factor out 5 from $5a_x + 3a_y$. However, you will not be able to factor out 5 from $3a_x + 2a_y$, because 5 and 3 are coprime as are a_x and a_y. This same reasoning can be applied to a_x. Therefore, the greatest common divisor of (x, y) and $(5x + 3y, 3x + 2y)$ must be g.

- The proof that the numbers of the set S are relatively prime is very similar to that of the Prime Number Theorem. We can show that the first few elements of the set {2,3,7,42,...} are coprime and that the n-th element is also coprime because:
$$\begin{gathered} s_n \equiv 1 \mod s_0 \\ s_n \equiv 1 \mod s_1 \\ \vdots \\ s_n \equiv 1 \mod s_{n-1} \end{gathered}$$
Since all elements are coprime to those in the set before it there will be no two elements of the set that have a gcd other than 1.

Problem 6

- Take two vectors x and y such that $x \neq y$. Suppose x and y differ in the ith spot such that $x_i = 0$ and $y_i = 1$. Choose all of M except the ith column. Since $x_i = 0$, $m(x)$ is fixed over any choice of the ith column. There are 2^8 possible bit combinations for the ith column, each of which gives a different value for $m(y)$. So the chance that $m(x) = m(y)$ is $\frac{1}{2^8}$, proving that this hash function family is universal.

- The universal hash function in 1.5.2 deals with hashing IP addresses (32-bit) to an array with 256 slots, so it is the same problem we are trying to solve in problem 6. The way they accomplish the hashing is by breaking the 32-bit word into 8-bit chunks. Each 8-bit chunk is multiplied by another 8-bit random number and those products are summed for all 4 chunks. After the sum is computed the new number is then taken modulo 256. In total the hashing takes a total of $8 \cdot 4 = 32$ random bits opposed to our $8 \cdot 32 = 256$ random bits. Both hashing functions are universal, but the matrix hash described in problem 6 takes more random bits.

Problem 7

- The numbers that are modular multiplicatives of themselves module n are the numbers 1 and $n-1$. 1 is the modulo multiplicative inverse of itself because $1 \cdot 1 \equiv 1 \mod n$. $n-1$ is the modular multiplicative inverse of itself because $(n-1)(n-1) = n^2 - 2n + 1 \equiv 1 \mod n$.

- To show that $(n-1)! \equiv -1 \mod n$ and n must be prime, we must first observe that $n - 1 \equiv -1 \mod n$. Using this we can see that $(n-1)! \equiv -(n-2)! \mod n$. From the numbers 2 to $n-2$ we can also observe that their inverses will also be in the range from 2 to $n-2$ and be unique. We can be assured that all of 1 to $n-1$ inverse to exist because if a number a is picked from $(1, n-1)$, then $gcd(a, n)$ will be 1 because n is prime and $a < n$. We can also be assured they're unique because assume that a and b are picked from $(1, n-1)$. Then $a \cdot x \equiv 1 \mod n$ and $b \cdot x \equiv 1 \mod n$ and $a \cdot x \equiv b \cdot x \mod n$. Dividing the equivalence by x we obtain $a \equiv b \mod n$ and since $1 \leq a, b < n$, we can conclude that $a = b$ so that each value is unique (one-to-one mapping). Furthermore the inverse of a number's inverse is itself, meaning that inverses come in pairs (i.e. 2 and 6 $\mod 11$). From that fact we can show that $(n-2)(n-1)\cdots 3\cdot 2 \equiv 1 \mod n$ and from part one we know that $1 \equiv 1 \mod n$. Altogether it can be shown that $(n-1)! \equiv -1 \mod n$.

- If n is not prime, then the fundamental theorem of arithmetic says that $\exists a$ such that $1 < a < n$, $gcd(a, n) > 1$. From that fact we can conclude that $(n-1)! \not\equiv -1 \mod n$ because $gcd((n-1)!, n) > 1$

- One main reason for not implementing this primality test is because it does not scale well with the size of the input. The algorithm might be able to determine with certainty the difference between and prime and not prime, but will take ages to compute. In the study of algorithms the second fastest growing algorithm besides n^n is $n!$, not to mention that a modulus is also being applied.

Problem 8

- See Table 1.

- To start the proof assume that $\exists(p, q)$ such that $q \equiv m \mod x$, $q \equiv n \mod y$ and $p \equiv m \mod x$, $p \equiv n \mod y$ and $p \neq q$, $0 \leq p, q \leq xy$, x and y are prime. That means that I can write $p = m + i \cdot x$ and $q = m + j \cdot x$ where $i \neq j$. Substituting in we obtain $m + i \cdot x \equiv n \mod y$ and $m + j \cdot x \equiv n \mod y$. We know from our assumptions that $0 \leq m + i \cdot x, m + j \cdot x < xy$. We can also note that $i, j < y$. By subtracting from both sides we get $ix \equiv n - m \mod y$ and $jx \equiv n - m \mod y$. Since $i, j < y$ we can conclude that $i = j$ which contradicts our assumption. This means that $p = q$ and that p is unique to a pair of (m, n). This problem shows the basis of the Chinese Remainder Theorem.

- This is the Chinese Remainder Theorem for $n = 2$. Let

$$q = (m \cdot c_x + n \cdot c_y) \mod xy$$

where $c_x = y(y^{-1} \mod x)$ and $c_y = x(x^{-1} \mod y)$. Take $q \pmod x$. Then

$$q = (m \cdot c_x + n \cdot c_y) \mod xy \pmod x$$

Because $c_y \pmod x = 0$,

$$q = mc_x \pmod x$$

Table 1: Problem 8A

n	mod 5	mod 7	n	mod 5	mod 7
0	0	0	19	4	5
1	1	1	20	0	6
2	2	2	21	1	0
3	3	3	22	2	1
4	4	4	23	3	2
5	0	5	24	4	3
6	1	6	25	0	4
7	2	0	26	1	5
8	3	1	27	2	6
9	4	2	28	3	0
10	0	3	29	4	1
11	1	4	30	0	2
12	2	5	31	1	3
13	3	6	32	2	4
14	4	0	33	3	5
15	0	1	34	4	6
16	1	2	35	0	0
17	2	3	36	1	1
18	3	4	37	2	2

Because $c_x \pmod{x} = 1$,

$$q = m \pmod{x}$$

Taking $q \pmod{y}$, you get $q = n \pmod{y}$ by the same reasoning. So q is a solution to both $q = m \pmod{x}$ and $q = n \pmod{y}$.

- This will be the case of the Chinese Remainder Theorem for $n = 3$. The above equations and properties will still hold for 3 primes. That is that $c_x \equiv 1 \mod x$, $c_x \equiv 0 \mod y$ and $c_x \equiv 0 \mod z$. The corresponding q can be computed as follows and is useful for number 9.

$$q = m\cdot(yz\cdot((yz)^{-1} \mod x)) + n\cdot(xz\cdot((xz)^{-1} \mod y)) + o\cdot(xy\cdot((xy)^{-1} \mod z)) \mod xyz$$

Problem 9

Mallory used the function $b = a^e \mod N$ to encrypt her messages where a is the plaintext and b is the ciphertext. Notice that $e = 3$ for every encryption function. Let $q = a^3$ so $q = b \mod N$. Using the three intercepted messages, you can uncover the following equations:

$$q \equiv m \mod x$$

$$q \equiv n \mod y$$

$$q \equiv o \mod z$$

where m, n, o are the encrypted messages and $m = 674, n = 36, o = 948, x = 3337, y = 187$, and $z = 1219$. Since x, y, z are coprime, use the Chinese Remainder Theorem to solve for q which says:

$$q = m \cdot c_x + n \cdot c_y + o \cdot c_z \mod xyz$$

where:

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

Substituting gives:

$$q = 674 \cdot c_x + 36 \cdot c_y + 948 \cdot c_z \mod (3337 \cdot 187 \cdot 1219)$$

$$c_x = 187 \cdot 1219 \cdot (189 \cdot 1219^{-1} \mod 3337)$$
$$c_y = 3337 \cdot 1219 \cdot (3337 \cdot 1219^{-1} \mod 187)$$
$$c_z = 187 \cdot 3337 \cdot (187 \cdot 3337^{-1} \mod 1219)$$

So $q = 74088$ and the plaintext $a = \sqrt[3]{q} = \boxed{42}$.