Homework 2

Trixie Roque

03/02/15

1. (a)$f = \Theta(g)$

   $(b) f = O(g)$

   $(c) f = \Theta(g)$

   $(d) f = \Theta(g)$

   $(e) f = \Theta(g)$

   $(f) f = \Theta(g)$

   $(g) f = O(g)$

   $(h) f = \Omega(g)$

   $(i) f = O(g)$

   $(j) f = O(g)$

   $(k) f = \Omega(g)$

   $(l) f = O(g)$

   $(m) f = O(g)$

   $(n) f = \Theta(g)$

   $(o) f = \Omega(g)$

   $(p) f = O(g)$

   $(q) f = \Theta(g)$

2. (a)

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} (a)(e) + (b)(g) & (a)(f) + (b)(h) \\ (c)(e) + (d)(g) & (c)(f) + (d)(h) \end{pmatrix} = \begin{pmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{pmatrix}$$

   (b) It takes f = O ($\log(n)$) to compute X^n since we can just use the method of repeatedly squaring X^1 and finding powers of 2 such that when multiplied together, the exponents equal n. For example, consider X^8. Then $X^8 = X^{1*2*2*2*2} = X^{2*2*2} = X^{4*2} = X^8$. Then we can see that it is faster to just double the exponents each time (which is the same as multiplying a matrix by itself) in order to reach a power of 8 instead of multiplying the matrix X^1 8 times. Using the repeated squaring method for X^8 took only 3 multiplications compared to 8 which is $\log_2(8)$ times faster. Therefore, for X^n, it suffices to have $\log_2(n)$ matrix multiplications.

3. x = number of bits

   $\frac{x \log_2(2)}{\log_2(10)} = \frac{x}{0.301} = x * 3.3 < 4 * x$

so a decimal with x digits will have, at most, 4 times as many digits when converted to binary. The ratio of digits between binary and decimal numbers is $\frac{\log_2(2)}{\log_2(10)}$ or about 3.3

4. We first show the upper bound: n! $= n^n$

   $\log(n!) = \log(1) + \log(2) + \log(3) + ... + \log(n) \leq \log(n) + \log(n) + \log(n) + ... + \log(n)$

   We replaced $\log(1)$+...+$\log(n{-}1)$ with $\log(n)$ which guarantees that $\log(n)$+...+$\log(n)$ is greater than $\log(1)$+...+$\log(n)$. Since the number of $\log(n)$'s is n, the upper bound is O(n$\log(n)$).

   We then show the lower bound using the same method: $n! = \frac{n}{2}^{\frac{n}{2}}$

   $\log(n!) = \log(1) + \log(2) + ... + \log(\frac{n}{2}) + ... + \log(n) \geq \log(\frac{n}{2}) + ... + \log(n) \geq \log(\frac{n}{2}) + ... + \log(\frac{n}{2})$

   Here we truncate the first half of $\log(1) + ... + \log(n)$ which guarantees that $\log(\frac{n}{2}) + ... + \log(n)$ is less than $\log(1) + ... + \log(n)$. Then since there are only half as many $\log(\frac{n}{2})$, the number of $\log(\frac{n}{2})$ we have is $\frac{n}{2}$. Therefore, the lower bound is $\frac{n}{2}\log(\frac{n}{2})$ or $O(n\log(n))$

5. Yes (ran in Python)

   (4^{1536} - 9^{4824}) mod 35 = [(4^{1024}*4^{512}) - (9^{4096} * 9^{512} * 9^{128} * 9^{64} *9^{16} * 9^8)]mod 35 = 0

6. Yes (ran in Python)

   (5^{30000} - 6^{123456}) mod 31 = 0

7. Consider b = 15. Then $a^{15} = a^7 * a^7 * a^1$. Then this method only uses 3 multiplications where the exponents = 7 + 7 + 1 = 15. Compared to the repeated squaring method which uses 4 multiplications, $a^1 * a^2 * a^4 * a^8$, this other method finds the result with less calculations.

8. 2^{125} mod 127 = (2^{64} * 2^{32} * 2^{16} * 2^8 * 2^2 * 2^2 * 2^1) mod 127

   2^{125} mod 127 = ((((((2^{64} * 2^{32}) mod 127 * 2^{16}) mod 127 * 2^8) mod 127 * 2^2) mod 127 * 2^2) mod 127 * 2^1) mod 127

   2^{125} mod 127 = 64 (method used: plugged into Python)

9. See Github repo: Homework 2 "lcm.py"

10. Idea taken from Wikipedia on Wilson's theorem:

    Since Wilson's theorem uses factorials, the bigger n gets, the more complex the computation will be thus the computation will have a longer running time.

11. See Github repo: Homework 2 "exponential_mod.py"