CMSI 282 - Homework 2

Janine Leano

March 1st 2015

1. (a.) f = Θ(g)
   (b.) f = O(g)
   (c.) f = Θ(g)
   (d.) f = Θ(g)
   (e.) f = Θ(g)
   (f.) f = Θ(g)
   (g.) f = Ω(g)
   (h.) f = Ω(g)
   (i.) f = Ω(g)
   (j.) f = Ω(g)
   (k.) f = Ω(g)
   (l.) f = O(g)
   (m.) f = O(g)
   (n.) f = Θ(g)
   (o.) f = Ω(g)
   (p.) f = O(g)
   (q.) f = Θ(g)

2. (a.)
$$\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} = \begin{bmatrix} x_{11} \times x_{11} + x_{12} \times x_{21} & x_{11} \times x_{12} + x_{12} \times x_{22} \\ x_{21} \times x_{11} + x_{22} \times x_{21} & x_{21} \times x_{12} + x_{22} \times x_{22} \end{bmatrix}$$

   (b.) In order to get X^8,
   Start with $X^2 = X^2 \times X^2$
   $X^4 = X^2 \times X^2$
   $X^8 = X^4 \times X^4$
   n is an exponential of 2
   In the general case X^n where $n = 2^k$, at every iteration, n is doubled from before. The running time is O(log(n)) since it takes $k = \log_2(n)$ matrix multiplications in order to compute X^n.

3. N = number given
   d = number of digits in N
   In decimal: $10^{d-1} = N$
   $d1 = \log_{10}(N) + 1$
   In binary: $d2 = \log_2(N) + 1$
   $\log_2(N) = \frac{\log_{10}(N)}{\log_{10}(2)} \leq 4$

4. Upper bound:
   $n! = 1 \times 2 \times 3 \times \ldots n$
   $n^n = n \times n \times n \times \ldots n$
   with both of equal n lengths. $n! < n^n$
   So $n! = O(n^n)$
   Lower bound:
   $n! = 1 \times 2 \times 3 \times \ldots n$

$(\frac{n}{2})^{\frac{n}{2}} =$
with both of equal n lengths. $n! > (\frac{n}{2})^{\frac{n}{2}}$
$\log(n!) > \log(\frac{n}{2})^{\frac{n}{2}}$
$\log(n!) > \frac{1}{2} \times n \times \log(\frac{1}{2} \times n)$
$\frac{1}{2}s$ are constants so
$\log(n!) > n\log(n)$

Therefore, $\log(n!) = \Theta(n\log(n))$

5. $4^{1536} \equiv 9^{4824} \mod (35)$, so yes

6. According to Fermat's Theorem:
$5^{30000} \equiv 1 \mod (31) \equiv 125 \mod (31) \equiv 6^{123456}$, so yes

7. Given $b = 15$. Repeated squaring yields $2^{15} = 2^8 \times 2^4 \times 2^2 \times 2^1$ (4 multiplications)
Repeated x^5-ing yields $2^{15} = 2^5 \times 2^5 \times 2^5$ (3 multiplications)

8. Using modular exponentiation, $2^{125} \mod (127) = 64$

9.
```
def lcm(x, y):
        return (x * y) / gcd(x, y)

def gcd(x, y):
        if (x == 0):
                return y
        return gcd((y % x), x)
```

Running time is polynomial.

10. We can't immediately base a primality test using Wilson's theorem because it is not efficient. It is harder to evaluate the larger the input. (source: Wikipedia article on Wilson's theorem)

11.
```
# algorithm for modular exponentiation credit:
# http://aditya.vaidya.info/blog/2014/06/27/modular-exponentiation-python/
def expmod(a, b, c):
        x = 1
        while(b > 0):
                if (b & 1 == 1):
                        x = (x * a) % c
                a = (a * a) % c
                b >>= 1
        return x % c

def stacked_expmod(a, b, c, p):
        d = expmod(b, c, p - 1)
        return expmod(a, d, p)


print(str(expmod(2,125,127)))
```