Alex Valentino

Homework
CS 344

1. (a) $n - 100 = \Theta(n - 200)$ since their limit equals 1
   (b) $n^{1/2} = O(n^{2/3})$ since their limit equals 0 with $n^{2/3}$ in the denominator
   (c) $n+(\log(n))^2 = O(100n+\log(n))$ since the limit of $(n+(\log(n))^2)/(100n+\log(n))$ goes to 0
   (d) $n\log(n) = \Theta(10n\log(10n))$ since by log rules $10n\log(10n) = 10n(\log(10) + \log(n))$, thus their quotient is 10 in the limit.
   (e) $\log(2n) = \Theta(\log(3n))$ since by log rules $\log(cn) = \log(c)+\log(n)$, therefore taking their quotient and limit yields 1.
   (f) $10\log(n) = \Theta(\log(n^2))$ since $\log(n^2) = 2\log(n)$, thus giving their quotient and limit a constant value.
   (g) $n\log^2(n) = O(n^{1.01})$
   (h) $n(\log(n))^2 = O(n^2/\log(n))$
   (i) $(\log(n))^{10} = O(x^{0.1})$
   (j) $\frac{n}{\log(n)} = O((\log(n))^{\log(n)})$
   (k) $(\log(n))^3 = O(\sqrt{n})$
   (l) $n^{1/2} = O(5^{\log_2(n)})$
   (m) $n2^n = O(3^n)$
   (n) $2^n = \Theta(2^{n+1})$
   (o) $2^n = O(n!)$
   (p) $(\log(n))^{\log(n)} = O(2^{(\log_2(n))^2})$
   (q) $\sum_{i=1}^{n} i^k = \Theta(n^{k+1})$
2. (a) If $c < 1$ then $1+c+c^2+\cdots$ converges to a constant value, thus you can trivially bound it from above and below.
   (b) If $c = 1$ then $\sum_{i=1}^{n} c^i = nc$. Thus it is a constant factor off of n, putting it in $\Theta(n)$.
   (c) Note that $\sum_{i=1}^{n} c^i = \frac{c^{n+1}-1}{c-1}$. Therefore $\lim_{n\to\infty} \frac{c^{n+1}-1}{c^n(c-1)} = \lim_{n\to\infty} \frac{c-c^{-n}}{c-1} = \frac{c}{c-1}$. Thus since $\sum_{i=1}^{n} c^i$ is bounded above by a positive constant multiple of c^n, then $\sum_{i=1}^{n} c^i = O(c^n)$.
3. $4^{1536} - 9^{4824}$ is divisible by 35 since $4^{1536} \equiv 4^0 = 1 \mod 35$ and $9^{4824} \equiv 9^0 = 1 \mod 35$ by fermat's little theorem, since once can compute that $1536 \equiv 4824 \equiv 0 \mod \phi(35)$.
4. Since $2^{2023} \equiv 0 \mod 2$ then $2^{2^{2023}} \equiv 2^0 = 1 \mod 3$.
5. Technically, we can computer the fibonacci numbers mod 5 via a lookup table, giving an algorithm which performs in $O(1)$. However assume that we don't know that the Fibonacci sequence loops after a finite number of iterations mod n. Consider the matrix $M = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$. It was shown in a previous chapter that $(F_n, F_{n+1}) = M^n(F_0, F_1)$. Note

that we can now find the eigenvalues of M and compute it's diagonalization over $\mathbb{Z}_5$, in which we can use modular exponentiation on the eigenvalues to compute our desired fibonacci numbers in $O(\log(n))$

6. Let $A(n) = (\log(n))^{\log(n)}, B(n) = fracn\log(n)$. Then $\log\left(\frac{A(n)}{B(n)}\right) = \log(n)\log(\log(n)) - \log(n) + \log(\log(n))$. Since $\log()$ is an increasing function then $\log(\log(n))$ is increasing and additionally for $n > e^e, \log(\log(n)) > 1$ gives us that $\log(A(n)/B(n))$ is going off to infinity. Therefore by the aforementioned motonicity of $\log, \lim \frac{A(n)}{B(n)} \to \infty$ giving us that $B(n)$ is the more efficient algorithm.

7. For the naive approach we have a run time of $O(n^2)$ for the first iteration since we're doing an n by n bit multiplication. For the i-th multiplication we have $O(in^2)$. Therefore the total runtime for the naive approach is $\sum_{i=1}^{y-1} O(in^2) = O(y^2n^2)$ This is in opposition to the iterative squaring method from the base level recursive call has a run time complexity of $O(n^2)$ since you only square the number up to $m = [\log(y)]$ times, where at each ith squaring you have a runtime of $O(4^i n^2)$. Thus you have a final time complexity of $O(2^{2m}n^2)$ since the sum of all the previous time complexities are still smaller than the final one listed above. Based on this analysis the ideal approach is repeated squaring, since $2^{2m} = [\log(y)]^2 \leq \log(y)^2$.

8.
   - $20^{-1} \mod 79 \equiv 4$
   - $3^{-1} \mod 62 \equiv 21$
   - $5^{-1} \mod 23 \equiv 14$