

CS 170 Spring 2008 - Solutions to Homework #1

[0.1] 1 point each, 17 points total.

a) $n - 100 = \Theta(n - 200)$

Proof: $\frac{n - 100}{n - 200} = \frac{1 - 100/n}{1 - 200/n} \rightarrow 1$ as $n \rightarrow \infty$

b) $n^{1/2} = O(n^{2/3})$

Proof: $\frac{n^{1/2}}{n^{2/3}} = n^{1/2-2/3} = n^{-1/6} \rightarrow 0$

c) $100n + \log n = \Theta(n + (\log n)^2)$

Proof: $100n + \log n = \Theta(n)$ and $n + (\log n)^2 = \Theta(n)$. Then both stay within a constant factor of n so they stay within a constant factor of each other.

d) $n \log n = \Theta(10n \log(10n))$

Proof: $10n \log(10n) = 10n(\log n + \log 10) = \Theta(n \log n) + O(n) = \Theta(n \log n)$

e) $\log 2n = \Theta(\log 3n)$

Proof: $\log 2n = \log 2 + \log n = \Theta(\log n)$, $\log 3n = \log 3 + \log n = \Theta(\log n)$.

f) $10 \log n = \Theta(\log(n^2))$

Proof: $\log(n^2) = 2 \log n$

g) $n^{1.01} = \Omega(\log^2 n)$

Proof: Polynomials of any (positive) exponent dominate logs raised to any exponent: $n^a / \log^b n \rightarrow \infty$ for any $a > 0$ and any b

h) $n^2 / \log n = \Omega(n \log^2 n)$

Proof: $f(n)/g(n) = n / \log^3 n \rightarrow \infty$

i) $n^{0.1} = \Omega((\log n)^{10})$

Proof: Same as g). Note that if you plotted them and got the impression that $n^{0.1}$ was smaller, you simply haven't plotted far enough. $n^{0.1}$ eventually takes over, somewhere between 10^{100} and 10^{1000} , a range where many programs refuse to plot.

j) $(\log n)^{\log n} = \Omega(n / \log n)$

Proof: You can change variables. Let $x = \log n$, then $f(n)/g(n) = \frac{x^{x+1}}{e^x} = x \left(\frac{x}{e}\right)^x$ where the second factor is increasing at least when $x > e$ and the first factor x goes to ∞ .

k) $\sqrt{n} = \Omega((\log n)^3)$

Proof: Same as g). Recall that \sqrt{n} is the same as $n^{1/2}$.

l) $n^{1/2} = O(5^{\log_2 n})$

Proof: n appears in the exponent but that does not make $g(n)$ an exponential since it appears in a log. In fact, $g(n) = 5^{\log_2 n} = (2^{\log_2 5})^{\log_2 n} = 2^{(\log_2 5)(\log_2 n)} = n^{\log_2 5}$. Since $\log_2 5$ is a constant, $g(n)$ is a polynomial (if we accept to call "polynomial" a non-integer power of n). Now since $\log_2 5 > 1/2$, $g(n)$ dominates $f(n)$.

m) $n2^n = O(3^n)$

Proof: $f(n)/g(n) = n \left(\frac{2}{3}\right)^n \rightarrow 0$ since exponentials dominate n

n) $2^n = \Theta(2^{n+1})$

Proof: $2^{n+1} = 2 * 2^n$

o) $n! = \Omega(2^n)$

Proof: $n! = 1.2.3 \dots n \geq 1.2.2 \dots 2 = 2^{n-1} = (1/2)2^n$

p) $(\log n)^{\log n} = O(2^{(\log_2 n)^2})$

Proof: We can change variables again. Let $x = \log n$. Then $\log_2 n = (\log n)/(\log 2) = x/\log 2$. Therefore $\frac{f(n)}{g(n)} = \frac{x^x}{2^{x^2/\log^2 2}} = \left(\frac{x}{2^{x/\log^2 2}}\right)^x$. Since $\frac{x}{2^{x/\log^2 2}} \rightarrow 0$, so does $f(n)/g(n)$.

Another method is to study the ratio $k(n) = f(n)/g(n)$ by taking its log:

$$\begin{aligned} \log k(n) &= \log((\log n)^{\log n}) - \log(2^{(\log_2 n)^2}) \\ &= (\log n)(\log \log n) - (\log_2 n)^2 \log 2 \\ &= (\log n)(\log \log n) - (\log n)^2 / \log 2 \\ &= \underbrace{(\log n)}_{\rightarrow +\infty} \underbrace{(\log \log n - (\log n) / \log 2)}_{\rightarrow -\infty} \rightarrow -\infty \end{aligned}$$

So $k(n) = e^{\log k(n)} \rightarrow 0$.

Yet another method is to use the following equation: $x^{\log y} = y^{\log x}$.

$$\begin{aligned} f(n) &= (\log n)^{\log n} = n^{\log \log n} \\ g(n) &= 2^{(\log_2 n)^2} = 2^{\log_2 n \log_2 n} = (2^{\log_2 n})^{\log_2 n} = n^{\log_2 n} \\ \frac{f(n)}{g(n)} &= \frac{1}{n^{\log_2^{-1} \log n - \log \log n}} \rightarrow 0. \end{aligned}$$

Therefore $(\log n)^{\log n} = O(2^{(\log_2 n)^2})$.

$$\mathbf{q)} \sum_{i=1}^n i^k = \Theta(n^{k+1})$$

Proof:

$$\begin{aligned} \sum_{i=n/2}^{n-1} (n/2)^k &\leq \sum_{i=1}^n i^k \leq \sum_{i=1}^n n^k \\ (n/2)(n/2)^k &\leq \sum_{i=1}^n i^k \leq nn^k \\ \frac{n^{k+1}}{2^{k+1}} &\leq \sum_{i=1}^n i^k \leq n^{k+1} \end{aligned}$$

Therefore, $\sum_{i=1}^n i^k$ is $\Theta(n^{k+1})$.

[0.4] 4 points each, 20 points total.

a) Consider the product of any 2×2 matrix:

$$\begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} \begin{pmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{pmatrix} = \begin{pmatrix} x_{11}y_{11} + x_{12}y_{21} & x_{11}y_{12} + x_{12}y_{22} \\ x_{21}y_{11} + x_{22}y_{21} & x_{21}y_{12} + x_{22}y_{22} \end{pmatrix}$$

This shows that every entry of the result matrix is the addition of two products of the entries of the original matrices. Since every entry can be computed with two multiplications and one addition, all four entries requires at most 8 multiplications and 4 additions.

b) First, consider the case where $n = 2^k$ for some positive integer k . To compute X^{2^k} we can recursively compute $Y = X^{2^{k-1}}$ and then square Y to obtain $Y^2 = X^{2^k}$. Unfolding the recursion, this can be seen as repeatedly squaring X to obtain $X^2, X^4, \dots, X^{2^k} = X^n$. At every squaring, we are doubling the exponent of X so that it must take $k = \log n$ matrix multiplications to produce X^n . This method can be easily generalized to members that are not powers of 2, using the following recursion:

$$X^n = \begin{cases} (X^{\lfloor n/2 \rfloor})^2 & \text{if } n \text{ is even} \\ X \cdot (X^{\lfloor n/2 \rfloor})^2 & \text{if } n \text{ is odd} \end{cases}$$

This is exactly the French exponentiation method you saw in lecture, except that X is a matrix. The algorithm requires $\lceil \log n \rceil$ iterations (one iteration for each bit in the binary representation of n), implying $O(\log n)$ matrix multiplications.

c) After iteration i , the entries in the resulting matrix are the addition of the products of the entries from the resulting matrix of iteration $i - 1$.

The maximum number of additions per entry occurs when the iteration

$$\text{has odd } n, \text{ when we must additionally multiply by } X. \quad X \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix}^2 =$$

$$\begin{pmatrix} x_{11}x_{21} + x_{21}x_{22} & x_{12}x_{21} + x_{22}x_{22} \\ x_{11}x_{11} + x_{12}x_{21} + x_{21}x_{21} + x_{22}x_{22} & x_{11}x_{12} + x_{12}x_{22} + x_{12}x_{21} + x_{22}x_{22} \end{pmatrix}$$

shows there are at most 3 additions of products per entry. Therefore, if the maximum-length matrix entry after iteration $i-1$ is $L(i-1)$ bits, then the maximum-length matrix entry after iteration i is at most $2L(i-1) + 3$ bits. The appropriate recurrence relation is $L(i) \leq 2L(i-1) + 3$, with the initial condition $L(0) = 1$. Solving the recurrence gives $L(i) \leq 4(2^i) - 3$. By (b) there are $O(\log n)$ iterations, hence $L(i)$ is $O(n)$ for all iterations i of the algorithm (because $L(\log n) \leq 4n - 3$). This can be restated as all intermediate (and final) matrix entries are $O(n)$ bits long.

d) The algorithm performs $O(\log n)$ matrix multiplications by (b); each matrix multiplication consists of the constant number of arithmetic operations by (a). Each arithmetic operation is on results of size $O(n)$ by (c) and hence takes at most time $O(M(n))$. This implies the algorithm runs in time $O(M(n) \log n)$.

e) Let $T(n)$ be the running time of the algorithm of input size n . First, we recursively first run the algorithm for inputs of size $\lfloor n/2 \rfloor$, which takes time $T(\lfloor n/2 \rfloor)$. Second, we square the results and possibly multiply by X to obtain the final answer: from (a) and (c) we deduce these secondary operations require at most 8 multiplications and 6 additions. Since addition takes linear time, which is $O(M(n))$, squaring and multiplying by X can be done in $O(M(\lfloor n/2 \rfloor))$ time (the results of the recursive call have bit size $O(\lfloor n/2 \rfloor)$ from part (c). This implies:

$$T(n) \leq T(\lfloor n/2 \rfloor) + kM(\lfloor n/2 \rfloor) \leq T(n/2) + kM(n/2)$$

for some constant k . Expanding the recursion and applying the formula for a geometric series, we obtain:

$$\begin{aligned} T(n) &\leq k(M(n/2) + M(n/4) + M(n/8) + \dots + M(1)) \\ &= k\left(\frac{n^c}{2^c} + \frac{n^c}{4^c} + \dots + 1^c\right) \\ &\leq kn^c \sum_{i=1}^{\infty} \frac{1}{2^{ic}} \\ &= \frac{k}{2^c - 1} n^c. \end{aligned}$$

Thus, $T(n) = O(n^c) = O(M(n))$

[1.4] **12 points.** We can lower bound $n!$ as

$$\underbrace{\left(\frac{n}{2}\right) \cdots \left(\frac{n}{2}\right)}_{\frac{n}{2} \text{ terms}} \leq 1 \cdot 2 \cdot 3 \cdots \left(\frac{n}{2}\right) \cdot \underbrace{\left(\frac{n}{2} + 1\right) \cdots n}_{\frac{n}{2} \text{ terms}}$$

and upper bound it as

$$\underbrace{1 \cdot 2 \cdot 3 \cdots n}_n \leq \underbrace{n \cdots n}_n$$

Hence,

$$\begin{aligned} \left(\frac{n}{2}\right)^{\frac{n}{2}} &\leq n! \leq n^n, \\ \frac{n}{2} \log\left(\frac{n}{2}\right) &\leq \log(n!) \leq n \log n, \\ \frac{1}{2}(n \log n - n) &\leq \log(n!) \leq n \log n. \end{aligned}$$

[1.5] **12 points.**

Upper bound:

$$\begin{aligned} \sum_{i=1}^n \frac{1}{i} &= 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots \leq \\ &\leq 1 + \frac{1}{2} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \cdots + \underbrace{\frac{1}{2^k} + \cdots + \frac{1}{2^k}}_{2^k \text{ terms}} = \\ &= \underbrace{1 + 1 + \cdots + 1 + 1}_{O(\log n) \text{ terms}} = \\ &= O(\log n) \end{aligned}$$

Lower bound:

$$\begin{aligned} \sum_{i=1}^n \frac{1}{i} &= 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots \geq \\ &\geq 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} \cdots + \underbrace{\frac{1}{2^k} + \cdots + \frac{1}{2^k}}_{2^{k-1} \text{ terms}} = \\ &= 1 + \underbrace{\frac{1}{2} + \frac{1}{2} + \cdots + \frac{1}{2} + \frac{1}{2}}_{\Omega(\log n) \text{ terms}} = \\ &= \Omega(\log n) \end{aligned}$$