Solutions for CS 521 Homework #1

October 13, 2011

Problem 1

(a) From least to greatest, with equivalent functions on the same line:

1, $n^{1/\log n}$ $n^{1/\log n} = 2$ (take its log)
$\log(\log^* n)$
$\log^* n$, $\log^*(\log n)$ $\log^*(\log n) = \log^* n - 1$ for large enough n
$2^{\log^* n}$
$\ln\ln n$
$\sqrt{\log n}$
$\ln n$ recall $\ln n = \frac{\log n}{\log e}$
$\log^2 n$
$2^{\sqrt{2\log n}}$
$(\sqrt{2})^{\log n}$ $(\sqrt{2})^{\log n} = (2^{1/2})^{\log n} = (2^{\log n})^{1/2} = n^{1/2}$
n, $2^{\log n}$ they are equal
$n\log n$, $\log n!$ by Stirling's approximation
n^2, $4^{\log n}$ they are equal: $4^{\log n} = (2^2)^{\log n} = (2^{\log n})^2 = n^2$
n^3
$(\log n)!$
$n^{\log\log n}$, $(\log n)^{\log n}$ they have the same logarithm
$(3/2)^n$
2^n
$n \cdot 2^n$
e^n
$n!$
$(n+1)!$
2^{2^n}
$2^{2^{n+1}}$

It is clear that for large enough n,

$$n^{1/2} < n < n^3 < n^{\log\log n} < n^{\log n}$$

and

$$(3/2)^n < 2^n < e^n < n!$$

The task then becomes relating the rest of the functions to these. Most of the relationships are obvious or can be shown by taking limits of quotients. We list below some of the ones that require a little ingenuity.

$n^{\log\log n}$ versus $(3/2)^n$: If we take the log of each, we get $\log n \log\log n$ versus $n\log(3/2)$, and it is clear that the second one grows faster. (As an exercise, try to show that $f(n) = o(g(n))$ implies $2^{f(n)} = o(2^{g(n)})$. The analogous statement for big-O is false.)

For $(\log n)!$: Consider the limits $\lim_{n\to\infty} \dfrac{(\log n)!}{(\log n)^{\log n}}$ and $\lim_{n\to\infty} \dfrac{n^3}{(\log n)!}$. Hint: In the former limit, let $m = \log n$ and substitute.

For $2^{\log^* n}$: It stands to reason that $2^{\log^* n}$ will grow faster than $\log^* n$. As for $2^{\log^* n}$ versus $\log\log n$, if we take their logs, we get $\log^* n$ versus $\log\log\log n$, and the second one grows faster.

For $2^{\sqrt{2\log n}}$: Take the logs of $\log^2 n$, $2^{\sqrt{2\log n}}$, and $(\sqrt{2})^{\log n}$. We obtain $2\log\log n$, $\sqrt{2\log n}$, and $\frac{1}{2}\log n$.

Proof of $\log^*(\log n) = \log^* n - 1$: Let $T = \log^* n$. We have to take T logs of n to get to a value of 1 or less. Then if we start from $\log n$, we have to take $T - 1$ logs, since we have already taken one log of n. We conclude $\log^*(\log n) = T - 1$.

(b) Set

$$f(n) = \begin{cases} 1/n & \text{if } n \text{ is even,} \\ 2^{2^{n+1}} \cdot n & \text{if } n \text{ is odd.} \end{cases}$$

Then $f(n)$ dominates every g_i at odd values of n but is dominated by every g_i at even values. This means $f(n) \neq O(g_i(n))$ and $f(n) \neq \Omega(g_i(n))$ for every i.

Problem 2

(a) To answer the second part first: set $f(n) = n$, and set $g(n) = 1$ when n is even, n^2 when n is odd. Then $f(n) \neq O(g(n))$ and $f(n) \neq \Omega(g(n))$.

Let's answer the first part now. We can just set $c = 1$. Define $S = \{n \in \mathbb{Z}^+ : f(n) \geq g(n)\}$. If S is infinite, then $f(n) = \overset{\infty}{\Omega}(g(n))$. If S is finite, it has a maximum value n_0; then for all $n > n_0$, $f(n) < g(n)$, so $f(n) = O(g(n))$.

Since S is either infinite or finite, we have shown that at least one of $f(n) = \overset{\infty}{\Omega}(g(n))$ and $f(n) = O(g(n))$ must hold true. It is possible that they are both true, e.g. when $f(n) = g(n)$.

(b) The clear disadvantage of the $\overset{\infty}{\Omega}$-notation is that the relationship between $f(n)$ and $g(n)$ is only known to hold for some n and not for all n above some threshold. On the other hand, if the relationship holds for "most" n, we can state this with $\overset{\infty}{\Omega}$ but not with Ω.

(c) For convenience we define A and B to be the two parts of Theorem 3.1, rewritten to use O' instead of O:

A: $f(n) = \Theta(g(n))$ B: $f(n) = O'(g(n))$ and $f(n) = \Omega(g(n))$

We wish to determine whether A implies B and whether B implies A. First suppose that B is true. For some $c > 0$ and for large enough n, we have $|f(n)| \leq c \cdot g(n)$ and $f(n) \geq c \cdot g(n)$. Since $f(n) \leq |f(n)|$, we also know $f(n) \leq c \cdot g(n)$. Therefore $f(n) = \Theta(g(n))$, and A is true.

Suppose A is true; B may be false. The above proof does not work in reverse, because $f(n) \leq c \cdot g(n)$ does not imply $|f(n)| \leq c \cdot g(n)$. A simple counterexample is $f(n) = g(n) = -n$. We have $f(n) = \Theta(g(n))$, but $f(n) \neq O'(g(n))$.

We conclude B implies A but A does not imply B.

(d) $\widetilde{\Omega}$ is defined similarly to $\widetilde{O}$. We define $\widetilde{\Theta}$ with two constants k_1 and k_2:

$$\widetilde{\Theta}(g(n)) = \{f(n) : \text{ there exist positive constants } c_1, c_2, k_1, k_2, n_0 \text{ such that}$$

$$0 \leq c_1 g(n) \log^{-k_1} n \leq f(n) \leq c_2 g(n) \log^{k_2} n \text{ for all } n \geq n_0\}.$$

Suppose $f(n) = \widetilde{\Theta}(g(n))$. Then for some positive constants c_1, c_2, k_1, k_2, n_0, we have $0 \leq c_1 g(n) \log^{-k_1} n \leq f(n) \leq c_2 g(n) \log^{k_2} n$ for all $n \geq n_0$. Pulling this apart, we have $0 \leq c_1 g(n) \log^{-k_1} n \leq f(n)$ and $0 \leq f(n) \leq c_2 g(n) \log^{k_2} n$. These statements are the definitions of $f(n) = \widetilde{\Omega}(g(n))$ and $f(n) = \widetilde{O}(g(n))$, respectively. The proof of the other direction works similarly.

Problem 3

For parts (a)-(g), the answers are the ceilings ($\lceil \cdot \rceil$) of the following functions. We assume that n is sufficiently large.

(a) n
(b) $\log^* n$
(c) $\log n$
(d) $\log n - 1$
(e) $\log \log n$
(f) ∞
(g) $\log_3 \log n$.

In (e), notice that 4 is the largest number that takes 1 step to reach 2, 16 is the largest number that takes 2 steps, 256 is the largest number that takes 3 steps, etc. In fact, the number 2^{2^i} takes i steps, since

$$\sqrt{2^{2^i}} = (2^{2^i})^{1/2} = 2^{2^i/2} = 2^{2^{i-1}};$$

the value of the exponent decreases by 1.

(g) is similar.

In (f), a number greater than 1 can never be reduced to 1 by taking square roots. This is why we use $n = 2$ as the base case when we have a recurrence $T(n)$ containing a recursive term $T(\sqrt{n})$.

The value for part (h) is $O(\log n)$ and $\Omega(\log n / \log \log n)$. Suppose our iteration starts with n_0 and proceeds as $n_1, n_2, ..., n_k$, where $n_k \leq 2$. It is clear that $n_i / n_{i+1} \leq \log n_0$ — in order words, to get from n_i to n_{i+1}, we divide by something less than or equal to $\log n_0$. Thus the number of iterations is greater than if we were dividing by $\log n_0$ each time; in that case, the number of iterations would be $\lceil \log_{\log n}(n/2) \rceil = \lceil (\log n - 1)/\log \log n \rceil$. For the upper bound, notice that these divisors are always (except maybe near the end) at least 2; thus the number of steps is less than if we were dividing by 2 each time. In that case, the number of iterations would be $\lceil \log(n/2) \rceil$.

Problem 4

(a) $T(n)$ is $\Theta(n^{\log_3 4})$.

We have $n^{\log_b a} = n^{\log_3 4}$ and $f(n) = n \log n$. Case 1 of the master theorem applies.

(b) $T(n)$ is $\Theta(n \log \log n)$. The master theorem cannot be used here: $n^{\log_b a} = n$ and $f(n) = n/\log n$, but the gap between these functions is not polynomial.

It will be helpful to rewrite $\dfrac{n}{\log n} = \dfrac{n}{\log_3 n / \log_3 2} = \dfrac{n \cdot \log_3 2}{\log_3 n}$. So

$$T(n) = 3T(n/3) + \frac{n \log_3 2}{\log_3 n}.$$

Let us look at the value of $T(n)$ for some small values of n. Say $T(1) = C$.

$T(3) = 3T(1) + (3 \log_3 2)/\log_3 3 = 3C + (3 \log_3 2)/1$.

$T(9) = 3T(3) + (9 \log_3 2)/\log_3 9 = 3(3C + (3 \log_3 2)/1) + (9 \log_3 2)/2 = 9C + (9 \log_3 2)/1 + (9 \log_3 2)/2$.

$T(27) = 9T(3) + (27 \log_3 2)/\log_3 27 = 3(9C + (9 \log_3 2)/1 + (9 \log_3 2)/2)) + (27 \log_3 2)/3 = 27C + (27 \log_3 2)/1 + (27 \log_3 2)/2 + (27 \log_3 2)/3$.

Apparently, for n a power of 3, we have

$$T(n) = Cn + (n \log_2 3)\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{\log_3 n}\right).$$

You can prove this formula using induction. Since $\sum_{i=1}^{n} = \Theta(\log n)$, the summation $\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{\log_3 n}$ is $\Theta(\log \log_3 n) = \Theta(\log \log n)$, and we have

$$T(n) = \Theta(n \log \log n).$$

(c) $T(n)$ is $\Theta(n^2\sqrt{n})$.

We have $n^{\log_b a} = n^2$ and $f(n) = n^{2.5}$. Case 3 of the master theorem applies, but we must check that the regularity condition, $af(n/b) \leq cf(n)$ for some $c < 1$, holds. Substituting for f, we obtain $4 \cdot n^{2.5}/2^{2.5} \leq c \cdot n^{2.5}$, or $1/\sqrt{2} \leq c$. We can therefore take $c = 1/\sqrt{2}$.

(d) $T(n)$ is $\Theta(n \log n)$. We cannot use the master theorem because of the -2 in the recursive call. Without the -2, the master theorem would give an answer of $\Theta(n \log n)$. This is correct, but we must prove it.

We can use induction. Let's show the lower bound first. We want to prove $T(n) \geq cn \log n$ and find a valid value for c. By induction, we assume $T(m) \geq cm \log m$ for all $m < n$.

$$\begin{aligned} T(n) &= 3T(n/3 - 2) + n/2 \\ &\geq 3c(n/3 - 2)\log(n/3 - 2) + n/2 \\ &= c(n-6)(\log(n-6) - \log 3) + n/2. \end{aligned}$$

We need this to be $\geq cn \log n$.

$$\begin{aligned} c(n-6)(\log(n-6) - \log 3) + n/2 &\geq cn \log n \\ \frac{n/2}{n \log n - (n-6)(\log(n-6) - \log 3)} &\geq c. \end{aligned}$$

Note that the denominator is positive (for large n), so we did not have to switch the inequality.

We want to show that some value of c will work for all n greater than some n_0. The easiest way is probably to find the limit of the fraction (as $n \to \infty$) with l'Hôpital's rule. To apply the rule, we need the denominator to go to infinity. The denominator can be rewritten as $(n \log n - (n-6)\log(n-6) + (n-6)\log 3)$, and here $(n \log n - (n-6)\log(n-6))$ is positive while $(n-6)\log 3$ goes to infinity.

Now we need to take the derivatives. The numerator has derivative $1/2$; the derivative of the denominator is

$$1 + \log n - \frac{n-6}{n-6} - (\log(n-6) - \log 3) = \log \frac{n}{n-6} + \log 3 \to \log 3.$$

Thus, by l'Hôpital's rule, the limit of the fraction is $1/(2 \log 3)$. There exists some n_0 such that $n \geq n_0$ implies

$$\frac{n/2}{n \log n - (n-6)(\log(n-6) - \log 3)} \in \left[\frac{1}{3 \log 3}, \frac{1}{\log 3}\right]$$

(these values were chosen arbitrarily; they just have to be positive and on either side of $1/\log 2$). We can therefore set $c = 1/(3 \log 3)$ and run through the proof again; this shows $T(n) \geq 1/(3 \log 3) \cdot n \log n$ for large n, q.e.d.

To show the upper bound, the proof is nearly identical. We switch $\geq$ to $\leq$, and we can use $c = 1/\log 3$. This shows $T(n) \leq 1/(\log 3) \cdot n \log n$ for large n, q.e.d.

(e) $T(n)$ is $\Theta(n \log \log n)$. The proof looks much like the one for part (b).

(f) $T(n)$ is $\Theta(n)$.

Clearly $T(n) \geq n$. In fact, $T(n) = O(n)$ as well: we show this by induction. We want to find a value of c that will let us prove $T(n) \leq cn$. Assume that for all $m < n$, $T(m) \leq cm$. Then

$$\begin{aligned} T(n) &= T(n/2) + T(n/4) + T(n/8) + n \\ &\leq cn/2 + cn/4 + cn/8 + n \\ &\leq n \cdot (1 + 7c/8). \end{aligned}$$

We want this quantity to be $\leq cn$. Solving $n \cdot (1 + 7c/8) \leq 8n$, we obtain $c \geq 8$. So we can set $c = 8$. Then, assuming that $T(m) \leq 8m$ for all $m < n$, we have $T(n) \leq 8n$, q.e.d.

(g) Supposing $T(0) = 0$, we have $T(1) = 1$, $T(2) = 1 + 1/2$, $T(3) = 1 + 1/2 + 1/3$, etc. Clearly $T(n) = \sum_{i=1}^{n} 1/n$, the n-th harmonic sum, and thus $T(n) = \Theta(\log n)$.

(h) Supposing $T(0) = 0$, we have $T(1) = \log 1$, $T(2) = \log 1 + \log 2$, $T(3) = \log 1 + \log 2 + \log 3$, etc. So $T(n) = \sum_{i=1}^{n} \log i = \log n! = \Theta(n \log n)$.

(i) $T(n)$ is $\Theta(n/\log n)$.

Assuming $T(0) = 0$, we have $T(n) = 1/\log 2 + 1/\log 4 + 1/\log 6 + ... + 1/\log n$. Since every term in the sum is at least $1/\log n$, we know $T(n) \geq n/2 \cdot \log n$. We still have to show an upper bound on $T(n)$.

Let k be the smallest integer satisfying $2^k > n$. We have $T(n) \leq T(2^k - 2)$. We take the summation for $T(2^k - 2)$ and replace each term $1/\log j$ with $1/\log j'$, where j' is the largest power of 2 not exceeding j. This increases the sum, and the term $1/\log 2^i$ will occur 2^{i-1} times. We have

$$T(2^k - 2) \leq \frac{1}{\log 2} + \frac{2}{\log 4} + \frac{4}{\log 8} + \frac{8}{\log 16} + ... + \frac{2^{k-2}}{\log 2^{k-1}} = \sum_{i=1}^{k-1} \frac{2^{i-1}}{i} = \frac{1}{2} \cdot \sum_{i=1}^{k-1} \frac{2^i}{i}.$$

The summation behaves like its largest term $2^{k-1}/(k-1)$. A simple way to see this is that for any two consecutive terms a and b (except the first and second), we have $a \leq 3/4 \cdot b$. Thus the series converges faster than a geometric series with initial term $2^{k-1}/(k-1)$ and ratio $3/4$. As a result, we can state $T(2^k - 2) \leq 1 + \frac{1}{2} \cdot 4 \cdot 2^{k-1}/(k-1) = 1 + 2^k/(k-1)$.

By definition, $2^k \leq 2n$, and so $k \leq \log 2n$. Since the function $x/\log x$ is increasing, we have $2^k/(k-1) \leq 2n/(\log 2n - 1) = 2n/\log n$. We conclude

$$\frac{n}{2 \log n} \leq T(n) \leq \frac{2n}{\log n} + 1.$$

(j) $T(n)$ is $\Theta(n \log \log n)$.

As in part (b), let us start out by finding the values of $T(n)$ for small n. Say $T(2) = C$.

$T(4) = 2T(2) + 4 = 2C + 4$.

$T(16) = 4T(4) + 16 = 8C + 16 + 16$.

$T(256) = 16T(16) + 256 = 128C + 256 + 256 + 256$.

The pattern is clear; it seems that $T(n) = Cn/2 + n \log \log n$. You can prove this equation with induction.

Problem 5

(a) Suppose $M \geq \lceil n/2 \rceil$ of the chips are bad and the rest are good. Each bad chip might report that its fellow bad chips are good, and report that the good chips are bad. (The good chips, of course, report that good chips are good and that bad chips are bad.) Then we have a situation where the good chips claim that they are good and the bad chips are bad, while the bad chips claim that they are good and that the good chips are bad. (Imagine two groups of people in an argument where each side says they are right and the other side is wrong.) There is not much we can do in this situation. We cannot automatically say that the smaller group of chips is good — the smaller group could actually be bad chips, who are reporting the same answers that a batch of good chips would report!

To summarize, suppose M chips are bad (call this set group A) and the other chips are either all good or all bad (call this set group B). Further suppose that chips in group A say that other chips in group A are good and that chips in group B are bad, and vice versa. Then there is no way to tell whether group B consists of all good chips or all bad chips.

(b) Given a set S of chips where more than half the chips are good, we will demonstrate how to construct a set T of chips, more than half of which are good, such that $|T| \leq \lceil |S|/2 \rceil$.

For our purposes, we will distinguish two results of a test of a pair of chips C_1 and C_2: (1) each chip says the other chip is good, or (2) at least one chip says the other is bad. In case 1, we know that C_1 and C_2 are the same kind of chip (good or bad). In case 2, we know that at least one of C_1 and C_2 is bad. (Note that if C_1 and C_2 are both bad, the test could return either result.) When case 1 occurs, we have a kind of redundancy — we have two of the same kind of chip, and it makes sense to throw one away. In case 2 we cannot do that, since we may end up keeping a bad one and throwing away a good one. The answer to this problem is, perhaps surprisingly, to throw *both* of them away. When n is *even*, the following process does what we want: `Let` T `be an initially empty set. Pair off the chips in` S`. For each pair` (C_1, C_2)`, run the test on them. If the result is (1), add` C_1 `to` T`. If the result is (2), do not add either chip to` T`.`

When this process is done, at least half the chips in T will be good. This is not hard to prove: S started with a certain ratio of good chips — let us say this ratio was r. Each time a result of type (2) occurs, we end up eliminating either a good chip and a bad chip, or two bad chips. The latter case will obviously increase the ratio of good chips in the remaining set. The former case will also: if we start with g good and b bad, with $g > b$, observe

$$\frac{g-1}{(g-1)+(b-1)} > \frac{g}{g+b}$$

(since this simplifies to $g > b$). Another way of saying this is that in a population of two kind of objects, removing one object of each type skews the ratio further away from 50%. Thus, after we have handled every result of type (2), the remaining set has a ratio of good chips of at least r. Now we handle the results of type (1): from each pair of good chips we choose one good one, and from each pair of bad chips we choose one bad one. This keeps the same ratio; we conclude that more than half the chips in T are good.

When n is odd, the situation is more complicated. As before, we pair off the chips in S and use them to construct T. This time there will be one chip C' left over in S. What do we do with C'? The correct answer turns out to be that we add C' to T if $|T|$ is even. In other words, we need $|T|$ to be odd at the end, and we add or withhold C' according to this condition. To prove that this works, we will handle separately the cases of C' being good and C' being bad. Define $S' = S - \{C'\}$ and $m = (n-1)/2$. When C' is good, at least half the chips in S' are good; in the worst case exactly half of these chips are good, so let us assume this for now. There will be m pairs formed from S'; the number of good-good pairs must equal the number of bad-bad pairs. The rest of the chips will form good-bad pairs. All the good-good pairs will contribute one chip to T; some of the bad-bad pairs will too. If every bad-bad pair contributes a chip to T, its size will be even; we add C' to T and we are done. Now a bare majority of the chips in T are good, and we are satisfied. If not every bad-bad pair contributed a chip to T, then C' might not be added to T; but since fewer bad chips were added, T will still contain a majority of good chips.

If more than half the chips in S' are good, this process will still work: we can view the situation as starting with exactly m good chips and m bad chips, pairing them off, and replacing some of the bad chips with good ones. Every time we replace a bad chip with a good one, it will, in fact, cause one less bad chip to be added to T or cause one more good chip to be added. As a result, even if we end up not adding C' to T in the end, a majority of the chips in T will still be good.

The reasoning in the preceding paragraph also shows why the process works when C' is bad: S' will have at least two more good chips than bad chips, ensuring that its contribution to T will have at least two more good chips than bad chips. Even if we then add C', in the end a majority of the chips in T will still be good.

(c) To identify the good chips, we repeatedly use the method from part (b) to identify a good chip. Starting with a set S_0, the method generates a new set S_1 whose size is approximately half that of S_0. From S_1 we generate a set S_2, etc., until we have a set of size 1. That will be our good chip; we then test the other chips against it. The number of tests needed to run part (b) is given by the recurrence $T(n) \leq T(\lceil n/2 \rceil) + \Theta(n)$; by the master theorem,[1] $T(n) = O(n)$. Testing the other chips against our good chip will require $\Theta(n)$ tests, so that the total number of tests will be $\Theta(n)$.

[1] or by direct solution