CS 521: Data Structures and Algorithms I
Homework 1

Dustin Ingram

October 4, 2011

1. **Problem 3-3, pg 61**

   (a) A simple way to get a nearly-correct ordering of functions by asymptotic groth rate is to choose a sufficiently large n and evaluate each function. Below, each ranked function is also shown with its value in the example case of $n = 1000$.

   We can see that for comparison with functions using iterated logartithms, $n = 1000$ is still not sufficiently large, but we know that such functions grow *so* slowly that they will not out-grow any non-iterative functions. However, to properly order them amongst themselves, we must now use a sufficiently *small* n, which again can be $n = 1000$, to reveal their comparative growth rates.

$g_i(n)$	$g_i(1000)$
$2^{2^{n+1}}$	(large)
2^{2^n}	(less large)
$(n+1)!$	4.02×10^{2570}
$n!$	4.2×10^{2567}
e^n	1.9×10^{434}
$n * 2^n$	1.0×10^{304}
2^n	1.0×10^{301}
$(3/2)^n$	1.2×10^{176}
$n^{\lg(\lg(n)}$	8.9×10^9
$(\lg(n))^{\lg(n)}$	8.9×10^9
$(\lg(n))!$	3.34×10^6
n^3	1000000000
n^2	1000000
$4^{\lg(n)}$	1000000
$n\lg(n)$	9965

$g_i(n)$	$g_i(1000)$
$\lg(n!)$	8529
n	1000
$2^{\lg(n)}$	1000
$\lg^2(n)$	99.3
$(\sqrt{2})^{\lg(n)}$	31.6
$2^{\sqrt{2\lg(n)}}$	22.07
$\ln(n)$	6.9
$\sqrt{\lg(n)}$	3.15
$n^{1/(\lg(n))}$	2
$\ln(\ln(n))$	1.93
$2^{\lg^*(n)}$	16
$\lg^*(n)$	4
$\lg^*(\lg(n))$	3
$\lg(\lg^*(n))$	2
1	1

   (b) This is satisfied by making a faster-growing function trigonometric, e.g. $f(n) = cos(n) * 2^{2^{n+1}}$.

2. **Problem 3-5, pg 62**

   (a) If $f(n) = O(g(n))$, then there exists some n after which $0 \leq f(n) \leq c*g(n)$. If the function behaves in a trigonometric fashion, there may be more than one value for n after which $f(n) \geq c * g(n) \geq 0$, which would make $f(n) = \overset{\infty}{\Omega}(g(n))$, while $f(n) = O(g(n))$ would remain true.
   However, if $f(n) = \Omega(g(n))$, then for all n after some value, $f(n) \geq c(g(n) \geq 0$, and therefore $f(n) = O(g(n))$ cannot also be true.

   (b) An advantage to using $\overset{\infty}{\Omega}$ instead of Ω to characterize the run time of programs is that it would reveal to us the instances in which we could not expect there to be a single n after which $f(n) \geq c*g(n)$. A disadvantage is that it is generally more useful to know that $f(n) \geq c * g(n)$ for all values from n to infinity.

   (c) Theorem 3.1 states:
   For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
   If we substitute O' for O, but still use Ω, the Ω remains the same, but the direction of $f(n) = O(g(n))$ becomes $f(n) \leq O(g(n))$.

   (d) Definitions are as follows:
   $\widetilde{\Omega}(g(n)) = \{f(n)$: there exists positive constants c, k and n_0 such that $cg(n)lg^k(n) \geq f(n) \geq 0$ for all $n \geq n_0$
   $\widetilde{\Theta}(g(n)) = \{f(n)$: if an only if there exists positive constants c, and n_0 such that $f(n) = \widetilde{O}(cg(n))$ and $f(n) = \widetilde{\Theta}(cg(n))$ for all $n \geq n_0$

3. **Problem 3-6, pg 63** The following bounds were determined by generating a subset of values for each function (generally where $n = \{1, ..., 32\}$ would suffice) and then attempting to fit a growth function to the results.

	$f(n)$	c	$f_c^*(n)$
a.	$n-1$	0	$\Theta(n)$
b.	$\lg(n)$	1	$\Theta(\lg(n))$
c.	$n/2$	1	$\Theta(\sqrt{2}^{\lg(n)})$
d.	$n/2$	2	$\Theta(\ln(n))$
e.	$\sqrt{n}$	2	$\Theta(\ln(n))$
f.	$\sqrt{n}$	1	$\Theta(10^n)$
g.	$\sqrt{n^{1/3}}$	2	$\Theta(\lg(\lg^*(n))$
h.	$n/\lg(n)$	2	$\Theta(\ln(n))$

4. **Problem 4-3, pg 108**

   (a) $T(n) = 4T(n/3) + n \lg n$
   Here, $a = 4$, $b = 3$, $f(n) = n \lg n$, and thus we have that $n^{\log_b a} =$

$n^{\log_3 4} = O(n^{1.26})$. Since $f(n) = \Omega(n^{\log_3 4+\epsilon})$ where $\epsilon \approx 0.74$, the ratio $f(n)/n^{\log_b a} = (n \lg n)/(n^2) = \lg n/n$ is asymptotically less than n^ϵ.

(b) $T(n) = 3T(n/3) + n/\lg n$
Here, $a = 3$, $b = 3$, $f(n) = n/\lg n$, and thus we have that $n^{log_b a} = n$. Since the ratio $f(n)/n^{\log_b a} = (n/\lg n)/n$ is neither asympotically less or more than n^ϵ, the Master Theorem cannot be applied. Via substitution:

$$\begin{aligned} T(n) &= 3T(n/3) + n/\lg n \\ &= 3(3T(n/9) + (n/3)/(\lg n/3)) + n/\lg n \\ &= 9T(n/9) + n/\lg n/3 + n/\lg n \\ &= 3^i T(n/3^i) + \sum_{j=1}^{i-1} n/\lg(n/3^j) \end{aligned}$$

(c) $T(n) = 4T(n/2) + n^2\sqrt{n}$
Here, $a = 4$, $b = 2$, $f(n) = n^2\sqrt{n}$, and thus we have that $n^{\log_b a} = n^{\log_2 4} = O(n^2)$ via the Master Theorem.

(d) $T(n) = 3T(n/3 - 2) + n/2$
Here, we must solve via substitution:

$$\begin{aligned} T(n) &= 3T(n/3 - 2) + n/2 \\ &= 3T(n/9 - 2/3 - 2) + n/6 - 2 \\ &= 3T(n/9 - 8/3) + n/6 + 1 \\ &= 3T(n/27 - 8/9 - 2) + n/18 + n/6 - 2 - 2 \\ &= 3T(n/27 - 26/9) + n/18 + n/6 - 4 \end{aligned}$$

(e) $T(n) = 2T(n/2) + n/\lg n$
Here, $a = 2$, $b = 2$, $f(n) = n/\lg n$, and thus we have that $n^{log_b a} = n$. Since the ratio $f(n)/n^{\log_b a} = (n/\lg n)/n$ is neither asympotically less or more than n^ϵ, the Master Theorem cannot be applied. Via substitution:

$$\begin{aligned} T(n) &= 2T(n/2) + n/\lg n \\ &= 2(2T(n/4) + (n/2)/(\lg n/2)) + n/\lg n \\ &= 4T(n/4) + n/\lg n/2 + n/\lg n \\ &= 2^i T(n/2^i) + \sum_{j=1}^{i-1} n/\lg(n/2^j) \end{aligned}$$

(f) $T(n) = T(n/2) + T(n/4) + T(n/8) + n$

(g) $T(n) = T(n-1) + 1/n$

(h) $T(n) = T(n-1) + \lg n$

(i) $T(n) = T(n-2) + 1/\lg n$

(j) $T(n) = \sqrt{n}T(\sqrt{n}) + n$
Here, a is not a constant, so the Master Theorem cannot be used.

5. **Problem 4-5, pg 109** For this problem, reference the following cases:

   - *Case 1.* Both chips declared 'good' – both are either 'good' or 'bad'
   - *Case 2.* Both chips declared 'bad' – at least one is 'bad'
   - *Case 3.* One chip declared 'good', one 'bad' – at least one is 'bad'

   (a) If more than $n/2$ chips are 'bad', pairwise testing will not reveal all the 'good' chips. Since the number of 'bad' chips exceed the number of good chips, at every iteration of a pairwise testing methodology (like the one described in Part b), it can never be guaranteed that the number of 'good' chips outnumber 'bad' chips, and therefore a single 'good' chip cannot be found.

   (b) If the goal is to find only one 'good' chip amongst n chips, of which the number of 'good' chips g is greater than the number of 'bad' chips b, the problem set (originally size n) can be quickly reduced by half using $\lfloor n/2 \rfloor$ pairwise tests by simply rejecting every pair of chips which result in either *Case 2* or *Case 3*.
   This step alone, however, does not guarantee a reduction in the problem set to size $n/2$ – for example, it is possible that for an even-sized set (with even-sized amounts of both 'good' and 'bad' chips) would not result in the rejection of any chips at all. Therefore, it would be necessary to maintain groups of 'A' chips and 'B' chips; in the event that the rejection set is less than n chips, both of these groups would contain indentical amounts of 'good' and 'bad' chips, and one could be rejected, resulting in a set of chips with size $< n/2$ where $g > b$ is still guaranteed to hold true.

   (c) *This is assuming that "Show that the good chips can be identified with* $\Theta(n)$ *pairwise tests" in fact means 'Show that each good chip can be individually found in* $\Theta(n)$ *pairwise tests' and not 'Show that every good chip in a set of size* n *can be found in* $\Theta(n)$ *pairwise tests'...*
   Using the method described in Part b, the recurrence is simple: each iteration of the method performs $\lfloor n/2 \rfloor$ comparisons, and leaves (as a worst-case) $n/2$ chips for the next iteration. With the assumption

that $T(n) = \Theta(n)$, we can induce:

$$\begin{aligned} T(n) &\leq T(n/2) + \lfloor n/2 \rfloor \\ &\leq (c(n/2)) + n/2 \\ &\leq c(n/2 + n/2) \\ &\leq cn \end{aligned}$$