ECE-250 – Algorithms and Data Structures (Winter 2012)

Tutorial 1 (2012-01-19) – Suggested Solutions

1 - Show, using both the formal definition and the limit criterion, that the relationship Θ is symmetric; that is, show that $\mathrm{f}(n) = \Theta(\mathrm{g}(n)) \implies \mathrm{g}(n) = \Theta(\mathrm{f}(n))$

Solution:

(a) Formal definition: Given that $\mathrm{f}(n) = \Theta(\mathrm{g}(n))$, then we know there exist $c_1 > 0$, $c_2 > 0$ and $N > 0$ such that

$$0 \leqslant c_1\mathrm{g}(n) \leqslant \mathrm{f}(n) \leqslant c_2\mathrm{g}(n) \qquad \forall n \geqslant N$$

Since c_1 is positive and both $\mathrm{f}(n)$ and $\mathrm{g}(n)$ are non-negative, then we have that $\mathrm{g}(n) \leqslant c_1^{-1}\mathrm{f}(n)$, and since c_2 is positive, then we have $c_2^{-1}\mathrm{f}(n) \leqslant \mathrm{g}(n)$.

Combining both, we obtain

$$0 \leqslant c_2^{-1}\mathrm{f}(n) \leqslant \mathrm{g}(n) \leqslant c_1^{-1}\mathrm{f}(n) \qquad \forall n \geqslant N$$

Showing that $\mathrm{g}(n) = \Theta(\mathrm{f}(n))$

(b) Limit criterion: Since $\displaystyle\lim_{x\to\infty} \frac{\mathrm{f}(n)}{\mathrm{g}(n)} = L$, where L is finite and non-zero, then we must have that $\displaystyle\lim_{x\to\infty} \frac{\mathrm{g}(n)}{\mathrm{f}(n)} = L^{-1}$, also finite and non-zero, showing that $\mathrm{g}(n) = \Theta(\mathrm{f}(n))$.

Note: the only tricky/questionable detail in here is that knowing that $\mathrm{f}(n) = \Theta(\mathrm{g}(n))$ does NOT imply that the above limit exists and is finite and non-zero (the implication goes the other way).

But since the exercise asked for the limit criterion to show it, then we would assume that the limit exist, and at least the argument proves that the relationship is symmetric in the cases where the limit exists.

2 - Explain why the following statement is or is not true: Given two asymptotically non-negative functions $\mathrm{f}(n)$ and $\mathrm{g}(n)$, with $\mathrm{f}(n) = \Theta(\mathrm{g}(n))$, then it must be true that either $\mathrm{f}(n) - \mathrm{g}(n) = o(\mathrm{f}(n))$ or $\mathrm{f}(n) - \mathrm{g}(n) = o(\mathrm{g}(n))$.

Solution:

Not much to say, really — a simple counter-example shows that the statement is false: take any non-negative function $\mathrm{f}(n)$ and take $\mathrm{g}(n) = 2\mathrm{f}(n)$.

3 - (a) Assuming that the array contains random values, evenly distributed, show that the variable `max` gets assigned $\lg n$ times on average:

```
int find_max (const int * array, int n)
{
    max = array[0];

    for (int i = 1; i < n; ++i)
    {
        if (array[i] > max)
        {
            max = array[i];
        }
    }

    return max;
}
```

Solution:

We'll work more or less by "intuition" — the idea being that in an interval of size k, the chances of having the highest value in the first half or in the second half are 50 - 50. So, if we have processed k elements with the corresponding max so far, when processing the following k elements, we'll have a 50% probability that we will write a new value of max.

But then, we'll double $\lg n$ times to complet the loop, showing that we'll assign $\Theta(\lg n)$ times on average.

For your entertainment, a more formal way to obtain that result is noticing that the probability that one given element be the highest in a group of k elements is $1 / k$ (you'll work more on this during your Probability Theory course next year, but something like this should be reasonably intuitive... right?).

Assuming also that the values are independent of each other, so that the fact that at iteration k the probability is $1 / k$, and regardless whether or not we do find the highest at that position, the probability for the following iteration is $1 / (k+1)$, then the average number of assignments to max will be proportional to the sum of those probabilities:

$$\#\mathit{Assignments} \;=\; C \cdot \sum_{k=1}^{n-1} \frac{1}{k}$$

This does not look like either arithmetic or geometric sums, or any that we've seen in class; but fortunately it does look a lot like an integral that we all know; we can, then, approximate that sum by the following integral:

$$\sum_{k=1}^{n-1} \frac{1}{k} \approx \int_1^n \frac{1}{x}\,dx \;=\; \ln n \;=\; \Theta(\lg n)$$

(b) Given the above result, determine the run time (in asymptotic notation) for the function `find_max` (as a function of the argument n representing the size of the array)

Solution:

Somewhat of a trick question, in that the above result does not affect the run time of the function — either worst-case or average-case.

The loop is executed n times (exactly n times, regardless of the data); the body of the loop evaluates a condition ($\Theta(1)$), and may or may not execute one additional statement; that tells us that each pass of the loop executes in $\Theta(1)$, for a total run time of $\Theta(n)$.

4 - Solve the following recurrence relation: $\mathrm{f}(n) = 4\mathrm{f}(n/2) + \Theta(n)$, with $\mathrm{f}(1) = \Theta(1)$

Solution:

Intuition should quickly make us suspect quadratic growth — the function n^2, when we double its argument n, its value increases by a factor of 4, which is what the recurrence relation indicates.

There is the additional n component, but we might suspect that the effect of that (given only $\lg n$ recursive calls) might be buried under the more significant quadratic term. Still, with repeated substitution we avoid the possible ambiguity in our intuition.

Replacing the term $\Theta(n)$ by a "generic" representative, $c_1 n$, we obtain:

$$\begin{aligned}
\mathrm{f}(n) &= 4(4\mathrm{f}(n/4) + c_1 n/2) + c_1 n \\
&= 16\mathrm{f}(n/4) + 3c_1 n \\
&= 16(4\mathrm{f}(n/8) + c_1 n/4) + 3c_1 n \\
&= 64\mathrm{f}(n/8) + 7c_1 n
\end{aligned}$$

The pattern being:

$$\mathrm{f}(n) = 4^k \mathrm{f}(n/2^k) + (2^k - 1)c_1 n$$

At $k = \lg n$ we reach $\mathrm{f}(1) = c_2$, so we substitute to obtain:

$$\begin{aligned}
\mathrm{f}(n) &= (2^2)^{\lg n}\mathrm{f}(1) + (2^{\lg n} - 1)c_1 n \\
&= n^2 c_2 + c_1(n-1)n \\
&= \Theta(n^2)
\end{aligned}$$