Informatics 2 – Introduction to Algorithms and Data Structures

Tutorial 1: Asymptotic Notation

SOLUTIONS

1. *For each of the following five functions g, identify a function f_i from the list such that $g = \Theta(f_i)$. Justify your answers as clearly as you can.*

   (a) $g(n) = n(n+1)(2n+1)/6$.

   Growth rate is $\Theta(n^3)$. An adequate justification (which can be made fully rigorous) is that when this is expanded as a polynomial, the highest-degree term is $n^3/3$ – and as we saw in lectures, quadratic and lower-order terms are $o(n^3)$. This implies that $\Theta(n^3)$ is the essential growth rate.

   Note, incidentally, that this is the formula for $\Sigma_{k=1}^{n} k^2$.

   (b) $g(n) = n \textit{ div } 57$ *(integer division, rounding down).*

   This is $\Theta(n)$. Indeed, $g(n)$ differs from $n/57$ (exact division) by at most 1. More rigorously, we can see that once $n \geq 57$, we'll have $g(n) > n/114$ (for instance), so that $g(n)$ is sandwiched between $n/114$ and $n/57$.

   (c) $g(n) = n \textit{ mod } 57 + 1$.

   This is $O(1)$, because we have $1 \leq g(n) \leq 57$ for all n. (Note that without the '+1', it would be $O(1)$ but not $\Omega(1)$ (i.e. not eventually bounded below by a positive constant), because n mod 57 would be zero infinitely often.)

   (d) $g(n) = n \lg n + (\lg n)^3 + e^{-n}$. *You may assume here that* $\lg n = o(\sqrt{n})$.

   This one shows the usefulness of asymptotic notation for cleaning up a messy formula. We claim $g(n) = \Theta(n \lg n)$, as this is the dominant term. The term e^{-n} can clearly be ignored as it is always ≤ 1. And from $\lg n = o(\sqrt{n})$ it follows easily that $(\lg n)^2 = o(n)$, whence $(\lg n)^3 = o(n \lg n)$, so the second term also becomes negligible relative to $n \lg n$. Again, this can all be made completely rigorous with a little effort.

   (e) $\star$ *Where would the factorial function fit into this picture? Does $n!$ have the same growth rate as one of the above functions f_i? Or does it fall between f_i and f_{i+1} for some i?*

   The growth rate of $n!$ falls strictly between that of 2^n and 2^{2^n}.

To see that $2^n = o(n!)$, let's look at the ratio $n!/2^n$, which is

$$\frac{1 \times 2 \times 3 \times \cdots \times n}{2 \times 2 \times 2 \times \cdots \times 2}$$

It's easy to see that this is at least $n/2$ (once $n \geq 2$), which tends to infinity as n does.

To see that $n! = o(2^{2^n})$, we can note that for $n \geq 4$,

$$n! \;<\; n^n \;<\; (2^n)^n \;=\; 2^{n^2} \;\leq\; 2^{2^n}$$

2. (a) *Show directly from the definition that* $100n^3 = o(n^4)$.

   Given $c > 0$, we need to pick a suitable N. We can arrive at this by working backwards: what needs to be true in order that $100n^3 < cn^4$? Cancelling n^3 from both sides, this is equivalent to $100 < cn$ (for positive n), which in turn is equivalent to $n > 100/c$.

   Having gone through something like this in rough working, we're now in a position to present the following polished solution, in which we appear to pull a rabbit from a hat:

   Given $c > 0$, consider any $N > 100/c$. Then for any $n \geq N$ we have

   $$100n^3 \;=\; c(100/c)n^3 \;<\; c.n.n^3 \;=\; cn^4.$$

   [The 'polished solution' is formally all we need to say to answer the question, but the 'rough working' is perhaps more illuminating.]

   (b) *Show that if* r, s *are any* real *numbers with* $0 \leq r < s$*, then* $n^r = o(n^s)$.

   Here we can argue "informally': if we take

   $$\lim_{n\to\infty} \frac{n^s}{n^r} = \lim_{n\to\infty} n^{s-r},$$

   it is easy to see that this is ∞, since $s > r$. To argue formally, again we can work backwards: we need an N large enough such that for every c, and every $n \geq N$, $n^r < c \cdot n^s$ holds. If we solve for n, it follows that $n > (1/c)^{1/(s-r)}$, i.e., taking $N > (1/c)^{1/(s-r)}$ suffices. Note that since $s > r$, the ratio $1/(s-r)$ in the exponent is always well-defined.

   [Tip: Looking at how the ratio of the two functions behaves is often a good way forward.]

   (c) *Writing 'lg' for log to base 2 and 'ln' for log to base e, show that* $\ln n = O(\lg n)$. *Deduce that* $\lg n = \Theta(\ln n)$.

   We will make use of the well-known formula

   $$\log_b x \;=\; (\log_b a)(\log_a x)$$

   to change the base of the logarithm. Applying the formula, we obtain that $\lg x = (\lg e) \cdot (\ln x)$. To show that $\ln n = O(\lg n)$, we need to find a constant $C > 0$ and an N such that for every $n \geq N$, $\ln n \leq C \cdot \lg n$.

Choosing $C = 1/\lg e$ works in this case. To show that $\lg n = \Theta(\ln n)$, we need to find constants $c_1, c_2 > 0$ and N such that for every $n \geq N$, we have $c_1 \ln n \leq \lg n \leq c_2 \ln n$. For c_2, we can take $c_2 = 1/C = \lg e$. For c_1, we can again take $c_1 = 1/C = \lg e$. Here $\lg e$ is an absolute constant, so this gives $\lg n = \Theta(\ln n)$.

(d) *Is it likewise true that $2^n = \Theta(e^n)$?*

Most certainly not! Here it suffices to argue "informally". Indeed, if we look at

$$\lim_{n \to \infty} \frac{e^n}{2^n} = \lim_{n \to \infty} \left(\frac{e}{2}\right)^n ,$$

this goes to ∞, and hence it will surpass any given $C > 0$ as n increases (specifically, once $n > \ln C / \ln(e/2)$).

3. *Recall the methods you learned at school for addition, long multiplication and long division. For each of these,* informally *analyse the asymptotic worst-case runtime on inputs of at most n decimal digits. You may take 'time' to mean the number of times you have to write a symbol on the page.*

In this question, we shall satisfy ourselves with an informal, non-rigorous style of analysis.

For numbers of at most n digits, addition takes 'time' $\Theta(n)$. We have to write the at most $n + 1$ digits of the answer, plus (at worst) a similar number of carry digits.

For long multiplication of two n-digit numbers, we in effect construct a list of n numbers each of at most $n + 1$ digits, then add them. Not hard to convince oneself that all of this takes time $\Theta(n^2)$.

For integer long division (e.g. resulting in a div b and a mod b). The division will proceed in $\leq n$ 'rounds', in each of which we perform a subtraction of size $\leq n + 1$. (The necessary values of $b, 2b, \ldots, 9b$ can be precomputed at the start, taking just time $\Theta(n)$.) So the overall runtime is clearly $O(n^2)$. To see that the worst-case runtime is also $\Omega(n^2)$, consider the situation of dividing an n-digit a by an $n/2$-digit b. Clearly this can require around $n/2$ subtractions of size $n/2$.