Asymptotic Notation
-draft-

M. Pellegrini*

March 6, 2002

1 Concrete Mathematics

Ronald L. Graham, Donald E. Knuth and Oren Patashnik [GKP89] give a nice introduction to the issue of asymptotic notation and asymptotic analysis. The motivation is that of giving a relative order of magnitude to functions $f : N \to R$ as their argument tends to infinity , with minimal level of detail specified of f. Similarly we handle functions $f : R \to R$ that tend to infinity as the argument tends to zero.
The $\prec$ relation. Given two functions f and g defined on the natural numbers N with values in the reals R, we define the following relation $\prec$:

$$f(n) \prec g(n) \iff \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

Informally, $g(n)$ goes to infinity faster than $f(n)$. The following properties hold:

(a) (Transitivity) if $f(n) \prec g(n)$ and $g(n) \prec h(n)$, then $f(n) \prec h(n)$;

(b) (Polynomials) $n^\alpha \prec n^\beta \iff \alpha < \beta$, for α, β any arbitrary real.

We can rank functions in hierarchies using the $\prec$ relation, for example:

$$1 \prec \log\log n \prec \log n \prec n^\epsilon \prec n^c \prec n^{\log n} \prec c^n \prec n^n \prec c^{c^n}$$

for arbitrary reals $0 < \epsilon < 1 < c$. So far we have seen functions that go to infinity as n tends to infinity, but we can deal also with function that go to zero as follows:

$$f(n) \prec g(n) \iff \frac{1}{g(n)} \prec \frac{1}{f(n)}$$

*Istituto di Matematica Computazionale del CNR, pellegrini@imc.pi.cnr.it

So we can invert the above list to obtain a ranking of small functions. We can combine this initial ranking to compare more complex functions. it is not difficult to prove that:

$$n^{\alpha_1}(\log n)^{\alpha_2}(\log\log n)^{\alpha_3} \prec n^{\beta_1}(\log n)^{\beta_2}(\log\log n)^{\beta_3} \iff (\alpha_1,\alpha_2,\alpha_2) \overset{lex}{<} (\beta_1,\beta_2,\beta_3)$$

where the relation $\overset{lex}{<}$ indicate the lexicographic order of tuples. Other important properties are:

$$1 \prec f(n) \prec g(n) \Rightarrow e^{|f(n)|} \prec e^{|g(n)|}$$

Exercise 1 *Use the rules learned so far to prove* $\log n \prec e^{\sqrt{\log n}} \prec n^{\epsilon}$*, for every* $\epsilon > 0$.

The $\asymp$ relation. We formalize the intuitive notion that two functions have the same rate of growth with the following definition:

$$f(n) \asymp g(n) \iff |f(n)| \leq C|g(n)| \; and \; |g(n)| \leq C|f(n)|, \text{ for some C, } n \text{ large enough}$$

where C is independent of n; and n large enough means that there is an n_0, independent of n, so that for all $n > n_0$ the right hand side holds.
The $\sim$ relation. A slightly stronger definition is the following:

$$f(n) \sim g(n) \iff \lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$$

Logarithmico-exponential functions. G.H. Hardy introduced the class of *logarithmico-exponential functions* $\mathcal{L}$. This class is vast and it will contains all the function we typically need in the analysis of algorithms, but its main interesting property is that it is structured as an asymptotic hierarchy, meaning that, for every pair of functions $f(n)$ and $g(n)$ in $\mathcal{L}$, one and only one of the following three cases occurs: either $f(n) \prec g(n)$, or $g(n) \prec f(n)$, or $f(n) \asymp g(n)$. Moreover, in the last case there exists a constant c, depending on f and g, such that $f(n) \sim cg(n)$. (proof not supplied). The class $\mathcal{L}$ of *logarithmico-exponential functions* is defined as follows:

(a) For any real α, the constant function $f(n) = \alpha$ is in $\mathcal{L}$.

(b) The identity function, $f(n) = n$, is in $\mathcal{L}$.

(c) If $f(n)$ and $g(n)$ are in $\mathcal{L}$, also $f(n) - g(n)$ is in $\mathcal{L}$.

(d) If $f(n)$ is in $\mathcal{L}$, also $e^{f(n)}$ is in $\mathcal{L}$.

(e) If $f(n)$ is in $\mathcal{L}$ and is eventually positive, also $\ln f(n)$ is in $\mathcal{L}$.

Hardy also proved that any function in $\mathcal{L}$ is either eventually positive, eventually negative, or identically zero; therefore also products and quotients of two functions in $\mathcal{L}$ is in $\mathcal{L}$, except that we cannot divide by a function that is identically to zero.
Big-Oh notation. The big-Oh notation is an alternative definition of asymptotics that has several advantages with respect to those seen so far, the main one is that it can be treated as an expression rather than a relation and so be subject to pseudo-arithmetic manipulations. In this notation, for an expression α, $O(\alpha)$ stands for a value that is at most a constant multiplied $|\alpha|$. The interesting fact is that the constant is not specified, thus even an existential certainty is sufficient to be able to use the big Oh notation. For example the result that:

$$H_n = \ln n + \gamma + O(1/n)$$

can be expressed in the previous notation in a weaker form:

$$H_n - \ln n - \gamma \prec \frac{\log \log n}{n}$$

or in a form more involved to be proved:

$$H_n - \ln n - \gamma \asymp \frac{1}{n}.$$

The Big-Oh notation makes sense when there is at least one variable quantity, and the formula:

$$f(n) = O(g(n))$$

means that there exist constants C and n_0 such that $|f(n)| < C|g(n)|$ for all $n > n_0$. Note that if $O(.)$ appears in several parts of an expression, formula or equation, the unspecified C's and n_0's may be different, however each one is constant with respect to n. For functions of a real variable x we write $f(x) = O(g(x))$ as $x \to 0$, meaning that there exist real positive constants ϵ and C so that for every x with $|x| < \epsilon$, $|f(x)| < C|g(x)|$. So in the innocent looking $O(.)$ notation there are at least two hidden unspecified constants. A second aspect must be handled with care, equalities involving $O(.)$ expressions cease to be fully reversible. While

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

is ok,

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

is plainly meaningless. The $O(.)$ notation is a coarsification of information conventionally taking place from the left hand side to right hand side. When the interpretation

of a big-Oh expression or equation is not obvious the proper way of proceeding is to fall back to set theoretic notation. We associate with $O(f(n))$ the set of functions satisfying its definition. The $=$ sign in equations is treated as $\subseteq$ and all arithmetic operations are to be understood as set operations, non-Oh functions are treated as singletons. For example the equation:

$$3n^3 + O(n^2) = O(n^3)$$

means that every function in the set of functions defined by the left hand side expression is also in the set of functions defined by the right hand side expression. With this caveats we can usually safely use the standard arithmetic manipulations. People sometimes abuses the big-Oh notation as if it were giving an exact order of growth, or it is used to specify lower bounds as well as upper bounds. Instead, if we prove for example $f(n) = O(n^k)$, this does not forbid that it might be true that also $f = O(n^h)$, with $h < k$. To put it differently, big-Oh limit how much unlucky you can be, not how lucky you can be. For lower bounds we have a second notation:

$$f(n) = \Omega(g(n)) \iff |f(n)| > C|g(n)|, \quad \text{for some } C > 0,\ n \text{ large enough.}$$

We have the following relation:

$$f(n) = \Omega(g(n)) \iff g(n) = O(f(n))$$

Finally both relation might hold and we define:

$$f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \ and \ f(n) = \Omega(g(n)).$$

Using the previous relational notation we have:

$$f(n) \asymp g(n) \iff f(n) = \Theta(g(n)).$$

Edmund Landau introduced also the little-oh notation, which however, should be replaced with $O(.)$ of functions tending to zero as $x \to 0$ whenever possible since the resulting expressions are easier to manipulate.

Bits of advice Asymptotic analysis of an algorithm using the big-Oh expressions to state the results of the analysis can be done at different level of detail. Which level of detail is appropriate depends on the purpose of the analysis. For example suppose that we establish that sorting algorithms S_1 and S_2 have respectively running time: $T_1(n) \sim c_1 n \log n$ and $T_2(n) \sim c_2 n \log n$. Choosing the algorithm only with respect to the relative value of c_1 and c_2 might be a bad policy. Indeed a finer analysis might uncover that $T_1(n) = c_1 n \log n + d_1 n + O(\log n)$ and $T_2(n) = c_2 n \log n + d_2 n + O(\log n)$. Usually small decrements in the first constant are paid by large increment of the second constants, therefore both factor must be take into consideration within the

range of values of n we want to target, taking into consideration that $\log n$ is a rather slowly growing function of n.

The base of the logs. In big-Oh notation we obtain the same class of functions by using any constant base: $O(\log n)$ [base 10], $O(\lg n)$ [base 2] and $O(\ln n)$ [base e]. Similarly the same class is denoted by: $O(\log\log n)$, $O(\lg\lg n)$ and $O(\ln\ln n)$.

Manipulation of big-Oh expressions. Some more rules:

$$\begin{aligned} n^h &= O(n^k) \text{ For every } k > h, \\ n^h &\notin O(n^k) \text{ For every } k < h, \\ O(f(n) + O(g(n)) &= O(|f(n)| + |g(n)|), \\ f(n) &= O(f(n)), \\ O(O(f(n)) &= O(f(n)), \\ O(f(n))O(g(n)) &= O(f(n)g(n)), \\ O(f(n)g(n)) &= f(n)O(g(n)). \end{aligned}$$

2 Thomas H. Cormen, Charles Eric Leiserson and Ronald L. Rivest

Thomas H. Cormen and Charles Eric Leiserson and Ronald L. Rivest [CLR90, page 23-32] introduce the asymptotic notation of big-Theta, Big-Oh, Big-Omega. Starting from the set interpretation, and moving to the arithmetic as an abuse of notation. Figure 2.1 is nice.

Big-Oh, Big-Omega and Big-Theta notation. Here the definition is restricted to eventually positive functions, thus the absolute value is avoided. For $g(n) \in \mathcal{L}$ eventually positive:

$$\begin{aligned} O(g(n)) &= \{f(n) \in \mathcal{L} | \exists C > 0,\ \exists n_0 > 0,\ \forall n > n_0 : 0 \leq f(n) \leq Cg(n)\} \\ \Omega(g(n)) &= \{f(n) \in \mathcal{L} | \exists C > 0,\ \exists n_0 > 0,\ \forall n > n_0 : 0 \leq Cg(n) \leq f(n)\} \\ \Theta(g(n)) &= \{f(n) \in \mathcal{L} | \exists C_1, C_2 > 0,\ \exists n_0 > 0,\ \forall n > n_0 : 0 \leq C_1 g(n) \leq f(n) \leq C_2 g(n)\} \end{aligned}$$

Bounds written in Θ notation are called *asymptotically tight*. Naturally $f(n) \in \Theta(g(n))$ implies $f(n) \in O(g(n))$, but not vice versa. With $\Theta(1)$ we denote a constant, although the variable for which it is a constant has to be understood from the context. When we say that the *worst case time function* $T_A^W(n)$ of algorithm A is $\Theta(g(n))$ this assertion has to be understood remembering that the worst case time function is the *maximum* of the actual time for all inputs of a length n. Since $T_A^W(n) = O(g(n))$, this implies that there exists constant C such that *for all* n and *for all* $x \in I$ with $|x| = n$, $T_A(x) \leq Cg(n)$. Since $T_A^W(n) = \Omega(g(n))$, this implies that there exists constant C such that *for all* n *there exists* $x \in I$ such that $T_A^W(x) \geq Cg(n)$. For every input length class n it is sufficient to find one bad input to force the lower bound to be high. On the other hand the upper bound holds for all input instances. Naturally for the *best case time function* the considerations are reversed, and the lower bound holds for all instances. To distinguish the two cases we use the expressions:

(i) *worst case running time lower bound* $\Omega(g(n))$ to denote the first case;

(ii) *running time lower bound* $\Omega(g(n))$ to denote the second case.

Care should be used since the two concepts are easily confused.

Asymptotics in equations. Suppose we are interested in the asymptotics of the solution of a recursive equation. In this case the known functions do not have to be expressed exactly, it is sufficient to indicate them asymptotically. The quality of the asymptotic solution will depend on the quality of the asymptotic replacement in the equation. For example, take

$$T(n) = 2T(n/2) + 3n + 2, \quad T(1) = 1 \quad \text{For } n \text{ power of 2.}$$

We can solve it directly and derive the solution $T(n) = O(n \log n)$. Otherwise, we can derive the same asymptotic result by upper bounding the known functions in the righthand side.

$$T'(n) = 2T'(n/2) + O(n), \quad T'(1) = 1 \quad \text{For } n \text{ power of 2.}$$

This solves in $T'(n) = O(n \log n)$, and it can be shown that it implies that also for that original function we have $T(n) = O(n \log n)$. However if we make a legal but low-quality upper bounding like:

$$T''(n) = 2T''(n/2) + O(n^2), \quad T''(1) = 1 \quad \text{For } n \text{ power of 2.}$$

we get $T''(n) = O(n^2)$, which implies that $T(n) = O(n^2)$, true but of a lower quality.

Asymptotics in summations. Several occurrences of big-Oh in the same expression or equation refer to different constants C and n_0, however in an expression like:

$$\sum_{i=1}^{n} O(i)$$

there is only one single anonymous function $f(k) \in O(k)$ involved, and this *is not* equivalent to $O(1) + O(2) + ... + O(n)$ which does not have a clean interpretation of its own.

Other properties of Big-Oh, Big-Omega and Big-Theta notation.

(a) Big-Oh, Big-Omega and Big-Theta are *transitive*.

(b) Big-Oh, Big-Omega and Big-Theta are *reflexive*

(c) Big-Oh, Big-Omega have *transposed symmetry*.

(d) Big-Theta is *symmetric*.

(e) We have seen that all functions in $\mathcal{L}$ are comparable under the $\prec, \succ, \asymp$ relations. Also, $f(n) \prec g(n)$ implies $f(n) = O(g(n))$, $f(n) \succ g(n)$ implies $f(n) = \Omega(g(n))$, and $f(n) \asymp g(n)$ implies $f(n) = \Theta(g(n))$. Naturally there are functions not in $\mathcal{L}$, and among these it is easy to find non comparable pairs of functions (for example n and $n^{1+\sin n}$).

References

[CLR90] Thomas H. Cormen, Charles Eric Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, Cambridge, Massachusetts, 1990.

[GKP89] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics : a Foundation for Computer Science*. Addison-Wesley Publishing Company, 1989.