September 7, 2016

Asymptotics

1 Definitions and Terminology

1.1 Admissibility and Limits

Define a function f to be *admissable* iff there is an integer n_0 such that $f(n)$ is defined and $f(n) > 0$ for all integers $n \geq n_0$.

We introduce here some terminology that reduces the need for explicitly quantifying mathematical statements. In the context of admissible functions, we will use the expression *almost everywhere* when applied to a statement to mean: " there is an integer n_0 such that the statement is true for all $n > n_0$". Using this terminology we can re-state the definition of adnissible function as follows:

A function f is *admissable* iff $f(n) \geq 0$ almost everywhere.

We also use some simplifying terminology in the context of limits. If f is an admissible function we will take the statement "*f trends to C*" to mean that the limit of $f(n)$ as n tends to infinity is equal to C:

$$\lim_{n \to \infty} f(n) = C$$

means f trends to C.

1.2 Big Oh, Big Omega, and Big Theta

The asymptotic notations Big Oh [$\mathcal{O}$], Big Omega [Ω] and Big Theta [Θ] are fundamental to the study of algorithms. These each relate to the "near infinity" behaviour of functions and are independent of multiplication by a constant and independent of any effects that relate only to a finite number of inputs.

Given an admissible function g, define $\mathcal{O}(g)$ to be the set of all admissible functions f such that

$$f(n) \leq Cg(n)$$

almost everywhere for some positive constant C. That is, there exists $C > 0$ and n_0 such that $f(n) \leq Cg(n)$ for all $n > n_0$. Similarly, define $\Omega(g)$ to be the set of all admissible functions f such that

$$f(n) \geq Cg(n)$$

almost everywhere for some constant $C > 0$. And finally define define $\Theta(g)$ to be the set of all admissible functions f such that

$$C_1 g(n) \leq f(n) \leq C_2 g(n)$$

almost everywhere, for some positive constants C_1, C_2.

1.3 Asymptotic Equivalence and the Tilde Relation

For admissable functions f and g, define $f \sim g$ to mean that

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

In Section 2 we show that $\sim$ is an equivalence relation. The terminology used for $f \sim g$ is that f and g are *asymptotically equivalent*. We denote by the asymptotic equivalence class of f as $\mathcal{A}[f]$.

Asymptotic equivalence is a more specialized notion that does not apply as broadly as Θ and also is a stronger relation than Θ when it does apply. We will typically use it as follows:

Suppose f is a function we wish to characterize asymptotically and that we know, or surmise, that $f \in \Theta(M)$ for some collection of "model functions" M. Then we may ask what specific model is asymptotically equivalent to f. For example, we may know that $f \in \Theta(n^d)$ and ask what positive constant A satisfies $f \sim An^d$. In that circumstance, we could call A the *growth factor* of f and d the *growth exponent* of f.

We return to calculation of growth constants in a later section.

2 Properties of the relations $\mathcal{O}$, Ω, Θ, and $\sim$

Proposition 2.1 (Reflexive Property). An admissible function is asymptotically related to itself. That is: if f is admissible then $f \in \mathcal{O}(f)$, $f \in \Omega(f)$, and $f \in \Theta(f)$.

Proof. Let $C = 1$. Then plainly

$$f(n) = Cf(n)$$

for all n, from which it is clear that the definitions of $f \in \mathcal{O}(f)$, $f \in \Omega(f)$, and $f \in \Theta(f)$ are all satisfied. □

Proposition 2.2. Assume that f and g are admissable functions. Then:
(a) (Anti-Symmetry) $f \in \mathcal{O}(g)$ if and only if $g \in \Omega(f)$.
(b) (Symmetry) $f \in \Theta(g)$ if and only if $g \in \Theta(f)$.

Proof (b). From the definition of Θ there are positive constants C_1 and C_2 such that $C_1 g(n) \leq f(n) \leq C_2 g(n)$ almost everywhere. Using algebra, we have:

$$\frac{1}{C_2} f(n) \leq g(n)$$

and

$$g(n) \leq \frac{1}{C_1} f(n).$$

Taking $D_1 = \frac{1}{C_2}$ and $D_2 = \frac{1}{C_1}$ we have

$$D_1 f(n) \leq g(n) \leq D_2 g(n)$$

showing that $g \in \Theta(f)$. □

Exercise 1. Supply a proof of (a).

Proposition 2.3 (Transitivity). Assume that f, g, and h are admissable functions. Then:
(a) If $f \in \mathcal{O}(g)$ and $g \in \mathcal{O}(h)$ then $f \in \mathcal{O}(h)$.
(b) If $f \in \Omega(g)$ and $g \in \Omega(h)$ then $f \in \Omega(h)$.
(c) If $f \in \Theta(g)$ and $g \in \Theta(h)$ then $f \in \Theta(h)$.

Proof (a). From the definition of $\mathcal{O}$ there are positive constants C_1 and C_2 such that

$$f(n) \leq C_1 g(n)$$

and

$$g(n) \leq C_2 h(n)$$

almost everywhere. Substituting the second into the first, and applying the transitive property of $\leq$, we have

$$f(n) \leq C_1 C_2 h(n)$$

almost everywhere. Taking $C = C_1 \times C_2$ the definition of $f \in \mathcal{O}(h)$ is satisfied. □

Exercise 2. Supply proofs of (b) and (c).

Proposition 2.4 (Dichotomy). If admissible functions f and g are Θ equivalent, then $f \in \mathcal{O}(g)$ and $f \in \Omega(g)$. Conversely, if $f \in \mathcal{O}(g)$ and $f \in \Omega(g)$ then $f \in \Theta(g)$.

A proof is a direct application of the definitions and is left as an exercise.

Propositions 1,2(b),3(c) above show that $f \in \Theta(g)$ is an equivalence relation, thus the Θ equivalence classes partition the set of admissible functions into mutually disjoint sets. Propositions 1,2(a),3(a),3(b),4 show that $\mathcal{O}$ and Ω behave analogously to the numerical order relations $\leq$ and $\geq$, with Θ playing the role of equality.

Terminology surrounding $\mathcal{O}$, Ω and Θ ranges from the set-theoretic introduced above to more informal. For example, when $f \in \Theta(g)$ it is often said that "f is $\Theta(g)$" and alternate notation $f = \Theta(g)$ may be used. To emphasize the properties analogous to numerical order relations we sometimes write $f \leq \mathcal{O}(g)$ or $g \geq \Omega(f)$. The set-theoretic versions, such as $\mathcal{O}(f) \subseteq \mathcal{O}(g)$, may also be used.

PROPOSITION 2.5. $\sim$ is an equivalence relation on the set of admissable functions.

To prove Prop 2.5 we need to verify that these three properties hold:

Reflexive: $f \sim f$ for all f

Proof. For any admissible function f, note that $\frac{f(n)}{f(n)}$ is defined and equal to 1 using straighforward algebra. Therefore

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

verifying that $f \sim f$. □

Symmetric: $f \sim g$ implies $g \sim f$ for all f, g

Proof. Suppose that $f \sim g$ for two admissible functions f and g. Then

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

Note that

$$\frac{g(n)}{f(n)} = \frac{1}{\frac{f(n)}{g(n)}}$$

and hence that

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

which verifies that $g \sim f$. □

Caution!

It's important to distinguish the above from the completely falacious argument:

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

Be sure you see why this argument is faulty.

Transitive: $f \sim g$ and $g \sim h$ implies $f \sim h$, for all f, g, h

Proof. Suppose that $f \sim g$ and $g \sim h$ for three admissible functions f, g, and h. Observe that

$$\frac{f(n)}{h(n)} = \frac{f(n)g(n)}{h(n)g(n)} = \frac{f(n)}{g(n)} \times \frac{g(n)}{h(n)}$$

from which it follows that

$$\lim_{n\to\infty} \frac{f(n)}{h(n)} = \lim_{n\to\infty} \left(\frac{f(n)}{g(n)} \times \frac{g(n)}{h(n)} \right) = \lim_{n\to\infty} \frac{f(n)}{g(n)} \times \lim_{n\to\infty} \frac{g(n)}{h(n)} = 1 \times 1 = 1$$

proving that $f \sim h$. □

Advisory

In general, it is legitimate to make the leap

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

if and only if it is independently verified (or a given) that

$$\lim_{n\to\infty} f(n)$$

is a finite number and

$$\lim_{n\to\infty} g(n)$$

is a finite non-zero number. Otherwise you end up with undefined expressions such as $\frac{\infty}{\infty}$, $\frac{\infty}{0}$, $\frac{0}{\infty}$, and $\frac{0}{0}$.

3 Relationships among $\mathcal{O}$, Ω, Θ and $\sim$

Proposition 3.1. Suppose that f and g are admissable and

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = C$$

where C is a constant. Then $f = \mathcal{O}(g)$ and $g = \Omega(f)$. Moreover, if $C > 0$ then $f = \Theta(g)$.

Proof. First note that C must be non-negative, because the quotient both $f(n)$ and $g(n)$ are non-negative almost everywhere and $g(n)$ must be positive almost everywhere in order for the limit to exist. By the definition of limit, $f(n)/g(n) \to C$, with $\epsilon = 1$, there exists a positive integer n_1 such that $f(n)/g(n) \le C + 1$ for $n \ge n_1$. Taking $C_1 = 1 + C$ we have $f(n)/g(n) \le C_1$ and after algebra

$$f(n) \le C_1 g(n)$$

for $n \ge n_1$. Therefore $f \le \mathcal{O}(g)$.

If in addition $C > 0$, again applying the definition of limit with $\epsilon = C/2$, there is a positive integer n_2 such that $f(n)/g(n) \ge C - \epsilon = C/2$ for $n \ge n_2$. Taking $C_2 = C/2$ we have $f(n)/g(n) \ge C_2$ and after algebra

$$C_2 g(n) \le f(n)$$

for $n \ge n_2$. Therefore $f \ge \Omega(g)$. □

PROPOSITION 3.2. If f and g are admissible and $f \sim g$ then $\Theta(f) = \Theta(g)$.

Proof. $f \sim g$ means that the quotient $f(n)/g(n)$ trends to 1. Since $1 > 0$ the result is a corollary to Prop 3.1. □

Proposition 3.2 states exactly what was alluded to earlier, that $\sim$ is a stronger relation than Θ. We also stated that $\sim$ is applicable to a smaller class of functions, and the reason for that is that the quotient $\frac{f(n)}{g(n)}$ may not have a limit at all (i.e., may not have a unique "trend" value). In the case where there is a trend for the quotient, there is a partial converse to 3.2 as follows:

PROPOSITION 3.3. Suppose that f and g are admissible and that $f(n)/g(n)$ trends to a positive constant C. Then $f \sim C \times g$.

Proof. Calculating with limits: $\lim_{n\to\infty} \frac{f(n)}{Cg(n)} = \frac{1}{C} \times \lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{1}{C} \times C = 1$. □

Exercise 3. Is the following inverse of Proposition 3.3 true? Suppose f and g are admissable and $f = \Theta(g)$. Then $f \sim Cg$ for some positive constant C. (True or false, with answer justified.)

4 Simplification Rules

Proposition 4.1. If f and g are admissable and $f \leq \mathcal{O}(g)$ then $\Theta(f+g) = \Theta(g)$.

Proof. Applying the definition of big-O, we find that there is a positive constant C and a positive integer n_1 such that

$$f(n) \leq Cg(n)$$

for $n \geq n_1$. Therefore we have

$$f(n) + g(n) \leq Cg(n) + g(n) = (C+1)g(n)$$

for $n \geq n_1$. Taking $C_1 = 1 + C$ we have

$$f(n) + g(n) \leq C_1 g(n)$$

and thus $f + g \leq \mathcal{O}(g)$.

On the other hand, note that by admissibility there exists a positive integer n_2 such that $f(n) \geq 0$ and therefore

$$g(n) \leq f(n) + g(n)$$

for all $n \geq n_2$. Taking $C_2 = 1$, we then have

$$C_2 g(n) \leq f(n) + g(n)$$

for all $n \geq n_2$ and thus $f + g \geq \Omega(g)$. It now follows that $f + g = \Theta(g)$. □

Exercise 4. Prove or supply a counterexample: $\Theta(1+g) = \Theta(g)$ for any admissable g.

Proposition 4.2. Suppose f and g are admissable and

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

Then $f + g \sim g$.

Proof. Before taking limits, observe that

$$\frac{f(n) + g(n)}{g(n)} = \frac{f(n)}{g(n)} + \frac{g(n)}{g(n)} = \frac{f(n)}{g(n)} + 1$$

and therefore

$$\lim_{n\to\infty} \left(\frac{f(n) + g(n)}{g(n)} \right) = \lim_{n\to\infty} \left(\frac{f(n)}{g(n)} + 1 \right) = \lim_{n\to\infty} \frac{f(n)}{g(n)} + \lim_{n\to\infty} 1 = \lim_{n\to\infty} \frac{f(n)}{g(n)} + 1 = 0 + 1 = 1.$$

Therefore $f + g \sim g$. □

Exercise 5. Prove true or false:

(a) $n \log n + \log n \sim n \log n$

(b) $n \log n + n \sim n \log n$

(c) $n \log n + n \log n \sim n \log n$

5 Polynomials

Look at these two functions of n:

$$P(n) = a_0 n^d + a_1 n^{d-1} + \ldots + a_n$$
$$Q(n) = n^d$$

(where we assume the leading coefficient $a_0 > 0$). $P(n)$ is the general form of an admissible polynomial of degree d, whereas $Q(n)$ is the much simpler form of the highest power term.

PROPOSOTION 5.1. With the definitions above, $P(n) \sim a_0 n^d$.

Proof. First note the calculation

$$\frac{P(n)}{Q(n)} = \frac{a_0 n^d}{n^d} + \frac{a_1 n^{d-1}}{n^d} + \ldots + \frac{a_{d-1} n}{n^d} + \frac{a_d}{n^d}$$
$$= a_0 + \frac{a_1}{n} + \frac{a_2}{n^2} + \ldots + \frac{a_{d-1}}{n^{d-1}} + \frac{a_d}{n^d}$$

from which it is apparent that $\frac{P(n)}{Q(n)}$ tends to a_0 as n becomes large. Since P and Q are admissible, the result follows from Prop 3.3. ☐

COROLLARY 5.2. $\Theta(P(n)) = \Theta(n^d)$.

Example applications of the various simplifying rules:

$$n(n+1)/2 = \Theta(n^2)$$
$$n^2 + \log n = \Theta(n^2)$$
$$n + \log n = \Theta(n)$$
$$5000n^2 + 2300\sqrt{n} = \Theta(n^2)$$

6 Estimating the growth constants from Data

In many cases a growth exponent and a growth factor associated with the asymptotic class of an algorithm can be estimated from data. Start by assuming that the algorithm runtime has Θ class one of these forms (aka "abstract models"):

$$\begin{aligned} &An^d + B\phi(n) && \text{[Model 0]} \\ &An^d \log n + B\phi(n) && \text{[Model 1]} \end{aligned}$$

where $A > 0$ and $\phi(n)$ is dominated by the first term: $\frac{\phi(n)}{n^d} \to 0$ as $n \to \infty$ [Model 0] or $\frac{\phi(n)}{n^d \log n} \to 0$ as $n \to \infty$ [Model 1].

Proposition 6.1. The abstract models have Θ class as follows:

$$\begin{aligned} &An^d + B\phi(n) = \Theta(n^d) && \text{[Model 0]} \\ &An^d \log n + B\phi(n) = \Theta(n^d \log n) && \text{[Model 1]} \end{aligned}$$

The proof is a direct application of Prop 4.1.

Thus we can "ignore" the second term (which might in fact be quite complicated, like the tail of a polynomial) when finding the exponent d and constant A in the models. In both cases we can find these growth constants using actual runtime data.

6.1 Estimating the Growth Exponent - Model 0

Assume that the asymptotic growth of an algorithm is modelled by $F(n) = An^d$ [Model 0] and that we have data gathered from experimentation to evaluate F at size n and again at size $10n$:

$$\begin{aligned} F(10n) &= (10n)^d \\ &= n^d 10^d \\ &= 10^d F(n) \end{aligned}$$

which shows that raising the input size by one order of magnitute increases the runtime by d orders of magnitude. For instance, when $d = 2$ (the quadratic case), increasing the size of the input by one decimal place increases the runtime by two decimal places. Another way to phrase the result is as a ratio:

$$\frac{F(10n)}{F(n)} = \frac{(10n)^d}{n^d} = 10^d$$

which can be stated succintly as

$$d = \log_{10}\left(\frac{F(10n)}{F(n)}\right).$$

If we have actual timing data $T(n)$ for an algorithm modelled by F we can use the ratio to estimate d.

Example 1 - insertion_sort

Consider for example the insertion_sort algorithm, and use "comps", the number of data comparisons, as a measure of runtime. We know from theory that insertion_sort is modelled by F and we wish to know the exponent d. We have collected runtime data

$$\begin{aligned} T(1000) &= 244853 \\ T(10000) &= 24991950 \end{aligned}$$

The ratio $T(10000)/T(1000)$ is

$$\begin{aligned} \frac{T(10000)}{T(1000)} &= \frac{24991950}{244853} \\ &= 102.07\ldots \\ &\simeq 100\pm \\ &= 10^2 \end{aligned}$$

yielding an estimate of $d = 2$, or quadratic runtime. Your eye might have noticed this in the data itself: $T(10000)$ is about 100 times $T(1000)$.

6.2 Estimating the Growth Exponent - Model 1

The somewhat more complex Model 1 works in the same way. Assume that the asymptotic growth of an algorithm is modelled by $G(n) = An^d \log n$ [Model 1] and that we have data gathered from experimentation to evaluate G at size n and again at size $10n$:

$$\begin{aligned}
\frac{G(10n)}{G(n)} &= \frac{(10n)^d \log(10n)}{n^d \log n} \\
&= \frac{n^d 10^d \log(10n)}{n^d \log n} \\
&= \frac{10^d \log(10n)}{\log n} \\
&= 10^d \left(\frac{\log 10 + \log n}{\log n} \right) \\
&= 10^d \left(\frac{1 + \log n}{\log n} \right) \\
&= 10^d \left(1 + \frac{1}{\log n} \right) \\
&\to 10^d
\end{aligned}$$

because $\frac{1}{\log n} \to 0$ as $n \to \infty$. As in the pure exponential case, this concliusion can be stated in terms of logarithms:

$$d \simeq \log_{10} \left(\frac{F(10n)}{F(n)} \right).$$

Example 2 - List::Sort

Consider the bottom-up merge_sort specifically for linked lists, implemented as List::Sort. It is known from theory that the algorithm is modelled by G, and we have collected specific timing data as follows:

$$\begin{aligned}
T(10000) &= 123674 \\
T(100000) &= 1566259
\end{aligned}$$

Then:

$$\begin{aligned}
\frac{T(100000)}{T(10000)} &= \frac{1566259}{123674} \\
&= 11.66\ldots \\
&\simeq 10\pm \\
&= 10^1
\end{aligned}$$

predicting $d = 1$. Note here that the data will not likely be enough to discriminate between Models 1 and 2, so we must base that choice on other considerations.

6.3 Estimating the Growth Factor

We can refine an abstract model to a "concrete" version by finding the constant A such that $A \times Model(n)$ more accurately predicts runtime. The goal is to make timing data and the concrete model match as closely as possible:

$$T(n) \simeq A \times M(n) \;\; \text{for all } n$$

At this point, we are assuming one of two "abstract" models for the runtime cost of an algorithm:

$$F(n) = n^d$$
$$G(n) = n^d \log n$$

and further we have estimated a value for the (integer) exponent d. Given that, we want to calculate an estimate for the constant A for either of our models M by solving one of the evaluated equations obtained from data for A:

$$A = \frac{T(n)}{M(n)}$$

where T is timing data and M is the growth model (F or G). In fact, we get different estimates for A for each known pair $(n, T(n))$ in our collected data - a classic over-constrained system. Ideally we would use a method such as least squares (linear regression) to optimize a value for A using all of the collected runtime data. A decent substitute would be to interpolate a value using the two data points we used to estimate the exponent. Here are those calculations using the two examples already given above.

Example 1 (continued)

We have this data for insertion_sort:

$$T(1000) = 244853$$
$$T(10000) = 24991950$$

The data points give estimates of A as

$$A = \frac{T(1000)}{F(1000)} = \frac{244853}{1000^2}$$
$$= 0.2485$$

$$A = \frac{T(10000)}{F(10000)} = \frac{24991950}{10000^2}$$
$$= 0.2499$$

It is reasonable to settle for $A = 0.25$ to complete our concrete model:

$$M(n) = 0.25 \times n^2 \qquad \text{Concrete Model for insertion\_sort}$$

This model can be used to estimate runtimes for values of n where actual data is lacking. Note that the choice of the quadratic abstract model is based on theory and known to be a correct abstract model for insertion_sort.

Example 2 (continued)

We have this data collected for List::Sort:

$$T(10000) = 123674$$
$$T(100000) = 1566259$$

The data points give estimates of A as

$$A = \frac{T(10000)}{G(10000)} = \frac{123674}{10000 \log 10000} = \frac{123674}{10000 \times 4}$$
$$= 3.09185$$

$$A = \frac{T(100000)}{G(100000)} = \frac{1566259}{100000 \log 100000} = \frac{1566259}{100000 \times 5}$$
$$= 3.132518$$

It is reasonable to settle for $A = 3.1$ to complete our concrete model:

$$M(n) = 3.1 \times n \log n \qquad \text{Concrete Model for List::Sort}$$

This model can be used to estimate runtimes for values of n where actual data is lacking. Note that the choice of the linear×log abstract model is based on theory and known to be a correct abstract model for List::Sort (a version of bottom-up merge_sort).

Exercise 6. Extend the results of Sections 6.1-6.3 to include Model 2:
$H(n) = An^d(\log n)^2 + B\phi(n)$.

6.4 Cautions and Limitations

The reader was likely surprised that using the data as in Section 3 above is unable to distinguish between the pure power model F and the model G that is a power model multiplied by a logarithm. The reason at one level is simple: the quotients $G(10n)/G(n)$ and $F(10n)/F(n)$ differ by $10^d/\log n$. The numerator 10^d is a fixed number, whereas the denominator $\log n$ grows infinitely large with n (albeit rather slowly), so the difference gets ever smaller as n grows large. Given that data inevitably has some variation due to randomness, teasing out such a diminishingly fine distinction is problematic.

Another observation the reader likely made is that we used the base 10 logarithm instead of the more common base 2 logarithm. Any base could have been used. We chose base 10 because multiplying by 10 is a visually simple process - just move the decimal point - whereas if we used base 2 (and doubled our input size instead of multiplying it by 10) the results are similar, except it is less easy visually to recognize "approximately" $2n$ than "approximately" $10n$.

Different base logarithmic functions have the same Θ class, so when discussing Θ we are free to use any base log:

LEMMA 6.2. $\log_a x = \log_a b \times \log_b x$

which tells us that $\log_2 n = \Theta(\log_{10} n)$, the first being a constant multiple of the second, that constant being $\log_2 10$.

Finally, and most important, we need to keep in mind that using the techniques of Section 5 are (1) only estimates - "estimate" being another word for "educated guess" - and (2) dependent on a choice of model. The choice of model may also be an educated guess, or it could be from theoretical considerations, or it could be a simplification from known theoretical constraints.

As in all of science, a model is an approximation of reality.

The Bottom Line

Simplifying formulas

- An admissible polynomial of degree d is $\Theta(n^d)$ and $\mathcal{A}[a_0 n^d]$
- When finding Θ, ignore $\mathcal{O}$ terms
- When finding $\sim$, ignore terms of strictly lower asymptotic class

Finding model constants

- Growth Exponent $d \simeq \log_{10} T(10n_0)/T(n_0)$
- Growth Factor $A \simeq T(n_0)/M(n_0)$

where n_0 is a specific size for which we have data, T is actual runtime data, and M is the abstract model. The concrete modelling formula is then

$$T(n) \simeq A \times M(n).$$