November 21, 2015

Homework 4: Estimating Runtime Constants

The Bottom Line

Simplifying formulas

- If P is a polynomial with leading term An^d $[A > 0]$ then $P \sim An^d$ and $P = \Theta(n^d)$
- When finding Tilda or Theta class, ignore terms of strictly lower asymptotic class

Finding model constants

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

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

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

1 Definitions, Notation, and Representations

Define a function F to be *admissable* iff it is defined on almost all of the non-negative integers and has positive real values on almost all of the domain. That is, there is an integer n_0 such that $F(n)$ is defined and $F(n) > 0$ for all integers $n > n_0$. Note that for polynomials, this condition requires that the leading coefficient is a positive number.

Note also we continue to use the terminology *almost all* of the non-negative integers to mean for all integers n that are greater than some fixed value n_0 – in other words, the condition holds for all but a finite number of possible exceptions. This terminology is easier to talk about and allows us to stop mentioning "$n > n_0$", simplifying the wording of many statements.

Finally, recall the concept of asymptotic equivalence: Two admissible functions F and G are *asymptotically equivalent* iff

$$\lim_{n\to\infty} \frac{F(n)}{G(n)} = 1.$$

We adopt the "tilde notation": $F \sim G$ means that F and G are asymptocially equivalent.

2 Simplifying Formulas

We begin by collecting some facts established in previous assignments:

LEMMA 0. The relation $\sim$ is an equivalence relation on the set of admissable functions.

Proof. This is Proposition 1 of Homework 2 ["Prop 2.1" for short]. $\square$

LEMMA 1. Suppose that F and G are admissible and the quotient $F(n)/G(n)$ tends to 1 as n grows large - that is, $F \sim G$. Then $F(n)$ and $G(n)$ are in the same Θ equivalence class.

Proof. This is Proposition 3 of Homework 2 ["Prop 2.3" for short]. $\square$

Look at these two functions of n:

$$\begin{aligned} F(n) &= a_0 n^d + a_1 n^{d-1} + \ldots + a_n \\ G(n) &= a_0 n^d \end{aligned}$$

(where we assume the leading coefficient a_0 is non-zero). $F(n)$ is the general form of a polynomial of degree d, whereas $G(n)$ is its leading term.

LEMMA 2. Suppose $F(n)$ is the general form of a polynomial of degree d with leading coefficient $A = a_0 > 0$. Then $F \sim An^d$.

Proof. (This is very similar to Prop 1.4. However a slightly different proof is as follows.) Let $G(n) = An^d$ and do the calculations:

$$\begin{aligned} \frac{F(n)}{G(n)} &= \frac{a_0 n^d}{a_0 n^d} + \frac{a_1 n^{d-1}}{a_0 n^d} + \ldots + \frac{a_d}{a_0 n^d} \\ &= 1 + \frac{a_1}{a_0} n^{-1} + \frac{a_2}{a_0} n^{-2} + \ldots + \frac{a_d}{a_0} n^{-d} \end{aligned}$$

(where $A = a_0$). It is apparent that $\frac{F(n)}{G(n)}$ tends to 1 as n becomes large. Therefore $F \sim G$ (and, by Lemma 1, $F = \Theta(G)$). $\square$

The setting for polynomials can be generalized. (Here H plays the role of a polynomial, h is the leading term, and ϕ is the "tail" consisting of all the lower order terms.)

Lemma 3. Suppose we have admissible functions

$$H(n) = h(n) + \phi(n)$$

where $\phi(n)$ has the property that $\frac{\phi(n)}{h(n)} \to 0$ as $n \to \infty$. Then $H \sim h$.

proof. This is embodied in both Prop1.1 and Prop 2.2. Another way to state a proof uses the same process we used above for polynomials:

$$\begin{aligned}\frac{H(n)}{h(n)} &= \frac{h(n)}{h(n)} + \frac{\phi(n)}{h(n)} \\ &= 1 + \frac{\phi(n)}{h(n)} \\ &\to 1 + 0 \text{ as } n \to \infty \\ &= 1\end{aligned}$$

□

Lemmas 2 and 3 can be loosely paraphrased as follows:

(1) (Polynomial Theta rule) A polynomial of degree d is $\Theta(n^d)$.
(2) (Polynomial Tilde rule) A polynomial of degree d and leading coefficient A is asymptotically equivalent to An^d.
(3) (Lower order cancellation rule) When finding the Theta or Tilda class of a function, terms with strictly lower asymptotic order can be ignored.

Example applications of these simplifying rules, showing a simple representative for the $\sim$ class and the Θ class:

Table 1: Class Representatives

$F(n)$	$\sim$	Θ
$n(n+1)/2$	$\frac{1}{2}n^2$	n^2
$0.05n^2 + 175\log n$	$0.05n^2$	n^2
$n + (\log n)^2$	n	n
$8n^3 + 5n^2\log n - 3n^2 + n\log n + 17$	$8n^3$	n^3
$5000n^2 + 2300\sqrt{n}$	$5000n^2$	n^2
$n^2 + n\sin n$	n^2	n^2

Exercise 1. Verify the statements in Table 1 by showing step-by-step simplification, citing a specific rule or Lemma for each step.

It is worth contemplating the relationship between Tilda and Theta. On the one hand, a limit need not exist in order for two functions to be Theta equivalent, so Theta applies in principle to a larger setting. On the other hand, most functions we encounter that represent the asymptotic growth of an algorithm have a polynomial or polynomial-like form, such as the examples above. Moreover, when it is applicable, asymptotic equivalence produces finer distinctions than Theta. As illustrated in the examples, Tilda [asymptotic equivalence] retains the leading coefficient, whereas Theta ignores it. It is reasonable to conclude that, in the cases where it is applicable, it is more useful to know the Tilda class than the Theta class (and, the Theta class is easily derived from the Tilda class in any case). The remaining sections are about finding ways to estimate the Tilde (and hence Theta) class from actual timing data.

3 Estimating the growth exponent from Data

In many cases an exponent 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”):

$$\begin{aligned} &An^d + \phi(n) && \text{[Model 1]} \\ &An^d \log n + \phi(n) && \text{[Model 2]} \end{aligned}$$

where $A \neq 0$ and $\phi(n)$ is dominated by the first term: $\frac{\phi(n)}{n^d} \to 0$ as $n \to \infty$ [Model 1] or $\frac{\phi(n)}{n^d \log n} \to 0$ as $n \to \infty$ [Model 2].

Lemma 4. The abstract models have Tilda and Theta class as follows:

$$\begin{aligned} An^d + \phi(n) &\sim An^d = \Theta(n^d) && \text{[Model 1]} \\ An^d \log n + \phi(n) &\sim An^d \log n = \Theta(n^d \log n) && \text{[Model 2]} \end{aligned}$$

The proof is a direct application of Lemma 3.

Thus we can “ignore” the second term ϕ when finding the exponent d in the models. (ϕ represents all of the “lower order details”.) In both cases we can find the exponent d and the leading coefficient (aka “growth constant”) A from actual runtime data.

Example 1: Model 1 and insertion_sort

Assume that the asymptotic growth of an algorithm is modelled by $F(n) = An^d$ [Model 1] and that we have data gathered from experimentation to evaluate F at size

n and again at size $10n$:

$$\begin{aligned} F(10n) &= A(10n)^d \\ &= An^d 10^d \\ &= A10^d F(n) \end{aligned}$$

which shows that raising the input size by one order of magnitude 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:

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

which can be stated succintly as

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

Note that the growth constant cancels in the quotient: calculating the exponent ignores A and hence we could use $A = 1$ in this setting.

If we have actual timing data $T(n)$ for an algorithm modelled by F we can use the ratio to estimate d. 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 $T(1000)$.

Example 2 - Model 2 and List::Sort

The somewhat more complex Model 2 works in the same way. Assume that the asymptotic growth of an algorithm is modelled by $G(n) = An^d \log n$ [Model 2] and that we have data gathered from experimentation to evaluate G at size n and again at size $10n$:

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

because $\frac{1}{\log n} \to 0$ as $n \to \infty$. (Again the growth constant A is made irrelavant by cancellation.) As in the pure exponential case, this conclusion can be stated in terms of logarithms:

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

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.

4 Estimating the growth constant

We can refine an abstract model to a "concrete" version by finding the leading coefficient (growth constant) A in the model. Knowing specific values for both A and d

will provide a model that can predict runtimes with useful accuracy. 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 such that $T(n) = A \times M(n)$ 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

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

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

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).

5 Cautions and Limitations

The reader was likely surprised that using the data as in Sections 3 and 4 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 5. $\log_a x = \log_a b \times \log_b x$

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 Sections 3 and 4 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.

Exercise 2. Find the model exponent d and growth constant A for each line in the following table:

Table 2: Measurement Data

n	$T(n)$	$T(10n)$	Model	d	A
10000	8395	84054	An^d		
80	179200	179200000	An^d		
10000	256040	3115695	$An^d \log n$		
10000	254848	25467962	An^d		
500	559016	17666669	An^d		

Report d as an integer or simple fraction and A to two decimal places.