Asymptotic Notation Basics (Updated April 16, 2013)

In computer science it is often difficult to precisely measure certain quantities, and even when they can be measured accurately their exact values are not essential. Accurate, but simple looking approximations could be more useful than complex exact formulas.

A typical example is that of a running time of an algorithm. The actual running time depends on the implementation, the compiler, on the machine, and on the conditions under which the program is executed. Thus it is simply not possible to tell what is the running time of an algorithm based only on its description. But in fact the exact formula is not even necessary. For example, if an algorithm makes $4n^3 + 3n + 7$ steps on inputs of length n, for large values of n the terms $3n$ and 7 are insignificant. It makes more sense to say that the running time is approximately $4n^3$. In fact, it makes sense to even ignore the constant 4, since this constant is implementation- and platform-dependent. So we can say that the running time will be proportional to n^3. In other words, it will be roughly cn^3, for some constant c. This observation motivates the definition of asymptotic notation.

Definition. For two functions $f(x)$ and $g(x)$, we say that $f(x)$ is of order $g(x)$, and write $f(x) = O(g(x))$, if there are constants c and x_0 such that $|f(x)| \leq cg(x)$ for all $x \geq x_0$.

In all useful contexts the functions we will deal with will be non-negative, in which case we can ignore the absolute value (we will in fact do this in most examples). Occasionally though we may encounter functions that may take some negative values for small values of x, so we need to take this into account in the definition. Note that the above definition forces $g(x)$ to be non-negative if x is large enough.

Example 1. We show that $7x + 20 = O(x)$. Indeed, say, for $x \geq 10$ we have $7x + 20 \leq 7x + 2x \leq 9x$, so the above definition applies with $x_0 = 10$ and $c = 9$.

But these choices are not unique. We could as well say that for $x \geq 20$ we have $7x + 20 \leq 7x + x \leq 8x$, etc.

Example 2. Let's show now that $2n^3 + 6n^2 - 2 = O(n^3)$. We have $2n^3 + 6n^2 - 2 \leq 2n^3 + 6n^3 = 8n^3$ for $n \geq 0$, so we can conclude that $2n^3 + 6n^2 - 2 = O(n^3)$.

Example 3. As a first more exciting example, we'll look at the harmonic numbers:

$$H(n) \;=\; \sum_{i=1}^{n} \frac{1}{i} \;=\; 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n}.$$

This function grows with n, but how fast? We claim that its growth rate is the same as that of logarithmic functions, that is $H(n) = O(\log n)$.

Let's now prove it. The idea of the proof is to divide the sequence $1, 1/2, 1/3, ..$ into about $\log n$ blocks, so that the sum of each block is between $\frac{1}{2}$ and 1. More specifically, we divide the sequence $1, 1/2, 1/3, ...$ into *blocks*

$$1 \qquad \frac{1}{2} + \frac{1}{3} \qquad \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} \qquad \dots \qquad \frac{1}{2^i} + \frac{1}{2^i + 1} + ... + \frac{1}{2^{i+1} - 1} \qquad \dots \qquad \frac{1}{2^k} + \frac{1}{2^k + 1} + ... + \frac{1}{n}$$

where k is chosen to be the integer such that $2^k \leq n < 2^{k+1}$. (Thus $k = \lfloor \log n \rfloor$.) So, for any i except k, the i-th block starts at $1/2^i$ and ends right before $1/2^{i+1}$. The last block is exceptional, since it ends at $1/n$ which could be smaller than $1/(2^{k+1} - 1)$.

Thus the sum of the i-th block (except the last) is

$$\frac{1}{2^j} + \frac{1}{2^j + 1} + \frac{1}{2^j + 2} ... + \frac{1}{2^{j+1} - 1}$$

In this sum, each term is at most $1/2^j$ and is greater than $1/2^{j+1}$. This block starts at the 2^j-th term and ends right before 2^{j+1}-st term, so it has $2^{j+1} - 2^j = 2^j$ terms. Therefore the sum of this block is at most

$2^j \cdot \frac{1}{2^j} = 1$ and at least $2^j \cdot \frac{1}{2^{j+1}} = \frac{1}{2}$. (Except the last block's sum, which is at most 1 but could be less than $\frac{1}{2}$.)

Putting it all together, we proceed as follows. We have $k+1$ blocks, with each block adding to at most 1, so $H_n \leq (k+1) \cdot 1 \leq \log n + 1$. On the other hand, all blocks except last add up to at least $\frac{1}{2}$, so $H_n \geq k \cdot \frac{1}{2} \geq \frac{1}{2}(\log n - 1)$. Summarizing, we have proved the following theorem.

Theorem 0.1 $\frac{1}{2}(\log n - 1) \leq H_n \leq \log n + 1$, *for all positive integers* n.

How can we express this using the big-Oh notation? Bounding H_n from above, we get that for $n \geq 2$, $H_n \leq \log n + 1 \leq \log n + \log n = 2 \log n$. So $H_n \leq 2 \log n$ for $n \geq 2$, and we can conclude that $H_n = O(\log n)$.

Much better estimates are known for H_n. It is known that $H_n \sim \ln n + \gamma$, where $\gamma \approx 0.57$ is called Euler's constant. More precisely, the difference $H_n - \ln n$ converges to γ with $n \to \infty$. We can express this better approximation using the big-Oh notation as well, by using the big-Oh notation for the approximation error rather than to the function itself: $H_n = \ln n + O(1)$. An even more accurate approximation is $H_n = \ln n + \gamma + O(1/n)$, as it shows that the approximation error $H_n - \ln n - \gamma$ vanishes when $n \to \infty$.

Example 4. The asymptotic notation has not been invented by computer scientists – it has been used in mathematics for over a 100 years, for describing approximation errors using various expansions (like Taylor series), and in number theory, to estimate the growth of some functions. Consider for example the following question: for a number n, how many prime numbers are between 2 and n? This value is denoted $\pi(n)$. It is known that $\pi(n) = O(n/\log n)$. In fact, more accurate estimates for $\pi(n)$ exist; for example $\pi(n) = n/\ln n + O(n/\log^2 n)$. This result is often called the Prime Number Theorem.

Example 5. We now consider an example that involves sequences defined by recurrence equations. Suppose we have a sequence $\{a_i\}$ defined by $a_0 = 3$, $a_1 = 8$, $a_n = 2a_{n-1} + a_{n-2}$. We claim that $a_n = O(2.75^n)$.

In order to prove this, we show by induction that $a_n \leq 3(2.75)^n$. Indeed, in the base cases, $a_0 = 3 = 3(2.75)^0$ and $a_1 = 8 < 3(2.75)^1$. In the inductive step, assume the claim holds for numbers less than n. For n, by the inductive assumption, we get

$$\begin{aligned} a_n &= 2a_{n-1} + a_{n-2} \\ &\leq 2 \cdot 3(2.75)^{n-1} + 3(2.75)^{n-2} \\ &\leq 3(2.75)^{n-2}(2 \cdot 2.75 + 1) \\ &\leq 3(2.75)^{n-2}(2.75)^2 \\ &\leq 3(2.75)^n, \end{aligned}$$

and thus the claim holds for n as well. Therefore it holds for all values of n.

Example 6. Ok, now let's talk about algorithms. What's the running time of the algorithm below?

Algorithm WHATSMYRUNTIME1 (n : **integer**)
  for $i \leftarrow 1$ **to** $6n$ **do** $z \leftarrow 2z - 1$
  for $i \leftarrow 1$ **to** $2n^2$ **do**
    for $j \leftarrow 1$ **to** $n + 1$ **do** $z \leftarrow z^2 - z$

The first loop makes $6n$ iterations, the second double loop makes $2n^2 \cdot (n+1) = 2n^3 + 2n^2$ iterations, for the total of $2n^3 + 2n^2 + 6n$ iterations. For $n \geq 1$ this is at most $2n^3 + 2n^3 + 6n^3 \leq 10n^3$, so the running time is $O(n^3)$.

Reference functions. The main reason for using the asymptotic notation is that it allows us to estimate a complicated or difficult to determine function in terms of a simple looking function, for example:

$$197n^5 + 13n^3 - n^2 + 21 \;=\; O(n^5).$$

So when we write $f(n) = O(g(n))$, we typically use some simple functions $g(n)$, whose behavior is well-understood. Thus, although it would be perfectly correct to write

$$n^5 \;=\; O(197n^5 + 13n^3 - n^2 + 21),$$

it would also be completely pointless, as n^5 is a much simpler function than $197n^5 + 13n^3 - n^2 + 21$.

These are four types of functions that are most commonly used in the big-O notation: 1, $\log n$, n^b, c^n, where $b > 0$ and $c > 1$. Occasionally powers of logarithms are used as well, that is functions $\log^a n$, for $a > 0$. In almost all applications, the asymptotic values are expressed in terms of these functions. For a warmup, we first compare functions n and 2^n.

Claim: $2^n \geq n + 1$ for all integers $n \geq 1$.

Indeed, this can be easily shown by induction. For $n = 1$, both sides are equal 2. For $n > 1$, assume the claim holds, that is $2^n \geq n + 1$. Then $2^{n+1} = 2 \cdot 2^n \geq 2(n + 1) \geq n + 2$. By the principle of induction, the claim holds.

This claim can be generalized to real numbers x, that is $2^x \geq x + 1$ for all real numbers $x \geq 1$. To see this, note first that it holds for $x = 1$. The derivative of the right-hand side is 1. The derivative of the left-hand side is $(2^x)' = 2^x \ln 2 > 1$ for $x \geq 1$. So the left-hand side grows faster than the right-hand side, and the inequality follows.

As a conclusion, by taking logarithms of both sides, we also get that $n > \log n$ for $n \geq 1$. So we have $n = O(2^n)$ and $\log n = O(n)$.

Logarithms. Consider first logarithmic functions, $f(n) = \log_r n$, where $r > 1$. We know that any $p > 1$

$$\log_r n \;=\; \frac{\log_p n}{\log_p r} \;=\; \frac{1}{\log_p r} \log_p n.$$

What this formula implies is that all logarithmic functions are within a constant factor of each other, so for the purpose of asymptotic notation *it does not matter what base we use*. In particular, we can as well use base 2. So we have $\log_r n = O(\log n)$ for all $r > 1$.

Polynomials. Polynomials are among the most common functions that appear in the analysis of algorithms. So we deal with polynomial functions first.

Theorem 0.2 *Let $f(x) = \sum_{i=0}^{k} a_i x^i$. Then $f(x) = O(x^k)$.*

Proof: Let $A = \max |a_i|$, be the maximum absolute value of the coefficient in $f(x)$. We can estimate $f(x)$ as follows. For $x \geq 1$ we have

$$\begin{aligned} f(x) &= a_k x^k + a_{k-1} x^{k-1} + ... + a_1 x + a_0 \\ &\leq A(x^k + x^{k-1} + ... + x + 1) \\ &\leq A(k + 1)x^k. \end{aligned}$$

Thus $f(x) \leq cx^k$ for $c = A(k + 1)$ and $x \geq 1$. The theorem follows. □

Theorem 0.3 *For all $a, b > 0$ and $c > 1$, we have:* (a) $1 = O(\log^a n)$. (b) $\log^a n = O(n^b)$. (c) $n^b = O(c^n)$.

Proof: Part (a) is obvious. We'll skip the proof of (b). We will now prove part (c). Take $d = c^{1/b}$. Since $c > 1$ and $b > 0$, we have $d > 1$. We then have

$$\begin{aligned} n &\leq 1 + d + d^2 + ... + d^{n-1} \\ &= \frac{d^n - 1}{d - 1} \\ &\leq Ad^n, \end{aligned}$$

where $A = 1/(d-1)$. The first inequality follows from $d > 1$ (which implies that $d^i > 1$ for all i). The middle equation uses the summation of the geometric sequences, and the in the last inequality we simply dropped the -1 in the numerator.

Now, recall that $d = c^{1/b}$. The derivation above implies then that $n \leq Ac^{(1/b)n}$, which is equivalent (raising both sides to the power of b) to $n^b \leq Bc^n$ for $B = A^b$. We conclude that $n^b = O(c^n)$, as claimed in (c). □

In particular, for example, we have $n^{100} = O(1.0000001^n)$. The left-hand side is a fast-growing polynomial. The right-hand side is a slow-growing exponential function. Still, eventually, the second function will overtake the first one.

Combinations of functions. Many functions can be expressed as sums or products of other more elementary functions. To determine the growth rate of some functions, we can often do this by finding the growth rate of some of their components and using the theorem below.

Theorem 0.4 *Suppose that $f_1(x) = O(g_1(x))$ and $f_2(x) = O(g_2(x))$. Then*
(a) $f_1(x) + f_2(x) = O(\max(g_1(x), g_2(x)))$. In fact, we also have $f_1(x) + f_2(x) = O(g_1(x) + g_2(x))$.
(b) $f_1(x)f_2(x) = O(g_1(x)g_2(x))$.

As a corollary, we get the following: if $f_1(x), f_2(x) = O(g(x))$ then $f_1(x) + f_2(x) = O(g(x))$ as well. Proofs and examples in the book.

Exercise 1: Prove that the O-relation is transitive, in the following sense: if $f(x) = O(g(x))$ and $g(x) = O(h(x))$ then $f(x) = O(h(x))$.

Exercise 2: We showed that all logarithms have the same rate of growth. What about exponential functions? More specifically, for any $a, b > 1$, is it always true that $a^x = O(b^x)$? Justify your answer.

Example 7. Determine the asymptotic value of function $f(n) = 5n^2 \log^6 n + n^3 \log n + 2\sqrt{n} + 1$. We can ignore all constant coefficients. Then we eliminate low order terms one by one. First, we can discard 1, because $1 = O(n^3 \log n)$. Next, let's look at $\sqrt{n}$. We have $\sqrt{n} = n^{1/2} = O(n^3)$, so $\sqrt{n} = O(n^3 \log n)$ as well. Finally, we compare $n^2 \log^6 n$ and $n^3 \log n$. Since $\log^5 n = O(n)$, multiplying both sides by $n^2 \log n$ we get $n^2 \log^6 n = O(n^3 \log n)$. Putting it all together, we get $f(n) = O(n^3 \log n)$.

Example 8. Determine the asymptotic value of function $f(n) = 7n^5 2^n + 3^n$. Again, we can ignore the constant coefficients. Comparing the two terms, the intuition is that 3^n grows much much faster than 2^n, even if we multiply 2^n by a polynomial. For a rigorous proof, divide both terms by 2^n, getting

$$\begin{aligned} n^5 2^n / 2^n &= n^5 \quad \text{and} \\ 3^n / 2^n &= (1.5)^n. \end{aligned}$$

We know that $n^5 = O(1.5^n)$. Now, multiplying it back by 2^n, we obtain $n^5 2^n = O(3^n)$. Therefore $f(n) = O(3^n)$.

Big-Ω and Θ notations. Notation $f(x) = O(g(x))$ means that $f(x)$ does not grow faster than $g(x)$, but it could be the case that $f(x)$ actually grows slower. For example, we have $n^2 + 3n + 1 = O(n^3)$, or $n^2 + 3n + 1 = O(2^n)$, etc. But these bounds are "wasteful", because we also have $n^2 + 3n + 1 = O(n^2)$, which is a more accurate and, in fact, the best possible estimate. This motivates the introduction of other notations for asymptotic growth of functions.

Definition. We say that $f(x)$ is $\Omega(g(x))$, and write $f(x) = \Omega(g(x))$, if there are constants C and k such that $|f(x)| \geq C|g(x)|$ for all $x > k$.

We have $f(x) = O(g(x))$ if and only if $g(x) = \Omega(f(x))$. (Easy exercise.)

We write $f(x) = \Theta(g(x))$ if $f(x) = O(g(x))$ and at the same time $f(x) = \Omega(g(x))$. Thus the Θ-notation provides the "exact" (asymptotic) estimate for function growth.

In other words, $f(x) = \Theta(g(x))$ if there are constants C_1, C_2, and k such that $C_1|g(x)| \leq f(x) \leq C_2|g(x)|$ for all $x > k$.

See examples in the book.

Example 9. Consider function $f(n) = 3n^2+2n-7$. We start with an upper bound: $f(n) = 3n^2+2n-7 \leq 3n^2 + 2n^2 = 5n^2$, so $f(n) = O(n^2)$. Next, we get a lower-bound estimate. For $n \geq 4$ we have $2n \geq 7$, so $f(n) = 3n^2 + 2n - 7 \geq 3n^2$. Therefore $f(n) = \Omega(n^2)$. Putting these two bounds together, we obtain that $f(n) = \Theta(n^2)$.

Example 10. Consider harmonic numbers again, $H_n = 1 + \frac{1}{2} + ... + \frac{1}{n}$. We proved earlier an upper bound $H_n \leq \log n + 1$, which implies $H_n = O(\log n)$.

But we also have a lower bound $H_n \geq \frac{1}{2}\log n - \frac{1}{2}$. Notice that $\frac{1}{4}\log n \geq \frac{1}{2}$ for $n \geq 4$. So for $n \geq 4$ we can estimate H_n as follows: $H_n \geq \frac{1}{2}\log n - \frac{1}{2} = \frac{1}{4}\log n + (\frac{1}{4}\log n - \frac{1}{2}) \geq \frac{1}{4}\log n$. This implies that $H_n = \Omega(\log n)$, and we can thus conclude that $H_n = \Theta(\log n)$.

Example 11. Let us consider again an algorithm consisting of a number of loops. Our goal is to determine its running time, using the asymptotic notation Θ. In the previous example, Algorithm WhatsMyRunTime1, the running time was $O(n^3)$, and in fact this was the best estimate, so the running time was actually $\Theta(n^3)$.

To make it more concrete, let us determine the number of words (rather than the running time) printed by the following pseudo-code:

Algorithm HowManyHellos (n : **integer**)
  for $i \leftarrow 1$ **to** $6n$ **do** print("HELLO")
  for $i \leftarrow 1$ **to** $4n + 1$ **do**
    for $j \leftarrow 1$ **to** $3i + 2$ **do** print("HELLO")

The first loop will print $6n$ words, that's easy. What about the second nested loop? Unlike in Algorithm WhatsMyRunTime1, we cannot simply multiply the ranges of the two loops, because the number of iterations in the inner loop depends on i; it changes from one iteration to next. For any given i, it will print $3i + 2$ words. Therefore we can write the total number of words using the summation notation as $6n + \sum_{i=1}^{4n+1}(3i + 2)$. Simplifying, and using the formula for the sum of an arithmetic sequence, we get

$$\begin{aligned} 6n + \sum_{i=1}^{4n+1}(3i+2) &= 6n + \sum_{i=1}^{4n+1}(3i) + \sum_{i=1}^{4n+1} 2 \\ &= 6n + 3\sum_{i=1}^{4n+1} i + 2(4n+1) \\ &= 6n + 3(4n+1)(4n+2)/2 + 2(4n+1) \\ &= 24n^2 + 32n + 5 \;=\; \Theta(n^2). \end{aligned}$$