ICS 353–Design and Analysis of Algorithms

Asymptotic Notations

Landau symbols

The following asymptotic notations are sometimes called *Landau symbols.* It is not difficult to see that these notations are very useful for comparing the performances of algorithms with respect to the consumed time and space. That is, they are used to describe the complexity classes of algorithms. We will use mainly the following five symbols.

Definition 1. Let $f(n)$ and $g(n)$ be any two functions mapping $\mathbb{N} \to (0, \infty)$. Then

1. **Big-Oh:** $f(n) = O(g(n))$ if and only if there is $n_1 \in \mathbb{N}$ and a constant $c_1 > 0$ such that
$$f(n) \le c_1 g(n), \qquad \text{for all} \quad n \ge n_1 \,.$$
2. **Big-Omega:** $f(n) = \Omega(g(n))$ if and only if there is $n_2 \in \mathbb{N}$ and a constant $c_2 > 0$ such that
$$f(n) \ge c_2 g(n), \qquad \text{for all} \quad n \ge n_2 \,.$$
3. **Theta:** $f(n) = \Theta(g(n))$ if and only if there is $n_0 \in \mathbb{N}$ and two constants $c_1, c_2 > 0$ such that
$$c_1 g(n) \le f(n) \le c_2 g(n), \qquad \text{for all} \quad n \ge n_0 \,.$$
4. **Small-oh:** $f(n) = o(g(n))$ if and only if
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \,.$$
5. **Small-omega:** $f(n) = \omega(g(n))$ if and only if
$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0 \,.$$

Remarks

Notice the following.

1. The **Big Oh** is an *upper bound* of a function. We will regularly use it to bound the *worst-case (or the maximum) running time* from above.
2. On the other hand, the **Big Omega** is a *lower bound* of a function. Usually we use it to obtain a lower bound on the *best-case (or the minimum) running time* of the algorithm.

3. The **Theta** combines both symbols the Big-Oh and the Big-Omega. Notice that if $f(n) = \Theta(g(n))$ then $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ at the same time. This means both functions are of the same order; in fact they both have the same main term but possibly with different multiplicative constants. For example, if $f(n) = 3n^2-3n+1$ and $g(n) = 5n^2+6n-4$ then $f(n) = \Theta(g(n)) = \Theta(n^2)$. One way to check that is to make sure that the limit $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ exists and it is a constant:
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{3n^2-3n+1}{5n^2+6n-4} = \lim_{n\to\infty} \frac{3-3/n+1/n^2}{5+6/n-4/n^2} = \frac{3}{5}\,.$$

4. The **small-oh** says that $g(n)$ is much bigger than $f(n)$ to the degree that $g(n)$ goes to infinity faster than $f(n)$, as $n \to \infty$.

5. The **small-omega** says that $g(n)$ is much smaller than $f(n)$ to the degree that $f(n)$ goes to infinity faster than $g(n)$, as $n \to \infty$.

6. In all of these cases, we only need to prove these statements for n large enough (i.e., for $n \geq n_0$). That's why it is called *asymptotic notations.*

7. The constants c_1 and c_2 are all hidden within the notations because they are not important at this stage. Suppose for example we know that the running time of a given algorithm is $O(n^2)$ (as in InsertionSort algorithm). Say we run the algorithm with $n = 1000$, and we find that it takes 4 seconds. If we want to find the time for $n = 10^6$, we don't have to run the algorithm over again. Since the time grows quadratically in this case, the estimated time for the problem should be close to $(1000)^2 \times 4$ seconds, or 4 million seconds. Had this algorithm been linear $O(n)$, the time would have been 1000×4 or 4000 seconds only! In both of these cases (quadratic and linear), the constant c was not needed.

8. Thus, when we analyze the performance of algorithms we should make our analysis independent from the type of machine and technology. Thus,
   - We should concentrate on the asymptotic performances (for large input size n),
   - We should concentrate on the main term and ignore the smaller ones and the constant factors,
   - The constant factors and other smaller terms are useful only to compare between two algorithms that have the same order of running time.

Examples

1. Let $f(n) = 35n$ and $g(n) = 2n + 3$. Then $f(n) = \Theta(g(n))$ because $1 \times g(n) \leq f(n) \leq 20 \times g(n)$, for all $n \geq 1$.

2. Similarly, for any constants $a > 0$ and b we have $f(n) = an + b = \Theta(n)$. Notice also that $f(n) = O(n^2) = O(n^3) = O(n^k)$ for any $k > 1$ because $an + b \leq (a + b)n^k$, for any $k > 1$ and $n \geq 1$.

3. If $f(n) = 5n^2 - 6n + 3$ and $g(n) = 2n + 8$, then $f(n) = \omega(g(n))$ and $g(n) = o(f(n))$ because
$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{2n+8}{5n^2 - 6n + 3} = \lim_{n\to\infty} \frac{2 + 8/n}{5n - 6 + 3/n} = 0\,.$$

4. Clearly, $\log n = O(n)$ and $n = \Omega(\log n)$. In fact, $\log n = o(n)$ and $n = \omega(\log n)$ because by L'Höpital's rule
$$\lim_{n\to\infty} \frac{\log n}{n} = 0\,.$$
In fact, if $c \in (0, 1)$ is any constant than $\log n = o(n^c)$ by applying the same rule.

5. Notice also that if k is any positive constant then $\log n^k = k \log n = o(n)$ and $n + \log n^k = \Theta(n)$ because $n \leq n + k \log n \leq 2kn$.

6. Similarly, $n + \sqrt{n} \log n = \Theta(n)$ because $\log n \leq \sqrt{n}$ and hence $n \leq n + \sqrt{n} \log n \leq 2n$.

7. Clearly, for any constant c we have $n = \omega(n^c) = \omega(\log n) = \omega(\log \log n) = \omega(1)$.

8. For any constant $c > 0$, we have $c^n = O(n!) = O(n^n)$ and $\log n! = \Theta(n \log n)$. See also Examples 1.12- 1.14 in the textbook.

9. We also use the notation $f(n) \prec g(n)$ to mean that $f(n) = o(g(n))$; that is the two functions are of different complexity classes. For example,
$$1 \prec \log^* n \prec \log\log n \prec \sqrt{\log n} \prec \frac{\log n}{\log\log n} \prec \log n \prec \sqrt{n} \prec n \prec n \log n \prec n^2 \prec e^n \prec n! \prec n^n\,.$$

10. Notice that if the running time of some algorithm is $T_n = \Theta(n)$, then the main term of T_n is cn for some constant $c > 0$ and all other terms in T_n are $o(n)$. For example, T_n could be one of the following.

    - $T_n = n/2 - \log n$
    - $T_n = 5n + 2\sqrt{n} - n^{5/7} \log n$
    - $T_n = \frac{7n^5 - 3n^3 + 2n}{4n^4 + \sqrt{n} - \log n} = \frac{n^5(7 - 3/n^2 + 2/n^4)}{n^4(4 + 1/n^{7/2} - (\log n)/n^4)}$