# 1 Asymptotic notation

The order of growth of the running time of an algorithm provides a good impression of an algorithm's efficiency. In this course, we shall often make use of asymptotic notation to compare the performances of various algorithms that are used to solve structured problems in computational linear algebra. To this end, this section reviews some common notation used to describe asymptotic growth of functions.

Interpreting the variable $n \in \mathbb{N}$ as the "size" of a specific problem instance, we recall the following well-known definitions:

$$\begin{aligned}
\Omega(f(n)) &= \{f(n) : \exists c, n_0 > 0 \text{ such that } 0 \le cg(n) \le f(n), \forall n > n_0\} \\
O(f(n)) &= \{f(n) : \exists c, n_0 > 0 \text{ such that } 0 \le f(n) \le cg(n), \forall n > n_0\} \\
\Theta(f(n)) &= \{f(n) : \exists c_1, c_2, n_0 > 0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n), \forall n > n_0\} \\
\omega(f(n)) &= \{f(n) : \exists \forall c > 0, \exists n_0 > 0 \text{ such that } 0 \le cg(n) \le f(n), \forall n > n_0\} \\
o(f(n)) &= \{f(n) : \exists \forall c > 0, \exists n_0 > 0 \text{ such that } 0 \le f(n) \le cg(n), \forall n > n_0\}
\end{aligned}$$

The notation $f(n) \in \Omega(g(n))$ ($f(n) \in O(g(n))$ ) describes an asymptotic lower (upper) bound for $f(n)$ in terms of $g(n)$ since it roughly states that the asymptotic growth of $f(n)$ is greater (less) or equal to the asymptotic growth of $g(n)$. On the other hand, $f(n) \in \Theta(g(n))$ describes an asymptotic tight bound for $f(n)$ in terms of $g(n)$ since it roughly states that the growth rates of $f(n)$ and $g(n)$ are about equal in the asymptotic sense. The aforementioned statements can be likened to the symbols "$\ge$" ("$\le$") and "=", respectively. Notice also that, similar to the statement "$a = b$ if and only if $a \le b$ and $a \ge b$", we have $f(n) \in \Theta(g(n))$ if, and only if, $f(n) \in \Omega(g(n))$ and $g(n) \in \Theta(f(n))$.

The notation $f(n) \in \omega(g(n))$ ($f(n) \in o(g(n))$ ) also describes an lower (upper) bound on the growth rate of $f(n)$ in terms $g(n)$, however this bound is not asymptotically tight. Since $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ ( $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$), the growth of $f(n)$ is strictly greater (less) than that of $g(n)$ , and hence, an analogy can be drawn with the symbol "$>$" ("$<$").

# 2 Complexity of divide-and-conquer algorithms

Divide-and-conquer is a frequently used technique for designing fast matrix computation algorithms. Strassen, FFT, fast polynomial division, HSS algorithms all employ elements of divide-and-conquer strategies. To evaluate the performance of these algorithms, it is paramount to understand how their complexity is evaluated. Central to this evaluation is solving a recurrence of the form

$$T(n) = \begin{cases} C & \text{if } n = 1 \\ aT(n/b) + f(n) & \text{if } n > 1 \end{cases}, \tag{1}$$

where the integers $a \ge 1$ and $b > 1$, and $C > 0$ is a positive real constant. Here, $T(n)$ describes the time complexity of the divide-and-conquer algorithm for a specific problem, and the recurrence states that the algorithms "splits" the problem into $a$ smaller problems (of the same kind) with size $n/b$ and then re-assembles the solution from the solutions of these smaller problems in $f(n)$ cost. We will discuss two methods for solving (1) to subsequently obtain a time complexity estimate of a divide-and-conquer algorithm.

The *first method* is that of guessing the solution and verifying the guess through substitution. We shall illustrate this through an example. Consider the recursion:

$$T(n) = 2T(n/2) + n.$$

We make the educated guess that $T(n) \in O(n \log n)$, i.e., there exists a $c > 0$ for which $T(n) \le cn \log n$.

Substitution yields

$$\begin{aligned} T(n) &\le 2T(n/2) + n \\ &= 2\left(cn/2\log n/2\right) + n \\ &= cn\log(n) - cn\log 2 + n \\ &\le cn\log(n) \qquad \text{if } c > 1/\log 2, \end{aligned}$$

which indeed confirms that $T(n) \in O(n\log n)$. Likewise, we can show that $T(n) \in \Omega(n\log n)$, i.e., there exists a $c > 0$ for which $T(n) \ge cn\log n$. Substitution gives this time around

$$\begin{aligned} T(n) &\ge cn\log(n) - cn\log 2 + n \\ &\ge cn\log(n) \qquad \text{if } 0 < c < 1/\log 2, \end{aligned}$$

which indeed shows that $T(n) \in \Omega(n\log n)$. Subsequently, we have confirmed that $T(n) \in \Theta(n\log n)$. Finding the right guess for a specific recursion can be hard at times, especially if one would like to obtain an asymptotically tight bound. To build intuition, one could resort to drawing a recursion tree to get an idea of how $T(n)$ grows; see e.g., the book of Cormen, Leiserson, Rivest and Stein for more details.

The *second method* is to invoke a general result on (1) to get a asymptotic complexity bound on $T(n)$. As it turns out, most (but not all) recursions fall into one of three categories described by the so-called *master theorem* here below.

**Theorem 1.** *Consider the recursion described in* (1)*. If*

1. $f(n) \in O\left(n^{\log_b a - \epsilon}\right)$ *for some* $\epsilon > 0$*, then* $T(n) \in \Theta\left(n^{\log_b a}\right)$.
2. $f(n) \in \Theta\left(n^{\log_b a}\right)$*, then* $T(n) \in \Theta\left(n^{\log_b a}\right)$.
3. $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ *for some* $\epsilon > 0$ *and if* $af(n/b) \le cf(n)$ *for some* $0 < c < 1$ *and sufficiently large* $n$*, then* $T(n) \in \Theta(f(n))$.

*Remark.* The statements in Theorem 1 still holds if $n/b$ in (1) is replaced with either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$.

We shall not formally prove the master theorem here, but we will however elobarate on where the result comes from so that a proper intuition can be built. To start off, let us consider the case where $f(n) = 0$, i.e., the cost to re-assemble the solution from the sub-problems is zero. If we assume that $n$ is a power of $b$, i.e., $n = b^L$ with $L = \log_b n$, we easily derive that

$$T(n) = T(b^L) = aT(b^{L-1}) = a^2T(b^{L-1}/2) = \ldots = a^LT(1) = a^{\log_b n}C = Cn^{\log_b a}.$$

For the more general case wherein $f(n) \neq 0$, repeated substitution yields on the other hand

$$\begin{aligned} T(n) &= aT(b^{L-1}) + f(b^L) \\ &= a^2T(b^{L-2}) + aT(b^{L-1}) + f(b^L) \\ &\vdots \\ &= a^LT(1) + \sum_{k=0}^{L-1} a^k f(b^{L-k}), \end{aligned}$$

leading to the expression

$$T(n) = Cn^{\log_b a} + \sum_{k=0}^{L-1} a^k f\left(\tfrac{n}{b^k}\right). \tag{2}$$

The asymptotic complexity of $T(n)$ is thus determined by which of the two terms, i.e., $Cn^{\log_b a}$ or $\sum_{k=0}^{L-1} a^k f(b^{L-k})$, dominates in (2). For each of the cases in Theorem 1, the following can be said.

1. If $f(n) \in O\left(n^{\log_b a-\epsilon}\right)$, we have

$$\begin{aligned}
T(n) &\leq Cn^{\log_b a} + D\sum_{k=0}^{L-1} a^k \left(\tfrac{n}{b^k}\right)^{\log_b a-\epsilon} \\
&= Cn^{\log_b a} + Dn^{\log_b a-\epsilon}\sum_{k=0}^{L-1} \left(\tfrac{a}{b^{(\log_b a-\epsilon)}}\right)^k \\
&= Cn^{\log_b a} + Dn^{\log_b a-\epsilon}\sum_{k=0}^{L-1} \left(b^{\epsilon}\right)^k \\
&= Cn^{\log_b a} + \frac{D}{1-b^{\epsilon}} n^{\log_b a-\epsilon}\left(1-\left(b^{\epsilon}\right)^L\right) \\
&= Cn^{\log_b a} + \frac{D}{1-b^{\epsilon}} n^{\log_b a-\epsilon}\left(1-n^{\epsilon}\right) \\
&= \left(C-\frac{D}{1-b^{\epsilon}}\right) n^{\log_b a} + \frac{D}{1-b^{\epsilon}} n^{\log_b a-\epsilon}.
\end{aligned}$$

   Since we also have $T(n) \geq Cn^{\log_b a}$, we thus have that $T(n) \in \Theta\left(n^{\log_b a}\right)$.

2. If $f(n) \in \Theta\left(n^{\log_b a}\right)$, there exists constants $C_1, C_2 > 0$ such that $C_1 n^{\log_b a} \leq f(n) \leq C_2 n^{\log_b a}$. Hence,

$$\begin{aligned}
Cn^{\log_b a} + C_1\sum_{k=0}^{L-1} a^k \left(\tfrac{n}{b^k}\right)^{\log_b a} &\leq T(n) &&\leq Cn^{\log_b a} + C_2\sum_{k=0}^{L-1} a^k \left(\tfrac{n}{b^k}\right)^{\log_b a} \\
Cn^{\log_b a} + C_1 n^{\log_b a}\sum_{k=0}^{L-1} \left(\tfrac{a}{b^{(\log_b a)}}\right)^k &\leq T(n) &&\leq Cn^{\log_b a} + C_2 n^{\log_b a}\sum_{k=0}^{L-1} \left(\tfrac{a}{b^{(\log_b a)}}\right)^k \\
Cn^{\log_b a} + C_1 n^{\log_b a}\sum_{k=0}^{L-1} 1 &\leq T(n) &&\leq Cn^{\log_b a} + C_2 n^{\log_b a}\sum_{k=0}^{L-1} 1 \\
Cn^{\log_b a} + C_1 n^{\log_b a}(\log_b n - 1) &\leq T(n) &&\leq Cn^{\log_b a} + C_2 n^{\log_b a}(\log_b n - 1)
\end{aligned}$$

   which shows that $T(n) \in \Theta\left(n^{\log_b a}\log n\right)$.

3. If $f(n) \in \Omega\left(n^{\log_b a+\epsilon}\right)$, it is relatively straightforward to see that $T(n) \in \Omega\left(f(n)\right)$. If we additionally assume that $af(n/b) \leq cf(n)$, we get

$$\begin{aligned}
T(n) &\leq Cn^{\log_b a} + \sum_{k=0}^{L-1} c^k f(n) \\
&\leq Cn^{\log_b a} + \frac{f(n)}{1-c}
\end{aligned}$$

   which also shows that $T(n) \in O\left(f(n)\right)$. Hence, $T(n) \in \Theta\left(f(n)\right)$.

Note that there are some gaps in the results of Theorem 1. For instance, the scenario wherein $f(n) \in o\left(n^{\log_b a}\right)$, but $f(n) \notin O\left(n^{\log_b a-\epsilon}\right)$ is not addressed. Fortunately, for the algorithms that we will discuss, we will not encounter these special cases.