2.1 Growth of Functions

Previously, we calculated the running time for insertion sort as $an^2 + bn + c$ for an input of size n for some constants a, b, and c. This quadratic runtime is also written as $\Theta(n^2)$ or $O(n^2)$. The running time function, $T(n) = f(n)$, is a function of size of input, n, to the algorithm. Meanwhile the asymptotic notations are primarily to describe the order of growth of $f(n)$. For instance, $\Theta(n^2)$ or $O(n^2)$ for the insertion sort is ignoring some details of the function, and merely focusing on the highest order of the function. This practice helps us capture the efficiency of the function. The value of a function is highly dominated by the term of with the highest order. In a polynomial function of x, $a_n x^n + a_{n-1}x^{n-1} + ... + ax + a_0$, the output of this function is mostly determined by the term $a_n x^n$ as x increases. You can run a few calculation for a function like $2x^2 + 500x + 10000$ and notice that the value of $2x^2 >> 500x + 10000$ as x grows. This also helps us classify algorithms with similar order of growth. If two algorithms have a running time of $50n^2 + 3$ and $0.5n^2 + 20n + 7$, clearly they do not have the exact running time, however, it's completely fair to say that they have a very similar efficiency for higher value of n, as their running time will be **asymptotic to** n^2.

Therefore, asymptotic notations are a great way to summarize the running time of a function. However, we cannot always ignore constants of the highest terms in practice. A huge overhead cost of an algorithm can sometimes be a bottleneck as the size of inputs we deal with in real life may not always have to be extremely large.

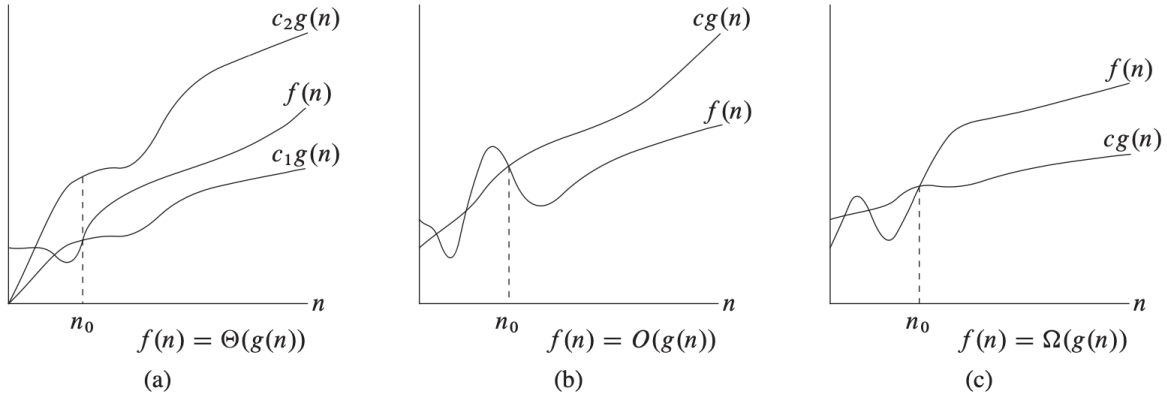

Figure 3.1 Graphic examples of the Θ, O, and Ω notations. In each part, the value of n_0 shown is the minimum possible value; any greater value would also work. **(a)** Θ-notation bounds a function to within constant factors. We write $f(n) = \Theta(g(n))$ if there exist positive constants n_0, c_1, and c_2 such that at and to the right of n_0, the value of $f(n)$ always lies between $c_1 g(n)$ and $c_2 g(n)$ inclusive. **(b)** O-notation gives an upper bound for a function to within a constant factor. We write $f(n) = O(g(n))$ if there are positive constants n_0 and c such that at and to the right of n_0, the value of $f(n)$ always lies on or below $cg(n)$. **(c)** Ω-notation gives a lower bound for a function to within a constant factor. We write $f(n) = \Omega(g(n))$ if there are positive constants n_0 and c such that at and to the right of n_0, the value of $f(n)$ always lies on or above $cg(n)$.

2.1.1 Asymptotic notation

Let's describe different asymptotic notations precisely.

Θ-notation

$\Theta(g(n)) = \{f(n) :$ there exists positive constants $c_1, c_2,$ and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0\}$.

A function $f(n)$ belongs to the set $\Theta(g(n))$ if there exist positive constants c_1 and c_2 such that it can be "sandwiched" between $c_1 g(n)$ and $c_2 g(n)$, for sufficiently large n. Note that $\Theta(g(n))$ is a **set** of functions, such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$. Therefore, to be precise "$f(n) \in \Theta(g(n))$", however, we usually write $f(n) = \Theta(g(n))$.

Figure 3.1(a) provides are very intuitive idea about $f(n) = \Theta(g(n))$. For $f(n) = \Theta(g(n))$, we are looking for $n \geq n_0$ where we can find $c_1 g(n)$ and $c_2 g(n)$ for c_1 and c_2 that can "sandwich" $f(n)$. We say that $g(n)$ is an ***asymptotically tight bound*** for $f(n)$.

Let's use the formal definition to show $f(n) = \frac{1}{2}n^2 - 3n = \Theta(n^2)$. To do so, we must determine positive constants c_1, c_2 and n_0 such that

$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$$

for all $n \geq n_0$. Dividing by n^2 yields

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2.$$

For the right-hand inequality to hold, for any $n \geq 1$, we can always choose $c_2 \geq \frac{1}{2}$. For the left-hand inequality to hold, we can choose $n \geq 7$ for any constant $c_1 \leq \frac{1}{14}$. We can choose the value of n_0 that holds for both the constants, which is $n_0 \geq 7$. For $c_1 = \frac{1}{14}$, $c_2 = \frac{1}{2}$, and $n_0 = 7$, we can verify that $\frac{1}{2}n^2 - 3n = \Theta(n^2)$. As long as we can find *some* positive constants c_1, and c_2 for $n \geq n_0$ that holds the definition above, we say that $g(n)$ is an asymptotically tight bound for $f(n)$.

How about $9n^3 = \Theta(n^2)$? If we follow the same formal definition above:

$$c_1 n^2 \leq 9n^3 \leq c_2 n^2$$

Dividing by n^2 yields

$$c_1 \leq 9n \leq c_2.$$

We can clearly see that we can never pick c_2 because $9n$ will always exceed the value of c_2 as n becomes arbitrarily large. Thus, $9n^3 \neq \Theta(n^2)$.

O-notation

You may have heard about "big-Oh" notation or O-notation several times. In fact, in most technical interviews, interviewees are required to state the O-notation for the solution program they write, and rightly so. Here is the formal definition of O-notation:

$O(g(n)) = \{f(n) :$ there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$.

Unlike, $\Theta(g(n)$ where we need both upper and lower bounds, here we are mostly interested in the upper bound that can contain $f(n)$ with some factor, c, of $g(n)$ for all $n \geq n_0$. In other words, we say that $g(n)$ is an ***asymptotic upper bound*** of $f(n)$. You can see the same idea is demonstrated in Fig 3.1.

From the definitions we can also see that Θ-notation is a bit stricter as it requires two bounds to contain $f(n)$ while O requires upper bound only. Thus, we can say that $f(n) = \Theta(g(n))$ implies $f(n) = O(g(n))$.

One must note that O-notation is defined under two constraints, c and n_0. Therefore, $O(n^2)$ is considered better (or worse) than $O(n^3)$ based on these two constraints.

Let's look at the value of c in two different functions. For instance, if $O(n^2)$ is for $f_1(n) = 1000000n^2$ and $O(n^3)$ is for $f_2(n) = 2n^3$, f_1 will only perform better than f_2 for a large value of n only.

Similarly, O-notation tells us nothing about the asymptotic bounds for the range $n < n_0$, meaning, for $n < n_0$, $f(n)$ might be way worse than $cg(n)$. Therefore, for a statement like $f(n) = O(n^2)$ for $n \geq 20$, a smaller input than 20 may take higher runtime than quadratic order of n.

Ω-notation

$\Omega(g(n)) = \{f(n) :$ there exist positive constants c and n_0 such that $0 \leq cg(n) \leq f(n)\ \forall n \geq n_0\}$

The "big-omega of g of n" is essentially a measure of lower asymptotic bound of $f(n)$. $f(n) = \Omega(n)$ implies that $f(n)$ takes **at least** cn runtime for a constant c and for all $n \geq n_0$.

Now that we know the definitions of Θ-notation, O-notation, and Ω-notation, can you prove Theorem 3.1 (pg 48 CLRS)?

2.2 Class Exercise

1. Prove that $f(n) = n^2 + 4n + 5$ is $O(n^2)$.
2. If $f(n) = O(n)$, then $f(n) = O(n^3)$. Explain.
3. If a function is $O(n)$, can it also be $\Omega(n^2)$?