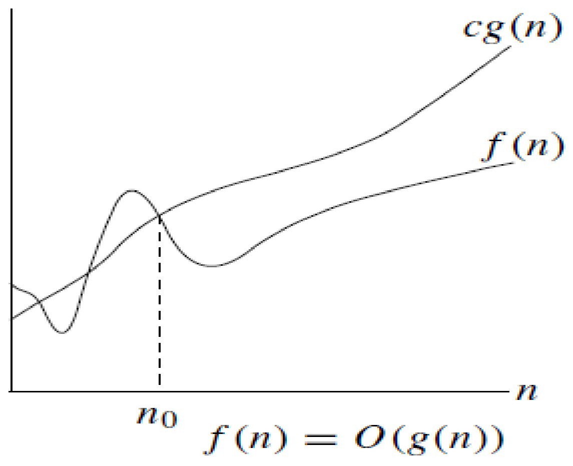
O-notation

- For a given function g(n), we denote by O(g(n)) (pronounced "big-oh of g of n") the set of functions:

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

- A function f (n) belongs to the set *O(g(n))* if there exist positive constant c such that *f(n)* is on or below c*g(n)* for sufficiently large $n \ge n_0$. (as shown in below figure)



- In other words, for all $n \ge n0$, the function *f (n)* is less than or equal to *g(n)* to within a constant factor. We say that g(n) is an **asymptotically upper bound** for f (n).

NOTE:

1) Note that $f(n) = \Theta(g(n))$ implies $f(n) = O(g(n))$, since Θ-notation is a stronger notion than *O*-notation. Written set-theoretically, we have $\Theta(g(n)) \subseteq O(g(n))$.
   Eg: we have seen that quadratic function $an2 + bn + c$, where $a > 0$, is in $\Theta(n^2)$, this implies that any such quadratic function is also in $O(n^2)$.
   Further, any *linear* function $an + b$ is in $O(n^2)$, which can be easily verified.

2) Since O-notation describes an upper bound, when we use it to bound the worst-case running time of an algorithm, we have a bound on the running time of the algorithm on every input. Thus, the $O(n^2)$ bound on worst-case running time of insertion sort also applies to its running time on every input. (But, the $\Theta(n^2)$ bound on the worst-case running time of insertion sort, however, does not imply a $\Theta(n^2)$ bound on the running time of insertion sort

on *every* input. For example, we know that when the input is already sorted, insertion sort runs in Θ *(n)* time.)

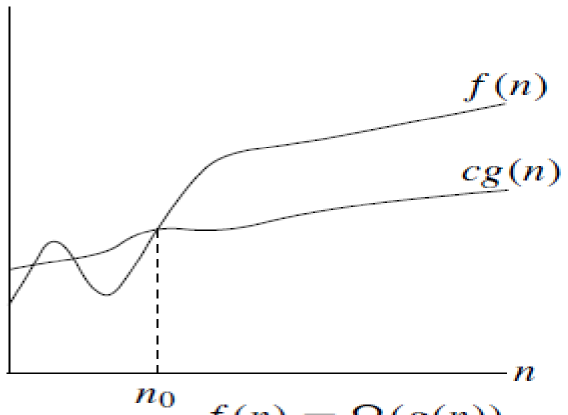
3) Technically, it is an abuse to say that the running time of insertion sort is *$O(n^2)$*, since for a given *n*, the actual running time varies, depending on the particular input of size *n*. But, when we say “the running time is $O(n^2)$,” we mean that there is a function f (n) that is $O(n^2)$ such that for any value of n, no matter what particular input of size n is chosen, the running time on that input is at most a constant time n^2, for sufficiently large n.
⇒ Equivalently, we mean that the worst-case running time is $O(n^2)$.
⇒ or running time is $O(n^2)$.

Ω-notation

- For a given function *g(n)*, we denote by **Ω***(g(n))* the set of functions:

$$\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0\} .$$

- A function f (n) belongs to the set Ω*(g(n))* if there exist positive constant c such that *f(n)* is on or above c*g(n)* for sufficiently large $n \geq n_0$. (as shown in below figure)



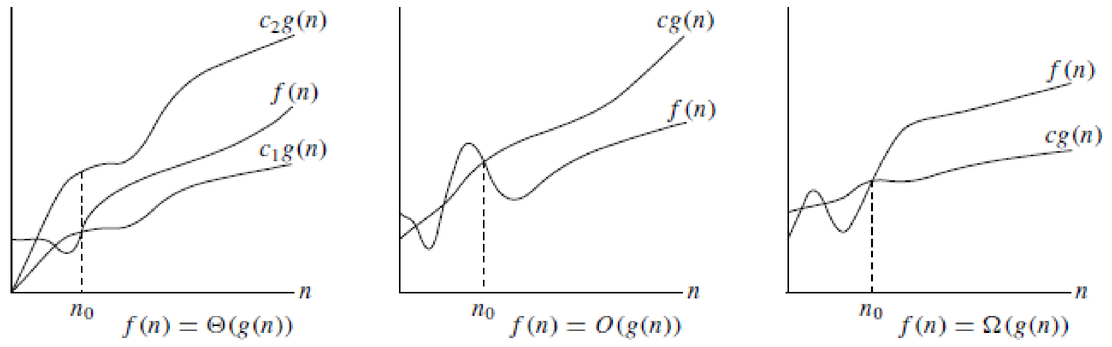
$f(n) = \Omega(g(n))$

- In other words, for all $n \geq n0$, the function *f (n)* is greater than or equal to *g(n)* to within a constant factor. We say that g(n) is an **asymptotically lower bound** for f (n).

NOTE:

1) Since Ω-notation describes a lower bound, when we use it to bound the best-case running time of an algorithm, by implication we also bound the running time of the algorithm on arbitrary inputs as well.
   For example, the best-case running time of insertion sort is Ω*(n).*
2) When we say that the *running time* (no modifier) of an algorithm is Ω *(g(n))*, we mean that *no matter what particular input of size n is chosen for each value of n*, the running time on that input is at least a constant times *g(n)*, for sufficiently large *n*.
   For example, the running time of insertion sort is Ω*(n).*
3) From the definitions of the asymptotic notations we have seen thus far, it is easy to prove the following important theorem:

   For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. ■



Example:

As an example of the application of this theorem, our proof that $an^2 + bn + c = \Theta(n^2)$ for any constants a, b, and c, where $a > 0$, immediately implies that $an^2 + bn + c = \Omega(n^2)$ and $an^2 + bn + c = O(n^2)$. In practice, rather than using Theorem 3.1 to obtain asymptotic upper and lower bounds from asymptotically tight bounds, as we did for this example, we usually use it to prove asymptotically tight bounds from asymptotic upper and lower bounds.