We say "f(n) is Ω(g(n))", or "f(n) is big-Omega of g(n)", if there exists a real constant c > 0 and an integer constant n0 >= 1 such that f(n) >= c.g(n) for every integer n >= n0.

Example 1.9: $3\log n + \log\log n$ *is* $\Omega(\log n)$.

Proof: $3\log n + \log\log n \geq 3\log n$, *for* $n \geq 2$.

We say "f(n) is Θ(g(n))", or "f(n) is big-Theta of g(n)",
if there exists real constants c1, c2 > 0 and an integer constant n0 >= 1
such that f(n) <= c1.g(n) and f(n) >= c2.g(n) for every integer n >= n0.

Example 1.10: $3\log n + \log\log n$ *is* $\Theta(\log n)$.

NB - 1) If f(n) is O(g(n)), then g(n) must be Ω(f(n))
2) If f(n) is Ω(g(n)), then g(n) must be O(f(n))
3) If f(n) is Θ(g(n)), then f(n) must be both O(g(n)) and Ω(g(n))
4) If f(n) is Θ(g(n)), then also g(n) must be Θ(f(n))

Theorem 1.7: *Let* $d(n)$, $e(n)$, $f(n)$, *and* $g(n)$ *be functions mapping nonnegative integers to nonnegative reals.*

1. *If* $d(n)$ *is* $O(f(n))$, *then* $ad(n)$ *is* $O(f(n))$, *for any constant* $a > 0$.
2. *If* $d(n)$ *is* $O(f(n))$ *and* $e(n)$ *is* $O(g(n))$, *then* $d(n)+e(n)$ *is* $O(f(n)+g(n))$.
3. *If* $d(n)$ *is* $O(f(n))$ *and* $e(n)$ *is* $O(g(n))$, *then* $d(n)e(n)$ *is* $O(f(n)g(n))$.
4. *If* $d(n)$ *is* $O(f(n))$ *and* $f(n)$ *is* $O(g(n))$, *then* $d(n)$ *is* $O(g(n))$.
5. *If* $f(n)$ *is a polynomial of degree d (that is,* $f(n) = a_0 + a_1 n + \cdots + a_d n^d$*), then* $f(n)$ *is* $O(n^d)$.
6. n^x *is* $O(a^n)$ *for any fixed* $x > 0$ *and* $a > 1$.
7. $\log n^x$ *is* $O(\log n)$ *for any fixed* $x > 0$.
8. $\log^x n$ *is* $O(n^y)$ *for any fixed constants* $x > 0$ *and* $y > 0$.

---

d(n) = 2n^2 is O(n^2)
e(n) = 4n^3 is O(n^3)
d(n) + e(n) = 2n^2 + 4n^3 is O(n^2 + n^3) = O(n^3)

f(n) = 10^100 . n
g(n) = 0.01 . n^2

efficient => polynomial running time => O(n^k) for some constant k > 0
not efficient => exponential running time => O(2^n)

f(n) = 3 . n^1001
g(n) = 2 . 2^n

It is considered poor taste to include constant factors and lower order terms in the big-Oh notation. For example, it is not fashionable to say that the function $2n^2$ is $O(4n^2 + 6n \log n)$, although this is completely correct. We should strive instead to describe the function in the big-Oh in ***simplest terms***.

logarithmic	**linear**	**quadratic**	**polynomial**	**exponential**
$O(\log n)$	$O(n)$	$O(n^2)$	$O(n^k)$ $(k \geq 1)$	$O(a^n)$ $(a > 1)$

A few words of caution about asymptotic notation are in order at this point. First, note that the use of the big-Oh and related notations can be somewhat misleading should the constant factors they "hide" be very large. For example, while it is true that the function $10^{100}n$ is $\Theta(n)$, if this is the running time of an algorithm being compared to one whose running time is $10n \log n$, we should prefer the $\Theta(n \log n)$-time algorithm, even though the linear-time algorithm is asymptotically faster. This preference is because the constant factor, 10^{100}, which is called "one googol," is believed by many astronomers to be an upper bound on the number of atoms in the observable universe. So we are unlikely to ever have a real-world problem that has this number as its input size. Thus, even when using the big-Oh notation, we should at least be somewhat mindful of the constant factors and lower order terms we are "hiding."

Some Functions Ordered by Growth Rate	Common Name
$\log n$	logarithmic
$\log^2 n$	polylogarithmic
$\sqrt{n}$	square root
n	linear
$n \log n$	linearithmic
n^2	quadratic
n^3	cubic
2^n	exponential

n	$\log n$	$\log^2 n$	$\sqrt{n}$	$n \log n$	n^2	n^3	2^n
4	2	4	2	8	16	64	16
16	4	16	4	64	256	4,096	65,536
64	6	36	8	384	4,096	262,144	1.84×10^{19}
256	8	64	16	2,048	65,536	16,777,216	1.15×10^{77}
1,024	10	100	32	10,240	1,048,576	1.07×10^{9}	1.79×10^{308}
4,096	12	144	64	49,152	16,777,216	6.87×10^{10}	10^{1233}
16,384	14	196	128	229,376	268,435,456	4.4×10^{12}	10^{4932}
65,536	16	256	256	1,048,576	4.29×10^{9}	2.81×10^{14}	10^{19728}
262,144	18	324	512	4,718,592	6.87×10^{10}	1.8×10^{16}	10^{78913}

We say that "f(n) is o(g(n))", or "f(n) is little-oh of g(n)", if for any constant c > 0, there exists a constant n0 > 0 such that f(n) <= c.g(n) for n >= n0.

We say that "f(n) is ω(g(n))", or "f(n) is little-omega of g(n)", if g(n) is o(f(n)).

Example 1.11: *The function* $f(n) = 12n^2 + 6n$ *is* $o(n^3)$ *and* $\omega(n)$.

Proof: *Let us first show that* $f(n)$ *is* $o(n^3)$. *Let* $c > 0$ *be any constant. If we take* $n_0 = (12+6)/c = 18/c$, *then* $18 \le cn$, *for* $n \ge n_0$. *Thus, if* $n \ge n_0$,

$$f(n) = 12n^2 + 6n \le 12n^2 + 6n^2 = 18n^2 \le cn^3.$$

Thus, $f(n)$ *is* $o(n^3)$.

To show that $f(n)$ *is* $\omega(n)$, *let* $c > 0$ *again be any constant. If we take* $n_0 = c/12$, *then, for* $n \ge n_0$, $12n \ge c$. *Thus, if* $n \ge n_0$,

$$f(n) = 12n^2 + 6n \ge 12n^2 \ge cn.$$

Thus, $f(n)$ *is* $\omega(n)$. ■

For the reader familiar with limits, we note that $f(n)$ is $o(g(n))$ if and only if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0,$$

provided this limit exists. The main difference between the little-oh and big-Oh notions is that $f(n)$ is $O(g(n))$ if ***there exist*** constants $c > 0$ and $n_0 \ge 1$ such that $f(n) \le cg(n)$, for $n \ge n_0$; whereas $f(n)$ is $o(g(n))$ if ***for all*** constants $c > 0$ there is a constant n_0 such that $f(n) \le cg(n)$, for $n \ge n_0$. Intuitively, $f(n)$ is $o(g(n))$ if $f(n)$ becomes insignificant compared to $g(n)$ as n grows toward infinity. As previously mentioned, asymptotic notation is useful because it allows us to concentrate on the main factor determining a function's growth.

Example of 'polylogarithmic' function : 11(log n)^4 + 5(log n)^2 + 3(log n) + 2