<http://forums.appleinsider.com/archive/index.php/t-63372.html>

Big-O: The asymptotic worst case performance of an algorithm. The function n happens to be the lowest valued function that will always have a higher value than the actual running of the algorithm. [constant factors are ignored because they are meaningless as n reaches infinity]  
  
Big-Omega. The opposite of Big-O. The asymptotic best case performance of an algorithm. The function n happens to be the highest valued function that will always have a lower value than the actual running of the algorithm. [constant factors are ignored because they are meaningless as n reaches infinity]  
  
Big-Theta. The algorithm is so nicely behaved that some function n can describe both the algorithm's upper and lower bounds within the range defined by some constant value c. An algorithm could then have something like this: BigTheta(n), O(c1n), BigOmega(-c2n) where n == n throughout.  
  
Little-o is like Big-O but sloppy. Big-O and the actual algorithm performance will actually become nearly identical as you head out to infinity. little-o is just some function that will always be bigger than the actual performance. Example: o(n^7) is a valid little-o for a function that might actually have linear or O(n) performance.  
  
Little-Omega is just the opposite. w(1) [constant time] would be a valid little omega for the same above function that might actually exihbit BigOmega(n) performance.

<http://en.wikipedia.org/wiki/Big_O_notation#Family_of_Bachmann.E2.80.93Landau_notations>

### Family of Bachmann–Landau notations

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Notation** | **Name** | **Intuition** | **As  n \to \infty, eventually...** | **Definition** |
| f(n) \in O(g(n)) | Big Omicron; Big O; Big Oh | *f* is bounded above by *g* (up to constant factor) asymptotically | |f(n)|  \leq  g(n)\cdot kfor some *k* | \exists k>0, n_0 \; \forall n>n_0 \; |f(n)| \leq |g(n)\cdot k|  or   \exists k>0, n_0 \; \forall n>n_0 \; f(n) \leq g(n)\cdot k |
| f(n) \in \Omega(g(n))(Note that, since the beginning of the 20th century, papers in number theory have been increasingly and widely using this notation in the weaker sense that *f* = o(*g*) is false) | Big Omega | *f* is bounded below by *g* (up to constant factor) asymptotically | f(n)  \geq  g(n)\cdot kfor some positive *k* | \exists k>0, n_0 \; \forall n>n_0 \; g(n)\cdot k \leq f(n) |
| f(n) \in \Theta(g(n)) | Big Theta | *f* is bounded both above and below by *g* asymptotically | g(n)\cdot k_1 \leq f(n) \leq g(n)\cdot k_2for some positive *k*1, *k*2 | \exists k_1,k_2>0, n_0 \; \forall n>n_0  g(n) \cdot k_1 \leq f(n) \leq g(n) \cdot k_2 |
| f(n) \in o(g(n)) | Small Omicron; Small O; Small Oh | *f* is dominated by *g* asymptotically | |f(n)| \le |g(n)|\cdot \varepsilonfor every ε | \forall \varepsilon>0 \; \exists n_0 \; \forall n>n_0 \; |f(n)| \le |g(n)\cdot \varepsilon| |
| f(n) \in \omega(g(n)) | Small Omega | *f* dominates *g* asymptotically | f(n) \ge g(n)\cdot kfor every *k* | \forall k>0 \; \exists n_0 \; \forall n>n_0 \; g(n)\cdot k \le f(n) |
| f(n)\sim g(n)\! | on the order of; "twiddles" | *f* is equal to *g* asymptotically | f(n)/g(n) \to 1 | \forall \varepsilon>0\;\exists n_0\;\forall n>n_0\;\left|{f(n) \over g(n)}-1\right|<\varepsilon |

Bachmann–Landau notation was designed around several [mnemonics](http://en.wikipedia.org/wiki/Mnemonic), as shown in the *As  n \to \infty, eventually...* column above and in the bullets below. To conceptually access these mnemonics, "omicron" can be read "o-*micro*n" and "omega" can be read "o-*mega*". Also, the lower-case versus capitalization of the Greek letters in Bachmann–Landau notation is mnemonic.

* The *o-****micro****n mnemonic*: The o-*micro*n reading of f(n) \in O(g(n))and of f(n) \in o(g(n))can be thought of as "O-*smaller than*" and "o-*smaller than*", respectively. This *micro*/smaller mnemonic refers to: for sufficiently large input parameter(s), *f* grows at a rate that may henceforth be **less** than *cg* regarding g \in O(f)or g \in o(f).
* The *o-****mega*** *mnemonic*: The o-*mega* reading of f(n) \in \Omega(g(n))and of f(n) \in \omega(g(n))can be thought of as "O-*larger than*". This *mega*/larger mnemonic refers to: for sufficiently large input parameter(s), *f* grows at a rate that may henceforth be **greater** than *cg* regarding g \in \Omega(f)or g \in \omega(f).
* The ***upper****-case mnemonic*: This mnemonic reminds us when to use the upper-case Greek letters in f(n) \in O(g(n))and f(n) \in \Omega(g(n)): for sufficiently large input parameter(s), *f* grows at a rate that may henceforth be **equal** to *cg* regarding g \in O(f).
* The ***lower****-case mnemonic*: This mnemonic reminds us when to use the lower-case Greek letters in f(n) \in o(g(n))and f(n) \in \omega(g(n)): for sufficiently large input parameter(s), *f* grows at a rate that is henceforth **inequal** to *cg* regarding g \in O(f).

Aside from Big *O* notation, the Big Theta Θ and Big Omega Ω notations are the two most often used in computer science; the Small Omega ω notation is rarely used in computer science.