**Big O**

The “ big-Oh” notation is widely used: we say that f (N ) is O(g (N )) if there exist constants c and N0 such that | f (N )| < c g (N ) for all N > N0. This notation is very useful in providing asymptotic upper bounds on the performance of algorithms, which is important in the theory of algorithms. But it is not useful for predicting performance or for comparing algorithms.

**Q. Why not?**

A. The primary reason is that it describes only an upper bound on the running time. Actual performance might be much better. The running time of an algorithm might be both O (N 2) and ~ a N log N. As a result, it cannot be used to justify tests like our doubling ratio test

**Q. So why is the big O notation so widely used?**

A. It facilitates development of bounds on the order of growth, even for complicated algorithms for which more precise analysis might not be feasible. **Moreover, it is compatible with the “ big-Omega” and “ big-Theta” notations that theoretical computer scientists use to classify algorithms by bounding their worst-case performance.**

The “big Omega” notation is typically used to describe a lower bound on the worst case

The “big-Theta” notation is typically used to describe the performance of algorithms that are optimal in the sense that no algorithm can have better asymptotic worst-case order of growth.

**Big Omega**

Omega(n) is *asymptotic lower bound*. If T(n) is Omega(f(n)), it means that from a certain n0, there is a constant C1 such that T(n) >= C1 \* f(n). Whereas big-O says there is a constant C2 such that T(n) <= C2 \* f(n)).

Big omega is used to develop the lower bound

**Big Theta**

Big theta gives the upper and lower bounds and is therefore used to classify algorithms

**Everything that is Theta(f(n)) is also O(f(n)), but not the other way around.**

**T(n) is said to be Theta(f(n)), if it is both O(f(n)) and Omega(f(n))**

For this reason **big-Theta is more informative than big-O** notation, so if we can say something is big-Theta, it's usually preferred. However, it is harder to prove something is big Theta, than to prove it is big-O

**For example**, [merge sort](http://en.wikipedia.org/wiki/Merge_sort) is both O(n\*log(n)) and Theta(n\*log(n)), but it is also O(n2), since n2 is asymptotically "bigger" than it. However, it is NOT Theta(n2), Since the algorithm is NOT Omega(n2).

the “big-Theta” notation is typically used to describe the performance of algorithms that are optimal in the sense that no algorithm can have better asymptotic worst-case order of growth.