Asymptotic Analysis of Functions

In order to analyze the efficiency of an algorithm, we consider its running time $t(n)$ as a function of the input size n. We look at large enough n such that only the order of growth of $t(n)$ is relevant. In such asymptotic analysis, we are interested in whether the function scales as exponential (*e.g.*, 10^n), polynomial (*e.g.*, n^3) or logarithmic (*e.g.*, $\log_2 n$), for example. We use the following asymptotic notations.

Θ* notation:** Given a function $g(n)$, $\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_2 and n_0 such that $c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0\}$. We say $\Theta(g(n))$ is an ***asymptotically tight bound for $f(n)$.

(Example) In the molecular dynamics (MD) program, `md.c`, the computational bottleneck is the sum over all distinct atom pairs (i, j) to compute interatomic forces, implemented in a doubly-nested loop (see function *ComputeAccel*()):

```
for (i=0; i<n-1; i++) {
   for (j=i+1; j<n; j++) {
      ...
   }
}
```

The running time of this loop is proportional to the total number of iterations,

$$f(n) = 1 + 2 + \cdots (n-1) = \frac{(n-1)(1+n-1)}{2} = \frac{n^2 - n}{2},$$

which is $\Theta(n^2)$.

(Proof)

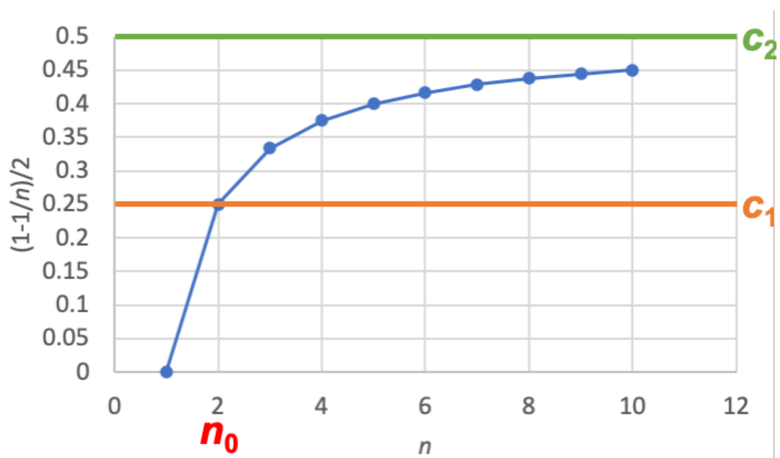
Let us consider inequalities

$$c_1 n^2 \le f(n) = \frac{n^2 - n}{2} \le c_2 n^2 \,. \qquad (1)$$

Dividing both sides by n^2 yields

$$c_1 \le f(n) = \frac{1}{2} - \frac{1}{2n} \le c_2 \,.$$

The right-hand inequality is satisfied for any positive n by choosing $c_2 \ge 1/2$. On the other hand, the left-hand inequality holds for all $n \ge 2$ if $c_1 \le 1/4$ (see the figure below). By choosing $c_1 = 1/4$, $c_2 = 1/2$ and $n_0 = 2$, Eq. (2) thus holds for all $n \ge n_0$. By definition, then $f(n)$ is $\Theta(n^2)$. //



O* (or "big-oh") notation:** Given a function $g(n)$, $O(g(n))$ = $\{f(n)$: there exist positive constants c and n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0\}$. We say $O(g(n))$ is an ***asymptotically upper bound for $f(n)$. Note that $O(g(n))$ is a superset of $\Theta(g(n))$. Outside computer science, the big-oh notation is most commonly used. While most bounds discussed in this class are tight bounds, we will loosely use the big-oh notation unless specific distinction is required.

References

1. A. Grama *et al*., *Introduction to Parallel Computing, Second Edition* (Addison Wesley, 2003), Appendix A.2—Order analysis of functions.
2. T. H. Cormen *et al*., *Introduction to Algorithms, Third Edition* (MIT Press, 2009), Chap. 3—Growth of functions.