CIS 575. Introduction to Algorithm Analysis
Material for January 22, 2024

Big O: Motivation & Definition

The topic of this note in covered in *Cormen*'s Section 3.2.

1 Big-O Notation

For a given algorithm, we shall want to

> estimate its running time (or space use), in terms of the size of its input.

In principle, the size of the input data must be measured as the number of bits needed to represent it. We shall see many algorithms that as input take an array of n integers; such an array may require $32n$ or $64n$ bits to represent, but to keep things simple, we shall often refer to it as having size n. In general, recall from the previous note that we don't really care about constants; it's thus also irrelevant if the running time is measured in ms, in μs, or something third.

We shall develop tools to compare various functions from natural numbers (input size) to positive reals (running time, or space use). A key concept is expressed as *Big-O* notation, where we may write $f \in O(g)$ to denote that g is an upper bound for f, but *not* in the most straightforward sense which would be that $f(n) \leq g(n)$ for all natural numbers n. Rather, as motivated in the previous note, we

- ignore constant factors, in that it suffices that $f(n) \leq c \cdot g(n)$ for some constant c
- only care about "the long run", in that we require $f(n) \leq c \cdot g(n)$ only for n above a certain threshold (often called n_0).

We have arrived at a key definition:

Definition 1.1 (Big-O) *Given two functions f, g from natural numbers into positive reals, we say that*

$$f \in O(g)$$

iff there exists $c > 0$ and $n_0 \geq 0$ such that

$$f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0.$$

Observe that $O(g)$ is a *set* of functions whereas f is *one* function. This explains that we write $f \in O(g)$ rather than $f = O(g)$. Unfortunately, the latter notation is widespread, and used even in the *Cormen* textbook whose reason for that choice (given on p. 58) is in my opinion rather unconvincing.

Examples We would certainly expect that

$$4n^2 + 7n + 8 \in O(n^2)$$

And in fact, when $n \geq \mathbf{1}$ we have $1 \leq n \leq n^2$ and thus the calculation

$$\begin{array}{rcl} 4n^2 + 7n + 8 & \leq & 4n^2 + 7n^2 + 8n^2 \\ & = & \mathbf{19}n^2 \end{array}$$

which by Definition 1.1, with $c = 19$ and $n_0 = 1$, shows the desired $4n^2 + 7n + 8 \in O(n^2)$. On the other hand, we would expect that

$$4n^2 + 7n + 8 \notin O(n)$$

An in fact, it can *not* be the case that $4n^2 + 7n + 8 \in O(n)$. For assume, to arrive at a contradiction, that there exists $c > 0$ and $n_0 \geq 1$ such that $4n^2 + 7n + 8 \leq cn$ for $n \geq n_0$. But for $n \geq n_0$ we would then also have $4n^2 \leq cn$ implying $4n \leq c$ which obviously cannot hold for arbitrarily large values of n.

Threshold We may want the threshold n_0 to be as low as possible. This can often be accomplished, at the price of a higher c, as we shall now illustrate by showing that

$$f(n) = 4n^2 - n + 3 \in O(n^2)$$

can be argued for (using Definition 1.1) in various ways:

- when $\mathbf{n} \geq \mathbf{3}$, we have $f(n) \leq 4n^2$ and thus we can use any $\mathbf{c} \geq \mathbf{4}$
- when $\mathbf{n} \geq \mathbf{2}$, we also need to make sure that $f(n) \leq cn^2$ when $n = 2$ which since $f(2) = 17$ will hold for any $\mathbf{c} \geq \mathbf{4.25}$
- when $\mathbf{n} \geq \mathbf{1}$, we also need to make sure that $f(n) \leq cn^2$ when $n = 1$ which since $f(1) = 6$ will hold for any $\mathbf{c} \geq \mathbf{6}$
- when $\mathbf{n} \geq \mathbf{0}$ we also need to make sure (assuming we consider 0 a natural number) that $f(n) \leq cn^2$ when $n = 0$ which since $f(0) = 3$ is — **impossible!**

In general, we see that:

> If $f \in O(g)$, where g is a function that is strictly positive (that is, never zero) then there exists $c > 0$ such that $f(n) \leq cg(n)$ holds for *all* n.