15-122: Principles of Imperative Computation

Recitation Week 2 — Josh Zimmerman, Nivedita Chopra

Big-O definition

The definition of big-O has a lot of mathematical symbols in it, and so can be very confusing at first. Let's familiarize ourselves with the formal definition and get an intuition behind what it's saying.

$O(g(n))$ is a *set* of functions, where $f(n) \in O(g(n))$ if and only if:

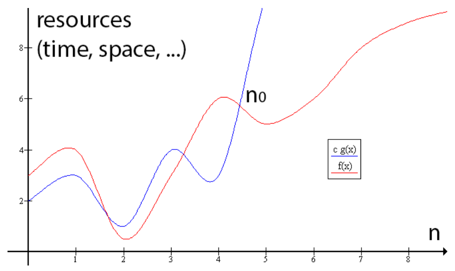
there is some ____

and some ____

such that for all ____,

____.

Big-O intuition



To the left of n_0, the functions can do anything.
To its right, $c * g(n)$ is always greater than or equal to $f(n)$.

There are actually *infinitely many functions* that are in $O(g(n))$: If $f(n) \in O(g(n))$, then $\frac{1}{2}f(n) \in O(g(n))$ and $\frac{1}{4}f(n) \in O(g(n))$ and $2f(n) \in O(g(n))$. In general, for any constant k, $k * f(n) \in O(g(n))$.

Checkpoint 0

Rank these big-O sets from left to right such that every big-O is a subset of everything to the right of it. (For instance, $O(n)$ goes farther to the left than $O(n!)$ because $O(n) \subset O(n!)$.) If two sets are the same, put them on top of each other.

$O(n!)$ $O(n)$ $O(4)$ $O(n\log(n))$ $O(4n+3)$ $O(n^2+20000n+3)$ $O(1)$ $O(n^2)$ $O(2^n)$
$O(\log(n))$ $O(\log^2(n))$ $O(\log(\log(n)))$

Checkpoint 1

Using the formal definition of big-O, prove that $n^3 + 300n^2 \in O(n^3)$.

Checkpoint 2

Using the formal definition of big-O, prove that if $f(n) \in O(g(n))$, then $k * f(n) \in O(g(n))$ for $k > 0$.

One interesting consequence of this is that $O(\log_i(n)) = O(\log_j(n))$ for all i and j (as long as they're both greater than 1), because of the change of base formula:

$$\log_i(n) = \frac{\log_j(n)}{\log_j(i)}$$

But $\frac{1}{\log_j(i)}$ is just a constant! So, it doesn't matter what base we use for logarithms in big-O notation.

When we ask for the *simplest, tightest bound* in big-O, we'll usually take points off if you write, for instance, $O(\log_2 n)$ instead of the simpler $O(\log n)$.

Simplest, tightest bounds

Something that will come up often with big-O is the idea of a *tight* bound on the runtime of a function.

It's technically correct to say that binary search, which takes around $\log(n)$ steps on an array of length n, is $O(n!)$, since $n! > \log(n)$ for all $n > 0$ but it's not very useful. If we ask for a *tight* bound, we want the closest bound you can give. For binary search, $O(\log(n))$ is a tight bound because no function that grows more slowly than $\log(n)$ provides a correct upper bound for binary search.

Unless we specify otherwise, we want the simplest, tightest bound!

Checkpoint 3

Simplify the following big-O bounds without changing the sets the represent:

$O(3n + 2)$ can be written more simply as ______________________________

$O(n^{2.5} + \log_2(n))$ can be written more simply as ______________________________

$O(\log_{10}(n) + \log_2(7n))$ can be written more simply as ______________________________

Checkpoint 4

Give the simplest, tightest bound for the following functions:

$f(n) = 16n^2 + 5n + 2 \in$ ______________________________

$g(n, m) = n^{1.5} \times 16m \in$ ______________________________

$h(x, y, z) = \max(x, y) + z^{16} \in$ ______________________________