Søren Sandmann Pedersen

Big-O Misconceptions

October 2012

In computer science and sometimes mathematics, big-O notation is used to talk about how quickly a function grows while disregarding multiplicative and additive constants. When classifying algorithms, big-O notation is useful because it lets us abstract away the differences between real computers as just multiplicative and additive constants.

Big-O is not a difficult concept at all, but it seems to be common even for people who should know better to misunderstand some aspects of it. The following is a list of misconceptions that I have seen in the wild.

But first a definition: We write

$$f(n) = O(g(n))$$

when $f(n) \leq Mg(n)$ for sufficiently large n, for some positive constant M.

Misconception 1: "The Equals Sign Means Equality"

The equals sign in

$$f(n) = O(g(n))$$

is a widespread travestry. If you take it at face value, you can deduce that since $5n$ and $3n$ are both equal to $O(n)$, then $3n$ must be equal to $5n$ and so $3 = 5$.

The expression $f(n) = O(g(n))$ doesn't type check. The left-hand-side is a function, the right-hand-side is a ... what, exactly? There is no help to be found in the definition. It just says "we write" without concerning itself with the fact that what "we write" is total nonsense.

The way to interpret the right-hand side is as a *set* of functions:

$$O(f) = \{g \mid g(n) \leq Mf(n) \text{ for some } M > 0 \text{ for large } n\}.$$

With this definition, the world makes sense again: If $f(n) = 3n$ and $g(n) = 5n$, then $f \in O(n)$ and $g \in O(n)$, but there is no equality involved so we can't make bogus deductions like $3 = 5$. We can however make the correct observation that $O(n) \subseteq O(n \log n) \subseteq O(n^2) \subseteq O(n^3)$, something that would be difficult to express with the equals sign.

Misconception 2: "Informally, Big-O Means 'Approximately Equal'"

If an algorithm takes $5n^2$ seconds to complete, that algorithm is $O(n^2)$ because for the constant $M = 7$ and sufficiently large n, $5n^2 \leq 7n^2$. But an algorithm that runs in constant time, say 3 seconds, is also $O(n^2)$ because for sufficiently large n, $3 \leq n^2$.

So informally, big-O means *approximately less than or equal*, not *approximately equal*.

If someone says "Topological Sort, like other sorting algorithms, is $O(n \log n)$", then that is *technically* correct, but severely misleading, because Toplogical Sort is also $O(n)$ which is a subset of $O(n \log n)$. Chances are whoever said it meant something false.

If someone says "In the worst case, any comparison based sorting algorithm must make $O(n \log n)$ comparisons" that is *not* a correct statement. Translated into English it becomes:

> *"In the worst case, any comparison based sorting algorithm must make fewer than or equal to $Mn \log(n)$ comparisons"*

which is not true: You can easily come up with a comparison based sorting algorithm that makes more comparisons in the worst case.

To be precise about these things we have other types of notation at our disposal. Informally:

$O()$:	*Less than or equal, disregarding constants*
$\Omega()$:	*Greater than or equal, disregarding constants*
$o()$:	*Stricly less than, disregarding constants*
$\Theta()$:	*Equal to, disregarding constants*

and some more. The correct statement about lower bounds is this: "In the worst case, any comparison based sorting algorithm must make $\Omega(n \log n)$ comparisons". In English that becomes:

> *"In the worst case, any comparison based sorting algorithm must make at least $Mn \log(n)$ comparisons"*

which is true. And a correct, non-misleading statement about Topological Sort is that it is $\Theta(n)$, because it has a lower bound of $\Omega(n)$ and an upper bound of $O(n)$.

Misconception 3: "Big-O is a Statement About Time"

Big-O is used for making statements about functions. The functions can measure time or space or cache misses or rabbits on an island or anything or nothing. Big-O notation doesn't care.

In fact, when used for algorithms, big-O is almost never about time. It is about primitive operations.

When someone says that the time complexity of MergeSort is $O(n \log n)$, they usually mean that the number of comparisons that MergeSort makes is $O(n \log n)$. That in itself doesn't tell us what the *time* complexity of any particular MergeSort might be because that would depend how much time it takes to make a comparison. In other words, the $O(n \log n)$ refers to *comparisons* as the primitive operation.

The important point here is that when big-O is applied to algorithms, there is always an underlying model of computation. The claim that the *time* complexity of MergeSort is $O(n \log n)$, is implicitly referencing an model of

computation where a comparison takes constant time and everything else is free.

Which is fine as far as it goes. It lets us compare MergeSort to other comparison based sorts, such as QuickSort or ShellSort or BubbleSort, and in many real situations, comparing two sort keys really does take constant time.

However, it doesn't allow us to compare MergeSort to RadixSort because RadixSort is not comparison based. It simply doesn't ever make a comparison between two keys, so its time complexity in the comparison model is 0. The statement that RadixSort is $O(n)$ implicitly references a model in which the keys can be lexicographically picked apart in constant time. Which is also fine, because in many real situations, you actually can do that.

To compare RadixSort to MergeSort, we must first define a shared model of computation. If we are sorting strings that are k bytes long, we might take "read a byte" as a primitive operation that takes constant time with everything else being free.

In this model, MergeSort makes $O(n \log n)$ string comparisons each of which makes $O(k)$ byte comparisons, so the time complexity is $O(k \cdot n \log n)$. One common implementation of RadixSort will make k passes over the n strings with each pass reading one byte, and so has time complexity $O(nk)$.

Misconception 4: Big-O Is About Worst Case

Big-O is often used to make statements about functions that measure the worst case behavior of an algorithm, but big-O notation doesn't imply anything of the sort.

If someone is talking about the randomized QuickSort and says that it is $O(n \log n)$, they presumably mean that its *expected running time* is $O(n \log n)$. If they say that QuickSort is $O(n^2)$ they are probably talking about its worst case complexity. Both statements can be considered true depending on what type of running time the functions involved are measuring.