Multi-Variable Code Analysis

So far we've made code models and found the $\mathcal{O}$, Ω, and Θ when we have one input (or at least one input whose size can increase). Real code often has multiple inputs, how do we do big-$\mathcal{O}$ analysis on code like that?

It's mostly the same, this note will just point out a few subtleties.

Instead of just referring to the input size as n, we have multiple variables: one for each input that could have a large size.[1] Usually we use n then m then p as our variable names.

Once we've decided on variable names, generating the code model is the same as we've always done (just now with more variables). The more interesting part is finding the big-$\mathcal{O}$.

We won't try to formally define big-$\mathcal{O}$ with multiple variables (we won't ask you to do any proofs like this). Here's an informal definition:

- We still don't care about constant factors.
- We only care about what happens as the variables get arbitrarily big.
- Unless the context of the problem tells us otherwise, we make no assumptions about how big the variables are relative to each other.

The third point is the key, and is easy to forget. If m and n are both variables, usually we don't know whether they're close or far from each other in value, or which is bigger, just that they are both big (because of point 1).

An example might help here. Let's assume that we're just given two lists, one of length m and one of length n with no further information.

Suppose our model is $3n^2m + mn$, can we simplify? Yes! Since n^2 dominates n and m dominates m, the first term dominates the second, and we can get rid of the second. The simplified big-$\mathcal{O}$ is $\mathcal{O}\left(n^2m\right)$.

What about $5n^2m+3m^2n$, can we simplify? If m and n are about the same size, these functions are interchangeable in big-$\mathcal{O}$ terms. So can we just pick one? No! If n is much bigger than m (imagine $n = m^2$) the first term will dominate, but if m is much bigger than n (imagine $m = n^2$) the second term will dominate. We need them both to accurately describe all the experiences you might have running this piece of code. Our tight, simplified big-$\mathcal{O}$ is: $\mathcal{O}\left(n^2m + m^2n\right)$.

It's ok to have that plus sign in the $\mathcal{O}$ notation, because neither term dominates the other.

Let's talk about that "Unless the context of the problem tells us otherwise" disclaimer. What might that look like? Usually this happens when one data structure is explicitly a subset of another. For example, we might have a list, and a second list which we're told is the first one with duplicates removed. We then definitely know that the second list is no larger than the first[2].

Some practice: find the tight, simplified big-$\mathcal{O}$ for each of these code models. Assume you know nothing about the relationship between m and n.

(a) $f_1(n,m) = n^3\log(m) + n^2m + nm$

(b) $f_2(n,m) = n^2 + m^2 + mn$

(c) $f_3(n,m) = n^2m + n^2$

(d) $f_4(n,m) = m\log(n) + m + \log(n)$

[1] so, for example, even though `binarySearch(int[] arr, int toFind)`, takes two parameters, we still only use one variable for the size of the array. `toFind`'s size doesn't affect the running time of the code (even though its value might).

[2] Another example, we'll see later in this course is that in a "simple" graph, the number of vertices and edges is related.

Answers:

- $f_1(n, m)$ is $\mathcal{O}\left(n^3 \log(m) + n^2 m\right)$.
- $f_2(n, m)$ is $\mathcal{O}\left(n^2 + m^2\right)$. Notice that regardless of which of m, n is bigger, one of the first two terms dominates the last one, so we can actually get rid of it.
- $f_3(n, m)$ is $\mathcal{O}\left(n^2 m\right)$.
- $f_4(n, m)$ is $\mathcal{O}\left(m \log(n)\right)$.