CS 31: Algorithms (Spring 2019): Lecture 2

Date: 27th March, 2019 (X-hour)
Topic: The Big-Oh Notation
Disclaimer: These notes have not gone through scrutiny and in all probability contain errors. Please email errors to deeparnab@dartmouth.edu.

1 The Big-Oh Notation

Given a function $g : \mathbb{N} \to \mathbb{Z}$, $O(g(n))$ denotes a *set* of functions defined as follows:

Definition 1. We say $f(n) \in O(g(n))$ if there exists two *constants* $a, b > 0$ such that for all $n \geq b$, we have $f(n) \leq a \cdot g(n)$.

Colloquially, $f(n)$ is said to be Big-Oh (O for order) of $g(n)$. In plain English, it means for large enough n ($n \geq b$) $f(n)$ is dominated by some scaled version of $g(n)$ (that is, $a \cdot g(n)$).

Remark: *Although $O(g(n))$ is a set and $f(n)$ belongs in the set, the world often abuses and says $f(n) = O(g(n))$ when it means $f(n) \in O(g(n))$. This is important to keep in mind. $O(g(n))$ is a set, and can't be, for instance, added together. Yet, we will see us adding these things. In this lecture we will make precise what these mean – these are all definitions and used for convenience.*

There are two other very similar definitions.

Definition 2. We say $f(n) \in \Omega(g(n))$ if there exists two *constants* $a, b > 0$ such that for all $n \geq b$, we have $f(n) \geq a \cdot g(n)$.

Observation 1. If $f(n) = O(g(n))$ then $g(n) = \Omega(f(n))$.

Proof. There exists $a, b > 0$ such that for all $n \geq b$, $f(n) \leq a \cdot g(n)$. But that means $g(n) \geq (1/a) \cdot f(n)$. □

Definition 3. We say $f(n) \in \Theta(g(n))$ if there exists three *constants* $b, a_1, a_2 \geq 0$ such that for all $n \geq b$, we have $a_1 \cdot g(n) \leq f(n) \leq a_2 \cdot g(n)$.

Observation 2. $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Proof. One direction is easy, and the other is easier. The easier direction is given $f(n) = \Theta(g(n))$, we directly get $f(n) = O(g(n))$ (certified by b, a_2) and $f(n) = \Omega(g(n))$ (certified by b, a_1).

The easy direction follows thus: since $f(n) = O(g(n))$, there exists constants $b_2, a_2 \geq 0$ such that for all $n \geq b_2$, $f(n) \leq a_2 \cdot g(n)$. Similarly, $f(n) = \Omega(g(n))$ implies constants $b_1, a_1 \geq 0$ such that for all $n \geq b_1$, $f(n) \geq a_1 \cdot g(n)$. But this means for all $n \geq b = \max(b_1, b_2)$, we have $a_1 \cdot g(n) \leq f(n) \leq a_2 \cdot g(n)$. □

Why are these notions useful? As we saw last time, in the analysis of algorithms, we are interested in understanding the running times of algorithms. In particular, we wish to understand the function $T(n)$ with respect to the size parameter n. As a first cut, we are interested in the *asymptotic* behavior of $T(n)$, that is, the behavior of $T(n)$ as n grows larger and larger. The Big-Oh notation is perfect to capture this: we don't care about how T behaves in the small n regime, and we also don't care about the scaling factor of T. The latter is relevant in running time analysis since scaling boils down to "change of units". And as was clear last time, what unit to use is not obvious. Should time be measured in number of BIT-ADDs? Should it be measured in number of operations? Or Wall Clock time? The Big-Oh notation tries to capture the "big-picture"[1] .

Remark: *An algorithm $\mathcal{A}$ is a* ***polynomial time*** *algorithm if there exists a constant $k \geq 1$ such that $T_{\mathcal{A}}(n) = O(n^k)$. If $k = 1$, the algorithm is a linear-time, or sometimes simply linear, algorithm. If $k = 2$, then the algorithm is a quadratic-time, or simply quadratic, algorithm.*

Example 1. $f(n) = 10n + 4$. We claim that $f(n) = O(n)$. To show this, we need to find a, b such that for all $n \geq b$, we have $10n + 4 \leq a \cdot n$. If we set $b = 4$, then we get for all $n \geq 4$, the LHS $10n + 4 \leq 10n + n \leq 11n$. Thus, we can set $b = 4, a = 11$ to prove $10n + 4 = O(n)$.

Example 2 (Counterexample)**.** If $f(n) = n$ and $g(n) = \frac{n^2}{3} - 7$, then $g(n) \notin O(f(n))$ or as is more prevalent $g(n) \neq O(f(n))$. To prove this, let us suppose for contradiction's sake that $g(n) = O(f(n))$. Therefore, there must exist constants $a, b > 0$ such that for all $n \geq b$, we have $\frac{n^2}{3} - 7 \leq a \cdot n$. But take $n = \max(3(a+1), 7, b) + 1$. Then, we get

$$\begin{aligned}
\frac{n^2}{3} &= n \cdot \frac{n}{3} \\
&> n \cdot \frac{3(a+1)}{2} \qquad \text{since } n > 3(a+1) \\
&= n \cdot (a+1) \\
&= an + n \\
&> an + 7
\end{aligned}$$

which in turn implies $\frac{n^2}{3} - 7 > an$ which is a contradiction, since $n > b$ as well.

Theorem 1 (Some operations involving the notations)**.** Below all functions take non-negative values.

1. If $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$, then $f(n) = O(h(n))$ as well.
2. If $f(n) \in \Theta(g(n))$ and $g(n) \in \Theta(h(n))$, then $f(n) \in \Theta(h(n))$ as well.

[1] This is not to say the small picture is not important. When we really want performance in our implementation of an algorithm, the constants are game-changers. Nevertheless, the really good algorithms often have wall clock times great too. Of course there are exceptions, and you should always implement your algorithm to check which is the case.

3. If $f(n), h(n)$ are both in $O(g(n))$, and $s(n) := f(n) + h(n)$, then we have $s(n) \in O(g(n))$ as well.
4. If $f(n) \in O(g_1(n))$ and $h(n) \in O(g_2(n))$, then $f(n) \cdot h(n) \in O(g_1(n) \cdot g_2(n))$.
5. Let $f(n) \in O(g(n))$, and $h()$ is a *non-decreasing, non-negative* function of n. Recall composition of functions: $(f \circ h)(n) := f(h(n))$. Then, $(f \circ h)(n) \in O((g \circ h)(n))$.

Proof.

1. $f(n) = O(g(n))$ means there exists constants $a, b \geq 0$ such that $f(n) \leq a \cdot g(n)$ for all $n \geq b$. $g(n) = O(h(n))$ means there exists constants $a', b' \geq 0$ such that $g(n) \leq a' \cdot h(n)$ for all $n \geq b'$. Thus, for all $n \geq \max(b, b')$, we have $f(n) \leq (a \cdot a') \cdot h(n)$. This proves that $f(n) = O(h(n))$.

2. Very similar proof as above, except there are three variables a_1, a_2, b to play with. Please do this yourself (recommended).

3. As before, there exists $a, b \geq 0$ and $a', b' \geq 0$ such that for all $n \geq b$, we have $f(n) \leq a \cdot g(n)$ and for all $n \geq b'$ we have $h(n) \leq a' \cdot g(n)$. That is, for all $n \geq \max(b, b')$, we have $s(n) \leq (a + a') \cdot g(n)$. Thus, $s(n) = O(g(n))$.

4. As before, there exists $a, b \geq 0$ and $a', b' \geq 0$ such that for all $n \geq b$, we have $f(n) \leq a \cdot g_1(n)$ and for all $n \geq b'$ we have $h(n) \leq a' \cdot g_2(n)$. Therefore, for all $n \geq \max(b, b')$, we get $f(n) \cdot h(n) \leq (a \cdot a') \cdot (g_1(n) \cdot g_2(n))$.

5. There exists $a, b > 0$ such that for all $n \geq b$, we have $f(n) \leq a \cdot g(n)$. Now since h is increasing, for all $n \geq h^{-1}(b)$ we have $h(n) \geq h(h^{-1}(b)) = b$. Therefore, For all $n \geq h^{-1}(b)$, we get $(f \circ h)(n) = f(h(n)) \leq a \cdot g(h(n)) = (g \circ h)(n)$.

□

You should try out some specific settings of functions to get an idea. For instance, if we take $h(n) = \log_2 n$ to be an example of a non-negative, non-decreasing function. Then taking $f(n) = n$ and $g(n) = n^2$ gives us $\log n = O(\log^2 n)$.

Unspecified Constants We often will come across functions $f(n) = O(1)$. By definition, it means there exists constants $a, b > 0$ s.t. for all $n \geq b$, we get $f(n) \leq a$. But note that $B := \max_{n<b} f(n)$ is some constant (it doesn't grow with n). Therefore, $F(N) \leq \max(a, B)$ for all n. Implying $f(n)$ is at most some *unspecified* constant. ✍

Exercise: *Are the sets $\Theta(1)$ and $O(1)$ the same?*

2 Using limits to figure out the answer

The following theorem often allows us to argue about the Big-Oh relations for large classes of functions.

Theorem 2. Suppose $L := \lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right)$ exists. $f(n) = \Theta(g(n))$ if and only if $0 < L < \infty$.

Proof. Do we recall limits from Calculus?
If the limit exists and is $L < \infty$, then this means for any $\varepsilon > 0$, there is a number b_ε such that for all $n \geq b_\varepsilon$, we have

$$\left|\left(\frac{f(n)}{g(n)}\right) - L\right| < \varepsilon$$

In particular, we get for all $n > b_\varepsilon$, $L - \varepsilon \leq f(n)/g(n) \leq L + \varepsilon$. That is, $(L-\varepsilon)g(n) \leq f(n) \leq (L+\varepsilon)g(n)$. Aha! Choose $b = b_{L/2}$, $a_1 = L/2$, and $a_2 = 3L/2$ to get for all $n \geq b$, $a_1 \cdot g(n) \leq f(n) \leq a_2 \cdot g(n)$. Since L is a constant > 0, we get that a_1, a_2, b are all constants.

On the other hand, if $f(n) = \Theta(g(n))$, then for all $n > b$, we have $a_1 \cdot g(n) \leq f(n) \leq a_2 \cdot g(n)$. That is, $a_1 \leq f(n)/g(n) \leq a_2$. Therefore, if $\lim_{n\to\infty}(f(n)/g(n))$ exists, then it must be $a_1 \leq L \leq a_2$. □

A corollary obtained from the above is the following

Theorem 3. If $\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = L < \infty$, then $f(n) = O(g(n))$.

Theorem 4. If $\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \infty$, then $f(n) \neq O(g(n))$.

Some Calculus textbooks will slaughter us here – "how can limit be ∞?". Indeed. So let's explain what we mean by the limit being ∞. We mean that given any M and b, there exists some $n > b$ such that $f(n)/g(n) > M$. This happens if $f(n)/g(n)$ is an increasing function of n. Take $f(n) = 4^n$ and $g(n) = 2^n$ – we get $f(n)/g(n) = 4^n/2^n = 2^n$ which goes to ∞ as $n \to \infty$.

Proof. With the above, the proof of the theorem follows from definition. Again, the tool is contradiction. Suppose, for contradiction's sake, that $f(n) = O(g(n))$. Then there exists some $a, b \geq 0$ such that for all $n > b$, we have $f(n)/g(n) \leq a$. But that's precisely what the "limit equal to infinity" disallows. □

The above two theorems help a lot often. ***But be wary when limits don't exist.***
We end with the "small o" notation as follows.

Definition 4. We say that $f(n) = o(g(n))$ if $\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = 0$.

2.1 Fun with Factorials and Logarithms

Recall the factorial function defined as $n! = 1 \cdot 2 \cdot 3 \cdots n$. What is its relation with n^n?

Claim 1. $n! = O(n^n)$

Proof. Let's look at $n!/n^n$. This equals $(1/n) \cdot (2/n) \cdots (n/n)$. As $n \to \infty$, each of the products goes to 0. Thus, the claim follows. □

Claim 2. $n^n \neq O(n!)$

Proof. Again the flipped rational function $n^n/n! = (n/1) \cdot (n/2) \cdots (n/n)$ which is an increasing function of n. Thus the limit is ∞ implying $n^n \neq O(n!)$. □

Now let us take logarithms.

Claim 3. $\log_2(n!) = \Theta(n \log_2 n)$.

Thus, we see that although $n! \neq \Theta(n^n)$, taking logs satisfies the $\Theta()$ relation. Again, this should not be surprising – we have see $f(n) = \Theta(g(n))$ but $2^{f(n)} \neq O(2^{g(n)})$.

Proof. Let's now prove the claim. Once again, let's look at the rational function $r(n) := \frac{\log_2(n!)}{n \log_2 n}$. First we note that the numerator is precisely $\sum_{i=1}^{n} \log_2 i$ since log of product is sum of logs. Since $\log_2 i \leq \log_2 n$ for all $i \leq n$, we get that the numerator is $\leq n \log_2 n$ implying $r(n) \leq 1$ for all n.

We now need to show that $\lim_{n\to\infty} r(n) >$ some positive number. To see this, we note that for all $i \geq n/2$, $\log_2 i > \log_2(n/2)$. Since there are $(n/2)$ numbers between 1 and n which are $\geq n/2$, we get that the numerator of $r(n)$ is $\geq (n/2)\log_2(n/2)$. Thus $r(n) \geq \frac{1}{2} \cdot \frac{\log_2 n - 1}{\log_2 n}$. As $n \to \infty$, the second fraction $\to 1$. Thus, $\lim_{n\to\infty} r(n) \geq 1/2$.

This proves the claim. □

Corollary 1. For any number n, the number of bits required to write $n!$ is $\Theta(n \log_2 n)$.

3 Abuse of Notation: what does it mean?

As if the abuse $f(n) = O(g(n))$ instead of $f(n) \in O(g(n))$ was not enough, in this course (and in the world at large), you will see statments of the form

$$\text{For every } n > 0, , \quad f(n) \leq g(n) + \Theta(h(n)) \tag{1}$$

In fact, this is most prevalent in recurrence inequalities. For MULT, the recurrence inequality we would write is

$$\text{For every } m > 0, , \quad T(n,m) \leq T(n+1, m-1) + \Theta(n)$$

As mentioned before, with the definition of $\Theta(h(n))$ as a set of functions, (1) sounds garbage. So what does it mean when we encounter such inequalities.

What it means is, that *there exists* a function $e(n)$, such that (a) for all $n > 0$, $f(n) \le g(n) + e(n)$, and (b) $e(n) \in \Theta(n)$. Observe in the recurrence for MULT above, $e(n) = 2n$.

The reason for this abuse is brevity – writing out the $e(n)$'s everytime is tedious, and one of the nice things about the Big Oh notation is the fact that we can express things succinctly. Of course brevity comes at a cost – awareness! Be wary, especially when you are solving recurrence inequalities with Big-Oh's and Theta's floating about. Let me point out a mistake which you should never make.

Theorem 5. Consider the following recurrence inequality:

This is a **wrong** theorem, and an erroneous proof.

$$T(1) = \Theta(1); \quad \forall n > 1, \quad T(n) = T(n-1) + \Theta(1)$$

The solution to the above is $T(n) = \Theta(1)$.

The above theorem is wrong, because we know what the solution to $T(n)$ is by the "opening up brackets" or the "kitty collection" method.

$$\begin{aligned} T(n) &= T(n-1) + \Theta(1) \\ &= T(n-2) + \Theta(1) + \Theta(1) \\ &\vdots \\ &= T(0) + \Theta(1) + \Theta(1) + \cdots + \Theta(1) = \Theta(n) \quad \text{since there are } n \text{ separate } \Theta(1)\text{'s adding up} \end{aligned}$$

Wrong Proof of Wrong Theorem 5. We apply induction. Base Case: When $n = 1$, $T(n) = \Theta(1)$. This is part of the input. Suppose for all $1 \le n < m$, we have $T(n) = \Theta(1)$. We need to show $T(m) = \Theta(1)$.

Well, $T(m) = T(m-1) + \Theta(1)$. Since $m - 1 < m$, and therefore $T(m-1) = \Theta(1)$ by induction. Therefore, $T(m) = \Theta(1) + \Theta(1)$. But adding a constant with another constant, is again a constant (see Theorem 1) So, $T(m) = \Theta(1)$. □

Whenever you see a wrong proof, you must find out where were it is wrong. And the mistake crept in due to the abuse of notation. Consider what the Theorem 5 is asserting : it says there is an (unspecified) constant $C \ge 0$ such that $T(n) \le C$ for all n. When we try to apply induction, we must first agree upon this constant. We don't know what it is, but it exists, and we stick with it. Base Case: $T(1) \le C$ – fine. Induction Hypothesis: for all $1 \le n < m$, we have $T(n) \le C$ – sure. Now we need to prove $T(m) \le C$ as well. What doe we now – we know that $T(m) \le T(m-1) + \Theta(1)$. Again what does this mean? It means there is some other unspecified constant $C' \ge 0$ such that $T(m) \le C + C'$. But now we are in trouble – we can't say $T(m) \le C$ unless $C' = 0$, and that is not specified.