

## CS31 (Algorithms), Spring 2020 : Lecture 2

Topic: The Big-Oh Notation

*Disclaimer: These notes have not gone through scrutiny and in all probability contain errors.*

*Please discuss in Piazza/email errors to [deeparnab@dartmouth.edu](mailto:deeparnab@dartmouth.edu)*

---

Given a function  $g : \mathbb{N} \rightarrow \mathbb{R}$ ,  $O(g(n))$  denotes a set of functions with domain  $\mathbb{N}$  and co-domain  $\mathbb{R}$ .

**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  $f(n) \leq a \cdot g(n)$ .

In plain English, it means for “large enough”  $n$  (more precisely,  $n \geq b$ )  $f(n)$  is dominated by some “scaled version” of  $g(n)$  (more precisely,  $a \cdot g(n)$ ).

**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.

**Remark:** Often when the function  $g(n)$  is “simple”, then one substitutes that function in the Big-Oh. For instance,  $O(n)$  is the set of functions where the  $g(n)$  is simply the identity function  $g(n) = n$ .

**Example 1.** Let  $f(n) = 10n + 4$ . We claim that  $f(n) = O(n)$ . To show this, we need to exhibit two positive constants  $a, b$  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  $10n + 4 = O(n)$ .

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  $f(n) \geq a \cdot g(n)$ .

In plain English, it means for “large enough”  $n$  (more precisely,  $n \geq b$ )  $f(n)$  dominates some “scaled version” of  $g(n)$  (more precisely,  $a \cdot g(n)$ ).

**Theorem 1.**  $f(n) \in O(g(n))$  if and only if  $g(n) \in \Omega(f(n))$ .

Again, the “English argument” makes sense. If after some point  $f$  is dominated by some scaled version of  $g$ , then after that point,  $g$  also must dominate a scaled version of  $f$ . And vice-versa. The proof below formalizes this. Make sure you are comfortable and confident writing such proofs.

*Proof.* There are two things to show. First, assume  $f(n) \in O(g(n))$ . That is, there exists  $a, b > 0$  such that for all  $n \geq b$ ,  $f(n) \leq a \cdot g(n)$ . Dividing by  $a$  both sides (since  $a > 0$ ) and rearranging, we get that  $g(n) \geq (1/a) \cdot f(n)$  for all  $n \geq b$ . This means  $g(n) \in \Omega(f(n))$ .

Second, assume  $g(n) \in \Omega(f(n))$ . That is, there exists  $a, b > 0$  such that for all  $n \geq b$ ,  $g(n) \geq a \cdot f(n)$ . Dividing by  $a$  both sides (since  $a > 0$ ) and rearranging, we get that  $f(n) \leq (1/a) \cdot g(n)$  for all  $n \geq b$ . This means  $f(n) \in O(g(n))$ .  $\square$

**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  $a_1 \cdot g(n) \leq f(n) \leq a_2 \cdot g(n)$ .

**Theorem 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  $a_1 \cdot g(n) \leq f(n) \leq a_2 \cdot g(n)$ .  $\square$

**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"<sup>1</sup>.

**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.

The next theorem shows that polynomials are in the Big-Oh class of the leading monomial (highest degree). In particular, it explains why the illustrative function  $f(n) = 10n + 4$  was  $O(n)$ .

**Theorem 3.** Suppose  $p(n)$  is a degree  $d$  polynomial, that is,  $p(n) = a_d n^d + a_{d-1} n^{d-1} + \dots + a_1 n + a_0$ . Then,  $p(n) \in O(n^d)$ .

*Proof.* We need to exhibit two constants  $a, b$  such that for all  $n \geq b$ ,  $f(n) \leq a \cdot n^d$ . Indeed, choose  $b = \max\{|a_d|, |a_{d-1}|, \dots, |a_0|\}$  and choose  $a = (|a_d| + d)$ .

Note that if  $n \geq b$ , then for each of the  $d$  indices  $0 \leq i \leq d-1$ , we get that the quantity  $a_i n^i \leq n^{i+1} \leq n^d$ . And therefore, for all  $n \geq b$ , we get

$$p(n) = a_d n^d + a_{d-1} n^{d-1} + \dots + a_1 n + a_0 \leq a_d n^d + d \cdot n^d \leq a \cdot n^d$$

This shows  $p(n) \in O(n^d)$ .  $\square$

It is sometimes important to show when some functions  $f(n)$  **are not**  $O(g(n))$  of some other function. Below is an example.

**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 commonly stated,  $g(n) \neq O(f(n))$ .

<sup>1</sup>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.

To prove this, let us suppose for contradiction's sake that  $g(n) = O(f(n))$ . Therefore, there exists 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

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

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

**Example 3** (Counterexamples). (Here is a common thing which confuses many students).  $4^n \neq O(2^n)$ . Although  $2n = O(n)$ , when the  $n$  is in the exponent the constant next to  $n$  matters. That is,  $2^{2n} \neq O(2^n)$ . The reason is actually simple:  $2^{2n}$  is not twice of  $2^n$  but *square* of  $2^n$ ; and squaring takes us to a different order of magnitude.

To prove formally, again we use contradiction. Suppose not, and say  $4^n = O(2^n)$ . Then there exists  $a, b \geq 0$  such that for all  $n > b$ ,  $4^n \leq a \cdot 2^n$ . Firstly,  $a > 1$  for  $4^n > 2^n$  for large  $n$ . Choose  $n = \max(b, a)$ . Then, we must have  $4^n = 2^n \cdot 2^n > 2^a \cdot 2^n > a \cdot 2^n$ . Thus we get a contradiction.

The following theorem is often useful.

**Theorem 4** (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.
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  $h(\cdot)$  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  $(f \circ h)(n) = f(h(n)) \leq a \cdot g(h(n)) = (g \circ h)(n)$ .

□

**Unspecified Constants** We often will meet functions and say  $f(n) = O(1)$ . What do we mean by that? By definition, it means there exists constants  $a, b > 0$  s.t. for all  $n \geq b$ , we get  $f(n) \leq a \cdot 1$ . Now suppose  $B := \max_{n < b} f(n)$  is the maximum value this function takes in the range  $\{1, 2, \dots, b-1\}$ . This  $B$  is some fixed **constant** (could be a bazillion, we don't care); it does not grow with  $n$ . Therefore, combining the above two, we see that  $F(n) \leq \max(a, B)$  for all  $n$ . Implying  $f(n)$  is at most some *unspecified* constant. 

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

## 1 Using limits to figure out the answer

The following theorem is very useful to argue about the Big-Oh relations for large classes of functions.

**Theorem 5.** Suppose  $L := \lim_{n \rightarrow \infty} \left( \frac{f(n)}{g(n)} \right)$  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  $b_\varepsilon$  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  $\lim_{n \rightarrow \infty} (f(n)/g(n))$  exists, then it must be  $a_1 \leq L \leq a_2$ . □

A corollary obtained from the above is the following

**Theorem 6.** If  $\lim_{n \rightarrow \infty} \left( \frac{f(n)}{g(n)} \right) = L < \infty$ , then  $f(n) = O(g(n))$ .

**Theorem 7.** If  $\lim_{n \rightarrow \infty} \left( \frac{f(n)}{g(n)} \right) = \infty$ , then  $f(n) \neq O(g(n))$ .

Let us explain what we mean by the limit being  $\infty$ . 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  $\infty$  as  $n \rightarrow \infty$ .

*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.  $\square$

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  $\lim_{n \rightarrow \infty} \left( \frac{f(n)}{g(n)} \right) = 0$ .

## 1.1 Fun with Factorials and Logarithms

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

**Theorem 8.**  $n! = O(n^n)$

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

**Theorem 9.**  $n^n \neq O(n!)$

*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  $\infty$  implying  $n^n \neq O(n!)$ .  $\square$

Now let us take logarithms.

**Theorem 10.**  $\log_2(n!) = \Theta(n \log_2 n)$ .

Thus, we see that although  $n! \neq \Theta(n^n)$ , taking logs satisfies the  $\Theta()$  relation. Again, this should not be surprising – we have seen  $f(n) = \Theta(g(n))$  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 (using log of product is sum of logs)

$$\log_2(n!) = \log_2(1 \times 2 \times \cdots \times n) = \log_2 1 + \log_2 2 + \cdots + \log_2 n = \sum_{i=1}^n \log_2 i$$

Now, since  $\log_2 i \leq \log_2 n$  for all  $i \leq n$  (since  $\log_2$  is an increasing function), we get that the numerator of  $r(n)$  is  $\leq n \log_2 n$ . That is, the denominator of  $r(n)$ . This implies  $r(n) \leq 1$  for all  $n$ . And in particular,  $\lim_{n \rightarrow \infty} r(n) \leq 1$ .

We now need to show that  $\lim_{n \rightarrow \infty} r(n) >$  some positive number. This will show  $\log_2(n!) \in \Theta(n \log_2 n)$ . To see this, we note that for all  $i \geq n/2$ ,  $\log_2 i > \log_2(n/2)$ . Therefore, we get that the numerator of  $r(n)$  is

$$\sum_{i=1}^n \log_2 i = \log_2 1 + \log_2 2 + \dots + \log_2 n \geq \log_2(n/2) + \log_2(n/2+1) + \dots + \log_2 n \geq (n/2) \cdot \log_2(n/2) = \frac{n}{2} \cdot (\log_2 n - 1)$$

Thus  $r(n) \geq \frac{1}{2} \cdot \frac{\log_2 n - 1}{\log_2 n}$ . As  $n \rightarrow \infty$ , the second fraction  $\rightarrow 1$ . Thus,  $\lim_{n \rightarrow \infty} r(n) \geq 1/2$ .  $\square$

Recall that given any natural number  $N$ , the number of bits required to describe it is  $\lceil \log_2(N+1) \rceil$ . In your problem set, you will show that  $\lceil x \rceil = \Theta(x)$ . Therefore, one requires  $\Theta(\log_2 N)$  bits to describe a natural number  $N$ . The above theorem, therefore, gives the following corollary.

**Theorem 11.** For any number  $n$ , the number of bits required to write  $n!$  is  $\Theta(n \log_2 n)$ .

## 2 Abuse of Notation: what does it mean?

*This section is not in the lecture video but is important.*

As if the abuse  $f(n) = O(g(n))$  instead of  $f(n) \in O(g(n))$  was not enough, in this course we will see statements of the form

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

Again recall,  $O(h(n))$  is a set of functions. What does it mean to “add a set”? In fact, this is most prevalent in recurrence inequalities. For instance, we will soon see recurrence inequalities of the form

$$T(1) = O(1), \quad \text{and for all } n \geq 2, \quad T(n) \leq T(n-1) + O(n) \tag{2}$$

What do these mean?

So, whenever you see a recurrence as above, what it really means is that there exists a function  $e(n)$ , such that (a) for all  $n \geq 2$ ,  $T(n) \leq g(n) + e(n)$ , and (b)  $e(n) \in O(n)$ .

Let us unpack what  $e(n) \in O(n)$  means. It means that there exists positive constants  $a, b$  such that for all  $n \geq b$ ,  $e(n) \leq a \cdot n$ . We are now going to simplify our lives a bit and see that we can actually state there exists a constant  $A$  such that

$$\forall n \geq 2, \quad e(n) \leq A \cdot n$$

How can we say this? Well, if  $n \geq b$ , we get this with  $A = a$ . Let  $B = \max_{n < b} e(n)$ ; this is some constant. Note, that for all  $2 \leq n \leq b$ ,  $e(n) \leq B \leq B \cdot n$ . Therefore,  $A = \max(a, B)$  satisfies our needs.

Putting all together, the recurrence (2) actually means the existence of a constant  $A, B$  such that

$$T(1) \leq B, \quad \text{and} \quad T(n) \leq T(n-1) + A \cdot n, \quad \text{for all } n \geq 2$$

The reason for this abuse is brevity – writing out these constants and keeping track of them is tedious. 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 12. Warning: This is a wrong theorem followed by an erroneous proof.**

Consider the following recurrence inequality:

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

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

**Wrong Proof of Wrong Theorem 12.** We proceed by induction. The predicate  $P(n)$  is true if  $T(n) = O(1)$

Base Case: When  $n = 1$ ,  $T(n) = O(1)$ . This is given to us.

We fix a  $k \geq 1$  and assume  $P(1), \dots, P(k)$  are all true. That is,  $T(m) = O(1)$  for all  $m \leq k$ . We need to show  $P(k+1)$  is true. That is, we need to show  $T(k+1) = O(1)$ .

Well,  $T(k+1) \leq T(k) + O(1)$ . Since  $P(k)$  is true, we get  $T(k) = O(1)$ . Therefore,  $T(k+1) = O(1) + O(1)$ . But adding a constant with another constant, is again a constant (see Theorem 4) So,  $T(k+1) = O(1)$ . Hence proved.  $\square$

The above theorem is wrong, because we know what the recurrence really means there exists some constants  $A, B$  such that

$$T(n) \leq \begin{cases} B & \text{if } n = 1 \\ T(n-1) + A & \text{if } n \geq 2 \end{cases}$$

what the solution to  $T(n)$  is by the “opening up brackets” or the “kitty collection” method. For any  $n \geq 2$ ,

$$\begin{aligned} T(n) &\leq T(n-1) + A \\ &\leq T(n-2) + A + A \\ &\vdots \\ &= T(1) + A(n-1) \leq An + (B-A) = O(n) \text{ by Theorem 3} \end{aligned}$$

**Where was the bug in the above induction proof?** 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 12 is asserting : it says there is an (unspecified) constant  $C \geq 0$  such that  $T(n) \leq C$  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:  $T(1) \leq C$  – fine. Induction Hypothesis: for all  $1 \leq n < m$ , we have  $T(n) \leq C$  – sure. Now we need to prove  $T(m) \leq C$  as well. What doe we now – we know that  $T(m) \leq T(m-1) + \Theta(1)$ . Again what does this mean? It means there is some other unspecified constant  $C' \geq 0$  such that  $T(m) \leq C + C'$ . But now we are in trouble – we can't say  $T(m) \leq C$  unless  $C' = 0$ , and that is not specified.