CS Theory (Spring '25) September 23, 2025

Lecture Note: Asymptotic Notation

Instructor: *Toniann Pitassi*

We have been studying different models of computation, including DFAs, NFAs and regular expressions. So far we focus on whether the model can solve a problem, and we haven't talked about how efficient the model can solve the problem. For example, we have seen the construction of a DFA from an NFA. Although they recognize the same language, it has a huge blow-up in the number of states (going from a NFA with k states can give a DFA with 2^k states). Starting now, we will not only focus on whether a computation model can, but also how efficient it can solve a problem.

1 Big-O Notation

The exact running time of an algorithm is often complicated, but we usually only care about an estimate. We use big-O notation to capture estimates. It helps us to understand how a function behaves on large inputs. We usually care more about large inputs, because even an inefficient algorithm can be fast on small inputs.

Example 1. $f(n) := 6n^3 + 2n^2 + 20n + 45 = O(n^3)$.

For polynomials, we consider only the highest order term and disregard coefficients to get an estimation in big-O form. [1]

Below we give a general definition for big-O notation.

Definition 2. *Let $f, g : \mathbb{N} \to \mathbb{R}^+$ be functions (positive integers to positive reals). We say $f(n) = O(g(n))$ if there exist constants $c, n_0 > 0$ such that for every $n \geq n_0$, we have $f(n) \leq c \cdot g(n)$.*

In the definition, we have the coefficient c so that we don't need to care about the coefficients in f, and we have $n \geq n_0$ so that we only need to care about big enough inputs.
Recall Example 1. Let $f(n) = 6n^3 + 2n^2 + 20n + 45, g(n) = n^3$. We can choose $c = 9$ and $n_0 = 45$ to fit in the above definition. [2] If $n \geq 45$, then $f(n) = 6n^3 + 2n^2 + 20n + 45$. Since $2 \leq n, 20 \leq n$ and $45 \leq n$, we have

$$f(n) \leq 6n^3 + n \cdot n^2 + n \cdot n + n = 7n^3 + n^2 + n \leq 7n^3 + n^3 + n^3 = 9n^3.$$

Example 3. $f(n) := 6n^3 + 2n^2 + 20n + 45 = O(n^4)$. Formally, if we pick $c = 9$ and $n_0 = 45$, then for all $n \geq 45$, we know $f(n) \leq 9 \cdot n^3 \leq 9 \cdot n^4$.

Example 4. $f(n) := 6n^3 + 2n^2 + 20n + 45 \neq O(n^2)$, because $6n^3$ grows faster than n^2.

Here's a more formal proof. Assume for contradiction that there exists c and n_0 such that for all $n \geq n_0$ we have $f(n) \leq cn^2$. $6n^3 + 2n^2 + 20n + 45 \leq cn^2$ implies that, $n^3 \leq cn^2$ which in turns

[1] For any polynomial $p(n) = c_0 + c_1 n + c_2 n^2 + \ldots + c_k n^k$, we have $p(n) \in O(n^k)$.

[2] Although $c = 7$ should work, picking a larger c makes the proof easier. There are a lot of possible choices of c and n_0.

means $n \le c$. Hence, we can't have $f(n) \le cn^2$ for all $n \ge n_0$, contradiction.

Intuition: We have $f(n) = O(g(n))$ if there exists a constant $c \ge 0$ such that $\lim_{n\to\infty} \frac{f(n)}{g(n)} \le c$. [3] Some simple examples:

- $2n^2 + n = O(n^2)$ since $\lim_{n\to\infty} \frac{2n^2+n}{n^2} = 2$.
- $2n^2 + n = O(n^3)$ since $\lim_{n\to\infty} \frac{2n^2+n}{n^3} = 0$.
- Finally $2n^2 + n \ne O(n)$ since $\lim_{n\to\infty} \frac{2n^2+n}{n} = \infty$.

Example 5. $\log_2(n) = O(\log_e(n))$. This is because $\log_2(n) = \frac{\log_e(n)}{\log_e(2)}$.

In particular, when using O notation, we often don't care so much what the base of the logarithm is.

Example 6. $\log(n) \ne O(\log_{\sqrt{n}}(n))$, since $\log_{\sqrt{n}}(n) = 2$. And $\log(n) \ne O(2)$.

Example 7. $\log_2(n) \ne O(\log_e \log_e(n))$. Substituting $k = \log_e(n)$ gives $k \ne O(\log k)$, since k grows faster than $\log(k)$.

Example 8. $n \cdot \log\log n = O(n \cdot \log n)$. It's easy to see $\log\log(n) = O(\log n)$ and we multiply both sides by n.

Example 9. $\log n \ne O((\log\log n)^{1000})$. If we substitute $k = \log\log(n)$, this becomes $2^k \ne O(k^{1000})$ [4]

Example 10. $\log n = O((\log\log n)^n)$. This is because $\log n = O(2^n) = O((\log\log n)^n)$ where we used the fact that for n large enough $\log\log(n) \ge 2$.

Example 11. $3^n \ne O(2^n)$. One way to see this is that $3^n/2^n = (1.5)^n \to \infty$ as $n \to \infty$. Another way is that $3^n = 2^{\log_2(3)\cdot n} = (2^n)^{\log_2(3)}$. If we substitute 2^n by k we have $k^{\log_2(3)} \notin O(k)$.

Example 12. $n\sqrt{2^n} = O(2^n)$. If we substitute 2^n by k we have $\log_2(k) \cdot \sqrt{k} = O(k)$. Which holds since for $k \ge 2$, $\log_2(k) \le \sqrt{k}$.

Below we use big-O notation for arithmetic expressions.

Example 13. $\sum_{i=1}^n i \le \sum_{i=1}^n n = n^2 = O(n^2)$.

There are some cases where we care about the leading constant and want a bound on the remaining terms, we can also use big-O notation, in a way similar to the following.

Example 14. $\sum_{i=1}^n i = \frac{1}{2}n^2 + \frac{1}{2}n = \frac{1}{2}n^2 + O(n)$.

[3] This isn't a formal definition for O. For instance this doesn't work if $g(n) = 0$ for some n. But this a helpful way to see if $f(n) = O(g(n))$.

[4] For any constant $c, c' > 0$, n^c always grows faster than $\log(n)^{c'}$. For instance: $n^{0.3} \ne O(\log(n)^{1000})$

Example 15. $\sum_{i=1}^{n} 2^i \le \sum_{i=1}^{n} O(2^n) \le O(n2^n)$ But this is not tight. Actually, $\sum_{i=1}^{n} 2^i = 2^{n+1} - 2 = O(2^n)$.

Example 16. $3^n = 2^{O(n)}$. The reason is $3^n = 2^{(\log_2 3)\cdot n} = 2^{O(n)}$. Here $O(n)$ is replacing an "anonymous function" that is bounded by $O(n)$. Compare to Example 11.

Example 17. $n^3 = 2^{O(\log n)}$. In fact, $n^3 = 2^{3\cdot \log_2 n}$.

$2^{O(\log n)}$ captures all polynomials in n (with positive leading coefficient). For example, $2^{100\cdot\log_2 n} = n^{100}$, $2^{10000\cdot\log_2 n} = n^{10000}$. Similarly, $n^{O(1)}$ also captures all polynomials in n. We use $\mathrm{poly}(n)$ to denote all polynomial in n.

Definition 18. *We use* $\mathrm{poly}(n)$ *to denote all polynomial in* n. *Formally* $f(n) = \mathrm{poly}(n)$ *if there exists constants* $c, n_0 \ge 0$ *such that for all* $n \ge n_0$ *we have* $f(n) \le n^c$.
In other words, $\mathrm{poly}(n) := n^{O(1)}$.

Example 19. $n \cdot \log^2(n) = \mathrm{poly}(n)$, since $n \cdot \log n = O(n^2)$ and $n^2 = \mathrm{poly}(n)$.

Example 20. $(\log n)^{100} = \mathrm{poly}(n)$. For n_0 large enough, we have that for all $n \ge n_0$, $\log(n) \le n$, meaning $(\log n)^{100} \le n^{100}$. So $(\log n)^{100} = \mathrm{poly}(n)$.

Example 21. $(\log n)^{\log\log n} = \mathrm{poly}(n)$. Note that $\log n = 2^{\log\log n}$, so

$$(\log n)^{\log\log n} = 2^{(\log\log n)^2} = 2^{O(\log(n))} = n^{O(1)}.$$

If the last step seems mysterious, here's more details. We know $(\log\log(n))^2 = O(\log(n))$. Hence, there exists constants c, n_0 such that for all $n \ge n_0$ we have: $(\log\log(n))^2 \le c\log(n)$. Hence for $n \ge n_0$ we have $(\log n)^{\log\log n} \le 2^{c\log(n)} = n^c$. Since c is a constant this implies $(\log n)^{\log\log n} = \mathrm{poly}(n)$.

Example 22. $\binom{n}{4} = \mathrm{poly}(n)$. This is because $\binom{n}{4} = \frac{n(n-1)(n-2)(n-3)}{24} = O(n^4)$ and $n^4 = \mathrm{poly}(n)$.

Example 23. It is not true that for all $0 \le k \le n$, $\binom{n}{k} = \mathrm{poly}(n)$. We always have $\binom{n}{k} = O(n^k)$ but k may depend on n and is not necessarily a constant. When $k = n/2$,

$$\binom{n}{n/2} = \frac{n!}{((n/2)!)^2} \approx \frac{2^n}{\sqrt{\pi n/2}},$$

However, if k is a fixed constant, then $\binom{n}{k}$ is a polynomial in n.

Here we formally prove it is false that $\binom{n}{k} \le poly(n)$ for all $0 \le k \le n$. Proof. Assume to the contrary that $\binom{n}{k} \le poly(n)$ for all k. Then,

$$\sum_{k=0}^{n} \binom{n}{k} \le \sum_{k=0}^{n} \mathrm{poly}(n) \le n \cdot \mathrm{poly}(n) \le \mathrm{poly}(n).$$

On the other hand, $\sum_{k=0}^{n} \binom{n}{k} = 2^n$. [5] This is not a poly$(n)$, so there is a contradiction.

[5] This can be proved by using two ways to count the number of subsets of a size-n set. On one hand, for each element, there are two possible choices: to be in the subset or not in the subset, so the total number of subsets is 2^n. On the other hand, the number of subsets of size k is $\binom{n}{k}$, so the total number is $\sum_{k=0}^{n} \binom{n}{k}$.

2 Big-Ω Notation

Definition 24 (Big-Ω Notation)**.** *Let $f, g : \mathbb{N} \to \mathbb{R}^+$ be functions. We say $f(n) = \Omega(g(n))$ if $g(n) = O(f(n))$*

Roughly speaking, using Ω means that $f(n)$ grows at least as fast as $g(n)$. Note that we can have $f(n) = O(g(n))$ and also $f(n) = \Omega(g(n))$.

Intuition: We have $f(n) = \Omega(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$.[6] Some simple examples:

- $2n^2 + n = \Omega(n^2)$ since $\lim_{n\to\infty} \frac{2n^2+n}{n^2} = 2$.
- $2n^2 + n = \Omega(n)$ because $\lim_{n\to\infty}(2n^2 + n)/n = \infty$
- Finally $2n^2 + n \notin \Omega(n^3)$ since $\lim_{n\to\infty}(2n^2 + n)/n^3 = 0$.

Example 25. $f(n) = 6n^3 + 2n^2 + 20n + 45$, then $f(n) = \Omega(n^3)$ and $f(n) = \Omega(n^2)$. However, $f(n) \neq \Omega(n^4)$.

Example 26. $\log_e(n) = \Omega(\log_2(n))$.

Example 27. $n^5 \log(n) = \Omega(n^5)$ but $n^5 \log(n) \neq \Omega(n^6)$. The inquality is because $\lim_{n\to\infty} n^5 \log(n)/n^6 = \lim_{n\to\infty} \log(n)/n = 0$

Example 28. $2^n \neq \Omega(3^n)$ and $2^n = 3^{\Omega(n)}$. The inequality holds because $3^n \notin O(2^n)$ (See example 11). For the equality we have $3^n = 2^{\log_3(2)\cdot n}$ and $\log_3(2) \cdot n = \Omega(n)$.

Example 29. $n^3 = 2^{\Omega(\log n)}$. Why ? $n^3 = 2^{3\log(n)}$ and $3\log(n) = \Omega(\log(n))$.

3 Little-o Notation

Definition 30 (Small-o)**.** *Let $f, g : \mathbb{N} \to \mathbb{R}^+$ be functions. We say $f(n) = o(g(n))$ if for all $c > 0$, there exists $n_0 > 0$ such that for all $n \geq n_0$, $f(n) < cg(n)$.*

Intuitively, o notation means that $f(n)$ grows at a strictly smaller speed than $g(n)$.

Theorem 31. *Let $f, g : \mathbb{N} \to \mathbb{R}^+$ be functions. We can't have both $f(n) = o(g(n))$ and $f(n) = \Omega(g(n))$.*

Intuition: If $\lim_{n\to\infty} f(n)/g(n) = 0$, then $f(n) = o(g(n))$. [7] Some simple examples:

- $2n^2 + n = o(n^3)$ since $\lim_{n\to\infty}(2n^2 + n)/n^3 = 0$.
- $2n^2 + n \neq o(n^3)$ since $\lim_{n\to\infty}(2n^2 + n)/n^2 = 2$

Example 32. $f(n) = 6n^3 + 2n^2$. Then $f(n) = o(n^4)$ but we have $f(n) \notin o(n^3)$.

Example 33. $\frac{1}{\sqrt{n}} = o(1)$. This is because for any $c > 0$, $\frac{1}{\sqrt{n}} < c$ once we take $n > \frac{1}{c^2}$. Another way to view is because $\lim_{n\to\infty} \frac{1}{\sqrt{n}} = 0$.

[6] This isn't a formal definition for Ω. But it's helpful to think about wether $f(n) = \Omega(g(n))$ or not.
[7] Again, this isn't a formal defintion.

Example 34. $\dfrac{n^2}{\log n} = o(n^2)$. Since $\lim_{n\to\infty} \frac{n^2}{\log n} \cdot \frac{1}{n^2} = \lim_{n\to\infty} \frac{1}{\log(n)} = 0$.

Example 35. $2^n = o(3^n)$, but $2^n \neq 3^{o(n)}$. For the first case, we have $2^n/3^n = (2/3)^n$ and this goes to 0 as $n \to \infty$. For the inequality $3^n = 2^{\log_2(3)\cdot n}$ and $n \neq o(\log_2(3) \cdot n)$.

4 So why all this notation ?

O-notation: We use it to show upper bounds. For instance, we want to say "problem X can be solved in time *at most* $O(n \log(n))$."

This means there exists an algorithm A for problem X which runs in time $f(n) = O(n \log(n))$.

Ω-notation: We use it to show lower bounds. For instance, we want to say "Any algorithm solving problem X can only be solved by an algorithm using at least $\Omega(n \log(n))$ time."

This means there doesn't exist an algorithm A, solving problem X which runs in time $f(n)$ where $f(n) \neq (n \log(n))$.

For instance, it's well known that for sorting an array of n integers, an algorithm needs $\Omega(n \log(n))$ time. On the other hand, Merge-Sort can sort an array in $O(n \log(n))$ time.

o-notation: We use it to mean strictly less. For an instance an algorithm with $o(n \log(n))$ time could have running time: $O(1), O(\sqrt{n}), O\left(\frac{n \log(n)}{\log\log(n)}\right)$.

We know that no algorithm for sorting an array of n elements can have running time $o(n \log(n))$ since we know there is an $\Omega(n \log(n))$ lower bound.

Another view: It's quite common in computer science to have the following scenario. There is problem X, where after many years of research, the best algorithm runs in $O(n^3)$ time. But we haven't been able to improve that upper bound. So, we wonder, is $o(n^3)$ running time possible ? I.e. can we get an asymptotically faster algorithm ? Or is there an $\Omega(n^3)$ lower bound ? Meaning no algorithm can be faster than $O(n^3)$. [8]

5 Some common tricks

Here's a list of common tricks used when trying to see if $f(n) = O(g(n))$.

- For any constants $a, b > 1$ we have $\log_a(n) = O(\log_b(n))$.
- Substituting $\log(n)$ with k. This is what we did in Example 7.
- Substituting $2^{f(x)}$ with k. This is what we did in Example 11.
- Use $n = 2^{\log(n)}$ (or $n^k = 2^{k\cdot\log(n)}$). More generally $f(x) = 2^{\log(f(x))}$. See for instance Examples 17, 21.
- We have that $2^{O(\log(n))} = \text{poly}(n)$. See Example 21
- If $f(n) = O(n^c)$, where c is come constant, then $f(n) = \text{poly}(n)$. See Examples 19 and 22.

[8] By Theorem 31: We can't have both an $\Omega(f(n))$ lower bound and $o(f(n))$ upper bound.

- If $f(n) = O(g(n))$ and $f'(n) = O(h(n))$, then $f(n) \cdot f'(n) = O(g(n) \cdot h(n))$.
- We have:

  1. $1 = O(\log(n))$
  2. $\log(n) = O(n^c)$ for any constant $c > 0$
  3. $n^c = O(n^{c+1})$ for any constant $c \geq 0$.
  4. $n^c = O(2^n)$ for any constant $c \geq 0$.
  5. For any polynomial $p(n)$ with highest power equal to n^k, we have $p(n) = O(n^k)$.