Lecture 13: Big-O Notation

Eric Vigoda
Discrete Mathematics for Computer Science

February 25, 2026

13 Asymptotic Growth and Big-O Notation

In this lecture we'll dive into $O()$ notation and how to compare functions $f(n)$ and $g(n)$ as $n \to \infty$.

13.1 Motivation: Comparing Growth Rates

In the study of algorithms, we will analyze the algorithm's running time in $O()$ notation, which captures its growth rate as n becomes large.

Let's begin with a simple, motivating problem.

Exercise.

- Let $f(n)$ be the number of n-bit strings. What is $f(n)$? _____
- Let $q(n)$ be the number of permutations of n people. What is $q(n)$? _____
- For large n, which quantity is dominant? (We will see the answer in a bit.)

$\square$ $f(n) = O(g(n))$ $\square$ $g(n) = O(f(n))$ $\square$ $f(n) = O(g(n))$ **and** $g(n) = O(f(n))$

More generally, we would like to compare expressions such as

$$n^2 \quad \text{and} \quad n \log n,$$
$$(\log n)^{100} \quad \text{and} \quad n^{0.01},$$
$$2^{\log_3 n} \quad \text{and} \quad 5^{\log_4 n}.$$

To reason about such questions systematically, we introduce asymptotic notation.

13.2 Graphical Intuition

Before giving a formal definition, it is helpful to think graphically. If we consider the following functions:

$$\log n, \quad n^{0.01}, \quad n, \quad n^2, \quad 2^n,$$

we observe a clear ordering in their growth rates, see Figure 1. For large n:

$$\log n \ll n^{0.01} \ll n \ll n^2 \ll 2^n.$$

The goal of Big-O notation is to make such comparisons precise.

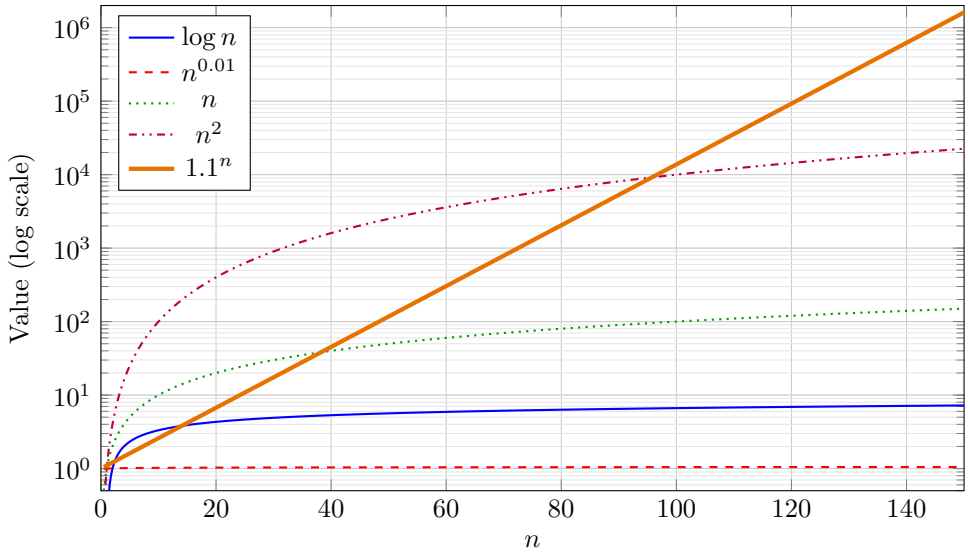

Figure 1: Growth comparison of representative functions on a logarithmic y-axis, for $0 \leq n \leq 150$.

13.3 Definition of Big-O

Formal Definition. For a function $g(n)$, we say that

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

if there exist constants $C > 0$ and $n_0 > 0$ such that

$$\forall n > n_0, \qquad g(n) \leq Cf(n).$$

Interpretation. For all sufficiently large n, the function $g(n)$ is at most a constant multiple of $f(n)$. Typically, you don't need to consider $n > n_0$ and it holds for all $n > 0$, possibly with a larger choice of C, but you can choose C as large as you'd like as long as it does not depend on n.

Big-O gives an **asymptotic upper bound**. It ignores:

- constant factors,
- lower-order terms.

13.4 Warm-Up Examples

Example 1.

$$f(n) = 50n^2 - 100n.$$

For large n, the n^2 term dominates. Thus,

$$f(n) = O(n^2).$$

Note, $f(n) = O(n^3)$ and $f(n) = O(2^n)$ but we are typically interested in the best possible upper bound which is $f(n) = O(n^2)$ in this case.

Example 2.

$$g(n) = 100n \log n + 500\sqrt{n}.$$

The $n \log n$ term dominates, so

$$g(n) = O(n \log n).$$

Since $n \log n = O(n^2)$, we conclude:

$$g(n) = O(f(n)), \qquad f(n) \neq O(g(n)).$$

13.5 Comparing log's

Example 3.

$$f(n) = 10\log_2 n, \qquad g(n) = \log_8 n$$

Since $8 = 2^3$, thus if $2^x = y$ then $8^x = (2^3)^x = (2^x)^3 = y^3$, and therefore:

$$\log_2(n) = \log_8(n^3) = 3\log_8(n).$$

Thus, $\log_2(n) = O(\log_8(n))$ and $\log_8(n) = O(\log_2(n))$.

> The base of the logarithm does not matter when comparing logarithms directly. However, when logarithms appear in exponents, the base can change the growth rate. Thus any fixed power of n dominates any fixed power of $\log n$.

13.6 A Key Recipe for Comparing Functions

To compare more complicated functions, we use the following method.

Step 0: To compare two functions $f(n)$ and $g(n)$, our first step is to express each as a polynomial in n; this means we try to express each function in the form n^c for a constant c. If this is not possible to do then we proceed as follows.

Step 1: Rewrite everything in terms of powers of 2.

Here are some convenient facts to keep in mind. By the definition of log we have the following two facts:

$$n = 2^{\log_2(n)}$$
$$\text{For any } b > 1\text{: } n = b^{\log_b(n)}$$

Furthermore, we have the following:

$$\text{For any } a > 0\text{:} \qquad n^a = 2^{a\log_2(n)}$$
$$\text{For any } k > 0\text{:} \qquad (\log n)^k = 2^{k\log\log n}$$

Finally, recall that

$$(n^a)^b = n^{ab} = \left(n^b\right)^a$$

Step 2: Compare exponents.

If we write two functions as

$$2^{A(n)} \quad \text{and} \quad 2^{B(n)},$$

then comparing growth reduces to comparing $A(n)$ and $B(n)$. This algebraic method avoids calculus and limits.

Growth Hierarchy Summary

For large n, the following ordering holds:

$$\log\log n \ll \log n \ll (\log n)^k \ll n^\epsilon \ll n \ll n\log n \ll n^2 \ll a^n \ll n!$$

for any fixed constants $k > 0$, $0 < \epsilon < 1$, and $a > 1$.

This ordering can often be verified by rewriting functions as powers of 2 and comparing their exponents.

13.7 Polynomial vs. Polylogarithm

Example 4. Compare

$$f(n) = (\log n)^{1000} \quad \text{and} \quad g(n) = n^{0.01}.$$

Rewrite:

$$f(n) = 2^{1000 \log\log n},$$
$$g(n) = 2^{0.01 \log n}.$$

Now compare the exponents:

$$1000 \log\log n \quad \text{vs.} \quad 0.01 \log n.$$

Since $\log n$ grows much faster than $\log\log n$, we conclude

$$(\log n)^{1000} = O(n^{0.01}),$$

but

$$n^{0.01} \neq O((\log n)^{1000}).$$

> Thus any fixed power of n dominates any fixed power of $\log n$.
>
> In other words, for every $\epsilon > 0$ and every constant $k > 0$,
>
> $$(\log n)^k = O(n^\epsilon), \qquad n^\epsilon \neq O((\log n)^k).$$

13.8 Factorial vs. Exponential

Example 5. Compare

$$f(n) = 2^n \quad \text{and} \quad g(n) = n!$$

We already have $f(n)$ written as a power of 2. Let us do so for $g(n)$ as well. When expanding $n!$, since the first $n/2$ terms are all at least $n/2$ we obtain the following:

$$g(n) = n! = n \times (n-1) \times (n-2) \times \cdots \times (1) > n \times (n-1) \times (n-2) \times \cdots \times (n/2) > (n/2)^{n/2} = 2^{Cn \log n}.$$

Since $n = O(n \log n)$ then $f(n) = O(g(n))$, but since $Cn \log n \neq O(n)$ then $g(n) \neq O(f(n))$.

13.9 Exponentials of Logarithms

Example 6. Compare

$$f(n) = 2^{\log_3 n} \quad \text{and} \quad g(n) = 5^{\log_4 n}.$$

Rewrite each as a polynomial. Since $2 = 3^{\log_3(2)}$ we can rewrite $f(n)$ in the following manner:

$$f(n) = 2^{\log_3 n} = \left(3^{\log_3(2)}\right)^{\log_3(n)} = \left(3^{\log_3(n)}\right)^{\log_3(2)} = n^{\log_3(2)},$$

and

$$g(n) = 5^{\log_4 n} = \left(4^{\log_4(5)}\right)^{\log_4(n)} = \left(4^{\log_4(n)}\right)^{\log_4(5)} = n^{\log_4(5)}$$

Then the comparison reduces to comparing the exponents

$$\log_3(2) < \log_3(3) = 1 \qquad \text{and} \qquad 1 = \log_4(4) < \log_4(5).$$

Thus, $f(n) = O(g(n))$, but $g(n) \neq O(f(n))$.

13.10 Useful Comparison Rules

- If $a \leq b$, then
$$n^a = O(n^b).$$
- Any polynomial dominates any polylogarithm:
$$(\log n)^{1000} = O(n^{0.01}), \qquad n^{0.01} \neq O((\log n)^{1000}).$$
- Any exponential a^n with $a > 1$ dominates any polynomial:
$$n^{1000} = O\left((1.01)^{0.001n}\right), \qquad (1.01)^{0.001n} \neq O(n^{1000})$$
- Logarithms with different bases differ only by constant factors:
$$\log_2 n = O(\log_{10} n) \quad \text{and} \quad \log_{10} n = O(\log_2 n).$$

13.11 Ω and Θ Notation

Whereas $O()$ notation corresponds to an upper bound, we use $\Omega()$ notation to capture a lower bound. We say
$$g(n) = \Omega(f(n))$$
if there exist constants $C > 0$ and n_0 such that
$$g(n) \geq Cf(n) \quad \text{for all } n > n_0.$$

And if a function $g(n)$ has a matching upper bound in $O()$ and lower bound in $\Omega()$ then we use $\Theta()$ to capture this. We say
$$g(n) = \Theta(f(n))$$
if both
$$g(n) = O(f(n)) \quad \text{and} \quad g(n) = \Omega(f(n)).$$

Thus $g(n) = \Theta(f(n))$ means $g(n)$ grows at the same rate as $f(n)$ up to constant factors.