COMPSCI 311 Section 1: Introduction to Algorithms

Lecture 2: Asymptotic Notation and Efficiency

Dan Sheldon

University of Massachusetts

Algorithm design

- Formulate the problem precisely
- Design an algorithm to solve the problem
- Prove the algorithm is correct
- Analyze the algorithm's running time

Example: Binary vs. Linear Search

An elegant algorithm you can teach to a 5-year old. You lose your page in 256-page book:

Linear search: 1, 2, 3, 4, . . . , 256. search up to 256 pages

Binary search: 128, 64, 32, 16, 8, 4, 2, 1. search up to 8 pages

# pages	linear	binary
256	256	8
512	512	9
1024	1024	10
2048	2048	11
n	$\leq n$	$\leq \log(n)$

Example: Binary vs. Linear Search

Board example: plot of n vs. $\log(n)$

Take-aways:

- Measure running time (# steps) as function of input size (n)
- Need tools to compare growth-rates of functions
- Big difference between brute-force and clever algorithms!

Big-O: Motivation

What is the running time of this algorithm? How many "primitive steps" are executed for an input array A of size n?

sum $= 0$
$n \leftarrow$ length of array A
for $i=$ 1 to n **do**
  for $j=$ 1 to n **do**
    sum $+=$ $A[i]$*$A[j]$

The (worst-case) running time as a function of n has the form

$$T(n) = an^2 + bn + c$$

We would like to coarsely categorize this as "order n^2" or $O(n^2)$

- Ignore constants, lower-order terms
- Need tools to compare growth rates of functions: "asymptotic order notation" (big-O)

Big-O: Formal Definition

Definition: The function $T(n)$ is $O(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that

$$T(n) \leq cf(n) \text{ for all } n \geq n_0$$

We say that f is an asymptotic upper bound for T.

Example:

$$\begin{aligned} T(n) &= 2n^2 + n + 2 \\ &\leq 2n^2 + n^2 + 2n^2 \qquad \text{if } n \geq 1 \\ T(n) &\leq \underbrace{5}_{c} n^2 \qquad \text{if } n \geq \underbrace{1}_{n_0} \end{aligned}$$

So $T(n)$ is $O(n^2)$

Example

Example: $T(n) = 2n^2 + n + 2$ is $O(n^3)$

$$
\begin{aligned}
T(n) &= 2n^2 + n + 2 \\
&\leq 2n^3 + n^3 + 2n^3 \qquad \text{if } n \geq 1 \\
T(n) &\leq \underbrace{5}_{c} n^3 \qquad\qquad \text{if } n \geq \underbrace{1}_{n_0}
\end{aligned}
$$

Big-O bounds do not need to be tight!

Big-O: Examples

Claim $n^2 + 10^6 n$ is $O(n^2)$

To prove this we need to show that

$$n^2 + 10^6 n \leq cn^2 \quad \text{for all } n \geq n_0$$

Clicker. Which values of c and n_0 make this statement true?

A. $c = 2$, $n_0 = 10^6$

B. $c = 10^6 + 1$, $n_0 = 1$

C. Both A and B

D. Neither A nor B

Big-O: Examples

- If $T(n) = n^2 + 10^6 n$ then $T(n)$ is $O(n^2)$
- If $T(n) = n^3 + n \log n$ then $T(n)$ is $O(n^3)$
- If $T(n) = 2^{\sqrt{\log n}}$ then $T(n)$ is $O(n)$

Clicker

Let $f(n) = 4n^2 + 23n \log_2 n + 500$. Which of the following are true?

A. $f(n)$ is $O(n^2)$

B. $f(n)$ is $O(n^3)$

C. Both A and B

D. Neither A nor B

The Big Idea: How to Use Big-O

Study pseudocode to determine running time $T(n)$ of an algorithm as a function of n:

$$T(n) = 2n^2 + n + 2$$

Prove that $T(n)$ is asymptotically upper-bounded by simpler function using big-O definition:

$$\begin{aligned} T(n) &= 2n^2 + n + 2 \\ &\le 2n^2 + n^2 + 2n^2 && \text{if } n \ge 1 \\ &\le 5n^2 && \text{if } n \ge 1 \end{aligned}$$

This is the right way to think about big-O, but too much work. We'll develop properties of big-O that simplify proving big-O bounds, **and use these properties to take shortcuts while analyzing algorithms** (you probably learned the shortcuts without knowing formal justification).

Properties of Big-O Notation

Claim (Transitivity): If f is $O(g)$ and g is $O(h)$, then f is $O(h)$.

Example:

- $\underbrace{2n^2 + n + 1}_{f(n)}$ is $O(\underbrace{n^2}_{g(n)})$
- $\underbrace{n^2}_{g(n)}$ is $O(\underbrace{n^3}_{h(n)})$
- Therefore, $2n^2 + n + 1$ is $O(n^3)$

Transitivity Proof

Claim (Transitivity): If f is $O(g)$ and g is $O(h)$, then f is $O(h)$.

Proof: we know from the definition that

- $f(n) \leq cg(n)$ for all $n \geq n_0$
- $g(n) \leq c'h(n)$ for all $n \geq n'_0$

Therefore

$$\begin{aligned} f(n) &\leq cg(n) && \text{if } n \geq n_0 \\ &\leq c(c'h(n)) && \text{if } n \geq n_0 \text{ and } n \geq n'_0 \\ &= \underbrace{cc'}_{c''} h(n) && \text{if } n \geq \underbrace{\max\{n_0, n'_0\}}_{n''_0} \\ f(n) &\leq c''h(n) && \text{if } n \geq n''_0 \end{aligned}$$

Know how to do proofs using Big-O definition.

Properties of Big-O Notation

Claims (Additivity):

- If f is $O(h)$ and g is $O(h)$, then $f + g$ is $O(h)$.

$$\underbrace{3n^2}_{O(n^5)} + \underbrace{n^4}_{O(n^5)} \text{ is } O(n^5)$$

- If f is $O(g)$, then $f + g$ is $O(g)$

$$\underbrace{n^3}_{g(n)} + \underbrace{23n + n\log n}_{f(n)} \text{ is } O(n^3)$$

Significance of Additivity

- OK to drop lower order terms:

$$2n^5 + 10n^3 + 4n\log n + 1000n \text{ is } O(n^5)$$

- Polynomials: Only highest-degree term matters. If $a_d > 0$ then:

$$a_0 + a_1 n + a_2 n^2 + \ldots + a_d n^d \text{ is } O(n^d)$$

- You are using additivity when you ignore the running time of statements outside for loops!

Other Useful Facts: Log vs. Poly vs. Exp

Fact: $\log_b(n)$ is $O(n^d)$ for all $b > 1, d > 0$

All polynomials grow faster than logarithm of any base

Fact: n^d is $O(r^n)$ when $r > 1$

Exponential functions grow faster than polynomials

Logarithm review

Definition: $\log_b(n)$ is the unique number c such that $b^c = n$

Informally: the number of times you can divide n into b parts until each part has size one

Properties:

- Log of product $\rightarrow$ sum of logs
  - $\log(xy) = \log x + \log y$
  - $\log(x^k) = k \log x$
- $\log_b(\cdot)$ is inverse of $b^{(\cdot)}$
  - $\log_b(b^n) = n$
  - $b^{\log_b(n)} = n$
- $\log_a n = \underbrace{\log_a b}_{\text{const.}} \cdot \log_b n$ (logs in any two bases are proportional)

When using big-O, it's OK not to specify base. Assume $\log_2$ if not specified.

Big-O comparison

Which grows faster?

$$n(\log n)^3 \quad \text{vs.} \quad n^{4/3}$$

simplifies to

$$(\log n)^3 \quad \text{vs.} \quad n^{1/3}$$

simplifies to

$$\log n \quad \text{vs.} \quad n^{1/9}$$

- We know $\log n$ is $O(n^d)$ for all $d > 0$
  - $\Rightarrow \log n$ is $O(n^{1/9})$
  - $\Rightarrow n(\log n)^3$ is $O(n^{4/3})$

Apply transformations (monotone, invertible) to both functions.
Try taking log.

Big-O: Correct Usage

Big-O: a way to categorize growth rate of functions relative to other functions.

Not: "***the*** running time of my algorithm".

Correct Usage:

- The worst-case running time of the algorithm in input of size n is $T(n)$.
- $T(n)$ is $O(n^3)$.
- The running time of the algorithm is $O(n^3)$.

Incorrect Usage:

- $O(n^3)$ is ***the*** running time of the algorithm. (There are many different asymptotic upper bounds to the running time of the algorithm.)