CS 312: Algorithms

Dan Sheldon

Mount Holyoke College

Last Compiled: September 10, 2018

Algorithm design

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

Big-O: Motivation

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

```
sum = 0
for i = 1 to n do
    for j = 1 to n do
        sum += A[i]*A[j]
    end for
end for
```

The running time is

$$T(n) = 2n^2 + n + 1 \ .$$

For large values of n, $T(n)$ is *less* than some multiple of n^2. We say $T(n)$ is $O(n^2)$ and we typically don't care about other terms.

Big-O: Formal Definition

Definition: The function $T(n)$ is $O(f(n))$ if there exist constants $c \geq 0$ 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.

Examples: work through / plot

- If $T(n) = n^2 + 1000000n$ 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 $O(n)$
- If $T(n) = n^3$ then $T(n)$ is $O(n^4)$ but it's also $O(n^3)$, $O(n^5)$ etc.

The Big Idea: How to Use Big-O

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

$$T(n) = 23n^2 + 17n + 15$$

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

$$\begin{aligned} T(n) &= 23n^2 + 17n + 15 \\ &\leq 23n^2 + 17n^2 + 15n^2 \quad \text{if } n \geq 1 \\ &\leq 55n^2 \quad \text{if } n \geq 1 \end{aligned}$$

This is the right way to think about big-O, but too much work. Therefore, we'll develop some mathematical properties of big-O that simplify proving big-O bounds for $T(n)$, **and use these properties to take shortcuts while analyzing algorithms** (that you probably already use).

Properties of Big-O Notation

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 $O(h)$.
- If $f_1, f_2, \dots, f_k$ are each $O(h)$, then $f_1 + f_2 + \dots + f_k$ is $O(h)$.
- If f is $O(g)$, then $f + g$ is $O(g)$.

We'll go through a couple of examples. . .

Consequences of Additivity

- OK to drop lower order terms. E.g., if

$$f(n) = 4.1n^3 + 23n + n \log n$$

  then $f(n)$ is $O(n^3)$
- Polynomials: Only highest degree term matters. E.g., if

$$f(n) = a_0 + a_1 n + a_2 n^2 + \dots + a_d n^d, \quad a_d > 0$$

  then $f(n)$ is $O(n^d)$

Other Useful Facts: Log vs. Poly vs. Exp

Fact: $\log_b(n)$ is $O(n^d)$ for all b and d

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

Informally: the number of times you can divide a 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 $b^{(\cdot)}$
  - $\log_b(b^n) = n$
  - $b^{\log_b(n)} = n$

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

Big-O sorting

Which grows faster, $n(\log n)^3$ or $n^{4/3}$?

Informal reasoning:

$$\begin{aligned} n(\log n)^3 &\le n^{4/3}? \\ (\log n)^3 &\le n^{1/3}? \\ \log n &\le n^{1/9}? \end{aligned}$$

Yes, because $\log n$ is $O(n^d)$ for all d. Therefore, $n(\log n)^3$ is $O(n^{4/3})$.

Apply transformations to both functions. Be careful that they preserve the inequality and are invertible. Try taking log.

Formal Proof

Informal reasoning from previous slide:

$$\begin{aligned} n(\log n)^3 &\le n^{4/3}? \\ (\log n)^3 &\le n^{1/3}? \\ \log n &\le n^{1/9}? \end{aligned}$$

Formal proof (go through transformations in reverse). We know $\log n$ is $O(n^{1/9})$, so there exist constants $c, n_0 \ge 0$ such that:

$$\begin{aligned} & \log n \le c n^{1/9} && \text{for all } n \ge n_0 \\ \iff \quad & (\log n)^3 \le c^3 n^{1/3} && \text{for all } n \ge n_0 \\ \iff \quad & n(\log n)^3 \le \underbrace{c^3}_{c'} n^{4/3} && \text{for all } n \ge n_0 \end{aligned}$$

Therefore, $n(\log n)^3$ is $O(n^{4/3})$.

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 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.)

Next time

- Big-Ω and Big-Θ notation