CMPSCI 311 Section 2: Introduction to Algorithms

Lecture 2: Asymptotic Notation and Efficiency

Dan Sheldon

University of Massachusetts

Last Compiled: January 25, 2017

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:

- 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.

Big-O: What it Is and Isn't

- **Is**: a way to categorize growth rate of (non-negative) functions relative to other functions.
- **Is not**: "the running time of my function"

Correct usage:

- The running time of my algorithm in input of size n is $T(n)$. Statement about algorithm only.
- $T(n)$ is $O(n^3)$. Statement about the function $T(n)$ only.
- The running time of my algorithm is $O(n^3)$. About algorithm **and** $T(n)$.

Incorrect usage:

- $O(n^3)$ is *the* running time of my algorithm

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

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

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 $f_1, f_2, \ldots, f_k$ are each $O(h)$, then $f_1 + f_2 + \ldots + f_k$ 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

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

  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?

$$n(\log n)^3 \text{ vs. } n^{4/3}$$
$$(\log n)^3 \text{ vs. } n^{1/3}$$
$$\log n \text{ vs. } n^{1/9}$$

- We know $\log n$ is $O(n^d)$ for all d
  - $\Rightarrow \log n$ 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-Ω Motivation

Algorithm **foo**
  for i= 1 to n **do**
    for j= 1 to n **do**
      do something...
    end for
  end for

Fact: run time is $O(n^3)$

Algorithm **bar**
  for i= 1 to n **do**
    for j= 1 to n **do**
      for k= 1 to n **do**
        do something else..
      end for
    end for
  end for

Fact: run time is $O(n^3)$

Conclusion: **foo** and **bar** have the same asymptotic running time.
What is wrong?

More Big-Ω Motivation

Algorithm **sum-product**

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

What is the running time of **sum-product**?

Easy to see it is $O(n^2)$. Could it be better? $O(n)$?

Big-Ω

Informally: T grows at least as fast as f

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

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

f is an asymptotic lower bound for T

Big-Ω

Exercise: let $T(n)$ be the running time of **sum-product**. Show that $T(n)$ is $\Omega(n^2)$

Algorithm **sum-product**

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

Do on board: easy way and hard way

Exercise review

Hard way

- Count exactly how many times the loop executes

$$1 + 2 + \ldots + n = \frac{n(n+1)}{2} = \Omega(n^2)$$

Easy way

- Ignore all loop executions where $i > n/2$ or $j < n/2$
- The inner statement executes at least $(n/2)^2 = \Omega(n^2)$ times

Big-Θ

Definition: the function $T(n)$ is $\Theta(f(n))$ if it is both $O(f(n))$ and $\Omega(f(n))$.

f is an asymptotically tight bound of T

Big-Θ example

How do we correctly compare the running time of these algorithms?

Algorithm **foo**

for i= 1 to n **do**
  for j= 1 to n **do**
    do something...
  end for
end for

Algorithm **bar**

for i= 1 to n **do**
  for j= 1 to n **do**
    for k= 1 to n **do**
      do something else..
    end for
  end for
end for

Answer: **foo** is $\Theta(n^2)$ and **bar** is $\Theta(n^3)$. They do not have the same asymptotic running time.

Additivity Revisited

Suppose f and g are two (non-negative) functions and f is $O(g)$

Old version: Then $f + g$ is $O(g)$
New version: Then $f + g$ is $\Theta(g)$

Example:

$$\underbrace{n^2}_{g} + \underbrace{42n + n \log n}_{f} \text{ is } \Theta(n^2)$$

Next Time

- When is an algorithm efficient?
- Graphs