CS 312: Algorithms

Dan Sheldon

Mount Holyoke College

Last Compiled: September 11, 2018

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:

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

Review: Asymptotics

Property	Definition / terminology
$f(n)$ is $O(g(n))$	$\exists c, n_0$ s.t. $f(n) \leq cg(n)$ for all $n \geq n_0$ g is an asymptotic upper bound on f
$f(n)$ is $\Omega(g(n))$	$\exists c, n_0$ s.t. $f(n) \geq cg(n)$ for all $n \geq n_0$ Equivalently: $g(n)$ is $O(f(n))$ g is an asymptotic lower bound on f
$f(n)$ is $\Theta(g(n))$	$f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$ g is an asymptotically tight bound on f

Algorithm design

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

Running Time Analysis

Mathematical analysis of worst-case running time of an algorithm as function of input size. Why these choices?

- Mathematical: describes the *algorithm*. Avoids hard-to-control experimental factors (CPU, programming language, quality of implementation).
- Worst-case: just works. ("average case" appealing, but hard to analyze)
- Function of input size: allows predictions. What will happen on a new input?

Efficiency

When is an algorithm efficient?

Stable Matching Brute force: $\Omega(n!)$
Propose-and-Reject?: $O(n^2)$

We must have done something clever

Polynomial Time

Working definition of efficient

Definition: an algorithm runs in polynomial time if its running time is $O(n^d)$ for some constant d

- Matches practice: almost all practically efficient algorithms have this property
- Usually distinguishes a clever algorithm from a "brute force" approach.
- Refutable: gives us a way of saying an algorithm is not efficient, or that no efficient algorithm exists.

Next Time

- Graphs