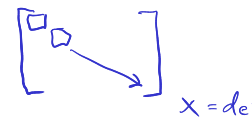
Algorithms and Operation Counting

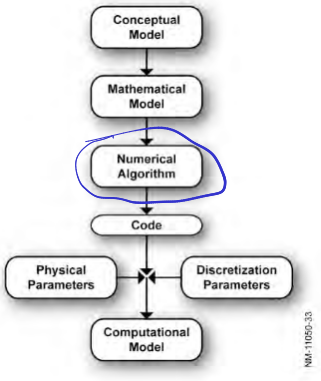
ASEN 3502 — Computational Methods

Please grab a handout in the back!

HW 1 - Due Thursday

Lab min through 1.3
rec through 1.5





Today
- Algorithms
- Operation Counting
- Asymptotic Notation

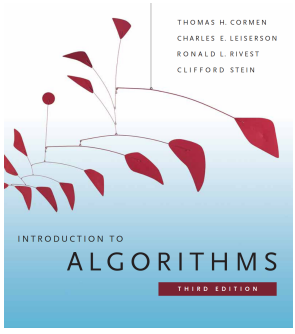
$O(n)$

ASME V&V 10, *Guide for Verification and Validation in Computational Solid Mechanics*.

What is an algorithm?

"Informally, an algorithm is any well-defined computational **procedure** that takes some value, or set of values, as **input** and produces some value, or set of values, as **output**."

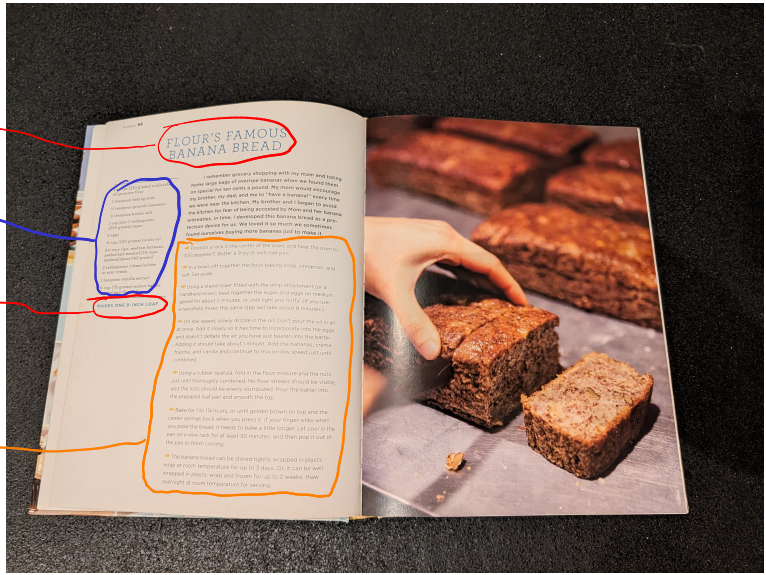
— Cormen et al.



Introduction to Algorithms
Cormen, Leiserson,
Rivest, and Stein

mitpress.mit.edu

Another type of algorithm



Why algorithms?

- Algorithms are mathematical objects. Why are they useful?

Why algorithms?

- Algorithms are mathematical objects. Why are they useful?
  - We can reason about properties like correctness and resource use

Why algorithms?

- Algorithms are mathematical objects. Why are they useful?
  - We can reason about properties like correctness and resource use

    > "An algorithm is said to be **correct** if, for every input instance, it halts with the correct output." — Cormen et al.

Why algorithms?

- Algorithms are mathematical objects. Why are they useful?
  - We can reason about properties like correctness and resource use

    > "An algorithm is said to be **correct** if, for every input instance, it halts with the correct output." — Cormen et al.
  - We can make precise and rigorous statements without having to deal with issues like roundoff error, memory allocation, etc.

How do we represent algorithms?

Matrix-vector multiplication: three ways

English

To get entry r of $\{b\}$, multiply each entry in row r of $[A]$ by the matching entry of $\{x\}$, and add up the results. Do this for every row.

Code

```
b = zeros(n,1);
for r = 1:n
    for c = 1:n
        b(r) = b(r) ...
             + A(r,c)*x(c);
    end
end
```

Pseudocode

input : $n \times n$ matrix $[A]$,
vector $\{x\}$
output : $\{b\} = [A]\{x\}$

for $r \leftarrow 1$ **to** n **do**
  $b[r] \leftarrow 0$
  for $c \leftarrow 1$ **to** n **do**
    $b[r] \leftarrow b[r] + A[r, c] \cdot x[c]$
  end
end

"What separates pseudocode from "real" code is that in pseudocode, we employ whatever expressive method is most clear and concise to specify a given algorithm." — Cormen et al.

Analysis of algorithms

- *Analyzing* an algorithm means predicting the resources that an algorithm requires
  - For today, we will count floating point operations ("Flops"): multiplication, division, addition, and subtraction of floating point numbers

Analysis of algorithms

- *Analyzing* an algorithm means predicting the resources that an algorithm requires
  - For today, we will count floating point operations ("Flops"): multiplication, division, addition, and subtraction of floating point numbers

- **Discuss with your neighbor:** Consider the equation $[A]\{x\} = \{b\}$. As n gets larger, which requires more floating point operations? Finding $\{x\}$ given $[A]$ and $\{b\}$, or finding $\{b\}$ given $[A]$ and $\{x\}$?

① $b = Ax$

② $Ax = b$ Harder if using Gaussian Elimination

Matrix-vector multiplication

input : $n \times n$ matrix $[A]$, vector $\{x\}$
output : $\{b\} = [A]\{x\}$

		flops	times
1	**for** $r \leftarrow 1$ **to** n **do**	0	$\sum_{r=1}^{n} 1 = n$
2	$b[r] \leftarrow 0$	0	n
3	**for** $c \leftarrow 1$ **to** n **do**	0	n^2
4	$b[r] \leftarrow b[r] + A[r,c] \cdot x[c]$	2	n^2
5	**end**		
6	**end**		

$$\sum_{r=1}^{n} \sum_{c=1}^{n} 1 = \sum_{r=1}^{n} n = n^2$$

FLOPS: $2n^2$

Gaussian elimination

input : $n \times n$ matrix $[A]$, right-hand side $\{b\}$
output : solution $\{x\}$ of $[A]\{x\} = \{b\}$

```
   // forward elimination
1  for c ← 1 to n − 1 do
2  |   for r ← c + 1 to n do
3  |   |   f ← A[r, c] / A[c, c]
4  |   |   for j ← c to n do
5  |   |   |   A[r, j] ← A[r, j] − f · A[c, j]
6  |   |   end
7  |   |   b[r] ← b[r] − f · b[c]
8  |   end
9  end

   // back substitution
10 for r ← n, n − 1, . . . , 1 do
11 |   s ← b[r]
12 |   for j ← r + 1 to n do
13 |   |   s ← s − A[r, j] · x[j]
14 |   end
15 |   x[r] ← s / A[r, r]
16 end
```

	flops	times
	0	$\sum_{i=1}^{n-1} 1 = n-1$
→	1	$\frac{n^2-n}{2}$
	0	
→	2	$\frac{n^3}{3} + O(n^2)$
→	2	$\frac{n^2-n}{2}$
	0	n
	0	n
	0	$\frac{n^2-n}{2}$
→	2	
→	1	n

Useful sum identities

$$\sum_{i=1}^{m} c f(i) = c \sum_{i=1}^{m} f(i) \qquad \sum_{i=1}^{m} f(i) + g(i) = \sum_{i=1}^{m} f(i) + \sum_{i=1}^{m} g(i)$$

$$\sum_{i=1}^{m} 1 = m \qquad \sum_{i=k}^{m} 1 = m - k + 1$$

$$\sum_{i=1}^{m} i = \frac{m(m+1)}{2} \qquad \sum_{i=1}^{m} i^2 = \frac{m(m+1)(2m+1)}{6} = \frac{m^3}{3} + O(m^2)$$

$$\sum_{c=1}^{n-1} \sum_{r=c+1}^{n} 1 = \sum_{c=1}^{n-1} n - c = \sum_{c=1}^{n-1} n - \sum_{c=1}^{n-1} c$$

$$= n(n-1) - \frac{(n-1)n}{2} = \frac{n^2-n}{2}$$

$$\sum_{c=1}^{n-1} \sum_{r=c+1}^{n} \sum_{j=c}^{n} 1 = \frac{n^3}{3} - \frac{n}{3}$$

see flop-count.pdf section 2.2

https://asen3502.com/lectures/020-algorithms/flop-count.pdf

FLOPS : $\frac{n^2-n}{2} + 2\left(\frac{n^3}{3} + O(n^2)\right) + 2\frac{n^2-n}{2}$

FLOPS $2\frac{n^3}{3} + O(n^2) + 2\frac{n^2-n}{2} + n$

```
function [C] = matmult([A], [B])
    for i = 1 to n
        for j = 1 to n
            c_ij = 0
            for k = 1 to n
                c_ij = c_ij + a_ik b_kj
            end
        end
    end
    return [C]
end
```

flops	times
0	n
0	n^2
0	⋮
0	n^3
2	n^3

Asymptotic notation

- Asymptotic notation is a tool for keeping track of only the terms that are most important as a function approaches an asymptote

Asymptotic notation

- Asymptotic notation is a tool for keeping track of only the terms that are most important as a function approaches an asymptote
- Chapra: $O(m^n)$ means "terms of order m^n and lower"

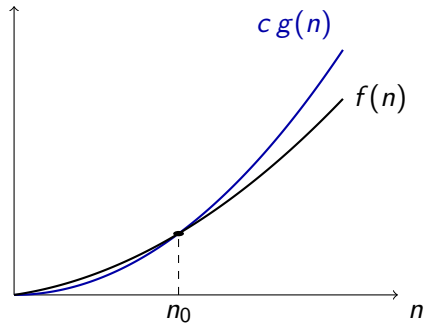
Asymptotic notation

- Asymptotic notation is a tool for keeping track of only the terms that are most important as a function approaches an asymptote
- Chapra: $O(m^n)$ means "terms of order m^n and lower"

Definition

$f(n) = O(g(n))$ as $n \to \infty$ if there exist positive constants c and n_0 such that

$$0 \leq f(n) \leq c\,g(n) \quad \text{for all } n \geq n_0$$



$f(n) \leq c\,g(n)$ for all $n \geq n_0$

← $f(n) = 3n^2 + 5n + 2$

$f(n) = 3n^2 + 5n + 2$

$c = 4.0$

$n_0 = 6$

then $0 \leq f(n) \leq c\,g(n)$

for all $n \geq n_0$

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

$c \geq$ sum of coefficients

$n_0 = 1$

$c = 10$

$n_0 = 1$

Useful Rules

1) $c\,f(n) = O(f(n)) \quad \forall\ c > 0$
2) if $f_1(n) = O(n^a)$ and $f_2(n) = O(n^b)$, $f_1(n) + f_2(n) = O(n^{\max(a,b)})$
3) if $f_1(n) = O(n^a)$ and $f_2(n) = O(n^b)$, $f_1(n) \cdot f_2(n) = O(n^{a+b})$
4) if $|f(n)|$ is bounded, $f(n) = O(1)$

Example: $f(n) = n^2 + 12n\sin(n) + 50 = O(n^2)$

n^2: $O(n^2)$; $12n\sin(n)$: $O(n)$ $O(1)$; 50: $50 \cdot 1$, $O(1)$

Gaussian elimination

input : $n \times n$ matrix $[A]$, right-hand side $\{b\}$
output : solution $\{x\}$ of $[A]\{x\} = \{b\}$

```
// forward elimination
for c ← 1 to n − 1 do                                  ≤ n
    for r ← c + 1 to n do                              ≤ n
        f ← A[r, c] / A[c, c]
        for j ← c to n do                              ≤ n
            A[r, j] ← A[r, j] − f · A[c, j]
        end
        b[r] ← b[r] − f · b[c]
    end
end
// back substitution
for r ← n, n − 1, . . . , 1 do
    s ← b[r]
    for j ← r + 1 to n do
        s ← s − A[r, j] · x[j]
    end
    x[r] ← s / A[r, r]
end
```

flops	Order of times called
×	$O(n)$
×	$O(n^2)$
✓	
×	$O(n^3)$
✓	
×	$O(n^2)$
✓	
×	$O(n)$
×	
×	$O(n^2)$
✓	
✓	$O(n)$

$O(n^3)$ Flops