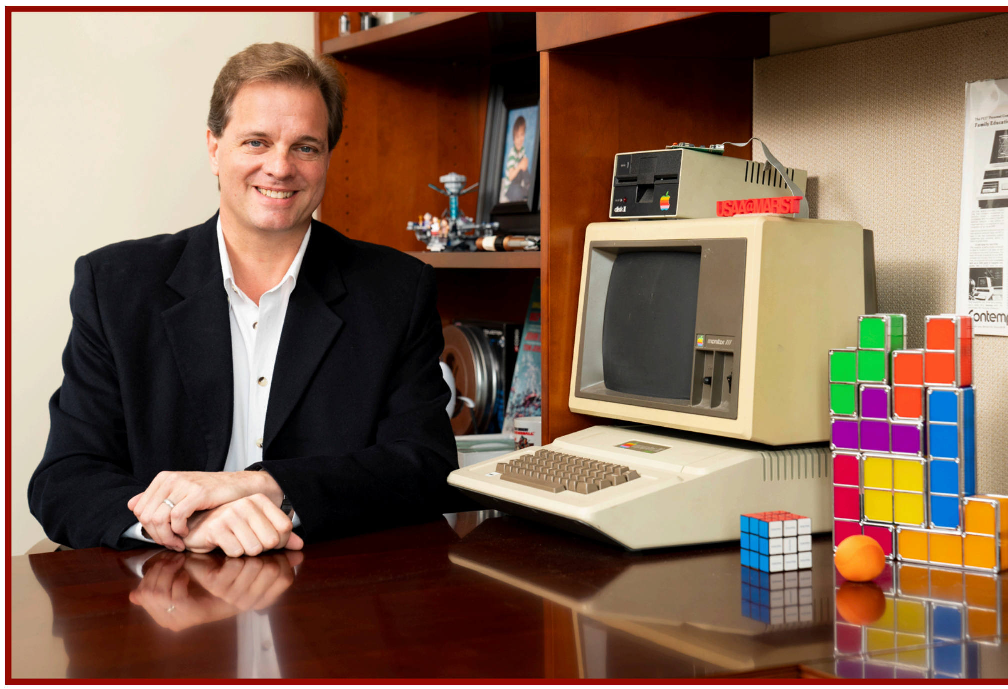
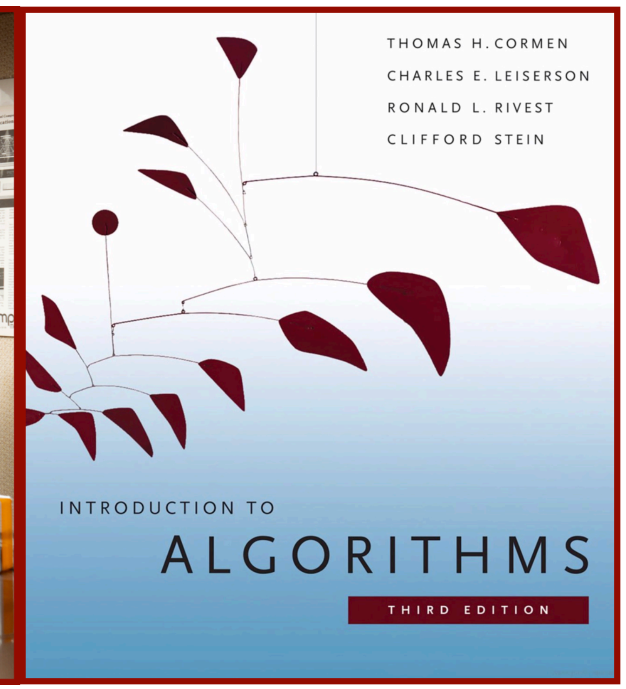
Studying Algorithms





Alan G. Labouseur, Ph.D.
Alan.Labouseur@Marist.edu

Algorithms

?

Algorithms

General recipes for solving problems…

?

Algorithms

General recipes for solving problems
not specific of any language or platform.

!

Algorithms

General recipes for solving problems
not specific of any language or platform.

We specify the input, desired output, and the steps to get from one to the other.

Algorithms

General recipes for solving problems
not specific of any language or platform.

We specify the input, desired output, and the steps to get from one to the other. Like a cookbook.

Coq au Vin (partial recipe)

While the liquid is boiling, in a small bowl, blend the 3 tablespoons flour and 2 tablespoons softened butter into a smooth paste.

Beat the flour/butter mixture into the approximately 2 cups hot cooking liquid with a whisk.

Simmer and stir for a minute or two **until** the sauce has thickened.

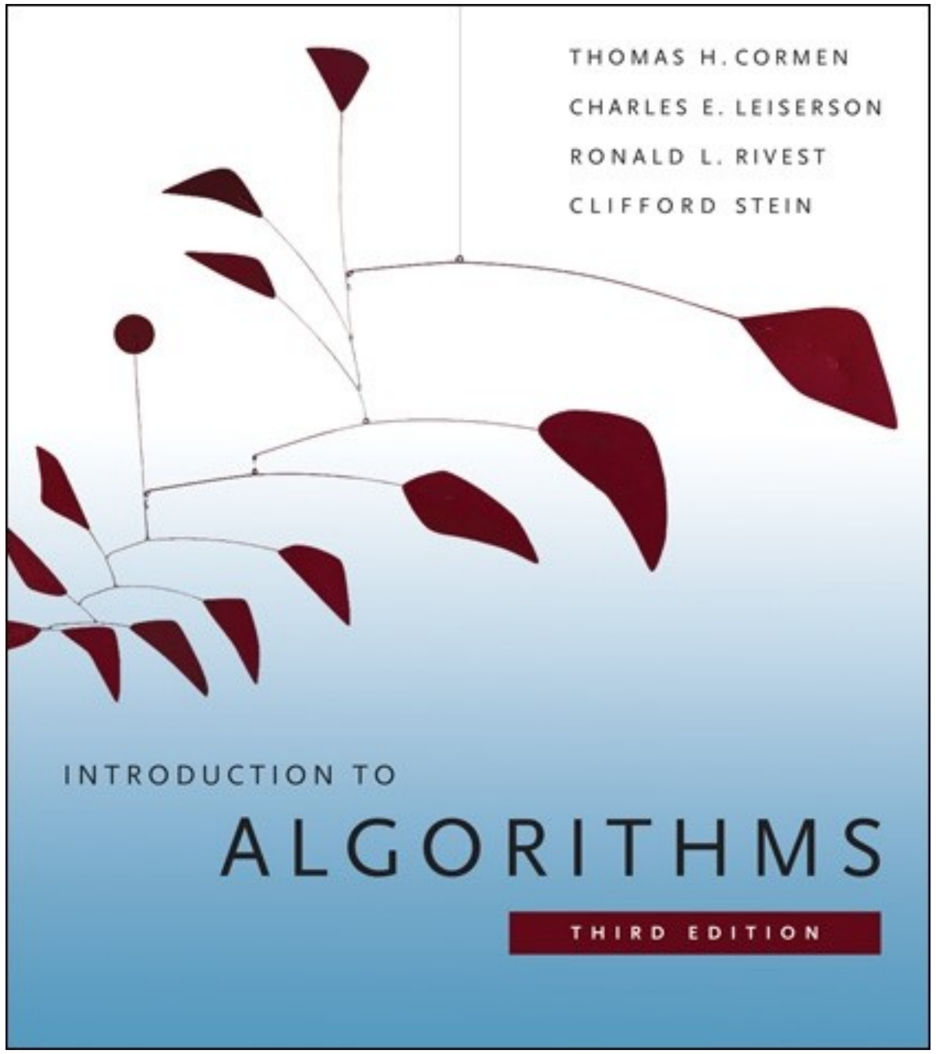
If the sauce doesn't thicken right away...

Sequence, Alternation, and Repetition.

Algorithms

General recipes for solving problems
not specific of any language or platform.

We specify the input, desired output, and the steps to get from one to the other. Like a ~~cookbook~~ textbook..



- Searching
- Sorting
- Data Structures
- Graphs
- Trees
- Dynamic Programming
- and more . . .

Algorithms

General recipes for solving problems
not specific of any language or platform.

Challenges:

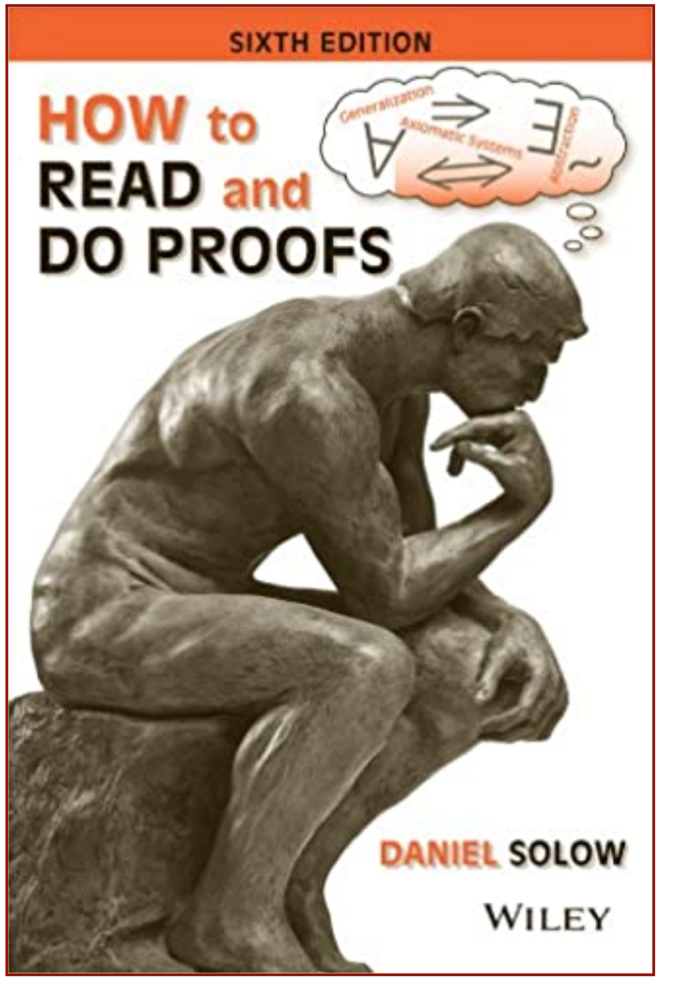
- Correctness
- Efficiency
- Applicability

Algorithms

General recipes for solving problems
not specific of any language or platform.

Challenges:

- Correctness
- Efficiency
- Applicability



Does it work? I.e.,
Is the output correct for for **all** inputs,
all instances, and **all** edge cases?

Also, does it halt?

And can you prove it?

Algorithms

General recipes for solving problems
not specific of any language or platform.

Challenges:

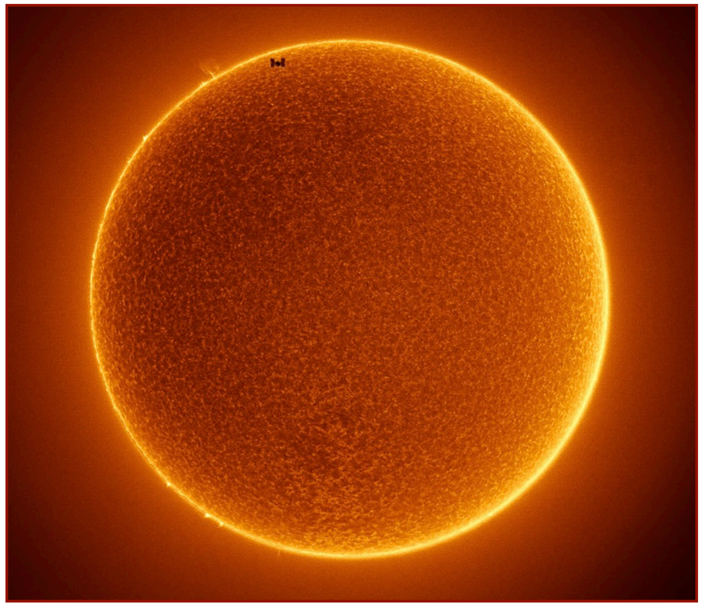
- Correctness
- Efficiency
- Applicability

Does it work

- in a reasonable amount of time?
- with a reasonable amount of effort?
- using a reasonable amount of resources?

Algorithms

General recipes for solving problems
not specific of any language or platform.



Challenges:

- Correctness
- Efficiency
- Applicability

Does it work

- in a **reasonable** amount of **time**?
- with a reasonable amount of effort?
- using a reasonable amount of resources?

By the way, what's reasonable?

Algorithms

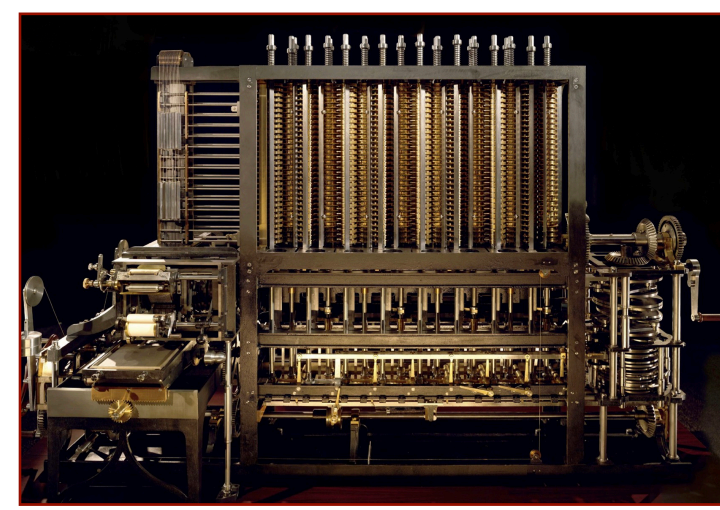
General recipes for solving problems
not specific of any language or platform.

Challenges:

- Correctness
- Efficiency
- Applicability

Does it work

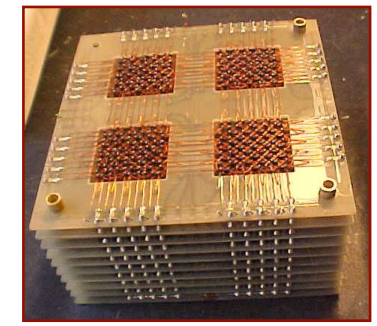
- in a reasonable amount of time?
- with a **reasonable** amount of **effort**?
- using a reasonable amount of resources?



By the way, what's reasonable?

Algorithms

General recipes for solving problems
not specific of any language or platform.

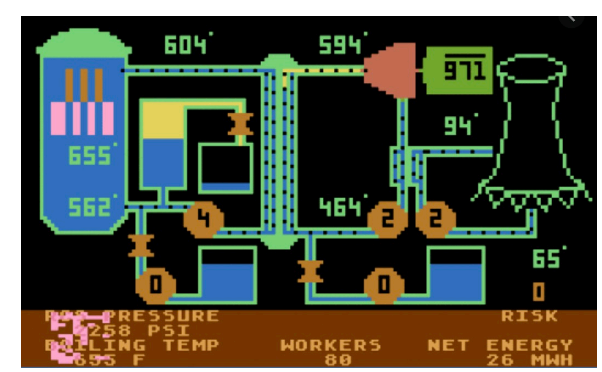


Challenges:

- Correctness
- Efficiency
- Applicability

Does it work

- in a reasonable amount of time?
- with a reasonable amount of effort?
- using a **reasonable** amount of **resources**?



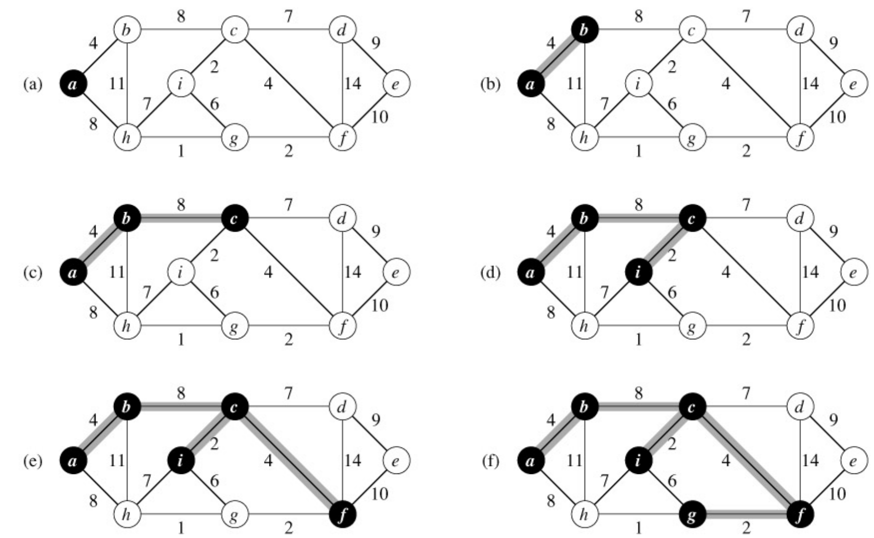
By the way, what’s reasonable?

Algorithms

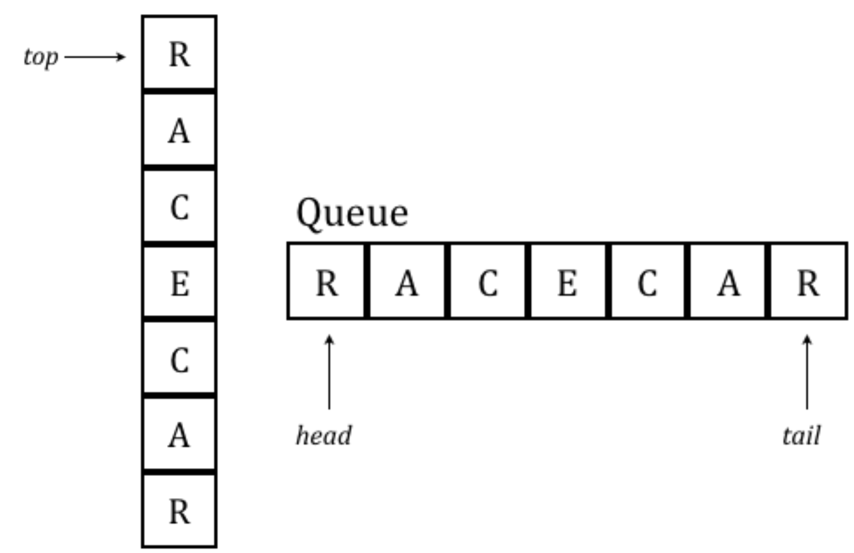
General recipes for solving problems
not specific of any language or platform.

Challenges:

- Correctness
- Efficiency
- Applicability



There are many algorithms.
There are many data structures.
Did you choose
  the right one at
  the right time in
  the right place for
  the right use case?



Stack

Or were you in the right place at the wrong time?

Algorithms

General recipes for solving problems
not specific of any language or platform.

Q: How can we characterize algorithms in a manner that's **not** specific of any language or platform?

?

Algorithms

General recipes for solving problems
not specific of any language or platform.

Q: How can we characterize algorithms in a manner that's **not** specific of any language or platform?

A: Growth functions.

Examples:

$O(n)$	"Order **n**" or "Big-oh of **n**"
$O(n^2)$	"Order **n squared**" or "Big-oh of **n squared**"
$O(\log_2 n)$	"Order **log to the base two of n**" or . . .

Algorithms

General recipes for solving problems
not specific of any language or platform.

Q: How can we characterize algorithms in a manner that's **not** specific of any language or platform?

A: Growth functions let us characterize how the *time*/*effort*/*space* required to execute the algorithm grows as the size of the input grows.

Think of this as "complexity".

We're concerned with the measures of effort/complexity needed to correctly solve a problem.

We're also concerned with how those measures change proportionally with the size of the input. I.e., how does the effort scale or grow with the input? What is its "*order of growth*"?

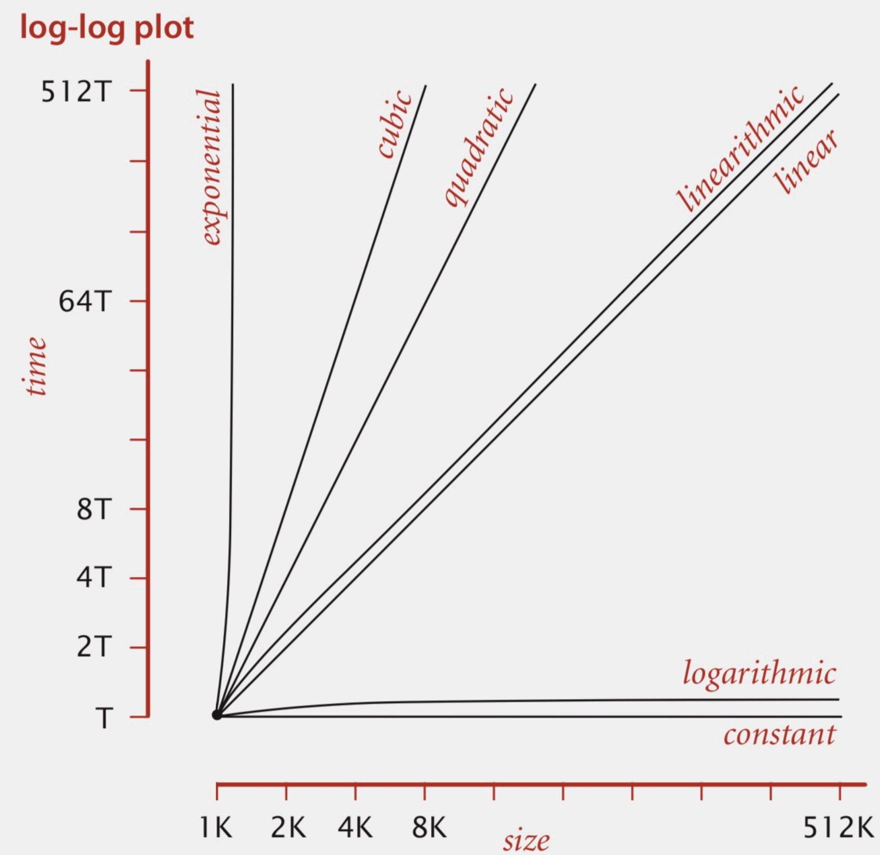
Algorithms

Characterizing algorithms in terms of complexity

Good news. The set of functions

$1,\ \log n,\ n,\ n \log n,\ n^2,\ n^3$, and 2^n

suffices to describe the order of growth of most common algorithms.



Typical orders of growth

from Robert Sedgewick and Kevin Wayne's Princeton Algorithms *course notes*

Characterizing algorithms in terms of complexity

Common order-of-growth classifications

order of growth	name	typical code framework	description	example	$T(2n) / T(n)$
1	**constant**	`a = b + c;`	statement	add two numbers	1
$\log n$	**logarithmic**	`while (n > 1)` `{ n = n/2; ... }`	divide in half	binary search	~ 1
n	**linear**	`for (int i = 0; i < n; i++)` `{ ... }`	single loop	find the maximum	2
$n \log n$	**linearithmic**	*mergesort*	divide and conquer	mergesort	~ 2

n^2	**quadratic**	`for (int i = 0; i < n; i++)`
		`for (int j = 0; j < n; j++)`
		`{ ... }`
			double loop	check all pairs	4

n^3	**cubic**	`for (int i = 0; i < n; i++)`
		`for (int j = 0; j < n; j++)`
		`for (int k = 0; k < n; k++)`
		`{ ... }`
			triple loop	check all triples	8

2^n	**exponential**	*combinatorial search*	exhaustive search	check all subsets	2^n

from Robert Sedgewick and Kevin Wayne's Princeton Algorithms *course notes*

Algorithms

Ponder this …

1.2-3
What is the smallest value of n such that an algorithm whose running time is $100n^2$ runs faster than an algorithm whose running time is 2^n on the same machine?

… and see if you can produce this graph in Desmos: