Analysis of Algorithms I

Dr. Mark Matthews

Lectures & Practicals

`

When	Where	Who
Tue 3pm	**B004**	**Mark**
Thurs 4pm	**B004**	**Mark**

When	Where	Who
Mon 4-6pm	**Rooms B108, B109, Arts Building**	**Luca**

Assessment

Assignment	30%
Final Examination	65%
Code repository submitted at end of semester.	5%

Practicals

Will be based on a recent topic from the course.
Will combine theory and/or implementation.

Algorithms

in which

you learn what an algorithm is and how to choose a good one.

Summary

- Brief history of algorithms
- What is an algorithm?
- Measuring the Performance of your Algorithms
- Turning the crank: time
- Complexity analysis

What is an algorithm?

What is an algorithm?

Search for a word

algorithm

/ˈalgərɪð(ə)m/

noun

A word used by programmers (and other humans) when they want to sound like they did something complex.

"I need to optimize my algorithm"

Translations, word origin and more definitions

What is an algorithm?

Search for a word

algorithm

/ˈalgərɪð(ə)m/

noun

a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.

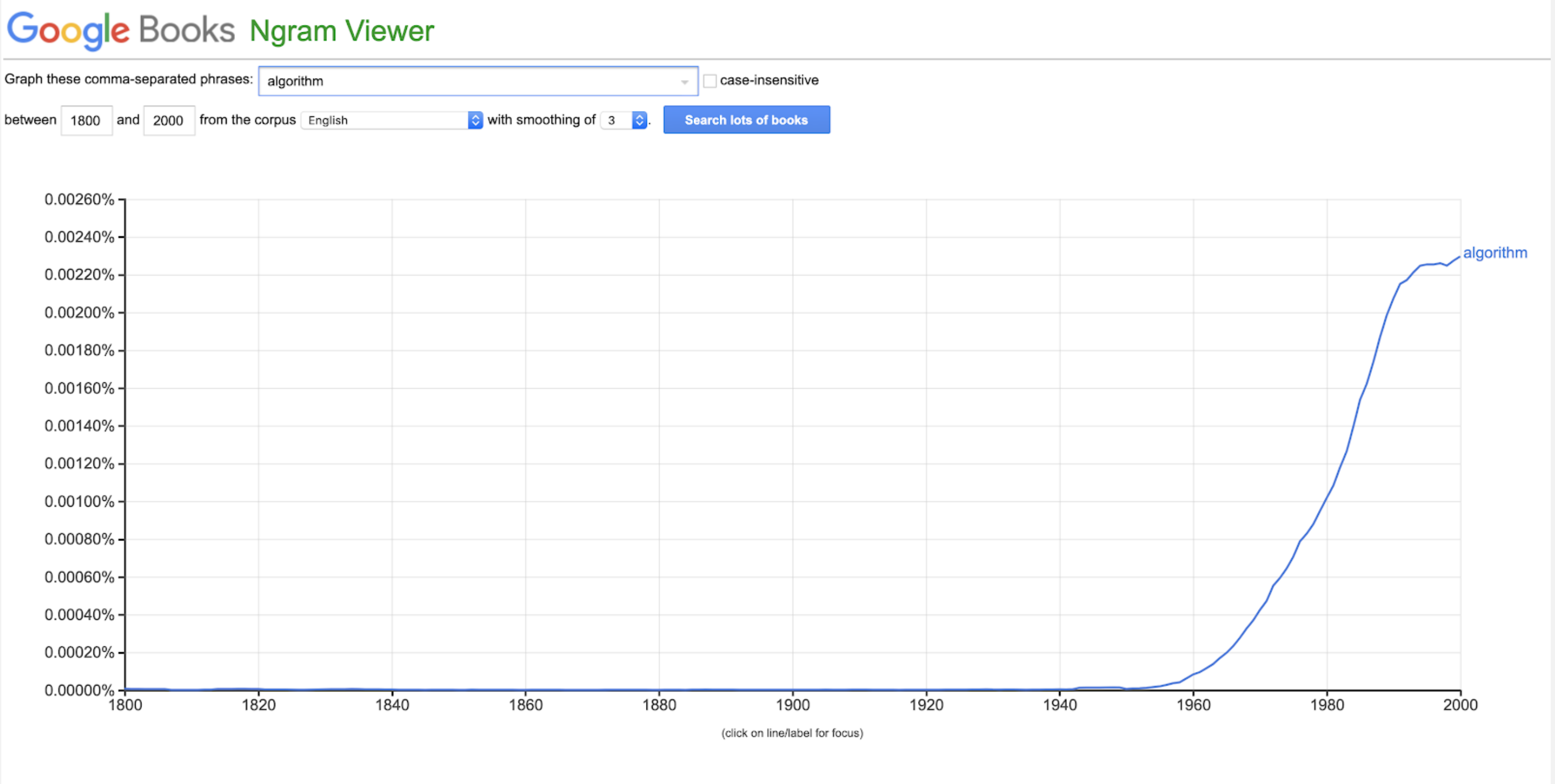
"a basic **algorithm for** division"

Translations, word origin and more definitions

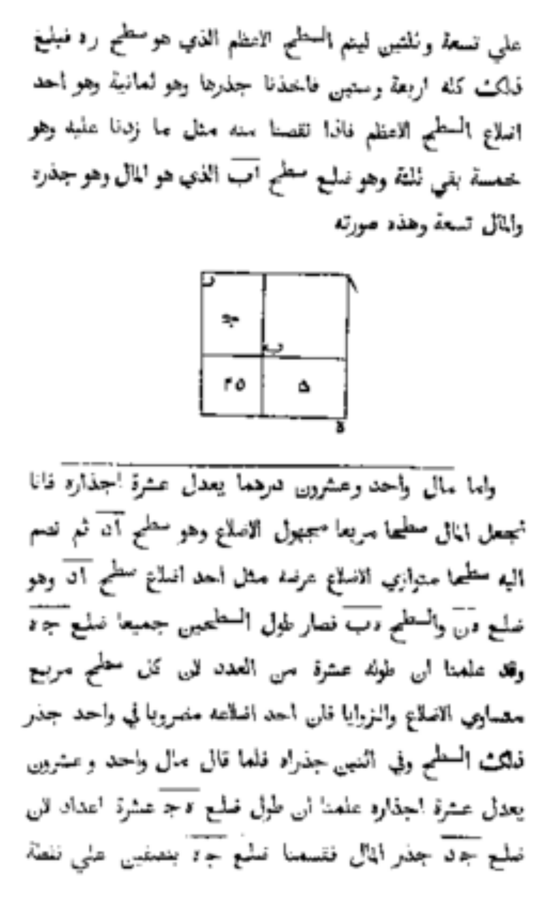
An algorithm is nothing fancy. It is just a step-by-step method of doing something.

What is an algorithm? Use of the word algorithm

Algorithms have likely been around since thousands of years ago. They have come to dominate human life over the past century with the advent of computers.



What is an algorithm?



علي تسعة وثلثين ليتم السطح الاعظم الذي هو سطح ره فبلغ
ذلك كله اربعة وستين فاخذنا جذرها وهو ثمانية وهو احد
اضلاع السطح الاعظم فاذا نقصنا منه مثل ما زدنا عليه وهو
خمسة بقي ثلثة وهو ضلع سطح اب الذي هو المال وهو جذره
والمال تسعة وهذه صورته

واما مال واحد وعشرون درهما يعدل عشرة اجذاره فانا
نجعل المال سطحا مربعا مجهول الاضلاع وهو سطح اد ثم نضم
اليه سطحا متوازي الاضلاع عرضه مثل احد اضلاع سطح اد وهو
ضلع دن والسطح هب فصار طول السطحين جميعا ضلع جه
وقد علمنا ان طوله عشرة من العدد لان كل سطح مربع
متساوي الاضلاع والزوايا فان احد اضلاعه مضروبا في واحد جذر
ذلك السطح وفي اثنين جذراه فلما قال مال واحد وعشرون
يعدل عشرة اجذاره علمنا ان طول ضلع هج عشرة اعداد لان
ضلع جد جذر المال فقسمنا ضلع جه بنصفين على نقطة

Muḥammad ibn Mūsā al-Khwārizmī
9th Century Persian Mathematician

The very first computer algorithm?

Ada Lovelace

Bernoulli Numbers

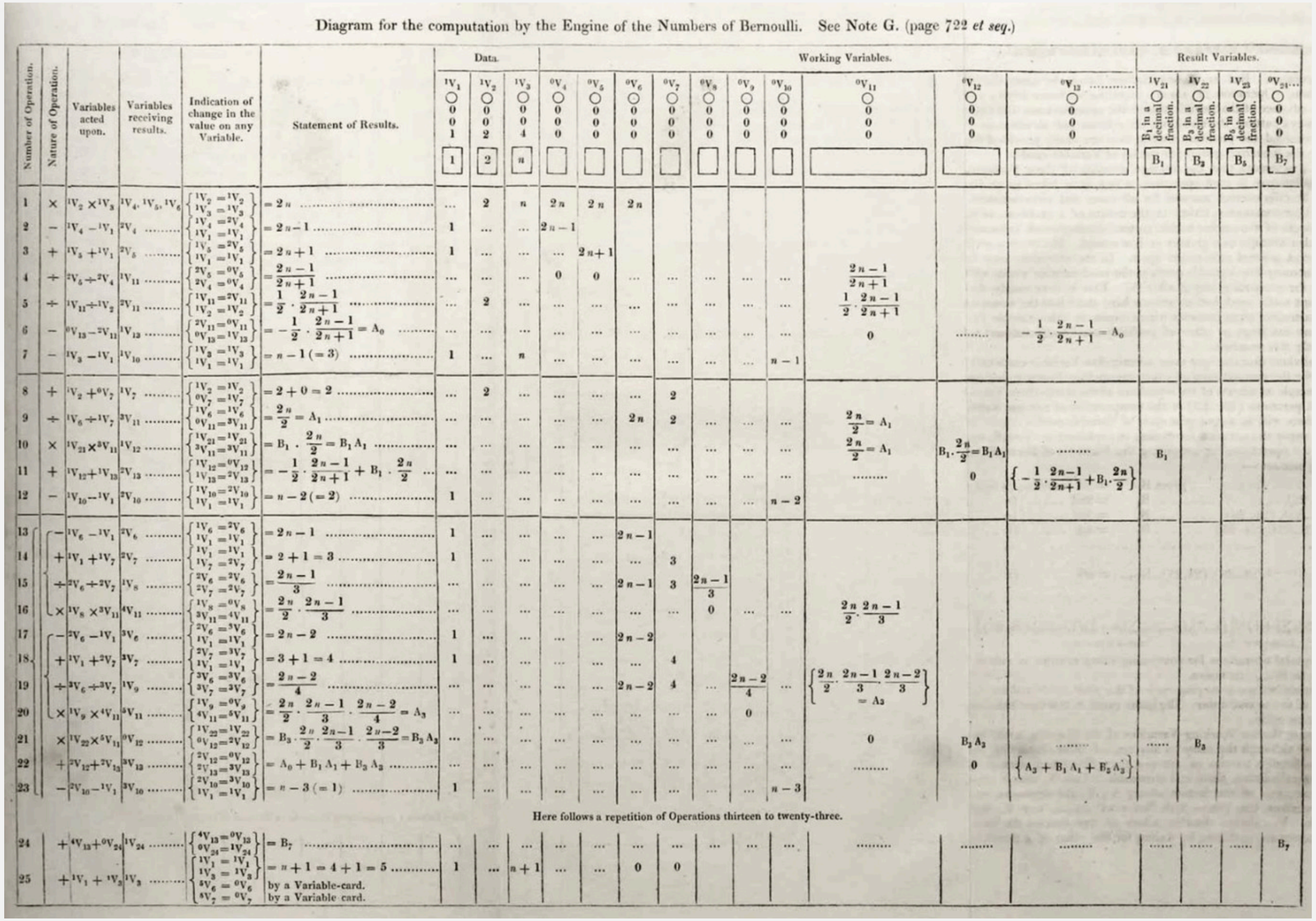


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.	Data. $^1V_1$ 0001 [1]	$^1V_2$ 0002 [2]	$^1V_3$ 0004 [n]	Working Variables. $^0V_4$ 0000	$^0V_5$ 0000	$^0V_6$ 0000	$^0V_7$ 0000	$^0V_8$ 0000	$^0V_9$ 0000	$^0V_{10}$ 0000	$^0V_{11}$ 0000	$^0V_{12}$ 0000	$^0V_{13}$ 0000	Result Variables. $^1V_{21}$ B_1 in a decimal fraction. [B_1]	$^1V_{22}$ B_3 in a decimal fraction. [B_3]	$^1V_{23}$ B_5 in a decimal fraction. [B_5]	$^0V_{24}$ 0000 [B_7]
1	×	$^1V_2 \times {}^1V_3$	$^1V_4, {}^1V_5, {}^1V_6$	$\{^1V_2 = {}^1V_2;\ ^1V_3 = {}^1V_3\}$	$= 2n$		2	n	$2n$	$2n$	$2n$											
2	−	$^1V_4 - {}^1V_1$	$^2V_4$	$\{^1V_4 = {}^2V_4;\ ^1V_1 = {}^1V_1\}$	$= 2n-1$	1			$2n-1$													
3	+	$^1V_5 + {}^1V_1$	$^2V_5$	$\{^1V_5 = {}^2V_5;\ ^1V_1 = {}^1V_1\}$	$= 2n+1$	1				$2n+1$												
4	÷	$^2V_5 \div {}^2V_4$	$^1V_{11}$	$\{^2V_5 = {}^0V_5;\ ^2V_4 = {}^0V_4\}$	$= \frac{2n-1}{2n+1}$				0	0						$\frac{2n-1}{2n+1}$						
5	÷	$^1V_{11} \div {}^1V_2$	$^2V_{11}$	$\{^1V_{11} = {}^2V_{11};\ ^1V_2 = {}^1V_2\}$	$= \frac{1}{2} \cdot \frac{2n-1}{2n+1}$		2									$\frac{1}{2} \cdot \frac{2n-1}{2n+1}$						
6	−	$^0V_{13} - {}^2V_{11}$	$^1V_{13}$	$\{^2V_{11} = {}^0V_{11};\ ^0V_{13} = {}^1V_{13}\}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$											0		$-\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$				
7	−	$^1V_3 - {}^1V_1$	$^1V_{10}$	$\{^1V_3 = {}^1V_3;\ ^1V_1 = {}^1V_1\}$	$= n - 1\ (= 3)$	1		n							$n-1$							
8	+	$^1V_2 + {}^0V_7$	$^1V_7$	$\{^1V_2 = {}^1V_2;\ ^0V_7 = {}^1V_7\}$	$= 2 + 0 = 2$		2					2										
9	÷	$^1V_6 \div {}^1V_7$	$^3V_{11}$	$\{^1V_6 = {}^1V_6;\ ^0V_{11} = {}^3V_{11}\}$	$= \frac{2n}{2} = A_1$						$2n$	2				$\frac{2n}{2} = A_1$						
10	×	$^1V_{21} \times {}^3V_{11}$	$^1V_{12}$	$\{^1V_{21} = {}^1V_{21};\ ^3V_{11} = {}^3V_{11}\}$	$= B_1 \cdot \frac{2n}{2} = B_1 A_1$											$\frac{2n}{2} = A_1$	$B_1 \cdot \frac{2n}{2} = B_1 A_1$		B_1			
11	+	$^1V_{12} + {}^1V_{13}$	$^2V_{13}$	$\{^1V_{12} = {}^0V_{12};\ ^1V_{13} = {}^2V_{13}\}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}$												0	$\left\{-\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}\right\}$				
12	−	$^1V_{10} - {}^1V_1$	$^2V_{10}$	$\{^1V_{10} = {}^2V_{10};\ ^1V_1 = {}^1V_1\}$	$= n - 2\ (= 2)$	1									$n-2$							
13	−	$^1V_6 - {}^1V_1$	$^2V_6$	$\{^1V_6 = {}^2V_6;\ ^1V_1 = {}^1V_1\}$	$= 2n - 1$	1					$2n-1$											
14	+	$^1V_1 + {}^1V_7$	$^2V_7$	$\{^1V_1 = {}^1V_1;\ ^1V_7 = {}^2V_7\}$	$= 2 + 1 = 3$	1						3										
15	÷	$^2V_6 \div {}^2V_7$	$^1V_8$	$\{^2V_6 = {}^2V_6;\ ^2V_7 = {}^2V_7\}$	$= \frac{2n-1}{3}$						$2n-1$	3	$\frac{2n-1}{3}$									
16	×	$^1V_8 \times {}^3V_{11}$	$^4V_{11}$	$\{^1V_8 = {}^0V_8;\ ^3V_{11} = {}^4V_{11}\}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3}$								0			$\frac{2n}{2} \cdot \frac{2n-1}{3}$						
17	−	$^2V_6 - {}^1V_1$	$^3V_6$	$\{^2V_6 = {}^3V_6;\ ^1V_1 = {}^1V_1\}$	$= 2n - 2$	1					$2n-2$											
18	+	$^1V_1 + {}^2V_7$	$^3V_7$	$\{^2V_7 = {}^3V_7;\ ^1V_1 = {}^1V_1\}$	$= 3 + 1 = 4$	1						4										
19	÷	$^3V_6 \div {}^3V_7$	$^1V_9$	$\{^3V_6 = {}^3V_6;\ ^3V_7 = {}^3V_7\}$	$= \frac{2n-2}{4}$						$2n-2$	4		$\frac{2n-2}{4}$								
20	×	$^1V_9 \times {}^4V_{11}$	$^5V_{11}$	$\{^1V_9 = {}^0V_9;\ ^4V_{11} = {}^5V_{11}\}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{4} = A_3$									0		$\left\{\frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3} = A_3\right\}$						
21	×	$^1V_{22} \times {}^5V_{11}$	$^0V_{12}$	$\{^1V_{22} = {}^1V_{22};\ ^0V_{12} = {}^2V_{12}\}$	$= B_3 \cdot \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3} = B_3 A_3$											0	$B_3 A_3$			B_3		
22	+	$^2V_{12} + {}^2V_{13}$	$^3V_{13}$	$\{^2V_{12} = {}^0V_{12};\ ^2V_{13} = {}^3V_{13}\}$	$= A_0 + B_1 A_1 + B_3 A_3$												0	$\{A_0 + B_1 A_1 + B_3 A_3\}$				
23	−	$^2V_{10} - {}^1V_1$	$^3V_{10}$	$\{^2V_{10} = {}^3V_{10};\ ^1V_1 = {}^1V_1\}$	$= n - 3\ (= 1)$	1									$n-3$							
					Here follows a repetition of Operations thirteen to twenty-three.																	
24	+	$^4V_{13} + {}^0V_{24}$	$^1V_{24}$	$\{^4V_{13} = {}^0V_{13};\ ^0V_{24} = {}^1V_{24}\}$	$= B_7$																	B_7
25	+	$^1V_1 + {}^1V_3$	$^1V_3$	$\{^1V_1 = {}^1V_1;\ ^1V_3 = {}^1V_3;\ ^5V_6 = {}^0V_6;\ ^5V_7 = {}^0V_7\}$	$= n + 1 = 4 + 1 = 5$; by a Variable-card.; by a Variable card.	1		$n+1$			0	0										

History of Algorithms

Computer Science algorithms were formalized by Turing and Church in 1936.

Some famous algorithms from 20th Century

THE TOP 10 LIST

- 1946: The Metropolis Algorithm
- 1947: Simplex Method
- 1950: Krylov Subspace Method
- 1951: The Decompositional Approach to Matrix Computations
- 1957: The Fortran Optimizing Compiler
- 1959: QR Algorithm
- 1962: Quicksort
- 1965: Fast Fourier Transform
- 1977: Integer Relation Detection
- 1987: Fast Multipole Method

Dantzig von Neumann Hestenes Householder Backus Hoare Greengard

What definition will we use?

A step-by-step procedure for solving a problem using a computer program

Procedure != programming language, computer, coding style
Procedure = method to solve the problem

"An algorithm is a sequence of finite computational steps that transforms an input into an output" Cormen and Leiserson, 2009

To get a computer to help us we need to specify an unambiguous step-by-step procedure or an **algorithm**

Algorithm structure

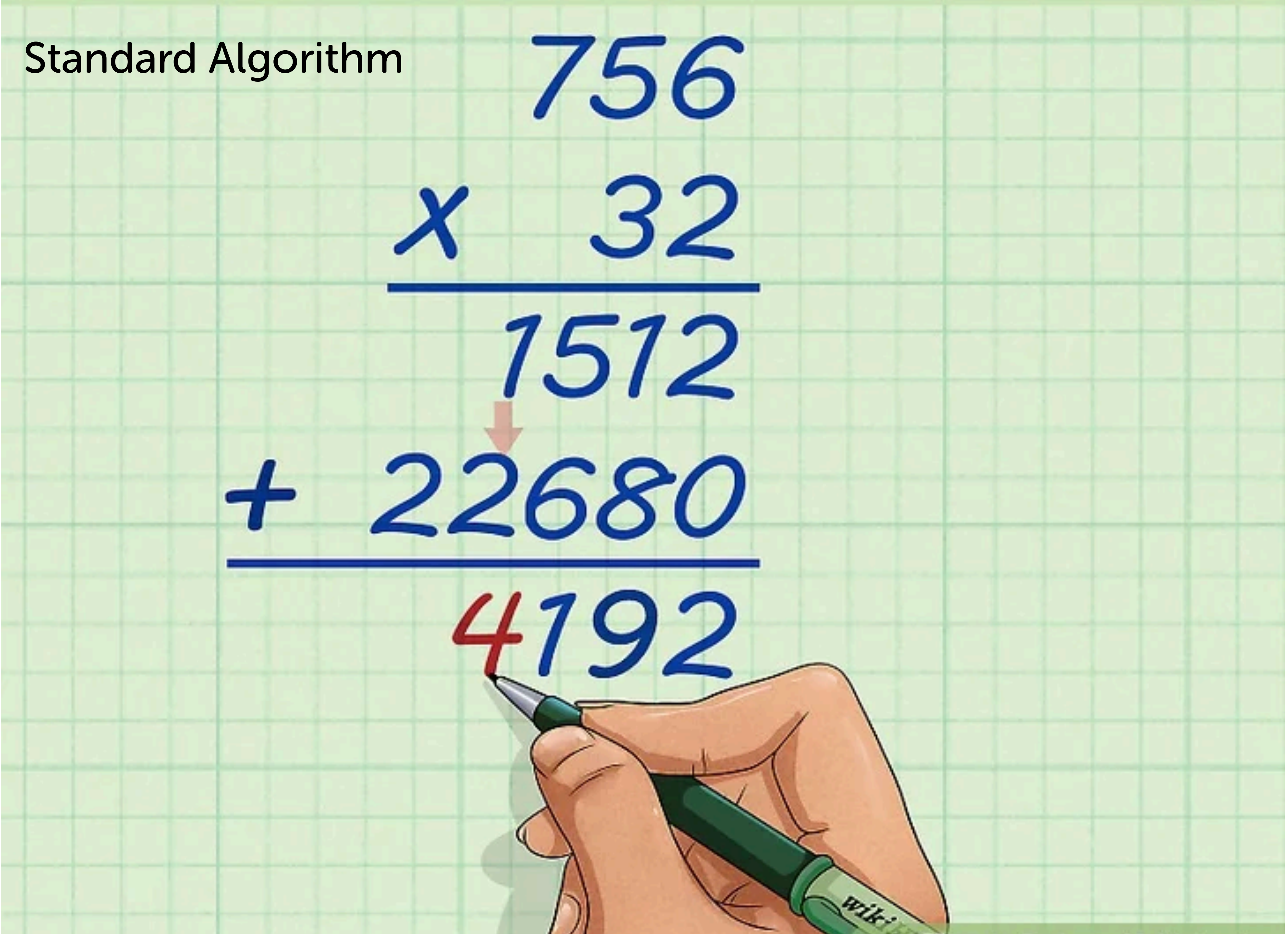
Algorithm

Input $\Rightarrow$ f(n) $\Rightarrow$ Output

Algorithm structure

Input		Algorithm		Output
24 x 16	→	f(n)	→	

Standard Algorithm



The Russian Peasant's Algorithm

This is a real method that people in Russia used to multiply numbers. Let's look at an example:

Input

24 x 16

Algorithm

f(n)

Output

half	double
24	16
12	32
6	64
3	128
1 (throw away the remainder)	256
	384

Cross out all even number rows and add the remaining double column

Why measure algorithmic performance?

Algorithm: a technique to solve a problem

16 x 24

Standard Algorithm

Russian Peasant's Algorithm
https://en.wikipedia.org/wiki/Ancient_Egyptian_multiplication

Karatsuba algorithm
https://en.wikipedia.org/wiki/Karatsuba_algorithm

We will see that while a given problem will often have multiple solutions there is usually one technique that solves the problem much faster

Why analyse algorithms?

- Predict performance of your program
- Compare algorithms
- Provide guarantees of performance
- Identify new efficient algorithms
- In practice? To avoid performance bugs

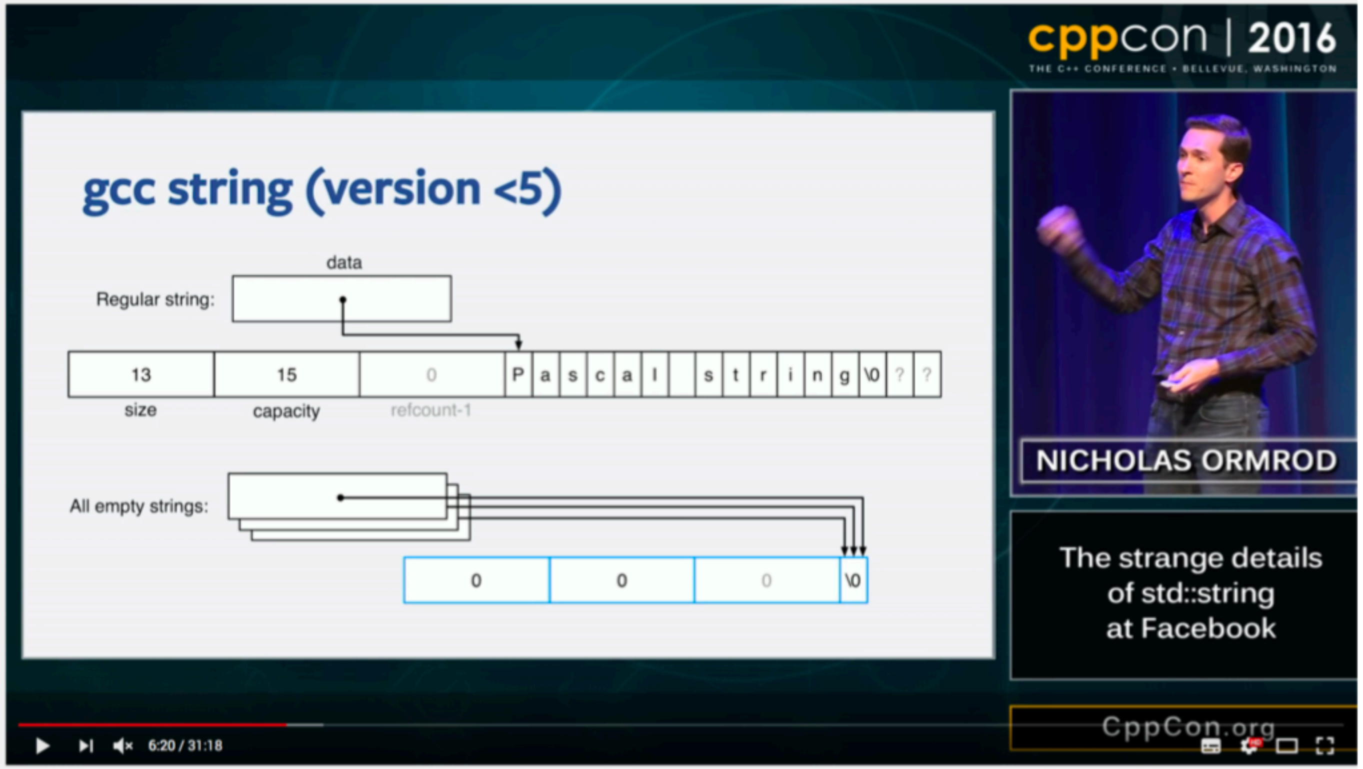
Why bother developing efficient algorithms?

- For nontrivial problems to be solved (i.e. implemented) by computers, the efficiency of the solution is a crucial property
- One cannot simply develop a program (to solve a problem) disregarding its efficiency, and then hope the power of the hardware will do the rest on embedded systems, dedicated processors are not all that powerful, either
- Even trivial problems must be implemented carefully
- If a simple operation is executed 1e9 times, then a reduction in the execution time by half can result in enormous improvements.
- We need a way to "measure" the efficiency of a program, one that allows us to draw conclusions such as "program A is more efficient that program B" (provided programs A and B actually "do the same thing")

Algorithmic efficiency

Facebook C++ string optimisation

saving even 1% in production is a massive benefit at Facebook scale



CppCon 2016: Nicholas Ormrod "The strange details of std::string at Facebook"

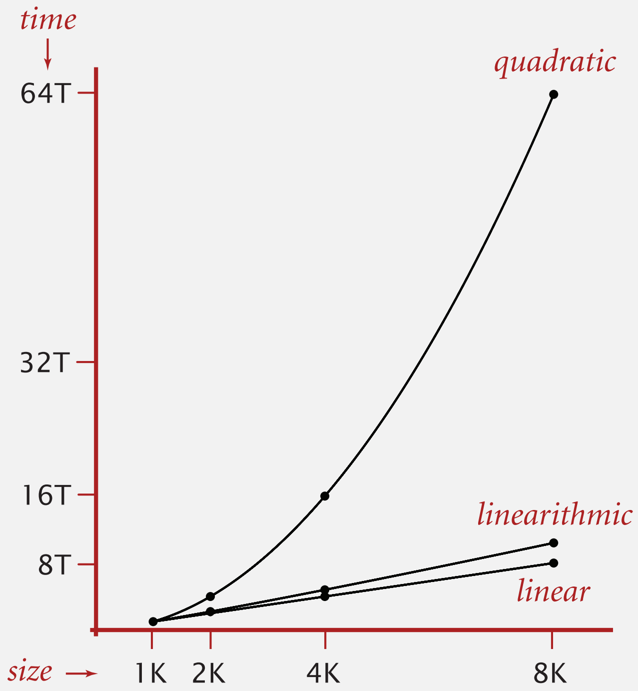
https://www.youtube.com/watch?v=kPR8h4-qZdk

Algorithmic efficiency

Discrete Fourier transform.

- Break down waveform of N samples into periodic components.
- Applications: DVD, JPEG, MRI, astrophysics, ...
- Brute force: N^2 steps.
- FFT algorithm: N log N steps, enables new technology.

Friedrich Gauss
1805



Runtime analysis

How do we know if our algorithm is good?

Two ways to assess:

1. Time
2. Space

Scientific method applied to algorithm analysis

A framework for predicting performance and comparing algorithms.

Scientific method

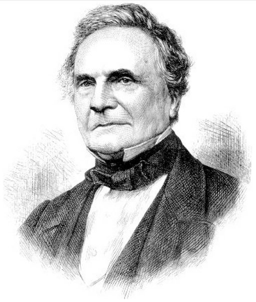
- Observe some feature of the natural world.
- Hypothesize a model that is consistent with the observations.
- Predict events using the hypothesis.
- Verify the predictions by making further observations.
- Validate by repeating until the hypothesis and observations agree.

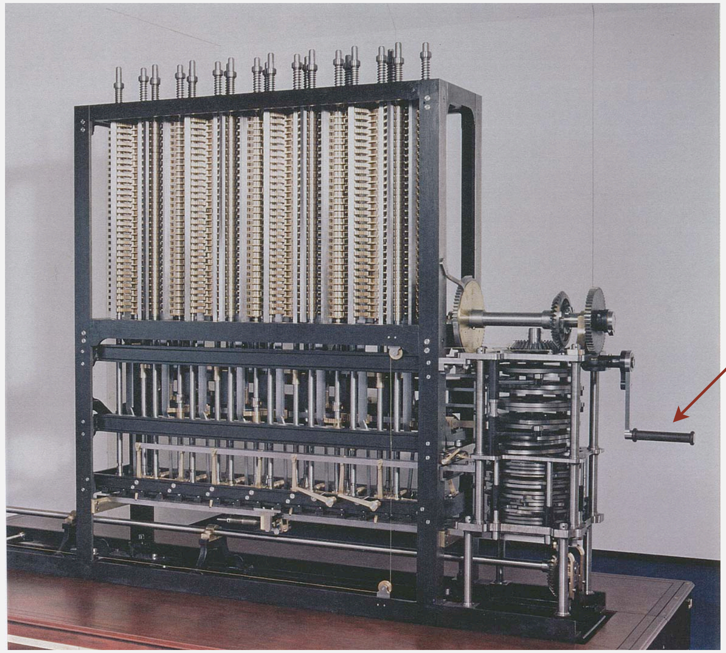
Principles

- Experiments must be reproducible.
- Hypotheses must be falsifiable.

Running time

" As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise—By what course of calculation can these results be arrived at by the machine in the shortest time? " — *Charles Babbage (1864)*





how many times do you have to turn the crank?

Measuring the run time

Q. How to time a program?
A. Manual.

Measuring the run time in Java

Q. How to time a program?
A. Automatic

Import Java.lang.System.currentTimeMillis()
This method returns the difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC(coordinated universal time)
Final long elapsedTime = System.currentTimeMillis()

```
public static void main(String[] args)
{
final long startTime = System.currentTimeMillis();
myFunction(N):
final long elapsedTime = System.currentTimeMillis() - startTime;
System.out.println(“the time taken ” + elapsedTime);
}
```

Empirical Analysis

Run the program for various input sizes and measure running time.

Empirical Analysis

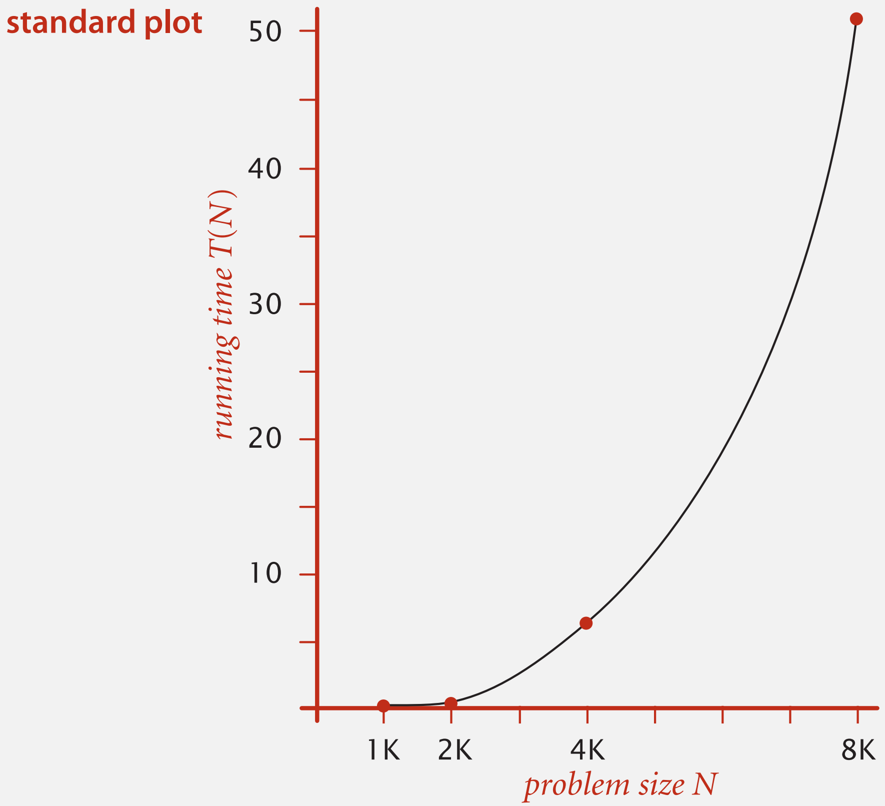
Run the program for various input sizes and measure running time.

N	time (seconds) †
250	0
500	0
1,000	0.1
2,000	0.8
4,000	6.4
8,000	51.1
16,000	?

Empirical Analysis

Standard plot. Plot running time $T(N)$ vs. input size N.

N	time (seconds) †
250	0
500	0
1,000	0.1
2,000	0.8
4,000	6.4
8,000	51.1
16,000	?



Prediction & Validation

Hypothesis. The running time is about $1.006 \times 10^{-10} \times N^{2.999}$ seconds.

Predictions.

- 51.0 seconds for N = 8,000.
- 408.1 seconds for N = 16,000.

Observations.

N	time (seconds) †
8,000	51.1
8,000	51
8,000	51.1
16,000	410.8

validates hypothesis!

Limitations of run-time analysis

Experimental approaches to evaluating algorithms have several limitations:

- It is necessary to implement the algorithm, which may be difficult
- Results may not be indicative of the running time on other inputs not included in the experiment
- In order to compare two algorithms, the same hardware and software environments must be used
- No guarantee that the algorithm will work in the same way when“live”.

We need guarantees of what will happen in the worse case.

Time complexity analysis

Pseudocode aside

Algorithms will be expressed in Java-style pseudocode

Our analysis will attempt to abstract away the implementation related issues (such as the hardware on which the program runs)

- High-level description of an algorithm
- More structured than English prose
- Less detailed than a program
- Preferred notations for describing algorithms
- Hides program design issues

A simple (non executable) language that allows more abstract descriptions of algorithms.

- It is not intended to be as rigorous as a programming language.
- Syntax not precisely defined.
- Combines programming style concepts with mathematical notation and natural language.
- Pseudo code should be clear enough to provide an easy mapping to a programming language.

Demo: Best case / worst case

Search (array, x)

Case 1: x is the first element in the array

How many checks? 1

Case 2: x is not in the array

How many checks? N where N is the number of elements in the array

Running time

Most algorithms transform input objects into output objects.

- The running time of an algorithm typically grows with the input size.
- Average case time is often difficult to determine.
- We often focus on the worst case running time.

Time complexity analysis

Time complexity is commonly estimated by counting the number of elementary operations performed by the algorithm, where an elementary operation takes a fixed amount of time to perform.

The running time of an algorithm on a particular input is the number of primitive operations or “steps” executed. It is convenient to define the notion of step so that it is as machine-independent as possible.

Measuring running time

When analysing the performance of an algorithm we can assign a fixed and equivalent amount of time to the following operations:

- Assigning a value to a variable
- Calling a method
- Performing an arithmetic operation
- Comparing two numbers
- Indexing into an array
- Following an object reference
- Returning from a method

Operation Counting Examples

1. var = 3;
2. A[i] = 4;

Demo: Worked Example

```
sum_up(N)
{
sum = 0;                      →  1
for (i = 1; i <= N; i++) {    →  1 + N+1 + N
sum = sum + i;                →  2N
}
return sum;                   →  1
}
```

The condition in the for loop checks N+1 times counting the last time when it is no longer <= to n and exits

T(N) = 4N + 4

T(N) = time taken by the sum_up algorithm in the worst case

Next Time: Theoretical Analysis

- Uses a high-level description of the algorithm instead of an implementation
- Characterise running time as a function of the input size, n.
- Take into account all possible inputs
- Allows us to evaluate the speed of an algorithm independent of the hardware/ software environment