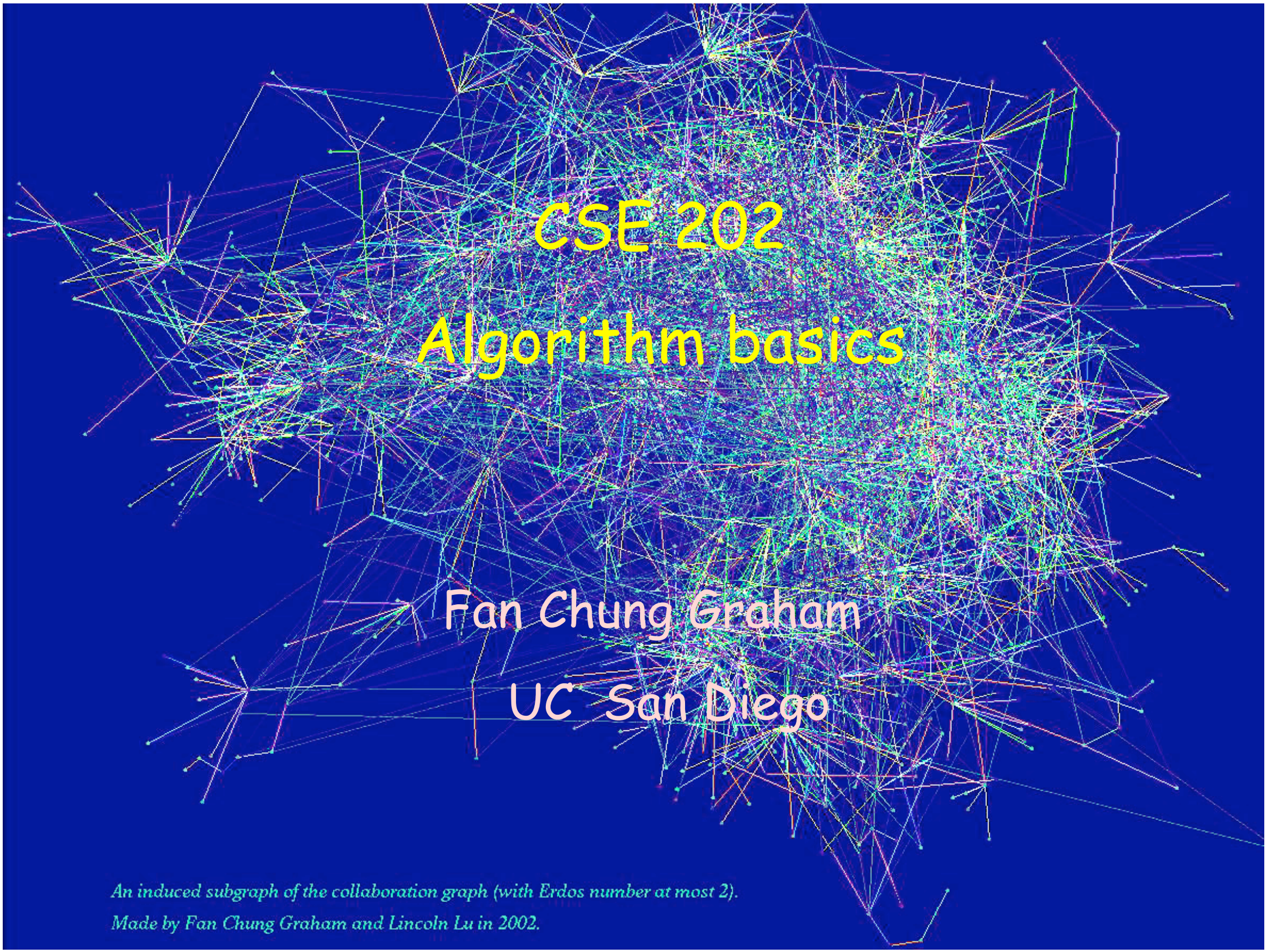
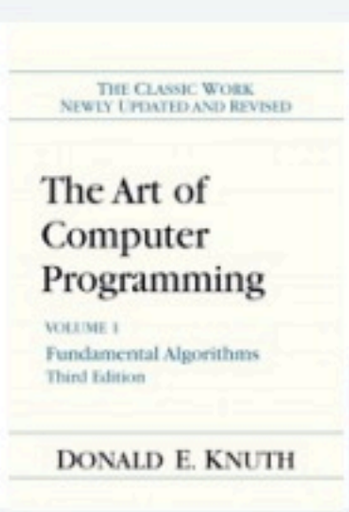

CSE 202
Algorithm basics
Fan Chung Graham
UC San Diego
An induced subgraph of the collaboration graph (with Erdos number at most 2).
Made by Fan Chung Graham and Lincoln Lu in 2002.

What is an algorithm?

" A procedure for solving a mathematical problem (as of finding the greatest common divisor) in a finite number of steps that frequently involves repetition of an operation. " — *webster.com*

" An algorithm is a finite, definite, effective procedure, with some input and some output. "

— *Donald Knuth*



Algorithm

An algorithm is a specific set of instructions for carrying out a procedure or solving a problem, usually with the requirement that the procedure terminate at some point. Specific algorithms sometimes also go by the name method, procedure, or technique. The word "algorithm" is a distortion of al-Khwārizmī, a Persian mathematician who wrote an influential treatise about algebraic methods. The process of applying an algorithm to an input to obtain an output is called a computation.

SEE ALSO: 196-Algorithm, Archimedes Algorithm, Brelaz's Heuristic Algorithm, Buchberger's Algorithm, Bulirsch-Stoer Algorithm, Bumping

Algorithm etymology

Etymology. [Knuth, TAOCP]

- *Algorism* = process of doing arithmetic using Arabic numerals.
- A misperception: *algiros* [painful] + *arithmos* [number].
- True origin: Abu 'Abd Allah Muhammad ibn Musa al-Khwarizm was a famous 9th century Persian textbook author who wrote *Kitāb al-jabr wa'l-muqābala*, which evolved into today's high school algebra text.

Abū ʿAbdallāh Muḥammad ibn Mūsā al-Khwārizmī (c. 780 – c. 850)

was a Persian mathematician, astronomer and geographer, a scholar in the House of Wisdom in Baghdad.

His *Kitab al-Jabr wa-l-Muqabala* presented the first systematic solution of linear and quadratic equations. He is considered the founder of algebra, a credit he shares with Diophantus. In the twelfth century, Latin translations of his work on the Indian numerals, introduced the decimal positional number system to the Western world. He revised Ptolemy's *Geography* and wrote on astronomy and astrology.

His contributions had a great impact on language. "**Algebra**" is derived from *al-jabr*, one of the two operations he used to solve quadratic equations. *Algorism* and *algorithm* stem from **Algoritmi**, the Latin form of his name.

From WIKI

Why study algorithms?

Internet. Web search, packet routing, distributed file sharing, ...

Biology. Human genome project, protein folding, ...

Computers. Circuit layout, databases, caching, networking, compilers, ...

Computer graphics. Movies, video games, virtual reality, ...

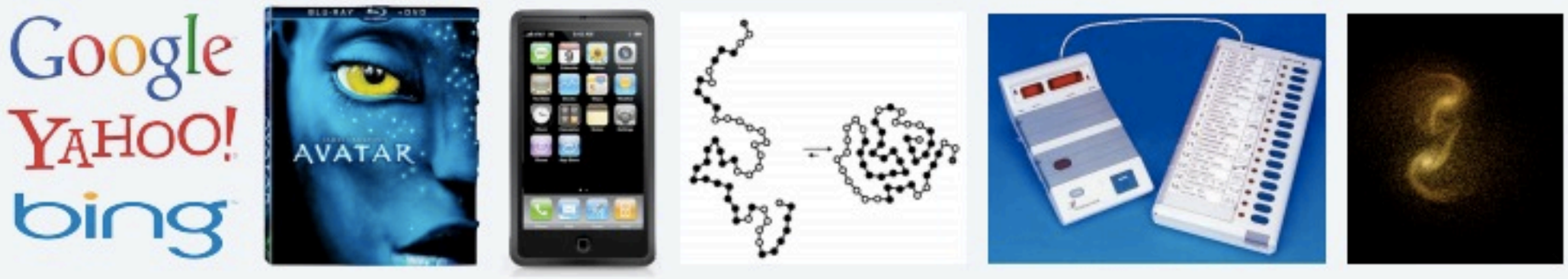
Security. Cell phones, e-commerce, voting machines, ...

Multimedia. MP3, JPG, DivX, HDTV, face recognition, ...

Social networks. Recommendations, news feeds, advertisements, ...

Physics. N-body simulation, particle collision simulation, ...

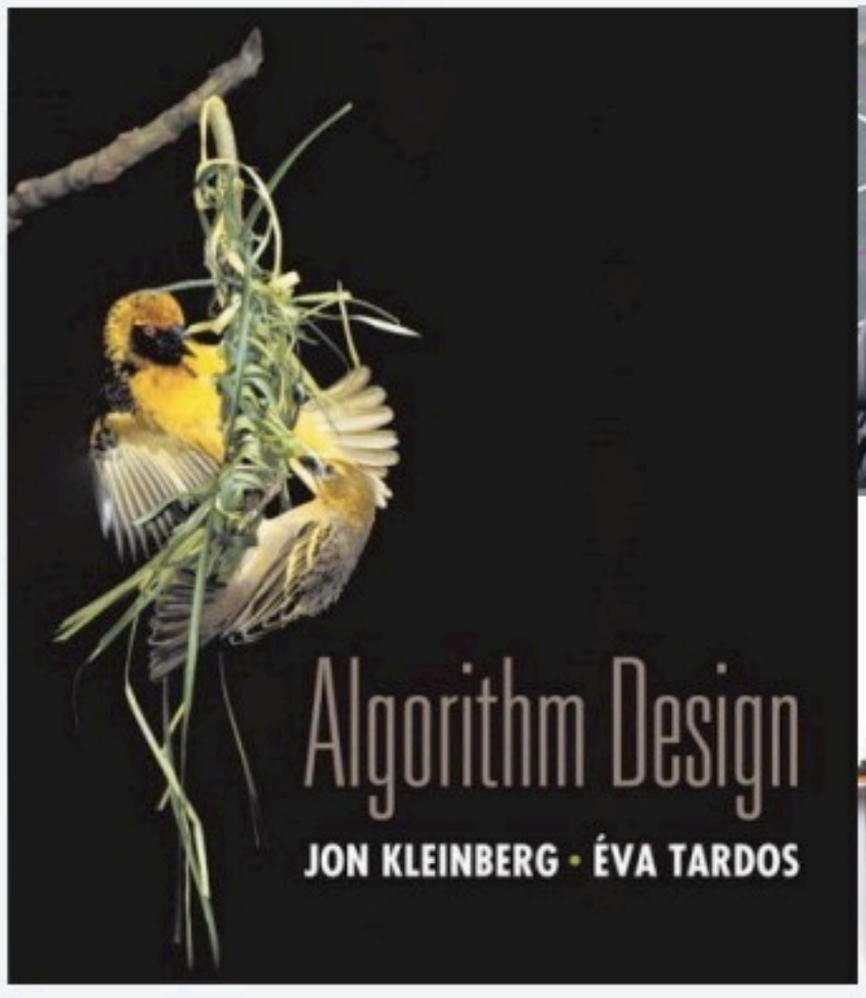
⋮



We emphasize algorithms and techniques that are useful in practice.

Textbook

Required reading. *Algorithm Design* by Jon Kleinberg and Éva Tardos. Addison-Wesley 2005, ISBN 978-0321295354.

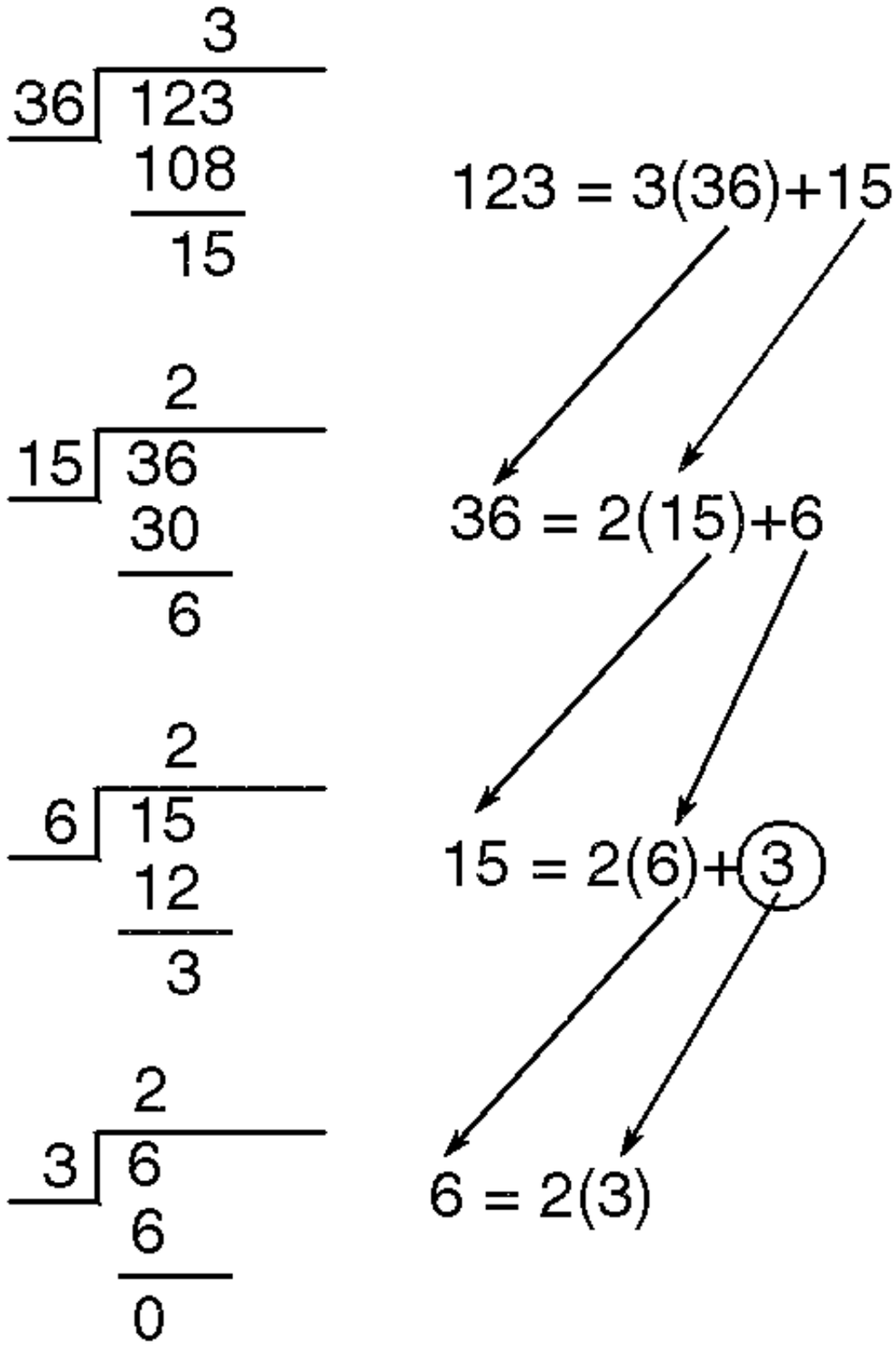


Euclidean algorithm:

Find the largest common factor between 36 and 123.

Euclidean algorithm:

Find the largest common factor between 36 and 123.

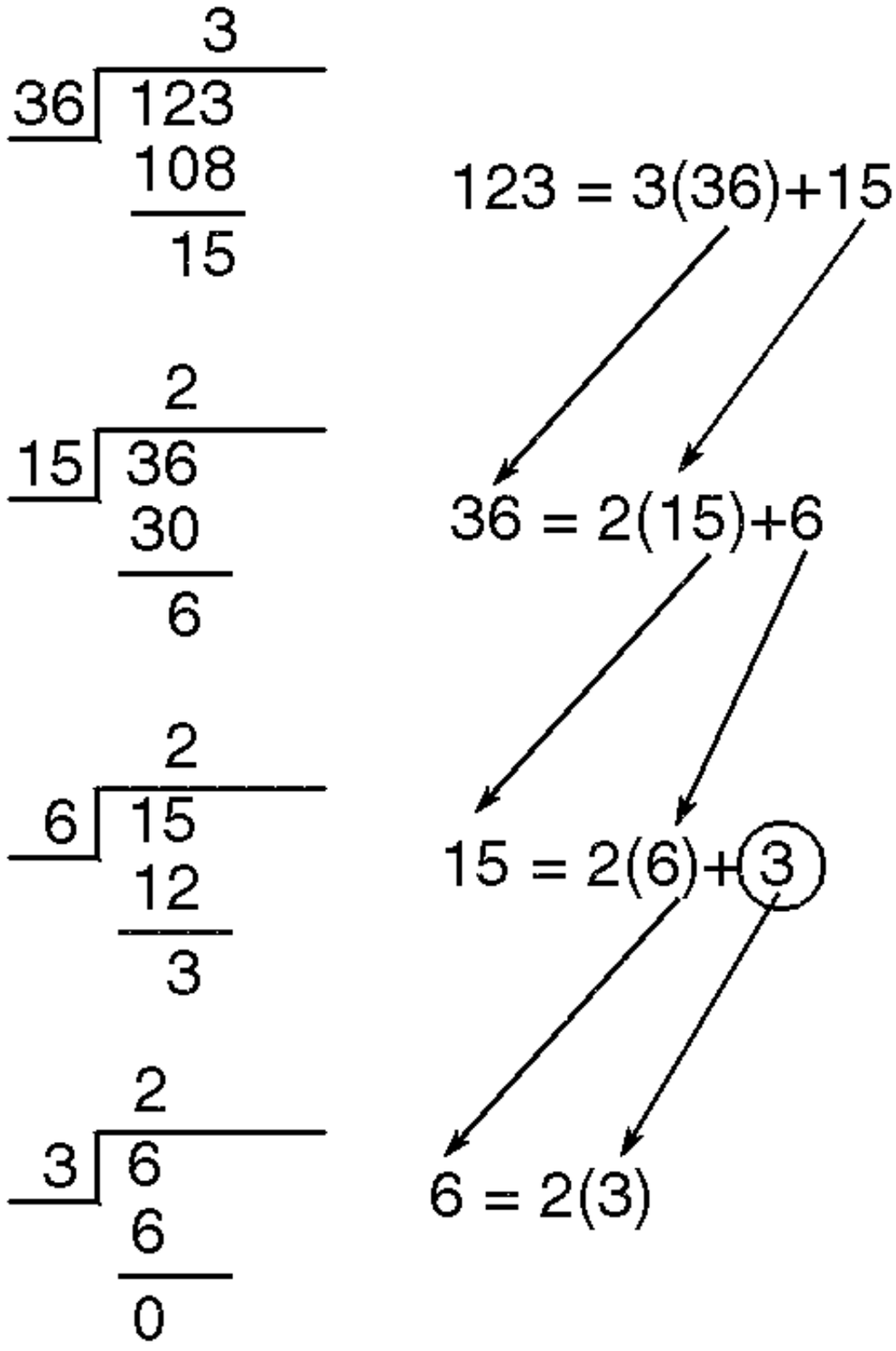


Euclidean of Alxandria (~325 BC)

Euclid of Alexandria is the most prominent mathematician of antiquity best known for his treatise on mathematics *The Elements*. The long lasting nature of *The Elements* must make Euclid the leading mathematics teacher of all time.

Euclidean algorithm:

Find the largest common factor between 36 and 123.



Euclidean algorithm:

Find the largest common factor between a and b.

Algorithm #1:

```
function gcd(a, b)
    while b ≠ 0
        t := b
        b := a mod b
        a := t
    return a
```

Algorithm #2:

```
function gcd(a, b)
    while a ≠ b
        if a > b
            a := a − b
        else
            b := b − a
    return a
```

Algorithm analysis:

- ☑ Termination?
- ☑ Correctness?
- ☑ Efficiency?

Emphasizes critical thinking, problem-solving

Algorithm Analysis

- Worst case running time.
- Average case running time.

Algorithm Analysis

- Worst case running time. Obtain bound on largest possible running time of algorithm on input of a given size N.
  - Generally captures efficiency in practice.
  - Draconian view, but hard to find effective alternative.

- Average case running time. Obtain bound on running time of algorithm on random input as a function of input size N.
  - Hard (or impossible) to accurately model real instances by random distributions.
  - Algorithm tuned for a certain distribution may perform poorly on other inputs.

Brute-Force Search

Brute force. For many non-trivial problems, there is a natural brute force search algorithm that checks every possible solution.

- Typically takes 2^N time or worse for inputs of size N.
- Unacceptable in practice.

n! for stable matching with n men and n women

- Not only too slow to be useful, it is an intellectual cop-out.
- Provides us with absolutely no insight into the structure of the problem.

> Proposed definition of efficiency. An algorithm is efficient if it achieves qualitatively better worst-case performance than brute-force search.

Polynomial-Time

Desirable scaling property. When the input size doubles, the algorithm should only slow down by some constant factor *C*.

> There exists constants c > 0 and d > 0 such that on every input of size N, its running time is bounded by cN^d steps.

A step. a single assembly-language instruction, one line of a programming language like C...

What happens if the input size increases from N to 2N?

Def. An algorithm is poly-time if the above scaling property holds.

Worst-Case Polynomial-Time

Def. An algorithm is efficient if its running time is polynomial.

Justification: It really works in practice!

- Although $6.02 \times 10^{23} \times N^{20}$ is technically poly-time, it would be useless in practice.
- In practice, the poly-time algorithms that people develop almost always have low constants and low exponents.
- Breaking through the exponential barrier of brute force typically exposes some crucial structure of the problem.

Exceptions.

- Some poly-time algorithms do have high constants and/or exponents, and are useless in practice.
- Some exponential-time (or worse) algorithms are widely used

Asymptotic Order of Growth

- We try to express that an algorithm's worst case running time is at most proportional to some function f(n).
- Function f(n) becomes a bound on the running time of the algorithm.

- Pseudo-code style.
  - counting the number of pseudo-code steps.
  - step. Assigning a value to a variable, looking up an entry in an array, following a pointer, a basic arithmetic operation…

"On any input size n, the algorithm runs for at most $1.62n^2 + 3.5n + 8$ steps."

Do we need such precise bound?

size / complexity	n = 10	20	30	40	50	100	1000
100 n	1 µsec	2 µsec	3 µsec	4 µsec	5 µsec	10 µsec	100 µsec
n^2	.1 µsec	.4 µsec	.9 µsec	1.6 µsec	2.5 µsec	10 µsec	1 msec
n^3	1 µsec	8 µsec	27 µsec	64 µsec	.13 msec	1 msec	1 sec
2^n	1 µsec	1 msec	1.1 sec	18.3 min	2.1 yr	2.4×10^{13} cent	----
n!	3.6 msec	1.8 yr	2.0×10^{15} cent	!	forget it	?	

Polynomial vs. **exponential** growth

(assuming 1,000,000,000 operations per second)

size / complexity	n = 10	20	30	40	50	100	1000
100 n	1 µsec	2 µsec	3 µsec	4 µsec	5 µsec	10 µsec	100 µsec
n^2	.1 µsec	.4 µsec	.9 µsec	1.6 µsec	2.5 µsec	10 µsec	1 msec
n^3	1 µsec	8 µsec	27 µsec	64 µsec	.13 msec	1 msec	1 sec
2^n	1 µsec	1 msec	1.1 sec	18.3 min	2.1 yr	2.4×10^{13} cent	----
n!	3.6 msec	1.8 yr	2.0×10^{15} cent	!	forget it	?	∞

Polynomial vs. **exponential** growth

(assuming 1,000,000,000 operations per second)