UNIVERSITÀ DEGLI STUDI DI PADOVA

Design and Analysis of Algorithms

Gianmaria Silvello

Department of Information Engineering
University of Padua
gianmaria.silvello@unipd.it
http://www.dei.unipd.it/~silvello/

So… What’s an Algorithm?

Algorithms

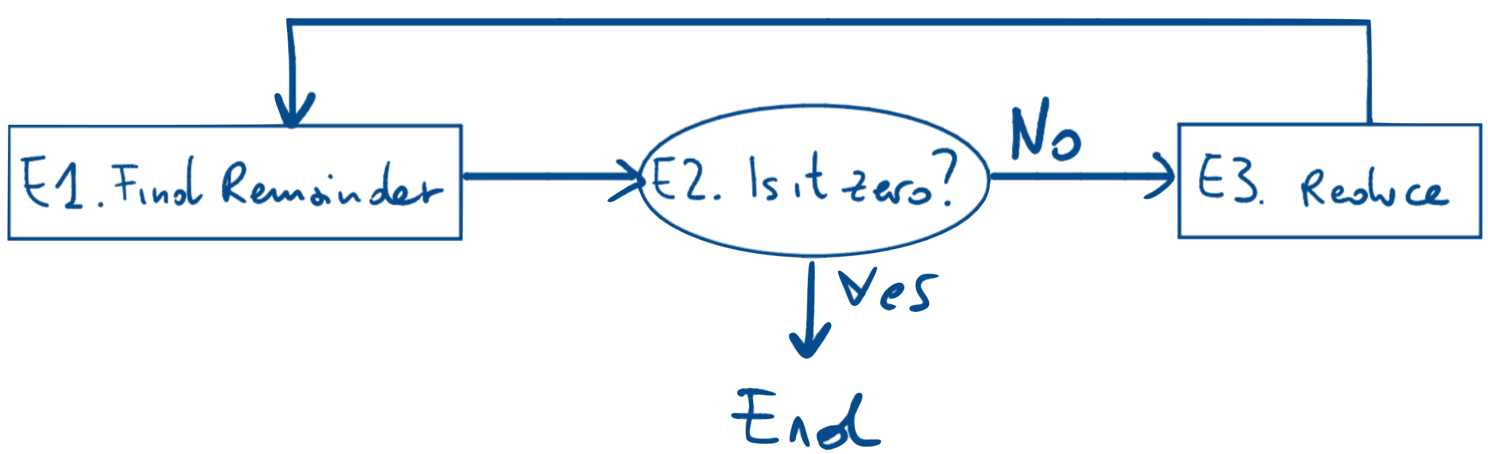
- An algorithm is a sequence of instructions to describe the solution of a problem

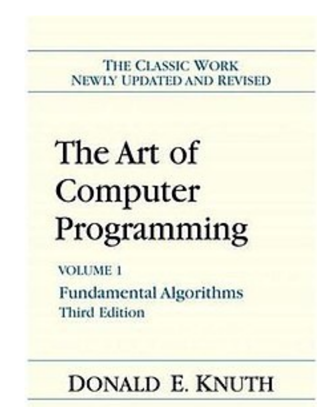
- An algorithm is a well-defined computational procedure that takes some value(s) as ***input*** and produces some value(s) as ***output***

- Algorithms give you a language to talk about problems and solutions
  - It’s a good way to talk about what we do

Euclid's algorithm

- Given two positive numbers n and m find their *greatest common divisor*.
  - E1. [Find remainder] Divide *m* by *n* and let r be the remainder ($0 \leq r < n$)
  - E2. [Is it zero?] If $r = 0$, the algorithm ends; *n* is the answer.
  - E3. [Reduce] Set $m \leftarrow n$, $n \leftarrow r$, and go back to step E1.





[From The art of Computer Programming, D. Knuth, Vol. 1]

Knuth's five rules

- An algorithm has to respect 5 rules:
  1. *Finiteness*: It must always terminate after a finite number of steps.
  2. *Definiteness*: Each step must be precisely defined.
  3. *Input*: It has zero or more inputs.
  4. *Output*: It has one or more outputs.
  5. *Effectiveness*: Its operations must all be sufficiently basic that they can in principle be done exactly in a finite amount of time by someone using pencil and paper.

- An algorithm without finiteness is a *computational method*

Algorithms

- Sorting problem
  - Input: a sequence of *n* numbers $<a_1, a_2, \ldots, a_n>$
  - Output: a permutation $<a'_1, a'_2, \ldots, a'_n>$ of the input such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$
- An algorithm is the sequence of operations to obtain the sorted list of numbers starting from the unordered one
- We can define the algorithm by using *pseudo-code* or any available programming language (e.g. Java, Python)

Algorithms

- For example, given the input sequence
<34, 2, 1, 45, 565>
a sorting algorithm returns the output sequence
<1, 2, 34, 45, 565>
- <34, 2, 1, 45, 565>
is called an *instance* of the sorting problem
- An algorithm is *correct* if it halts with the correct output, so we say that an algorithm is correct if it *solves* the given computational problem

Computational problem

- A computational problem $\mathbf{\Pi}$ is a mathematical relation between a set I of possible instances and a set S of possible solutions: $\mathbf{\Pi} \subseteq I \times S$ such that $\forall\ i \in I$ there exists ($\exists$) at least one solution $s \in S$ such that $(i,s) \in \mathbf{\Pi}$

- An algorithm A solves the problem $\mathbf{\Pi} \subseteq I \times S$ if $\forall i \in I$, $A(i) \rightarrow s$

The study of performance

Our main concern is performance

- But, what's more important than performance?
    - Correctness
    - Cost
    - Maintainability
    - Stability and robustness
    - Having a wide range of features
    - Modularity
    - Security
    - User-friendliness

So… why performance?

- Often performances define the line between feasible and unfeasible
  - Sometimes if it’s not real-time then it’s not useful
  - If it requires too much time then it is not usable

- Performance is a measure of value

- Speed is fun!

Algorithmic Analysis

- It is used to determine how *good* an algorithm is.
- The idea is to take an algorithm and to determine its *quantitative* behaviour
  - How many operations are the algorithm doing to solve a problem?
  - Given two algorithms A and B both solving the problem *p,* do we prefer A or B? Why?
- The core is to determine the performance characteristics of an algorithm

Efficiency

- Even though computers are getting faster and faster, we should still care about algorithmic efficiency and memory use
- Algorithmic efficiency is often measured in terms of the input size (*n* is the size of the input)
- *Insertion sort* requires $\mathbf{c_1 n^2}$ time to sort a sequence of *n* numbers
    - *Merge sort* requires $\mathbf{c_2\, n \lg(n)}$ time to sort a sequence of *n* numbers
        - (when we write lg(*n*) we mean $\log_2(n)$)

An example: Insertion sort vs merge sort

Input: sequence of 10^7 numbers

Insertion sort: $c_1 n^2$

Merge sort: $c_2\, n \lg(n)$

Fast computer: 10^{10} instr/sec

+

Good programmer: $c_1 = 2$

Slow computer: 10^7 instr/sec

+

bad programmer: $c_2 = 50$

$$\frac{2 \times (10^7)^2 \text{ instructions}}{10^{10} \text{ instructions/second}} = 20,000 \text{ secs} > 5.5h$$

$$\frac{50 \times (10^7) lg 10^7 \text{ instructions}}{10^7 \text{ instructions/second}} \approx 1163 \text{ secs} < 20 \text{ min}$$

Example from CLRS 3rd ed.