

Course «Algorithms» · Lecture 1

Introduction to Algorithms

Acting Professor Irina Maratovna Ualiyeva

Main Textbook: A. Levitin, Sections 1.1–1.4

Today’s goal

What this course covers, and what it means to specify, prove and compare algorithms.

- the plan of the master’s course **Algorithms**
- problems, instances and algorithms
- Euclid’s algorithm: correctness and efficiency
- problem solving, problem types, data structures

The plan of the course

Week	Topic
1	Introduction
2	Pseudocode, flowcharts
3	The analysis framework
4	Nonrecursive analysis
5	Recursive analysis
6	DP: main idea, examples
7	DP: knapsack, memory functions
8	DP: Warshall

Week	Topic
9	DP: Floyd
10	Greedy: Prim
11	Greedy: Dijkstra
12	Greedy: Huffman
13	P, NP, reductions
14	NP-complete problems
15	The future of algorithms

Tools of analysis, then design techniques, then the limits of efficiency.

Problems, instances, algorithms

Problem

A relation between inputs and outputs: for example, sorting.

Instance

One particular input: the sequence 5, 1, 4.

Algorithm

Unambiguous instructions that solve every legitimate instance in finite time.

- The set of legitimate inputs is part of the problem
- One failing legitimate instance makes the algorithm incorrect
- An algorithm isn’t a program: it has many implementations

Knuth’s five properties

Property	Meaning
Finiteness	It always terminates after a finite number of steps
Definiteness	Each step is precisely defined
Input	Zero or more inputs from a specified set
Output	One or more outputs related to the inputs
Effectiveness	Each step is basic enough to be carried out exactly

Three algorithms for gcd(m, n)

Consecutive integers

Try min(m, n), then one less, and so on. Fails for n = 0, so the input must be restricted.

Middle-school

Multiply common prime factors. Not an algorithm until “find the prime factors” is specified.

Euclid

Repeat (m, n) → (n, m mod n) until n = 0. Over two thousand years old.

The specification and the level of detail decide whether a procedure is an algorithm at all.

Euclid’s algorithm

```
ALGORITHM Euclid(m, n)
  //Input: nonnegative, not-both-zero integers m and n
  //Output: gcd(m, n)
  while n ≠ 0 do
     r ← m mod n
     m ← n
     n ← r
  return m
```

$$\gcd(60, 24) = \gcd(24, 12) = \gcd(12, 0) = 12$$

Why it is correct

$$m = qn + r, \qquad \gcd(m, n) = \gcd(n,\ m \bmod n)$$

- If d divides m and n, it divides r = m – qn
- If d divides n and r, it divides m = qn + r
- The pairs (m, n) and (n, r) have the same common divisors
- When n = 0, the answer is gcd(m, 0) = m

Termination: n is a nonnegative integer that strictly decreases.

Why it is fast

$$m \geq n > 0 \quad \Rightarrow \quad m \bmod n < \frac{m}{2}$$

- If n ≤ m/2, then m mod n < n ≤ m/2
- If n > m/2, then m mod n = m − n < m/2
- So every two steps halve the number: **O(log n)** iterations

Value versus length

Euclid: 10 iterations

for m = 31415, n = 14142

Checking integers: 14,142

iterations on the same input

- For b-bit numbers: **O(b)** iterations against about $\mathbf{2^b}$
- Linear in the value of n, exponential in its length
- We return to this in Lectures 7 and 13

Why the choice of algorithm matters

n	$n \log_2 n$	n^2	2^n
10	instant	instant	10^{-6} s
10^3	10^{-5} s	10^{-3} s	beyond the age of the universe
10^6	0.02 s	about 17 min	beyond the age of the universe

At 10^9 operations per second. A machine 1000 times faster:

- a 2^n algorithm handles inputs only about **10** larger
- an n^2 algorithm handles inputs about **32** times larger

Algorithmic problem solving

1. Understand the problem and its legitimate inputs
2. Choose the computational model (RAM)
3. Choose an exact or an approximate solution
4. Choose data structures and a design technique
5. Specify the algorithm in pseudocode
6. Prove correctness
7. Analyze time, space and optimality
8. Code and check against experiments

Main problem types

Type	Example	Lectures
Sorting, searching	Selection sort; sequential search	3–4
Combinatorial	Knapsack, traveling salesman	7, 13, 14
Graph	Closure, shortest paths, spanning trees	8–11
Coding, string	Huffman codes	12
Numerical	Matrix multiplication	4

Combinatorial problems are usually the hardest: candidates grow very fast with the input.

Data structures this course assumes

Structure	Efficient operations	Where it matters
Array	Access by index, $\Theta(1)$	DP tables
Stack, queue	Operations at the ends, $\Theta(1)$	Traversal, recursion
Binary heap	Priority queue, $\Theta(\log n)$	Prim, Dijkstra, Huffman
Adjacency matrix	Edge test $\Theta(1)$, memory $\Theta(n^2)$	Warshall, Floyd
Adjacency lists	Neighbor scan, memory $\Theta(n + m)$	Sparse graphs

The data structure decides the cost of the operations an algorithm relies on.

Pitfalls

- **Leaving the input unspecified:** consecutive checking fails for n = 0
- **Confusing value with size:** size is measured in bits
- **Hiding a computation in one step:** “find the prime factors”
- **Skipping termination:** a loop that never stops proves nothing

Summary

- A problem is a relation; an algorithm solves every legitimate instance
- gcd(m, n) = gcd(n, m mod n) proves Euclid’s algorithm correct
- Halving gives **O(log n)** iterations
- Input size is the length of the encoding, not the value
- Problem solving goes from specification to proof, analysis and code

Review questions

1. Define a computational problem, an instance and an algorithm. Why is a program not the same as an algorithm?
2. Prove that $\gcd(m, n) = \gcd(n, m \bmod n)$ for $n > 0$. Why does this alone not prove that Euclid's algorithm is correct?
3. Prove that $m \bmod n < m/2$ if $m \geq n > 0$, and derive a bound on the number of iterations.
4. Is a procedure that checks every integer from 2 to $\sqrt{n}$ polynomial in the size of the input n?

Reading: Levitin, Sections 1.1–1.4; Cormen et al., Chapter 1, Section 31.2.