

Course «Algorithms» · Lecture 2

Pseudocode and Flowcharts: Specifying Algorithms

Textbook: A. Levitin, Sections 1.1–1.2

Today’s goal

A specification precise enough to prove and analyze.

- four ways to specify an algorithm
- preconditions, postconditions and the RAM model
- traces and boundary cases
- **loop invariants** and variants
- flowcharts and the structured program theorem

Four ways to specify an algorithm

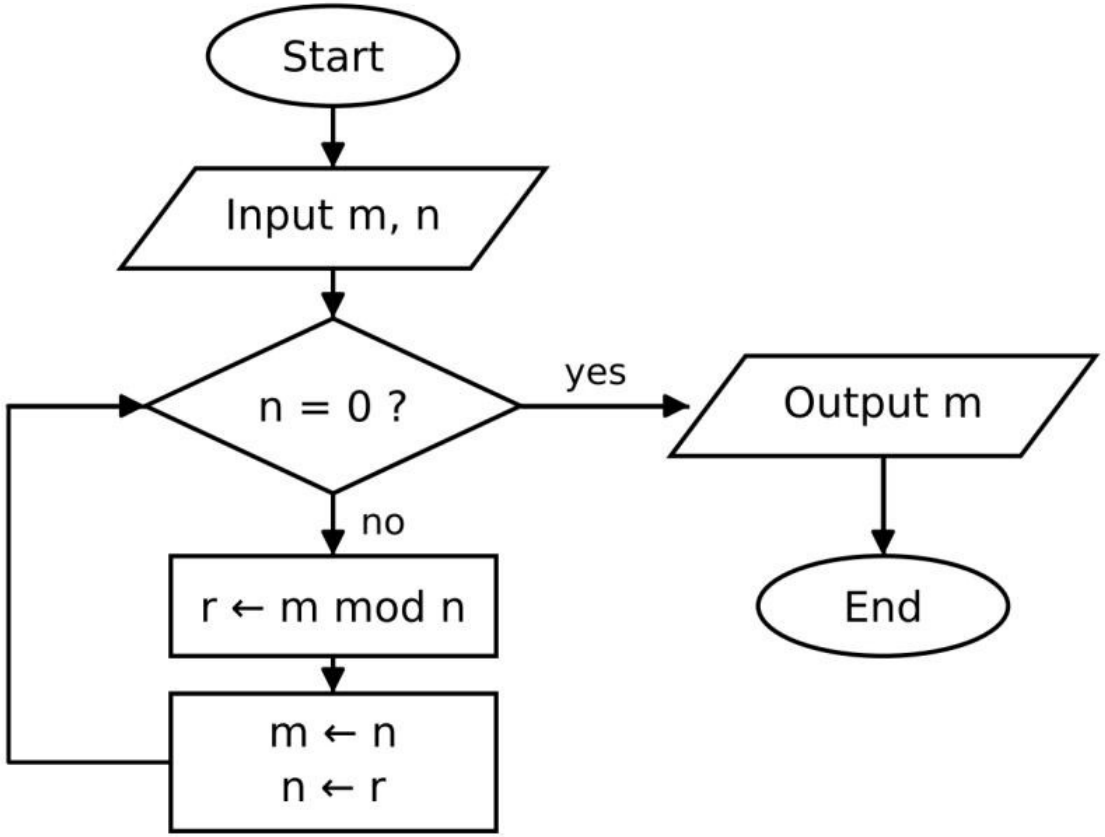
Form	Strength	Weakness
Natural language	Accessible	Ambiguous
Pseudocode	Precise, easy to count	No executable check
Flowchart	Shows control flow	Grows quickly, hard to edit
Program	Executable, testable	Includes irrelevant details

“Divide the bigger by the smaller until zero; the answer is the last one.” The last what?

Pseudocode: Euclid’s algorithm

```
ALGORITHM Euclid(m, n)
  //Computes gcd(m, n) by Euclid's algorithm
  //Input: Two nonnegative, not-both-zero integers m and n
  //Output: Greatest common divisor of m and n
  while n ≠ 0 do
      r ← m mod n
      m ← n
      n ← r
  return m
```

Flowchart: Euclid’s algorithm



Conventions of this course

Construct	Notation	Example
Assignment	←	maxval ← A[0]
Comparison	=, ≠, <, ≤, >, ≥	if n = 0
Array	A[0..n – 1]	Indices start at 0
Counting loop	for i ← a to b do	for i ← 1 to n – 1 do
Conditional loop	while condition do	while n ≠ 0 do
Logic	and, or, not	while i < n and A[i] ≠ K
Result	return value	return m

Preconditions and postconditions

Precondition

Must hold before execution: m, n ≥ 0 are integers, not both zero.

Postcondition

Must hold afterwards: the result equals gcd of the original m, n.

$$\{P\}\ S\ \{Q\}$$

- **Partial correctness:** if S terminates, Q holds
- **Total correctness:** S also terminates

What one line may cost: the RAM model

Instructions run one at a time; basic operations on values of bounded size take constant time.

Acceptable

swap A[i] and A[j]

Not acceptable

sort A, or find the maximum of A, as one step

One line means constant work, unless it calls a named algorithm. Numbers with thousands of digits aren’t O(1).

Tracing MaxElement: A = 3, 8, 2, 9, 5

Step	i	A[i]	A[i] > maxval?	maxval
start	–	–	–	3
1	1	8	yes	8
2	2	2	no	8
3	3	9	yes	9
4	4	5	no	9

$$C(n) = \sum_{i=1}^{n-1} 1 = n - 1$$

A trace is evidence, not a proof.

Boundary cases: sequential search

```
ALGORITHM SequentialSearch(A[0..n – 1], K)
  //Output: index of the first element equal to K, or –1
  i ← 0
  while i < n and A[i] ≠ K do
     i ← i + 1
  if i < n return i
  else return –1
```

- If K is absent, i reaches n, and A[n] doesn’t exist
- Swap the two conditions and only an unsuccessful search fails

Proving a loop: invariants

A **loop invariant** is true before every test of the loop condition.

Initialization

It holds before the first iteration.

Maintenance

If it holds and the condition is true, it holds after the body.

Termination

With the negated condition, it implies the postcondition.

To prove that the loop stops, use a variant: a nonnegative integer that strictly decreases.

Invariant for MaxElement

Before iteration i, maxval is the largest element of A[0..i – 1].

- **Initialization:** for i = 1, maxval = A[0], the largest of one element
- **Maintenance:** A[i] either becomes the new maximum or doesn’t; maxval is the largest of A[0..i]
- **Termination:** at i = n, maxval is the largest of the whole array

Variant: n – i.

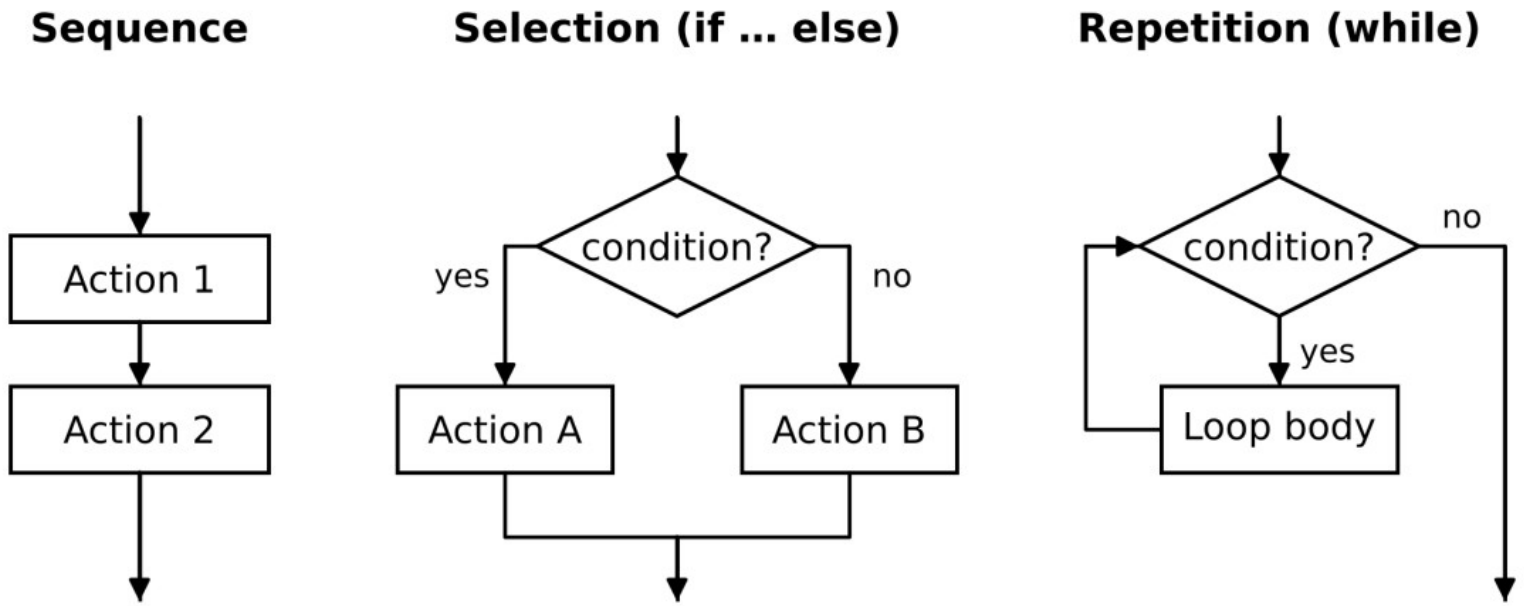
Invariant for Euclid's algorithm

$$\gcd(m, n) = \gcd(m_0, n_0)$$

- **Initialization** is immediate
- **Maintenance:** gcd(m, n) = gcd(n, m mod n), from Lecture 1
- **Termination:** n = 0, so m = gcd(m_0, n_0)

Variant: n, since 0 ≤ m mod n < n. The same logic supports proofs for DP and greedy algorithms.

Three building blocks



Böhm and Jacopini (1966): sequence, selection and repetition are sufficient.

Why structure helps analysis

Structure	In a proof or an analysis
Sequence	Costs add up
Selection	Both branches must be considered
Repetition	A loop becomes a sum, and needs an invariant

Flowcharts are inconvenient for all but very simple algorithms. From now on: pseudocode.

Pitfalls

- **Hiding work in a line:** “find the shortest path” isn’t one step
- **Omitting the specification:** correctness can’t even be stated
- **A weak invariant:** “maxval is an element of A” doesn’t imply the result
- **Ignoring termination:** an invariant gives only partial correctness
- **Treating a trace as a proof:** a trace checks one instance

Summary

- Pseudocode is the right level for proof and analysis
- Preconditions and postconditions define correctness
- In the RAM model, one line means constant work
- Traces find errors; invariants and variants prove correctness
- Sequence, selection and repetition are sufficient

Review questions

1. What does pseudocode deliberately leave out compared with a program? Why is this useful for analysis?
2. What is the difference between partial and total correctness?
3. State and prove a loop invariant for MaxElement. Which step uses the condition i = n?
4. Why does SequentialSearch fail if the two parts of its loop condition are swapped? Give an instance.

Reading: Levitin, Sections 1.1–1.2; Cormen et al., Sections 2.1–2.2.