Chapter 1: Introduction

1. The Notion of Algorithm

A sequence of unambiguous instructions for solving a problem, i.e. for obtaining the required output for any legitimate input in a finite amount of time

Important points:
- Nonambiguity
- Range of inputs
- The same algorithm can be represented in different ways
- Several algorithms for solving the same problem, can be based on different ideas, can have different running times

Examples: great common divisor, computing Fibonacci numbers

<table>
<thead>
<tr>
<th>Euclid(m,n)</th>
<th>Consecutive computation of gcd(m,n)</th>
</tr>
</thead>
</table>
| //Computes gcd(m,n) by Euclid’s alg.  
//Input: Two nonnegative  
// non 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 |
| //Computes gcd(m,n) by consecutive checks  
//Input: Two nonnegative  
// non both zero integers m and n  
//Output: Greatest common divisor of m and n |
| 1. t ← min(m,n)  
2. if m % t = 0 goto 3, else goto 4  
3. if n % t = 0 return t, else goto 4  
4. t ← t - 1  
5. goto 2 |

Questions:
What needs to be added to the input specifications in both algorithms?

What determines the choice of operations?

Estimate the number of operations in the above algorithms

Rewrite the algorithms as recursive ones
<table>
<thead>
<tr>
<th><strong>Fibonacci</strong>(n) recursive</th>
<th><strong>Fibonacci</strong>(n) with iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>//Computes the n-th Fibonacci number // by the formula F(n) = F(n-1) + F(n-2) //Input: positive integer n //Output: The n-th Fibonacci number <strong>procedure Fibonacci</strong>(n) if n = 1 or n = 2 return 1 else return Fibonacci(n-1) + Fibonacci(n-2)</td>
<td>//Computes the n-th Fibonacci number by // consecutive checks //Input: positive integer n //Output: The n-th Fibonacci number if n = 1 or n = 2 return 1 else n1 ← 1, n2 ← 1 for k = 3 to n do n3 ← n2 + n1 n1 ← n2 n2 ← n3 return n3</td>
</tr>
</tbody>
</table>

Question:

What is the complexity of the two algorithms?

2. **Algorithmic Problem Solving**
What does it mean to understand the problem?

What are the problem objects?

What are the operations applied to the objects?

Deciding on computational means

How the objects would be represented?

How the operations would be implemented?

Design an algorithm

Build a computational model of the solving process

Prove correctness: correct output for every legitimate input in finite time

Based on correct math formula
By induction

Tracing can prove incorrectness, it cannot prove correctness in the general case.

Analyze the algorithm

Efficiency: time and space

Simplicity

Generality (range of inputs, special cases)

Optimality
Optimal algorithm: no other algorithm can do better.

Coding

How the objects and operations in the algorithm are represented in the chosen programming language?

Can every problem be solved by an algorithm?
3. **Important problem types**

- Sorting
- Searching
- String processing
- Graph problems
- Combinatorial problems
- Geometric problems
- Numerical problems

The typology is made by operations and/or processed objects

4. **Fundamental data structures**

**Linear data structures:** lists – sequence of elements

- Array
- Linked list
- Stack
- Queue

  **Operations:** search, delete, insert
  **Implementation:** static (built-in array data type), dynamic (using reference data types)

**Non-linear data structures:**

- Graphs
- Trees: connected graph without cycles
  - Rooted trees
  - Ordered trees
  - Binary trees

  **Graph representation:** adjacency lists; adjacency matrix
  **Tree representation:** as graphs; binary nodes

**Sets, bags, dictionaries**

- **Set:** unordered collection of distinct elements
  - **Operations:** membership, union, intersection
  - **Representation:** bit string; linear structure

- **Bag:** unordered collection, elements may be repeated
- **Dictionary:** a bag with operations search, add, delete