# **Tutorial for Section 1.1**

# Introduction to design of algorithms

**Exercise 1**

Design an algorithm for computing  for any positive integer *n*. Besides assignment and comparison, your algorithm may only use the four basic arithmetic operations (+, -, ×, /).

**Hint**: Find the least integer m such that

**Exercise 2**

Consider the following algorithm for finding the distance between the two closest elements in an array of numbers.

**Algorithm** *MinDistance*(*A*[0*..n −* 1])

**Input**: Array *A*[0..*n −* 1] of numbers

**Output**: Minimum distance between two of its elements

*dmin ← ∞*

**for** *i ←* 0 **to** *n −* 1 **do**

**for** *j ← 0* **to** *n −* 1 **do**

**if** *i ≠*  *j* **and** *|A*[*i*] *− A*[*j*]*| < dmin*

*dmin ← |A*[*i*] *− A*[*j*]*|*

**return** *dmin*

Make as many improvements as you can in this algorithmic solution to the problem. If you need to, you may change the algorithm altogether; if not, improve the implementation given.

**Exercise 3**

Describe how one can implement each of the following operations on an array so that the time it takes does not depend on the array’s size *n*.

1. Delete the *ith* element of an array (1 ≤ i ≤ n).
2. Delete the *ith* element of a sorted array (the remaining array has to stay sorted, of course).

**Exercise 4**

Let *A* be the adjacency matrix of an undirected graph. Explain what property of the matrix indicates that:

* + The graph is complete.
  + The graph has a loop, i.e., an edge connecting a vertex to itself.
  + The graph has an isolated vertex, i.e., a vertex with no edges incident to it.

**Exercise 5**

*Anagram checking* Design an algorithm for checking whether two given words are anagrams, i.e., whether one word can be obtained by permuting the letters of the other. (For example, the words *tea* and *eat* are anagrams.)