# **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

**Algorithm** FloorofSquareRoot(n)

**Input**: A positive integer n

**Output**: The floor of square root of n

m 🡨 1

**while** m×m < n **do** //the loops stops when m2 ≥ n

m 🡨 m + 1

**if** m×m = n **return** m

**else return** m - 1

**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.

**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 −* 2 **do**

**for** *j ← i+1* **to** *n −* 1 **do**

**if** *|A*[*i*] *− A*[*j*]*| < dmin*

*dmin ← |A*[*i*] *− A*[*j*]*|*

**return** *dmin*

**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). Swap the ith element with the last element and decrement n by 1.
2. Delete the *ith* element of a sorted array (the remaining array has to stay sorted, of course). Replace the ith element by a sentinel element (null) to be avoided in the future processing of the array.

**Exercise 4**

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

* + The graph is complete. All the elements of A are 1 except the main diagonal is formed by 0s.
  + The graph has a loop, i.e., an edge connecting a vertex to itself. Some A[i[i] = 1.
  + The graph has an isolated vertex, i.e., a vertex with no edges incident to it. If the isolated vertex is labeled i (0 ≤ i ≤ n-1), then the row i and the column i of A are all made of 0s.

**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.)

**Algorithm** *Anagram* (w1[0*..n −* 1],w2[0..m-1])

**Input**: Two words w1 and w2.

**Output**: True if w1 and w2 are anagrams, false otherwise.

**if** n ≠ m

return false

**for** *i ←* 0 to *n −1* do

var 🡨 false

**for** *j ← 0* to *n −* 1 do

**if** w1[i] = w2[j]

delete w2[j] from w2

var 🡨 true

break

if var = false return false

**if** w2 is empty return true

**else** return false