



SAPIENZA
UNIVERSITÀ DI ROMA

Algorithms

Exercises 1 — Basic algorithms

Worked exercises on chapters 1–5: representation of numbers, pseudo-code, sorting, searching and numerical algorithms. Every exercise has a complete solution.

Fabio Furini

fabiofurini.github.io

Department of Computer, Control and Management Engineering — Sapienza University of Rome

Online version and downloadable material:

fabiofurini.github.io/algorithms/
github.com/fabiofurini/algorithms

Academic year 2026–2027

Contents

1	Representation of numbers	2
2	Pseudo-code	16
3	Sorting algorithms	33
4	Linear and binary search	98
5	Numerical algorithms	117



Representation of numbers

In the exercises of this chapter numbers are represented only with the definitions and observations of the chapter «Representation of numbers», summarized by the following formulas.

Natural numbers with k digits in base b :

$$(d_{k-1} \dots d_1 d_0)_b = \sum_{i=0}^{k-1} d_i b^i \quad (\text{value of a sequence of digits})$$

$$[0, b^k - 1] \quad (\text{representation interval})$$

$$\lfloor \log_b n \rfloor + 1 \quad (\text{minimum number of digits of } n \geq 1)$$

Integer numbers with k bits:

$$[-(2^{k-1} - 1), 2^{k-1} - 1] \quad (\text{sign-magnitude interval})$$

$$[-2^{k-1}, 2^{k-1} - 1] \quad (\text{two's complement interval})$$

$$n < 0 \mapsto 2^k + n \quad (\text{two's complement of a negative } n)$$

In two's complement, the representation of a negative n is also obtained by inverting the bits of $|n|$ and adding 1; a sum overflows if and only if two operands with the same sign give a result with the opposite sign bit.

Real numbers in IEEE 754 single precision (1 sign bit s , 8 bits of biased exponent E , 23 bits of fraction f), for $1 \leq E \leq 254$:

$$x = (-1)^s \cdot (1.f_1 f_2 \dots f_{23})_2 \cdot 2^{E-127}$$

Double precision: 1 + 11 + 52 bits, bias 1023.

Exercise 1.1 — Representation of numbers (E1)

Consider the representation of natural numbers in the binary system (base $b = 2$): representation intervals and minimum number of digits.

Questions

1. Compute the largest natural number that can be represented with $k = 6$ digits in binary and determine the associated representation interval.
2. For the following values of k (number of digits), determine the representation interval $[0, b^k - 1]$ in base $b = 2$:

$$k = 4, \quad k = 8, \quad k = 10, \quad k = 16$$

3. Determine the minimum number of digits needed to represent each of the following numbers in binary, using the formula $\lfloor \log_b n \rfloor + 1$:

$$n = 64, \quad n = 128, \quad n = 256, \quad n = 512$$

4. Given $k = 12$ bits:

- (a) determine the representation interval;
- (b) verify that $n = 2,048$ can be represented with $k = 12$ bits;
- (c) compute the minimum number of bits needed to represent $n = 2,048$.

Solution

1. The largest natural number that can be represented with $k = 6$ digits in base $b = 2$ is:

$$b^k - 1 = 2^6 - 1 = 64 - 1 = 63$$

This corresponds to the binary sequence $(111111)_2$. The associated representation interval is:

$$[0, 2^6 - 1] = [0, 63]$$

2. Representation intervals for the different values of k :

$$k = 4 : [0, 2^4 - 1] = [0, 16 - 1] = [0, 15]$$

$$k = 8 : [0, 2^8 - 1] = [0, 256 - 1] = [0, 255]$$

$$k = 10 : [0, 2^{10} - 1] = [0, 1,024 - 1] = [0, 1,023]$$

$$k = 16 : [0, 2^{16} - 1] = [0, 65,536 - 1] = [0, 65,535]$$

3. The minimum number of digits for each number:

$$n = 64 = 2^6 : \lfloor \log_2 64 \rfloor + 1 = \lfloor 6 \rfloor + 1 = 6 + 1 = 7$$

$$n = 128 = 2^7 : \lfloor \log_2 128 \rfloor + 1 = \lfloor 7 \rfloor + 1 = 7 + 1 = 8$$

$$n = 256 = 2^8 : \lfloor \log_2 256 \rfloor + 1 = \lfloor 8 \rfloor + 1 = 8 + 1 = 9$$

$$n = 512 = 2^9 : \lfloor \log_2 512 \rfloor + 1 = \lfloor 9 \rfloor + 1 = 9 + 1 = 10$$

For powers of 2, i.e., $n = 2^m$, we have $\log_2 n = m$, and therefore the minimum number of bits is $m + 1$.

4. For $k = 12$ bits:

(a) the representation interval is:

$$[0, 2^{12} - 1] = [0, 4,096 - 1] = [0, 4,095]$$

(b) since $n = 2,048 = 2^{11}$ and $2,048 \leq 4,095$, we have $2,048 \in [0, 4,095]$: the number $n = 2,048$ can be represented with $k = 12$ bits;

(c) the minimum number of bits needed to represent $n = 2,048 = 2^{11}$ is:

$$\lfloor \log_2 2,048 \rfloor + 1 = \lfloor 11 \rfloor + 1 = 12$$

We need exactly 12 bits to represent $2,048 = 2^{11}$, because its binary representation is one digit 1 followed by eleven digits 0:

$$2,048 = (100000000000)_2$$

Exercise 1.2 — Representation of numbers (E2)

Consider the representation intervals and the minimum number of digits of natural numbers in the binary system (base $b = 2$).

Questions

1. Complete the following table of representation intervals in binary:

k (digits)	b (base)	$b^k - 1$	representation interval
5	2		
8	2		
10	2		
12	2		

2. For each natural number n in the table below, compute the minimum number of digits needed to represent it in base $b = 2$, using the formula $\lfloor \log_b n \rfloor + 1$:

n	b	$\log_b n$	$\lfloor \log_b n \rfloor + 1$
7	2		
15	2		
31	2		
100	2		
255	2		

3. Using the relationship between bytes and bits, complete the table:

bytes	bits (k)	2^k	representation interval
1	8	$2^8 = 256$	$[0, 255]$
2			
4			

4. Convert $(11011011)_2$ to base ten, writing the complete sum $\sum_{i=0}^{k-1} d_i \cdot 2^i$.

Solution

1. Representation intervals in binary:

k (digits)	b (base)	$b^k - 1$	representation interval
5	2	$2^5 - 1 = 31$	$[0, 31]$
8	2	$2^8 - 1 = 255$	$[0, 255]$
10	2	$2^{10} - 1 = 1,023$	$[0, 1,023]$
12	2	$2^{12} - 1 = 4,095$	$[0, 4,095]$

2. Minimum number of digits for each natural number:

n	b	$\log_b n$	$\lfloor \log_b n \rfloor + 1$
7	2	$\log_2 7 \approx 2.807$	$\lfloor 2.807 \rfloor + 1 = 3$
15	2	$\log_2 15 \approx 3.907$	$\lfloor 3.907 \rfloor + 1 = 4$
31	2	$\log_2 31 \approx 4.954$	$\lfloor 4.954 \rfloor + 1 = 5$
100	2	$\log_2 100 \approx 6.644$	$\lfloor 6.644 \rfloor + 1 = 7$
255	2	$\log_2 255 \approx 7.994$	$\lfloor 7.994 \rfloor + 1 = 8$

For the numbers of the form $2^k - 1$ we need exactly k bits: $2^k - 1$ is the largest natural number with k bits.

3. Since 1 byte = 8 bits:

bytes	bits (k)	2^k	representation interval
1	8	$2^8 = 256$	$[0, 255]$
2	16	$2^{16} = 65,536$	$[0, 65,535]$
4	32	$2^{32} = 4,294,967,296$	$[0, 4,294,967,295]$

4. The sequence has $k = 8$ digits: $d_7 = 1, d_6 = 1, d_5 = 0, d_4 = 1, d_3 = 1, d_2 = 0, d_1 = 1, d_0 = 1$. Hence:

$$\begin{aligned}
 (11011011)_2 &= \sum_{i=0}^7 d_i \cdot 2^i \\
 &= d_0 \cdot 2^0 + d_1 \cdot 2^1 + d_2 \cdot 2^2 + d_3 \cdot 2^3 + d_4 \cdot 2^4 + d_5 \cdot 2^5 + d_6 \cdot 2^6 + d_7 \cdot 2^7 \\
 &= 1 \cdot 1 + 1 \cdot 2 + 0 \cdot 4 + 1 \cdot 8 + 1 \cdot 16 + 0 \cdot 32 + 1 \cdot 64 + 1 \cdot 128 \\
 &= 1 + 2 + 0 + 8 + 16 + 0 + 64 + 128 \\
 &= 219
 \end{aligned}$$

Therefore, $(11011011)_2 = (219)_{10}$.

Exercise 1.3 — Representation of numbers (E3)

Consider the complete enumeration of the sequences of digits in binary for a given number of bits, and the binary representation of some natural numbers.

Questions

1. Complete the table of all the sequences of $k = 5$ digits in binary (base 2) and their decimal values. Show the detailed calculation for at least the first three and the last three sequences:

Sequence of digits	b	k	decimal representation
$(00000)_2$	2	5	
$(00001)_2$	2	5	
$(00010)_2$	2	5	
$(00011)_2$	2	5	
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(11101)_2$	2	5	
$(11110)_2$	2	5	
$(11111)_2$	2	5	

- How many different sequences are possible with $k = 5$ bits? Verify the answer with the formula b^k , where $b = 2$.
- Find the binary representation of the following decimal numbers, by determining which powers of 2 sum to the given value:

$$n = 21, \quad n = 50, \quad n = 63$$

- For each sequence below, determine whether it represents a power of 2 and, if so, which power:

$$(10000)_2, \quad (10101)_2, \quad (100000000)_2$$

Solution

- All the sequences with $k = 5$ bits:

Sequence of digits	b	k	decimal representation
$(00000)_2$	2	5	$(0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0)_{10} = (0)_{10}$
$(00001)_2$	2	5	$(0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0)_{10} = (1)_{10}$
$(00010)_2$	2	5	$(0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)_{10} = (2)_{10}$
$(00011)_2$	2	5	$(3)_{10}$
$(00100)_2$	2	5	$(4)_{10}$
$(00101)_2$	2	5	$(5)_{10}$
$(00110)_2$	2	5	$(6)_{10}$
$(00111)_2$	2	5	$(7)_{10}$
$(01000)_2$	2	5	$(8)_{10}$
$(01001)_2$	2	5	$(9)_{10}$
$\vdots$	$\vdots$	$\vdots$	
$(11101)_2$	2	5	$(1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0)_{10} = (29)_{10}$
$(11110)_2$	2	5	$(1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)_{10} = (30)_{10}$
$(11111)_2$	2	5	$(1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0)_{10} = (31)_{10}$

2. The total number of different sequences with $k = 5$ bits in base $b = 2$ is:

$$b^k = 2^5 = 32$$

Indeed, the sequences go from $(00000)_2 = 0$ to $(11111)_2 = 31$, which gives exactly 32 different values (from 0 to 31 inclusive).

3. Binary representation of the decimal numbers:

- $n = 21$: we look for the powers of 2 that sum to 21, starting from the largest power not greater than 21:

$$21 = 16 + 5 = 16 + 4 + 1 = 2^4 + 2^2 + 2^0$$

Therefore $(21)_{10} = (10101)_2$.

- $n = 50$:

$$50 = 32 + 18 = 32 + 16 + 2 = 2^5 + 2^4 + 2^1$$

Therefore $(50)_{10} = (110010)_2$.

- $n = 63$:

$$63 = 32 + 31 = 32 + 16 + 8 + 4 + 2 + 1 = 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$$

Alternatively, $63 = 2^6 - 1$ is the largest natural number with 6 bits, so all its 6 bits are 1. Therefore $(63)_{10} = (111111)_2$.

4. Powers of 2:

- $(10000)_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 2^4 = 16$: it is a power of 2, namely 2^4 .
- $(10101)_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 16 + 4 + 1 = 21$: it is not a power of 2 (it is the sum of three different powers of 2).
- $(100000000)_2 = 1 \cdot 2^8 = 256$: it is a power of 2, namely 2^8 .

A sequence in binary represents a power of 2 if and only if it has exactly one digit equal to 1 and all the other digits equal to 0.

Exercise 1.4 — Representation of numbers (E4)

Consider the two's complement representation of integer numbers in binary.

Questions

1. Determine the representation interval of the integers in two's complement with the following numbers of bits:

$$k = 4, \quad k = 8, \quad k = 16, \quad k = 32$$

2. Complete the following table with all the two's complement representations with $k = 4$ bits:

Binary	Decimal	Sign bit
(0000) ₂		
(0001) ₂		
(0010) ₂		
(0011) ₂		
(0100) ₂		
(0101) ₂		
(0110) ₂		
(0111) ₂		
(1000) ₂		
(1001) ₂		
(1010) ₂		
(1011) ₂		
(1100) ₂		
(1101) ₂		
(1110) ₂		
(1111) ₂		

3. Compute the two's complement representation of -5 with 8 bits in three steps: (1) binary representation of 5, (2) inversion of all the bits, (3) addition of 1.
4. Verify that with $k = 5$ bits in two's complement we can represent the integer -16 but not the integer 16.

Solution

1. Representation intervals, with the formula $[-2^{k-1}, 2^{k-1} - 1]$:

$$k = 4 : [-2^{4-1}, 2^{4-1} - 1] = [-2^3, 2^3 - 1] = [-8, 7]$$

$$k = 8 : [-2^{8-1}, 2^{8-1} - 1] = [-2^7, 2^7 - 1] = [-128, 127]$$

$$k = 16 : [-2^{16-1}, 2^{16-1} - 1] = [-2^{15}, 2^{15} - 1] = [-32,768, 32,767]$$

$$k = 32 : [-2^{32-1}, 2^{32-1} - 1] = [-2^{31}, 2^{31} - 1] = [-2,147,483,648, 2,147,483,647]$$

2. All the two's complement representations with $k = 4$ bits: a sequence with sign bit 1 and natural value r represents $r - 16$.

Binary	Decimal	Sign bit
$(0000)_2$	0	0 (non-negative)
$(0001)_2$	1	0 (non-negative)
$(0010)_2$	2	0 (non-negative)
$(0011)_2$	3	0 (non-negative)
$(0100)_2$	4	0 (non-negative)
$(0101)_2$	5	0 (non-negative)
$(0110)_2$	6	0 (non-negative)
$(0111)_2$	7	0 (non-negative)
$(1000)_2$	$8 - 16 = -8$	1 (negative)
$(1001)_2$	$9 - 16 = -7$	1 (negative)
$(1010)_2$	$10 - 16 = -6$	1 (negative)
$(1011)_2$	$11 - 16 = -5$	1 (negative)
$(1100)_2$	$12 - 16 = -4$	1 (negative)
$(1101)_2$	$13 - 16 = -3$	1 (negative)
$(1110)_2$	$14 - 16 = -2$	1 (negative)
$(1111)_2$	$15 - 16 = -1$	1 (negative)

The representation interval with 4 bits is $[-8, 7]$.

3. Two's complement representation of -5 with 8 bits:

- Step 1: binary representation of 5 with 8 bits: $(00000101)_2$.

- Step 2: inversion of all the bits: $(11111010)_2$.
- Step 3: addition of 1 (the first row contains the carries; the carry above a column comes from the column on its right):

carry	0	0	0	0	0	0	0
	1	1	1	1	1	0	1
	0	0	0	0	0	0	1
sum	1	1	1	1	1	0	1

Therefore, the two's complement representation of -5 with 8 bits is $(11111011)_2$.
Check:

$$2^8 - 5 = 256 - 5 = 251 = (11111011)_2$$

4. With $k = 5$ bits the representation interval is:

$$[-2^{5-1}, 2^{5-1} - 1] = [-2^4, 2^4 - 1] = [-16, 15]$$

- Since $-16 \in [-16, 15]$, the integer -16 can be represented, and its representation is:

$$2^5 - 16 = 32 - 16 = 16 = (10000)_2$$

- Since $16 > 15$, we have $16 \notin [-16, 15]$: the integer 16 cannot be represented with 5 bits in two's complement.

Two's complement represents one more negative integer than positive integers, because the interval $[-2^{k-1}, 2^{k-1} - 1]$ is asymmetric.

Exercise 1.5 — Representation of numbers (E5)

Consider the addition of integer numbers in two's complement, the detection of overflows, and the comparison between sign-magnitude and two's complement.

Questions

1. Perform the following additions in two's complement with $k = 5$ bits, showing the binary representations and the column addition, and verify the results:

$$(-7) + 3, \quad (-8) + 8, \quad 5 + 6$$

2. Determine whether an overflow occurs in the following additions with $k = 5$ bits, and explain why:

$$7 + 7, \quad (-8) + (-8), \quad 7 + (-8), \quad 9 + 8$$

3. Compare the sign-magnitude and the two's complement representations with $k = 4$ bits:

- (a) what is the representation interval of each?
 (b) how many representations does zero have in each system?
 (c) which system is better for arithmetic operations, and why?
4. With $k = 8$ bits in two's complement, compute the representations of -64 and -128 , using the formula $2^k + n$ for the negative integers n .

Solution

In the column additions, the first row contains the carries: the carry written above a column comes from the column on its right, and the leftmost column receives the carry out of the most significant position, which is discarded.

1. With $k = 5$ bits the representation interval is $[-16, 15]$.

- $(-7) + 3 = -4$. Representations: $-7 \rightarrow 2^5 - 7 = 25 = (11001)_2$ and $3 = (00011)_2$.

carry	0	0	0	1	1
		1	1	0	0
		0	0	0	1
sum	0	1	1	1	0

The result $(11100)_2$ has natural value 28, hence it represents $28 - 32 = -4$.

- $(-8) + 8 = 0$. Representations: $-8 \rightarrow 2^5 - 8 = 24 = (11000)_2$ and $8 = (01000)_2$.

carry	1	1	0	0	0
		1	1	0	0
		0	1	0	0
sum	1	0	0	0	0

The result is $(00000)_2 = 0$.

- $5 + 6 = 11$. Representations: $5 = (00101)_2$ and $6 = (00110)_2$.

carry	0	0	1	0	0
		0	0	1	0
		0	0	1	1
sum	0	0	1	0	1

The result is $(01011)_2 = 11$.

2. Overflow detection with $k = 5$ bits (interval $[-16, 15]$):

- $7 + 7$ (expected 14). Representations: $7 = (00111)_2$.

carry	0	0	1	1	1
		0	0	1	1
		0	0	1	1
sum	0	0	1	1	0

Result: $(01110)_2 = 14$: both operands are positive and the result is positive: no overflow ($14 \leq 15$).

- $(-8) + (-8)$ (expected -16). Representations: $-8 \rightarrow 2^5 - 8 = 24 = (11000)_2$.

carry	1	1	0	0	0
		1	1	0	0
		1	1	0	0
sum	1	1	0	0	0

Result: $(10000)_2 = -16$: both operands are negative and the result is negative: no overflow ($-16 \geq -16$). The carry out is discarded.

- $7 + (-8)$ (expected -1). Representations: $7 = (00111)_2$ and $-8 \rightarrow 2^5 - 8 = 24 = (11000)_2$.

carry	0	0	0	0	0
		0	0	1	1
		1	1	0	0
sum	0	1	1	1	1

Result: $(11111)_2 = -1$: the operands have different signs, so an overflow cannot occur.

- $9 + 8$ (expected 17). Representations: $9 = (01001)_2$ and $8 = (01000)_2$.

carry	0	1	0	0	0
		0	1	0	0
		0	1	0	0
sum	0	1	0	0	1

Result: $(10001)_2 = -15$: both operands are positive, but the result has sign bit 1: overflow. Indeed $9 + 8 = 17 > 15$ cannot be represented with 5 bits.

Rule: an overflow occurs only when two integers with the same sign are added and the result has the opposite sign bit.

3. Comparison of sign-magnitude and two's complement with $k = 4$ bits:

- (a) representation intervals:

$$\text{sign-magnitude: } [-(2^{4-1} - 1), 2^{4-1} - 1] = [-7, 7]$$

$$\text{two's complement: } [-2^{4-1}, 2^{4-1} - 1] = [-8, 7]$$

two's complement represents one more integer, -8 ;

- (b) sign-magnitude has two representations of zero, $(0000)_2 = +0$ and $(1000)_2 = -0$; two's complement has only one, $(0000)_2 = 0$;

- (c) two's complement is better for arithmetic operations, because:

- zero has a unique representation;
- addition and subtraction are performed as for natural numbers (modulo 2^k), with the same circuits used for unsigned numbers;
- the sign bit needs no special handling, so the circuits are simpler and more efficient.

4. Representations with $k = 8$ bits:

$$-64: 2^k + n = 2^8 + (-64) = 256 - 64 = 192 = (11000000)_2$$

$$-128: 2^k + n = 2^8 + (-128) = 256 - 128 = 128 = (10000000)_2$$

The integer $-128 = -2^{8-1}$ is the smallest integer representable with 8 bits in two's complement.

Exercise 1.6 — Representation of numbers (E6)

Consider the representation of real numbers in floating point, according to the IEEE 754 standard.

Questions

1. Write the real number 13.25 in binary and in normalized binary scientific notation, and give its 32 bits in IEEE 754 single precision.
2. Determine the real number represented by the following 32 bits in IEEE 754 single precision:

0 01111100 010000000000000000000000

3. Using the double precision format (1 bit of sign, 11 bits of exponent with bias 1023, 52 bits of fraction), compute the largest finite number and the smallest positive normalized number. The values $E = 0$ and $E = 2^{11} - 1 = 2,047$ of the biased exponent are reserved.
4. In double precision, adding ten times the number 0.1 to a variable initialized to 0 gives 0.9999999999999999 and not 1. Explain why.

Solution

1. Representation of 13.25:

- Step 1 (binary): $13.25 = 8 + 4 + 1 + 0.25 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2}$, hence:

$$13.25 = (1101.01)_2$$

- Step 2 (normalization): the point moves three positions to the left:

$$(1101.01)_2 = (1.10101)_2 \cdot 2^3$$

hence $s = 0$, $e = 3$ and the fraction starts with 10101.

- Step 3 (biased exponent): $E = e + 127 = 130 = 128 + 2 = (10000010)_2$.
- Step 4 (fraction): 10101 completed with zeros up to 23 bits.

Pseudo-code

Exercise 2.1 — Pseudo-code (E1)

Given an array $a[1..n]$ of $n \geq 1$ integer numbers, consider the problem of determining the minimum value in the array.

Questions

1. Write the pseudo-code of an algorithm that finds the minimum value in an array a of n integer numbers. The algorithm should:
 - take as input an array $a[1..n]$ of $n \geq 1$ integer numbers;
 - return the minimum value min in the array;
 - use a single pass through the array.
2. Execute the algorithm step by step on the following instances:
 - Instance 1: $a = [9, 12, 3, 16, 19, 11, 12]$ with $n = 7$;
 - Instance 2: $a = [15, 7, 22, 4]$ with $n = 4$;
 - Instance 3: $a = [100]$ with $n = 1$.

For each instance, show the initial state of the variables, their state after each iteration, and the final result.

3. Write the pseudo-code of an algorithm that finds both the maximum and the minimum values in a single pass through the array.

Solution

1. Pseudo-code for finding the minimum value in an array:

Algorithm: FindMinimum**Input** : An array $a[1..n]$ of $n \geq 1$ integer numbers**Output**: The minimum value in the array a

```

1 function FindMinimum( $a, n$ )
2    $min := a[1]$ 
3   for  $i := 2$  to  $n$  do
4     if  $a[i] < min$  then
5        $min \leftarrow a[i]$ 
6     end
7   end
8   return  $min$ 
9 end

```

- We initialize min to $a[1]$, the first element (it exists, since $n \geq 1$).
- We iterate through each remaining element of the array, comparing it with the current minimum.
- If the current element is smaller, we update the minimum.
- Finally, we return the minimum value found.

2. Step-by-step execution on the instances.

Instance 1: $a = [9, 12, 3, 16, 19, 11, 12]$, $n = 7$

i	$a[i]$	$a[i] < min?$	min
–	–	–	9
2	12	No	9
3	3	Yes	3
4	16	No	3
5	19	No	3
6	11	No	3
7	12	No	3

Return: $min = 3$

Instance 2: $a = [15, 7, 22, 4]$, $n = 4$

i	$a[i]$	$a[i] < min?$	min
–	–	–	15
2	7	Yes	7
3	22	No	7
4	4	Yes	4

Return: $min = 4$

Instance 3: $a = [100]$, $n = 1$

i	$a[i]$	$a[i] < min?$	min
—	—	—	100
no iterations: the loop starts at $i = 2 > n = 1$			

Return: $min = 100$

3. Pseudo-code to find both the maximum and the minimum in a single pass:

Algorithm: FindMaxMin

Input : An array $a[1..n]$ of $n \geq 1$ integer numbers

Output: The pair (max, min) , where max is the maximum and min is the minimum of a

```

1 function FindMaxMin( $a, n$ )
2    $max := a[1]$ 
3    $min := a[1]$ 
4   for  $i := 2$  to  $n$  do
5     if  $a[i] > max$  then
6        $max \leftarrow a[i]$ 
7     end
8     if  $a[i] < min$  then
9        $min \leftarrow a[i]$ 
10    end
11  end
12  return  $max, min$ 
13 end

```

For example, on Instance 1 ($a = [9, 12, 3, 16, 19, 11, 12]$) the algorithm returns $(19, 3)$.

Exercise 2.2 — Pseudo-code (E2)

Given an array $a[1..n]$ of $n \geq 1$ integer numbers, consider the problem of computing the sum of its values.

Questions

- Write the pseudo-code of an algorithm that computes the sum of all the elements of an array a of n integer numbers. The algorithm should:
 - take as input an array $a[1..n]$ of $n \geq 1$ integer numbers;

- return the sum sum of all the elements;
 - initialize a variable sum to 0 and add each element to it.
2. Execute the algorithm step by step on the following instances:
- Instance 1: $a = [5, 12, 3, 8]$ with $n = 4$;
 - Instance 2: $a = [10, 20, 30]$ with $n = 3$;
 - Instance 3: $a = [7]$ with $n = 1$.

For each instance, show the initial value of the sum, its value after adding each element, and the final result.

3. Modify the pseudo-code to compute both the sum and the average $avg = \frac{sum}{n}$ of the elements, with a single pass through the array.

Solution

1. Pseudo-code for computing the sum of all the elements of an array:

Algorithm: ArraySum

Input : An array $a[1..n]$ of $n \geq 1$ integer numbers

Output: The sum of all the elements of the array a

```

1 function ArraySum( $a, n$ )
2    $sum := 0$ 
3   for  $i := 1$  to  $n$  do
4      $sum \leftarrow sum + a[i]$ 
5   end
6   return  $sum$ 
7 end

```

- We initialize sum to 0.
 - We iterate through each element of the array, adding it to the running sum .
 - Finally, we return the total sum .
2. Step-by-step execution on the instances.

Instance 1: $a = [5, 12, 3, 8]$, $n = 4$

i	$a[i]$	sum
—	—	0
1	5	5
2	12	17
3	3	20
4	8	28

Return: $sum = 28$

Instance 2: $a = [10, 20, 30]$, $n = 3$

i	$a[i]$	sum
—	—	0
1	10	10
2	20	30
3	30	60

Return: $sum = 60$

Instance 3: $a = [7]$, $n = 1$

i	$a[i]$	sum
—	—	0
1	7	7

Return: $sum = 7$

3. Pseudo-code to compute both the sum and the average:

Algorithm: ArraySumAndAverage

Input : An array $a[1..n]$ of $n \geq 1$ integer numbers

Output: The pair (sum, avg) , where sum is the total and avg is the average of the elements

```

1 function ArraySumAndAverage( $a, n$ )
2    $sum := 0$ 
3   for  $i := 1$  to  $n$  do
4      $sum \leftarrow sum + a[i]$ 
5   end
6    $avg := sum/n$ 
7   return  $sum, avg$ 
8 end

```

The precondition $n \geq 1$ guarantees that the division sum/n is defined. For example, on Instance 1 ($a = [5, 12, 3, 8]$, $n = 4$) the algorithm returns $(28, 7)$, since the sum is 28 and the average is $28/4 = 7$.

Exercise 2.3 — Pseudo-code (E3)

Given a two-dimensional array $b[1..m][1..n]$ of $m \times n$ integer numbers and a threshold value k , consider the problem of counting the number of its values strictly greater than k .

Questions

- Write the pseudo-code of an algorithm that counts how many elements of b are strictly greater than the threshold k . The algorithm should:
 - take as input a two-dimensional array $b[1..m][1..n]$, the dimensions m and n , and the threshold k ;
 - return the number of elements satisfying $b[i][j] > k$;
 - scan all the entries of the array with two nested loops.
- Execute the algorithm step by step on the following instances, showing the value of *count* after each update:
 - Instance 1: $b = \begin{bmatrix} 3 & 7 & 1 \\ 5 & 2 & 8 \end{bmatrix}$, $m = 2$, $n = 3$, $k = 4$;
 - Instance 2: $b = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$, $m = 3$, $n = 3$, $k = 5$.
- Modify the pseudo-code so that, in addition to the count, the algorithm also returns an array of pairs (i, j) listing the positions of all the elements of the two-dimensional array greater than k .

Solution

- Pseudo-code for counting the elements greater than a threshold:

Algorithm: CountGreaterThan

Input : A two-dimensional array $b[1..m][1..n]$, the dimensions m and n , and a threshold k

Output: The number *count* of elements with $b[i][j] > k$

```

1 function CountGreaterThan( $b, m, n, k$ )
2    $count := 0$ 
3   for  $i := 1$  to  $m$  do
4     for  $j := 1$  to  $n$  do
5       if  $b[i][j] > k$  then
6          $count \leftarrow count + 1$ 
7       end
8     end
9   end
10  return  $count$ 
11 end

```

- We initialize *count* to 0.
- The outer loop iterates over the rows $i = 1, 2, \dots, m$, the inner loop over the columns $j = 1, 2, \dots, n$.
- For each entry $b[i][j]$, if it exceeds k we increase *count*.
- Finally, we return *count*.

2. Step-by-step execution on the instances.

Instance 1: $b = \begin{bmatrix} 3 & 7 & 1 \\ 5 & 2 & 8 \end{bmatrix}$, $m = 2$, $n = 3$, $k = 4$

i	j	$b[i][j]$	$b[i][j] > 4?$	$count$
–	–	–	–	0
1	1	3	No	0
1	2	7	Yes	1
1	3	1	No	1
2	1	5	Yes	2
2	2	2	No	2
2	3	8	Yes	3

Return: $count = 3$ (elements 7, 5, 8)

Instance 2: $b = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$, $m = 3$, $n = 3$, $k = 5$

i	j	$b[i][j]$	$b[i][j] > 5?$	$count$
—	—	—	—	0
1	1	1	No	0
1	2	2	No	0
1	3	3	No	0
2	1	4	No	0
2	2	5	No	0
2	3	6	Yes	1
3	1	7	Yes	2
3	2	8	Yes	3
3	3	9	Yes	4

Return: $count = 4$ (elements 6, 7, 8, 9)

3. Pseudo-code returning also the positions of the elements greater than k . The array pos is declared with $m \cdot n$ elements, the largest possible number of positions; only its first $count$ elements are used.

Algorithm: CountAndListGreaterThan

Input : A two-dimensional array $b[1..m][1..n]$, the dimensions m and n , and a threshold k

Output: The number $count$ and the array $pos[1..count]$ of the pairs (i, j) with $b[i][j] > k$

```

1 function CountAndListGreaterThan( $b, m, n, k$ )
2    $count := 0$ 
3    $pos := \text{new array}[1..m \cdot n]$ 
4   for  $i := 1$  to  $m$  do
5     for  $j := 1$  to  $n$  do
6       if  $b[i][j] > k$  then
7          $count \leftarrow count + 1$ 
8          $pos[count] \leftarrow (i, j)$ 
9       end
10    end
11  end
12  return  $count, pos[1..count]$ 
13 end

```

For Instance 1, the algorithm returns $count = 3$ and $pos = [(1, 2), (2, 1), (2, 3)]$; for Instance 2, it returns $count = 4$ and $pos = [(2, 3), (3, 1), (3, 2), (3, 3)]$.

Exercise 2.4 — Pseudo-code (E4)

Given a two-dimensional array $b[1..m][1..n]$ of $m \times n$ integer numbers, with $m \geq 1$ and $n \geq 1$, consider the problem of finding the minimum value of each column.

Questions

- Write the pseudo-code of an algorithm that computes the minimum of each column of the two-dimensional array b . The algorithm should:
 - take as input a two-dimensional array $b[1..m][1..n]$ and the dimensions m, n ;
 - return an array $colMin[1..n]$, where the element $colMin[j]$ is the minimum of the column j ;
 - iterate over the columns and the rows with two nested loops.
- Execute the algorithm step by step on the following instances, showing each update of the array $colMin$:
 - Instance 1: $b = \begin{bmatrix} 4 & 9 & 2 \\ 7 & 1 & 6 \\ 3 & 5 & 8 \end{bmatrix}$, $m = 3$, $n = 3$;
 - Instance 2: $b = \begin{bmatrix} 8 & 3 \\ 5 & 7 \\ 2 & 4 \end{bmatrix}$, $m = 3$, $n = 2$.
- Modify the pseudo-code so that, in addition to the minimum value of each column, the algorithm also returns the first row index $rowIdx[j]$ where the minimum of column j is located.

Solution

- Pseudo-code for computing the column minima:

Algorithm: ColumnMinima

Input : A two-dimensional array $b[1..m][1..n]$, the dimensions $m \geq 1$ and $n \geq 1$

Output: An array $colMin[1..n]$, where $colMin[j]$ is the minimum of column j

```

1 function ColumnMinima( $b, m, n$ )
2    $colMin := \text{new array}[1..n]$ 
3   for  $j := 1$  to  $n$  do
4      $colMin[j] \leftarrow b[1][j]$ 
5     for  $i := 2$  to  $m$  do
6       if  $b[i][j] < colMin[j]$  then
7          $colMin[j] \leftarrow b[i][j]$ 
8       end
9     end
10  end
11  return  $colMin$ 
12 end

```

- The outer loop iterates over the columns $j = 1, 2, \dots, n$.
- We initialize $colMin[j]$ to $b[1][j]$, the first element of column j .
- The inner loop scans the rows $i = 2, 3, \dots, m$, comparing each entry with the current minimum of the column, and updates $colMin[j]$ when a smaller element is found.
- Finally, we return the array $colMin$.

2. Step-by-step execution on the instances.

Instance 1: $b = \begin{bmatrix} 4 & 9 & 2 \\ 7 & 1 & 6 \\ 3 & 5 & 8 \end{bmatrix}$, $m = 3$, $n = 3$

Column $j = 1$

i	$b[i][1]$	$<? colMin[1]$	
—	—	—	4
2	7	No	4
3	3	Yes	3

Column $j = 2$

i	$b[i][2]$	$<? colMin[2]$	
—	—	—	9
2	1	Yes	1
3	5	No	1

Column $j = 3$

i	$b[i][3]$	$<?$	$colMin[3]$
–	–	–	2
2	6	No	2
3	8	No	2

The column $<?$ reports the outcome of the comparison $b[i][j] < colMin[j]$.

Return: $colMin = [3, 1, 2]$

Instance 2: $b = \begin{bmatrix} 8 & 3 \\ 5 & 7 \\ 2 & 4 \end{bmatrix}$, $m = 3$, $n = 2$

Column $j = 1$

i	$b[i][1]$	$<?$	$colMin[1]$
–	–	–	8
2	5	Yes	5
3	2	Yes	2

Column $j = 2$

i	$b[i][2]$	$<?$	$colMin[2]$
–	–	–	3
2	7	No	3
3	4	No	3

Return: $colMin = [2, 3]$

3. Pseudo-code returning also the row index of each column minimum. The update uses the strict comparison $<$, so in case of ties the first row is kept.

Algorithm: ColumnMinimaWithIndex

Input : A two-dimensional array $b[1..m][1..n]$, the dimensions $m \geq 1$ and $n \geq 1$

Output: The arrays $colMin[1..n]$ and $rowIdx[1..n]$

```

1 function ColumnMinimaWithIndex( $b, m, n$ )
2    $colMin := \text{new array}[1..n]$ 
3    $rowIdx := \text{new array}[1..n]$ 
4   for  $j := 1$  to  $n$  do
5      $colMin[j] \leftarrow b[1][j]$ 
6      $rowIdx[j] \leftarrow 1$ 
7     for  $i := 2$  to  $m$  do
8       if  $b[i][j] < colMin[j]$  then
9          $colMin[j] \leftarrow b[i][j]$ 
10         $rowIdx[j] \leftarrow i$ 
11      end
12    end
13  end
14  return  $colMin, rowIdx$ 
15 end

```

For Instance 1, the algorithm returns $colMin = [3, 1, 2]$ and $rowIdx = [3, 2, 1]$; for Instance 2, it returns $colMin = [2, 3]$ and $rowIdx = [3, 1]$.

Exercise 2.5 — Pseudo-code (E5)

Given a two-dimensional array $b[1..m][1..n]$ of $m \times n$ integer numbers, consider the problem of computing its transposed two-dimensional array, i.e., the two-dimensional array $c[1..n][1..m]$ with $c[j][i] = b[i][j]$.

Questions

- Write the pseudo-code of an algorithm that computes the transpose of a two-dimensional array $b[1..m][1..n]$. The algorithm should:
 - take as input a two-dimensional array $b[1..m][1..n]$ and the dimensions m, n ;
 - allocate a new two-dimensional array $c[1..n][1..m]$ for the result;
 - return the transposed two-dimensional array c , such that $c[j][i] = b[i][j]$.
- Execute the algorithm step by step on the following instances, showing the two-dimensional array c being filled entry by entry:

- Instance 1: $b = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$, $m = 2$, $n = 3$;

- Instance 2: $b = \begin{bmatrix} 7 & 2 \\ 3 & 8 \\ 1 & 5 \end{bmatrix}$, $m = 3$, $n = 2$.
3. For a square two-dimensional array ($m = n$), write the pseudo-code of an algorithm that performs the transposition in place (without allocating a new two-dimensional array), by exchanging the elements $b[i][j]$ and $b[j][i]$ for all $i < j$.

Solution

1. Pseudo-code for the transposition of a two-dimensional array:

Algorithm: Transpose

Input : A two-dimensional array $b[1..m][1..n]$, the dimensions m and n

Output: The transposed two-dimensional array $c[1..n][1..m]$

```

1 function Transpose( $b, m, n$ )
2    $c := \text{new array}[1..n][1..m]$ 
3   for  $i := 1$  to  $m$  do
4     for  $j := 1$  to  $n$  do
5        $c[j][i] \leftarrow b[i][j]$ 
6     end
7   end
8   return  $c$ 
9 end
  
```

- We allocate c as a new $n \times m$ two-dimensional array.
 - The outer loop iterates over the rows $i = 1, 2, \dots, m$ of b , the inner loop over the columns $j = 1, 2, \dots, n$ of b .
 - Each entry $b[i][j]$ is copied to the position $c[j][i]$; every entry of c is assigned exactly once.
 - Finally, we return the transposed two-dimensional array c .
2. Step-by-step execution on the instances.

Instance 1: $b = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$, $m = 2$, $n = 3$, so c is 3×2

i	j	$b[i][j]$	assignment
1	1	1	$c[1][1] \leftarrow 1$
1	2	2	$c[2][1] \leftarrow 2$
1	3	3	$c[3][1] \leftarrow 3$
2	1	4	$c[1][2] \leftarrow 4$
2	2	5	$c[2][2] \leftarrow 5$
2	3	6	$c[3][2] \leftarrow 6$

Return: $c = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$

Instance 2: $b = \begin{bmatrix} 7 & 2 \\ 3 & 8 \\ 1 & 5 \end{bmatrix}$, $m = 3$, $n = 2$, so c is 2×3

i	j	$b[i][j]$	assignment
1	1	7	$c[1][1] \leftarrow 7$
1	2	2	$c[2][1] \leftarrow 2$
2	1	3	$c[1][2] \leftarrow 3$
2	2	8	$c[2][2] \leftarrow 8$
3	1	1	$c[1][3] \leftarrow 1$
3	2	5	$c[2][3] \leftarrow 5$

Return: $c = \begin{bmatrix} 7 & 3 & 1 \\ 2 & 8 & 5 \end{bmatrix}$

3. In-place transposition of a square two-dimensional array ($m = n$):

Algorithm: TransposeInPlace

Input : A square two-dimensional array $b[1..n][1..n]$, the dimension n

Output: The two-dimensional array b , transposed in place

```

1 function TransposeInPlace( $b, n$ )
2   for  $i := 1$  to  $n - 1$  do
3     for  $j := i + 1$  to  $n$  do
4        $tmp := b[i][j]$ 
5        $b[i][j] \leftarrow b[j][i]$ 
6        $b[j][i] \leftarrow tmp$ 
7     end
8   end
9   return  $b$ 
10 end
```

The inner loop starts at $j = i + 1$, so that each pair of symmetric positions is exchanged exactly once (exchanging it twice would restore the original array) and the elements $b[i][i]$ on the diagonal are never moved. For $i = n$ the inner loop would be empty, so the outer loop stops at $n - 1$. For example, on $b = \begin{bmatrix} 4 & 1 \\ 9 & 6 \end{bmatrix}$ ($n = 2$) the only iteration exchanges $b[1][2] = 1$ and $b[2][1] = 9$, giving $b = \begin{bmatrix} 4 & 9 \\ 1 & 6 \end{bmatrix}$.

Exercise 2.6 — Pseudo-code (E6)

Given a two-dimensional array $a[1..m][1..p]$ of $m \times p$ integer numbers and a two-dimensional array $b[1..p][1..n]$ of $p \times n$ integer numbers, consider the problem of computing their product, i.e., the two-dimensional array $c[1..m][1..n]$ with:

$$c[i][j] = \sum_{k=1}^p a[i][k] \cdot b[k][j]$$

Questions

1. Write the pseudo-code of an algorithm that computes the product $c = a \cdot b$. The algorithm should:
 - take as input the two-dimensional arrays $a[1..m][1..p]$ and $b[1..p][1..n]$, and the dimensions m, p, n ;
 - allocate a new two-dimensional array $c[1..m][1..n]$, initialized to zero;
 - use three nested loops (over i, j, k) to compute the product with the formula above;
 - return the product two-dimensional array c .
2. Execute the algorithm step by step on the following instances, showing the values of k and $c[i][j]$ at each step of the innermost loop:
 - Instance 1: $a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, $b = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$, $m = p = n = 2$;
 - Instance 2: $a = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 3 \end{bmatrix}$, $b = \begin{bmatrix} 4 & 5 \\ 6 & 7 \end{bmatrix}$, $m = 3, p = 2, n = 2$.

Solution

1. Pseudo-code for the product of two two-dimensional arrays:

Algorithm: MatrixProduct

Input : The two-dimensional arrays $a[1..m][1..p]$ and $b[1..p][1..n]$, the dimensions m, p, n

Output: The product two-dimensional array $c[1..m][1..n]$

```

1 function MatrixProduct( $a, b, m, p, n$ )
2    $c := \text{new array}[1..m][1..n]$ 
3   for  $i := 1$  to  $m$  do
4     for  $j := 1$  to  $n$  do
5        $c[i][j] \leftarrow 0$ 
6     end
7   end
8   for  $i := 1$  to  $m$  do
9     for  $j := 1$  to  $n$  do
10      for  $k := 1$  to  $p$  do
11         $c[i][j] \leftarrow c[i][j] + a[i][k] \cdot b[k][j]$ 
12      end
13    end
14  end
15  return  $c$ 
16 end

```

- We allocate c as an $m \times n$ two-dimensional array and initialize all its elements to zero with two nested loops.
- The outer loop iterates over the rows of c , $i = 1, 2, \dots, m$, and the middle loop over its columns, $j = 1, 2, \dots, n$.
- The inner loop computes the sum $\sum_{k=1}^p a[i][k] \cdot b[k][j]$, adding one product at each iteration.
- Finally, we return c .

2. Step-by-step execution on the instances.

Instance 1: $a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, $b = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$, $m = p = n = 2$

i	j	k	$a[i][k]$	$b[k][j]$	product	$c[i][j]$ (after)
1	1	1	1	5	$1 \cdot 5 = 5$	5
1	1	2	2	7	$2 \cdot 7 = 14$	19
1	2	1	1	6	$1 \cdot 6 = 6$	6
1	2	2	2	8	$2 \cdot 8 = 16$	22
2	1	1	3	5	$3 \cdot 5 = 15$	15
2	1	2	4	7	$4 \cdot 7 = 28$	43
2	2	1	3	6	$3 \cdot 6 = 18$	18
2	2	2	4	8	$4 \cdot 8 = 32$	50

Return: $c = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$

Instance 2: $a = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 3 \end{bmatrix}$, $b = \begin{bmatrix} 4 & 5 \\ 6 & 7 \end{bmatrix}$, $m = 3$, $p = 2$, $n = 2$

i	j	k	$a[i][k]$	$b[k][j]$	product	$c[i][j]$ (after)
1	1	1	1	4	4	4
1	1	2	0	6	0	4
1	2	1	1	5	5	5
1	2	2	0	7	0	5
2	1	1	0	4	0	0
2	1	2	1	6	6	6
2	2	1	0	5	0	0
2	2	2	1	7	7	7
3	1	1	2	4	8	8
3	1	2	3	6	18	26
3	2	1	2	5	10	10
3	2	2	3	7	21	31

Return: $c = \begin{bmatrix} 4 & 5 \\ 6 & 7 \\ 26 & 31 \end{bmatrix}$

The first two rows of a form the identity, so the first two rows of c are equal to the two rows of b .

Sorting algorithms

The exercises of this chapter use the three algorithms of the chapter «Sorting algorithms» (Insertion Sort, Exchange Sort and Selection Sort) in their pseudo-code version, with arrays indexed from 1. The following conventions hold in all the solutions.

- In a trace table, each row shows the state of the array *after* the operation described in the last column. The current key of Insertion Sort is highlighted in **red**; the cells modified by a shift, an insertion or a swap are highlighted in **blue**.
- A **key comparison** is the evaluation of a comparison between two keys: $a[j] > key$ in Insertion Sort (evaluated only when $j \geq 1$), $a[i] > a[j]$ in Exchange Sort, $a[j] < a[\text{minIdx}]$ in Selection Sort.
- A **step** is the execution of one line of the pseudo-code; the evaluation of the condition of a **while** loop counts as one step each time it is evaluated, including the last one, which ends the loop.
- For every $n \in \mathbb{Z}_{\geq 1}$:

$$\sum_{k=1}^{n-1} k = 1 + 2 + \cdots + (n-1) = \frac{n(n-1)}{2}.$$

Exercise 3.1 — Sorting algorithms (E1)

Consider the arrays

$$a = [3, 1, 2] \quad \text{and} \quad b = [2, 1, 2]$$

and the rearrangements of an array, as in the definition of the sorting problem.

Questions

1. List all the rearrangements of a and of b . Which of them are sorted in non-decreasing order?
2. Explain why the output of the sorting problem is not just any array sorted in non-decreasing order: give an example of an array that is sorted in non-decreasing order but is not a correct output for a .

3. How many rearrangements has an array of n distinct elements? Compute the number for $n = 5$, $n = 10$ and $n = 20$.
4. Consider the algorithm **BruteForceSort** that generates the rearrangements of the array one by one and stops when it finds a sorted one. How many rearrangements may it examine in the worst case for an array of n distinct elements? Compare with the $\frac{n(n-1)}{2}$ shifts executed in the worst case by Insertion Sort.

Solution

1. The array $a = [3, 1, 2]$ has $3! = 6$ rearrangements:

$$[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1].$$

Only $[1, 2, 3]$ is sorted in non-decreasing order.

The array $b = [2, 1, 2]$ contains the value 2 twice, so exchanging the two 2's gives the same array. Its distinct rearrangements are only 3:

$$[1, 2, 2], [2, 1, 2], [2, 2, 1],$$

and only $[1, 2, 2]$ is sorted. In both cases the sorted rearrangement is **unique**: the output of the sorting problem is uniquely determined by the input.

2. The output must be a sorted **rearrangement of the input**. For example, the arrays $[1, 1, 1]$ and $[1, 2, 4]$ are sorted, but they are not a correct output for $a = [3, 1, 2]$, since they do not contain the same elements. An “algorithm” that overwrites all the entries with 0 produces a sorted array, but it does not solve the sorting problem. This is why, when we argue that a sorting algorithm is correct, we must also show that the final array contains the original elements.
3. An array of n distinct elements has

$$n! = n \cdot (n - 1) \cdots 2 \cdot 1$$

rearrangements: there are n choices for the first position, $n - 1$ for the second one, and so on. Hence

$$5! = 120, \quad 10! = 3\,628\,800, \quad 20! = 2\,432\,902\,008\,176\,640\,000 \approx 2.4 \cdot 10^{18}.$$

4. In the worst case the sorted rearrangement is the last one generated, so **BruteForceSort** may examine all the $n!$ rearrangements (and checking each of them requires up to $n - 1$ comparisons). Insertion Sort executes at most $\frac{n(n-1)}{2}$ shifts:

n	$n!$	$\frac{n(n-1)}{2}$
5	120	10
10	3 628 800	45
20	2 432 902 008 176 640 000	190

Since $n! \geq 2^{n-1}$ for every $n \geq 1$, the number of rearrangements examined by the brute-force approach grows at least exponentially with n , while the number of shifts of Insertion Sort grows only quadratically. Already for $n = 20$, even examining 10^9 rearrangements per second, **BruteForceSort** could need

$$\frac{2.43 \cdot 10^{18}}{10^9} \text{ s} \approx 2.43 \cdot 10^9 \text{ s} \approx 77 \text{ years.}$$

Exercise 3.2 — Sorting algorithms (E2)

Consider the problem of **checking** whether an array $a[1..n]$ of $n \geq 1$ natural numbers is sorted in non-decreasing order, and the following algorithm.

Algorithm: IsSorted

Input : An array $a[1..n]$ of n natural numbers

Output: **true** if $a[1] \leq a[2] \leq \dots \leq a[n]$, **false** otherwise

```

1 function IsSorted( $a, n$ )
2    $i := 1$ 
3   while  $i \leq n - 1$  do
4     if  $a[i] > a[i + 1]$  then
5       return false
6     end
7      $i \leftarrow i + 1$ 
8   end
9   return true
10 end
```

Questions

1. Apply **IsSorted** to the arrays $[1, 3, 3, 7, 9]$, $[2, 5, 4, 8]$, $[9, 1, 2, 3]$ and $[1]$. For each array give the output and the number of steps executed.
2. Is **IsSorted** a comparison-based algorithm? How many key comparisons does it perform in the worst case?
3. Let $S(n)$ be the number of steps executed by **IsSorted** in the worst case, for an array of n elements. Compute $S(n)$, and compare the number of key comparisons of **IsSorted** with the worst-case number of key comparisons of the sorting algorithms of the chapter.
4. Prove that **IsSorted** returns **true** if and only if the array $a[1..n]$ is sorted in non-decreasing order.

Solution

1. Executions:

- $[1, 3, 3, 7, 9]$ ($n = 5$): the comparisons $1 \leq 3$, $3 \leq 3$, $3 \leq 7$, $7 \leq 9$ are all satisfied;

the loop ends when $i = 5$ and the output is **true**. Steps: line 2 once, line 3 5 times, lines 4 and 7 4 times each, line 9 once:

$$1 + 5 + 4 + 4 + 1 = 15.$$

- $[2, 5, 4, 8]$ ($n = 4$): $a[1] = 2 \leq 5$; then $a[2] = 5 > a[3] = 4$, so the output is **false** at $i = 2$. Steps: line 2 once, line 3 twice, line 4 twice, line 7 once, line 5 once:

$$1 + 2 + 2 + 1 + 1 = 7.$$

- $[9, 1, 2, 3]$ ($n = 4$): $a[1] = 9 > a[2] = 1$, so the output is **false** at $i = 1$. Steps: lines 2, 3, 4, 5 once each:

$$1 + 1 + 1 + 1 = 4.$$

- $[1]$ ($n = 1$): the condition $1 \leq 0$ is false and the output is **true**. Steps: lines 2, 3, 9 once each:

$$1 + 1 + 1 = 3.$$

2. Yes: the algorithm accesses the keys only to compare two of them (line 4); it never modifies them and never performs arithmetic on them. In the worst case it performs $n - 1$ key comparisons, one for each pair of consecutive positions.

3. The worst case is when the algorithm never returns at line 5, i.e., when the array is sorted. The number of times each line is executed is:

- line 2 is executed 1 time;
- line 3 is executed n times;
- line 4 is executed $n - 1$ times;
- line 5 is not executed;
- line 7 is executed $n - 1$ times;
- line 9 is executed 1 time.

Therefore

$$S(n) = 1 + n + (n - 1) + (n - 1) + 1 = 3n,$$

in agreement with point 1 ($15 = 3 \cdot 5$ for the sorted array of 5 elements, $3 = 3 \cdot 1$ for $n = 1$).

Checking whether an array is sorted requires at most $n - 1$ key comparisons, a number that grows linearly with n , while Insertion Sort, Exchange Sort and Selection Sort perform $\frac{n(n-1)}{2}$ key comparisons in the worst case, a number that grows quadratically with n . For $n = 100$: 99 comparisons against 4950.

4. The algorithm returns **false** only at line 5, and only after finding an index i with $a[i] > a[i + 1]$: in this case the array is not sorted, since two consecutive elements are in the wrong order. Hence, if the array is sorted, the algorithm never returns **false**; since i increases by 1 at every iteration, the loop ends after $n - 1$ iterations and the algorithm returns **true** at line 9.

Conversely, if the algorithm returns **true**, the loop has been completed, so the comparison at line 4 was false for every $i \in \{1, 2, \dots, n-1\}$:

$$a[1] \leq a[2], \quad a[2] \leq a[3], \quad \dots, \quad a[n-1] \leq a[n].$$

Chaining these inequalities, $a[1] \leq a[2] \leq \dots \leq a[n]$: the array is sorted. Hence **IsSorted** returns **true** if and only if the array is sorted in non-decreasing order.

Exercise 3.3 — Sorting algorithms (E3)

Consider the array of records $(key, label)$, in which only the keys are compared,

$$a = [(3, A), (1, B), (3, C), (2, D), (1, E)],$$

and the algorithms Insertion Sort and Selection Sort of the chapter (Insertion Sort shifts $a[j]$ to the right while $a[j] > key$; Selection Sort swaps $a[i]$ with the minimum of $a[i..n]$, choosing the first minimum found).

Questions

1. Apply Insertion Sort to a , showing the array after each iteration of the outer loop. Is the result stable?
2. Apply Selection Sort to a , showing the array after each iteration of the outer loop. Is the result stable?
3. A student replaces the condition $a[j] > key$ of Insertion Sort with $a[j] \geq key$. Is the new algorithm still correct? Is it stable?
4. Are Insertion Sort and Selection Sort in-place? Describe a sorting method that is not in-place.

Solution

1. Insertion Sort (the inserted key is highlighted in **red**):

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$
initial	(3, A)	(1, B)	(3, C)	(2, D)	(1, E)
2	(1, B)	(3, A)	(3, C)	(2, D)	(1, E)
3	(1, B)	(3, A)	(3, C)	(2, D)	(1, E)
4	(1, B)	(2, D)	(3, A)	(3, C)	(1, E)
5	(1, B)	(1, E)	(2, D)	(3, A)	(3, C)

For $i = 3$, $a[2] = (3, A)$ is not > 3 , so $(3, C)$ stays after $(3, A)$; for $i = 5$, the shifts stop at $(1, B)$, since $1 > 1$ is false. In the output B precedes E and A precedes C , as in the input: the result is **stable**. In general, Insertion Sort is stable, because a key is never moved before an equal key (the shift condition is a strict inequality).

2. Selection Sort (the two swapped positions are highlighted in **blue**):

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$minIdx$
initial	(3, A)	(1, B)	(3, C)	(2, D)	(1, E)	
1	(1, B)	(3, A)	(3, C)	(2, D)	(1, E)	2
2	(1, B)	(1, E)	(3, C)	(2, D)	(3, A)	5
3	(1, B)	(1, E)	(2, D)	(3, C)	(3, A)	4
4	(1, B)	(1, E)	(2, D)	(3, C)	(3, A)	4 (no swap)

The keys are sorted, but (3, C) precedes (3, A), while in the input A preceded C: the result is **not stable**. The swap at $i = 2$ moved (3, A) from position 2 to position 5, “jumping over” (3, C). Hence Selection Sort is not stable (an even smaller counterexample is [(2, A), (2, B), (1, C)], which becomes [(1, C), (2, B), (2, A)]).

3. With $a[j] \geq key$ the algorithm is still **correct**: the keys are still sorted, because key is inserted after all the elements $< key$ and before all the elements $\geq key$. However, it is **not stable**, since key is moved before the equal keys. On a :

$$[(3, A), (1, B), (3, C), (2, D), (1, E)] \longrightarrow [(1, E), (1, B), (2, D), (3, C), (3, A)],$$

where both pairs of equal keys are inverted. Moreover, it executes more shifts than necessary when there are equal keys.

4. Both algorithms are **in-place**: they rearrange the elements inside the array a , and they use only a constant number of auxiliary variables (i, j, key for Insertion Sort; $i, j, minIdx, tmp$ for Selection Sort), independently of n .

A method that is **not** in-place: create a new array $b[1..n]$; for $k \in \{1, 2, \dots, n\}$, find the minimum among the elements of a not yet copied, copy it into $b[k]$ and mark it as used (for example with an array of n boolean flags); finally return b . This method uses extra memory proportional to n (the array b and the flags), so it is not in-place.

Exercise 3.4 — Sorting algorithms (E4)

Consider the following arrays of natural numbers:

1. $a = [5, 2, 4, 6, 1, 3]$
2. $a = [8, 3, 7, 4, 9, 2]$
3. $a = [6, 5, 4, 3, 2, 1]$
4. $a = [1, 2, 3, 4, 5, 6]$
5. $a = [10, 5, 15, 3]$

Questions

1. Apply the **InsertionSort** algorithm to each array, to sort it in non-decreasing order. Show the execution step by step: for every iteration of the outer loop, give the key being inserted, every shift performed by the inner loop and the state of the array after the insertion.

2. For each array, count the key comparisons $a[j] > key$ and the shifts performed by the algorithm.

Solution

The trace tables follow the conventions stated in the introduction to the exercises of this chapter. Each item answers both questions.

1. Array $a = [5, 2, 4, 6, 1, 3]$ ($n = 6$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	—	—	5	2	4	6	1	3	initial state
2	1	2	5	2	4	6	1	3	$key \leftarrow a[2] = 2; j \leftarrow 1$
	0	2	5	5	4	6	1	3	$a[1] = 5 > 2$: shift $a[2] \leftarrow 5; j \leftarrow 0$
	—	2	2	5	4	6	1	3	$j = 0$: exit; insert $a[1] \leftarrow 2$
3	2	4	2	5	4	6	1	3	$key \leftarrow a[3] = 4; j \leftarrow 2$
	1	4	2	5	5	6	1	3	$a[2] = 5 > 4$: shift $a[3] \leftarrow 5; j \leftarrow 1$
	—	4	2	4	5	6	1	3	$a[1] = 2 \leq 4$: exit; insert $a[2] \leftarrow 4$
4	3	6	2	4	5	6	1	3	$key \leftarrow a[4] = 6; j \leftarrow 3$
	—	6	2	4	5	6	1	3	$a[3] = 5 \leq 6$: exit; insert $a[4] \leftarrow 6$
5	4	1	2	4	5	6	1	3	$key \leftarrow a[5] = 1; j \leftarrow 4$
	3	1	2	4	5	6	6	3	$a[4] = 6 > 1$: shift $a[5] \leftarrow 6; j \leftarrow 3$
	2	1	2	4	5	5	6	3	$a[3] = 5 > 1$: shift $a[4] \leftarrow 5; j \leftarrow 2$
	1	1	2	4	4	5	6	3	$a[2] = 4 > 1$: shift $a[3] \leftarrow 4; j \leftarrow 1$
	0	1	2	2	4	5	6	3	$a[1] = 2 > 1$: shift $a[2] \leftarrow 2; j \leftarrow 0$
	—	1	1	2	4	5	6	3	$j = 0$: exit; insert $a[1] \leftarrow 1$
6	5	3	1	2	4	5	6	3	$key \leftarrow a[6] = 3; j \leftarrow 5$
	4	3	1	2	4	5	6	6	$a[5] = 6 > 3$: shift $a[6] \leftarrow 6; j \leftarrow 4$
	3	3	1	2	4	5	5	6	$a[4] = 5 > 3$: shift $a[5] \leftarrow 5; j \leftarrow 3$
	2	3	1	2	4	4	5	6	$a[3] = 4 > 3$: shift $a[4] \leftarrow 4; j \leftarrow 2$
	—	3	1	2	3	4	5	6	$a[2] = 2 \leq 3$: exit; insert $a[3] \leftarrow 3$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	—	5	2	4	6	1	3	initial state
2	2	2	5	4	6	1	3	insert 2 at the beginning
3	4	2	4	5	6	1	3	insert 4 between 2 and 5
4	6	2	4	5	6	1	3	6 already in position
5	1	1	2	4	5	6	3	insert 1 at the beginning
6	3	1	2	3	4	5	6	insert 3 between 2 and 4

The algorithm performs 12 key comparisons and 9 shifts, and the sorted array is

$[1, 2, 3, 4, 5, 6]$.

2. Array $a = [8, 3, 7, 4, 9, 2]$ ($n = 6$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	—	—	8	3	7	4	9	2	initial state
2	1	3	8	3	7	4	9	2	$key \leftarrow a[2] = 3; j \leftarrow 1$
	0	3	8	8	7	4	9	2	$a[1] = 8 > 3$: shift $a[2] \leftarrow 8; j \leftarrow 0$
	—	3	3	8	7	4	9	2	$j = 0$: exit; insert $a[1] \leftarrow 3$
3	2	7	3	8	7	4	9	2	$key \leftarrow a[3] = 7; j \leftarrow 2$
	1	7	3	8	8	4	9	2	$a[2] = 8 > 7$: shift $a[3] \leftarrow 8; j \leftarrow 1$
	—	7	3	7	8	4	9	2	$a[1] = 3 \leq 7$: exit; insert $a[2] \leftarrow 7$
4	3	4	3	7	8	4	9	2	$key \leftarrow a[4] = 4; j \leftarrow 3$
	2	4	3	7	8	8	9	2	$a[3] = 8 > 4$: shift $a[4] \leftarrow 8; j \leftarrow 2$
	1	4	3	7	7	8	9	2	$a[2] = 7 > 4$: shift $a[3] \leftarrow 7; j \leftarrow 1$
	—	4	3	4	7	8	9	2	$a[1] = 3 \leq 4$: exit; insert $a[2] \leftarrow 4$
5	4	9	3	4	7	8	9	2	$key \leftarrow a[5] = 9; j \leftarrow 4$
	—	9	3	4	7	8	9	2	$a[4] = 8 \leq 9$: exit; insert $a[5] \leftarrow 9$
6	5	2	3	4	7	8	9	2	$key \leftarrow a[6] = 2; j \leftarrow 5$
	4	2	3	4	7	8	9	9	$a[5] = 9 > 2$: shift $a[6] \leftarrow 9; j \leftarrow 4$
	3	2	3	4	7	8	8	9	$a[4] = 8 > 2$: shift $a[5] \leftarrow 8; j \leftarrow 3$
	2	2	3	4	7	7	8	9	$a[3] = 7 > 2$: shift $a[4] \leftarrow 7; j \leftarrow 2$
	1	2	3	4	4	7	8	9	$a[2] = 4 > 2$: shift $a[3] \leftarrow 4; j \leftarrow 1$
	0	2	3	3	4	7	8	9	$a[1] = 3 > 2$: shift $a[2] \leftarrow 3; j \leftarrow 0$
	—	2	2	3	4	7	8	9	$j = 0$: exit; insert $a[1] \leftarrow 2$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	—	8	3	7	4	9	2	initial state
2	3	3	8	7	4	9	2	insert 3 at the beginning
3	7	3	7	8	4	9	2	insert 7 between 3 and 8
4	4	3	4	7	8	9	2	insert 4 between 3 and 7
5	9	3	4	7	8	9	2	9 already in position
6	2	2	3	4	7	8	9	insert 2 at the beginning

The algorithm performs 12 key comparisons and 9 shifts, and the sorted array is

$$[2, 3, 4, 7, 8, 9].$$

3. Array $a = [6, 5, 4, 3, 2, 1]$ ($n = 6$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	—	—	6	5	4	3	2	1	initial state
2	1	5	6	5	4	3	2	1	$key \leftarrow a[2] = 5; j \leftarrow 1$
	0	5	6	6	4	3	2	1	$a[1] = 6 > 5$: shift $a[2] \leftarrow 6; j \leftarrow 0$
	—	5	5	6	4	3	2	1	$j = 0$: exit; insert $a[1] \leftarrow 5$
3	2	4	5	6	4	3	2	1	$key \leftarrow a[3] = 4; j \leftarrow 2$
	1	4	5	6	6	3	2	1	$a[2] = 6 > 4$: shift $a[3] \leftarrow 6; j \leftarrow 1$
	0	4	5	5	6	3	2	1	$a[1] = 5 > 4$: shift $a[2] \leftarrow 5; j \leftarrow 0$
	—	4	4	5	6	3	2	1	$j = 0$: exit; insert $a[1] \leftarrow 4$
4	3	3	4	5	6	3	2	1	$key \leftarrow a[4] = 3; j \leftarrow 3$
	2	3	4	5	6	6	2	1	$a[3] = 6 > 3$: shift $a[4] \leftarrow 6; j \leftarrow 2$
	1	3	4	5	5	6	2	1	$a[2] = 5 > 3$: shift $a[3] \leftarrow 5; j \leftarrow 1$
	0	3	4	4	5	6	2	1	$a[1] = 4 > 3$: shift $a[2] \leftarrow 4; j \leftarrow 0$
	—	3	3	4	5	6	2	1	$j = 0$: exit; insert $a[1] \leftarrow 3$
5	4	2	3	4	5	6	2	1	$key \leftarrow a[5] = 2; j \leftarrow 4$
	3	2	3	4	5	6	6	1	$a[4] = 6 > 2$: shift $a[5] \leftarrow 6; j \leftarrow 3$
	2	2	3	4	5	5	6	1	$a[3] = 5 > 2$: shift $a[4] \leftarrow 5; j \leftarrow 2$
	1	2	3	4	4	5	6	1	$a[2] = 4 > 2$: shift $a[3] \leftarrow 4; j \leftarrow 1$
	0	2	3	3	4	5	6	1	$a[1] = 3 > 2$: shift $a[2] \leftarrow 3; j \leftarrow 0$
	—	2	2	3	4	5	6	1	$j = 0$: exit; insert $a[1] \leftarrow 2$
6	5	1	2	3	4	5	6	1	$key \leftarrow a[6] = 1; j \leftarrow 5$
	4	1	2	3	4	5	6	6	$a[5] = 6 > 1$: shift $a[6] \leftarrow 6; j \leftarrow 4$
	3	1	2	3	4	5	5	6	$a[4] = 5 > 1$: shift $a[5] \leftarrow 5; j \leftarrow 3$
	2	1	2	3	4	4	5	6	$a[3] = 4 > 1$: shift $a[4] \leftarrow 4; j \leftarrow 2$
	1	1	2	3	3	4	5	6	$a[2] = 3 > 1$: shift $a[3] \leftarrow 3; j \leftarrow 1$
	0	1	2	2	3	4	5	6	$a[1] = 2 > 1$: shift $a[2] \leftarrow 2; j \leftarrow 0$
	—	1	1	2	3	4	5	6	$j = 0$: exit; insert $a[1] \leftarrow 1$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	—	6	5	4	3	2	1	initial state
2	5	5	6	4	3	2	1	insert 5 at the beginning
3	4	4	5	6	3	2	1	insert 4 at the beginning
4	3	3	4	5	6	2	1	insert 3 at the beginning
5	2	2	3	4	5	6	1	insert 2 at the beginning
6	1	1	2	3	4	5	6	insert 1 at the beginning

The algorithm performs 15 key comparisons and 15 shifts, and the sorted array is

$$[1, 2, 3, 4, 5, 6].$$

4. Array $a = [1, 2, 3, 4, 5, 6]$ ($n = 6$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	—	—	1	2	3	4	5	6	initial state
2	1	2	1	2	3	4	5	6	$key \leftarrow a[2] = 2; j \leftarrow 1$
—	—	2	1	2	3	4	5	6	$a[1] = 1 \leq 2$: exit; insert $a[2] \leftarrow 2$
3	2	3	1	2	3	4	5	6	$key \leftarrow a[3] = 3; j \leftarrow 2$
—	—	3	1	2	3	4	5	6	$a[2] = 2 \leq 3$: exit; insert $a[3] \leftarrow 3$
4	3	4	1	2	3	4	5	6	$key \leftarrow a[4] = 4; j \leftarrow 3$
—	—	4	1	2	3	4	5	6	$a[3] = 3 \leq 4$: exit; insert $a[4] \leftarrow 4$
5	4	5	1	2	3	4	5	6	$key \leftarrow a[5] = 5; j \leftarrow 4$
—	—	5	1	2	3	4	5	6	$a[4] = 4 \leq 5$: exit; insert $a[5] \leftarrow 5$
6	5	6	1	2	3	4	5	6	$key \leftarrow a[6] = 6; j \leftarrow 5$
—	—	6	1	2	3	4	5	6	$a[5] = 5 \leq 6$: exit; insert $a[6] \leftarrow 6$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	—	1	2	3	4	5	6	initial state
2	2	1	2	3	4	5	6	2 already in position
3	3	1	2	3	4	5	6	3 already in position
4	4	1	2	3	4	5	6	4 already in position
5	5	1	2	3	4	5	6	5 already in position
6	6	1	2	3	4	5	6	6 already in position

The algorithm performs 5 key comparisons and 0 shifts, and the sorted array is

$$[1, 2, 3, 4, 5, 6].$$

5. Array $a = [10, 5, 15, 3]$ ($n = 4$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	—	10	5	15	3	initial state
2	1	5	10	5	15	3	$key \leftarrow a[2] = 5; j \leftarrow 1$
0	0	5	10	10	15	3	$a[1] = 10 > 5$: shift $a[2] \leftarrow 10; j \leftarrow 0$
—	—	5	5	10	15	3	$j = 0$: exit; insert $a[1] \leftarrow 5$
3	2	15	5	10	15	3	$key \leftarrow a[3] = 15; j \leftarrow 2$
—	—	15	5	10	15	3	$a[2] = 10 \leq 15$: exit; insert $a[3] \leftarrow 15$
4	3	3	5	10	15	3	$key \leftarrow a[4] = 3; j \leftarrow 3$
2	2	3	5	10	15	15	$a[3] = 15 > 3$: shift $a[4] \leftarrow 15; j \leftarrow 2$
1	1	3	5	10	10	15	$a[2] = 10 > 3$: shift $a[3] \leftarrow 10; j \leftarrow 1$
0	0	3	5	5	10	15	$a[1] = 5 > 3$: shift $a[2] \leftarrow 5; j \leftarrow 0$
—	—	3	3	5	10	15	$j = 0$: exit; insert $a[1] \leftarrow 3$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
–	–	10	5	15	3	initial state
2	5	5	10	15	3	insert 5 at the beginning
3	15	5	10	15	3	15 already in position
4	3	3	5	10	15	insert 3 at the beginning

The algorithm performs 5 key comparisons and 4 shifts, and the sorted array is

[3, 5, 10, 15].

Exercise 3.5 — Sorting algorithms (E5)

Consider the following arrays of natural numbers:

1. $a = [7, 2, 9, 4]$
2. $a = [15, 10, 5]$
3. $a = [3, 1, 4, 1, 5, 9, 2, 6]$
4. $a = [20, 18, 16, 14]$
5. $a = [11, 11, 11]$

Questions

1. Apply the **InsertionSort** algorithm to each array, to sort it in non-decreasing order. Show the execution step by step: for every iteration of the outer loop, give the key being inserted, every shift performed by the inner loop and the state of the array after the insertion.
2. For each array, count the key comparisons $a[j] > key$ and the shifts performed by the algorithm.

Solution

The trace tables follow the conventions stated in the introduction to the exercises of this chapter. Each item answers both questions.

1. Array $a = [7, 2, 9, 4]$ ($n = 4$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	—	7	2	9	4	initial state
2	1	2	7	2	9	4	$key \leftarrow a[2] = 2; j \leftarrow 1$
	0	2	7	7	9	4	$a[1] = 7 > 2$: shift $a[2] \leftarrow 7; j \leftarrow 0$
	—	2	2	7	9	4	$j = 0$: exit; insert $a[1] \leftarrow 2$
3	2	9	2	7	9	4	$key \leftarrow a[3] = 9; j \leftarrow 2$
	—	9	2	7	9	4	$a[2] = 7 \leq 9$: exit; insert $a[3] \leftarrow 9$
4	3	4	2	7	9	4	$key \leftarrow a[4] = 4; j \leftarrow 3$
	2	4	2	7	9	9	$a[3] = 9 > 4$: shift $a[4] \leftarrow 9; j \leftarrow 2$
	1	4	2	7	7	9	$a[2] = 7 > 4$: shift $a[3] \leftarrow 7; j \leftarrow 1$
	—	4	2	4	7	9	$a[1] = 2 \leq 4$: exit; insert $a[2] \leftarrow 4$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	7	2	9	4	initial state
2	2	2	7	9	4	insert 2 at the beginning
3	9	2	7	9	4	9 already in position
4	4	2	4	7	9	insert 4 between 2 and 7

The algorithm performs 5 key comparisons and 3 shifts, and the sorted array is

$$[2, 4, 7, 9].$$

2. Array $a = [15, 10, 5]$ ($n = 3$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	Action
—	—	—	15	10	5	initial state
2	1	10	15	10	5	$key \leftarrow a[2] = 10; j \leftarrow 1$
	0	10	15	15	5	$a[1] = 15 > 10$: shift $a[2] \leftarrow 15; j \leftarrow 0$
	—	10	10	15	5	$j = 0$: exit; insert $a[1] \leftarrow 10$
3	2	5	10	15	5	$key \leftarrow a[3] = 5; j \leftarrow 2$
	1	5	10	15	15	$a[2] = 15 > 5$: shift $a[3] \leftarrow 15; j \leftarrow 1$
	0	5	10	10	15	$a[1] = 10 > 5$: shift $a[2] \leftarrow 10; j \leftarrow 0$
	—	5	5	10	15	$j = 0$: exit; insert $a[1] \leftarrow 5$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	Action
—	—	15	10	5	initial state
2	10	10	15	5	insert 10 at the beginning
3	5	5	10	15	insert 5 at the beginning

The algorithm performs 3 key comparisons and 3 shifts, and the sorted array is

[5, 10, 15].

3. Array $a = [3, 1, 4, 1, 5, 9, 2, 6]$ ($n = 8$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	$a[7]$	$a[8]$	Action
—	—	—	3	1	4	1	5	9	2	6	initial state
2	1	1	3	1	4	1	5	9	2	6	$key \leftarrow a[2] = 1; j \leftarrow 1$
	0	1	3	3	4	1	5	9	2	6	$a[1] = 3 > 1$: shift $a[2] \leftarrow 3; j \leftarrow 0$
	—	1	1	3	4	1	5	9	2	6	$j = 0$: exit; insert $a[1] \leftarrow 1$
3	2	4	1	3	4	1	5	9	2	6	$key \leftarrow a[3] = 4; j \leftarrow 2$
	—	4	1	3	4	1	5	9	2	6	$a[2] = 3 \leq 4$: exit; insert $a[3] \leftarrow 4$
4	3	1	1	3	4	1	5	9	2	6	$key \leftarrow a[4] = 1; j \leftarrow 3$
	2	1	1	3	4	4	5	9	2	6	$a[3] = 4 > 1$: shift $a[4] \leftarrow 4; j \leftarrow 2$
	1	1	1	3	3	4	5	9	2	6	$a[2] = 3 > 1$: shift $a[3] \leftarrow 3; j \leftarrow 1$
	—	1	1	1	3	4	5	9	2	6	$a[1] = 1 \leq 1$: exit; insert $a[2] \leftarrow 1$
5	4	5	1	1	3	4	5	9	2	6	$key \leftarrow a[5] = 5; j \leftarrow 4$
	—	5	1	1	3	4	5	9	2	6	$a[4] = 4 \leq 5$: exit; insert $a[5] \leftarrow 5$
6	5	9	1	1	3	4	5	9	2	6	$key \leftarrow a[6] = 9; j \leftarrow 5$
	—	9	1	1	3	4	5	9	2	6	$a[5] = 5 \leq 9$: exit; insert $a[6] \leftarrow 9$
	6	2	1	1	3	4	5	9	2	6	$key \leftarrow a[7] = 2; j \leftarrow 6$
	5	2	1	1	3	4	5	9	9	6	$a[6] = 9 > 2$: shift $a[7] \leftarrow 9; j \leftarrow 5$
	4	2	1	1	3	4	5	5	9	6	$a[5] = 5 > 2$: shift $a[6] \leftarrow 5; j \leftarrow 4$
	3	2	1	1	3	4	4	5	9	6	$a[4] = 4 > 2$: shift $a[5] \leftarrow 4; j \leftarrow 3$
7	2	2	1	1	3	3	4	5	9	6	$a[3] = 3 > 2$: shift $a[4] \leftarrow 3; j \leftarrow 2$
	—	2	1	1	2	3	4	5	9	6	$a[2] = 1 \leq 2$: exit; insert $a[3] \leftarrow 2$
	8	6	1	1	2	3	4	5	9	6	$key \leftarrow a[8] = 6; j \leftarrow 7$
8	6	6	1	1	2	3	4	5	9	9	$a[7] = 9 > 6$: shift $a[8] \leftarrow 9; j \leftarrow 6$
	—	6	1	1	2	3	4	5	6	9	$a[6] = 5 \leq 6$: exit; insert $a[7] \leftarrow 6$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	$a[7]$	$a[8]$	Action
—	—	3	1	4	1	5	9	2	6	initial state
2	1	1	3	4	1	5	9	2	6	insert 1 at the beginning
3	4	1	3	4	1	5	9	2	6	4 already in position
4	1	1	1	3	4	5	9	2	6	insert 1 between 1 and 3
5	5	1	1	3	4	5	9	2	6	5 already in position
6	9	1	1	3	4	5	9	2	6	9 already in position
7	2	1	1	2	3	4	5	9	6	insert 2 between 1 and 3
8	6	1	1	2	3	4	5	6	9	insert 6 between 5 and 9

The algorithm performs 14 key comparisons and 8 shifts, and the sorted array is

[1, 1, 2, 3, 4, 5, 6, 9].

4. Array $a = [20, 18, 16, 14]$ ($n = 4$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
–	–	–	20	18	16	14	initial state
2	1	18	20	18	16	14	$key \leftarrow a[2] = 18; j \leftarrow 1$
	0	18	20	20	16	14	$a[1] = 20 > 18$: shift $a[2] \leftarrow 20; j \leftarrow 0$
	–	18	18	20	16	14	$j = 0$: exit; insert $a[1] \leftarrow 18$
3	2	16	18	20	16	14	$key \leftarrow a[3] = 16; j \leftarrow 2$
	1	16	18	20	20	14	$a[2] = 20 > 16$: shift $a[3] \leftarrow 20; j \leftarrow 1$
	0	16	18	18	20	14	$a[1] = 18 > 16$: shift $a[2] \leftarrow 18; j \leftarrow 0$
	–	16	16	18	20	14	$j = 0$: exit; insert $a[1] \leftarrow 16$
4	3	14	16	18	20	14	$key \leftarrow a[4] = 14; j \leftarrow 3$
	2	14	16	18	20	20	$a[3] = 20 > 14$: shift $a[4] \leftarrow 20; j \leftarrow 2$
	1	14	16	18	18	20	$a[2] = 18 > 14$: shift $a[3] \leftarrow 18; j \leftarrow 1$
	0	14	16	16	18	20	$a[1] = 16 > 14$: shift $a[2] \leftarrow 16; j \leftarrow 0$
	–	14	14	16	18	20	$j = 0$: exit; insert $a[1] \leftarrow 14$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
–	–	20	18	16	14	initial state
2	18	18	20	16	14	insert 18 at the beginning
3	16	16	18	20	14	insert 16 at the beginning
4	14	14	16	18	20	insert 14 at the beginning

The algorithm performs 6 key comparisons and 6 shifts, and the sorted array is

$[14, 16, 18, 20]$.

5. Array $a = [11, 11, 11]$ ($n = 3$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	Action
–	–	–	11	11	11	initial state
2	1	11	11	11	11	$key \leftarrow a[2] = 11; j \leftarrow 1$
	–	11	11	11	11	$a[1] = 11 \leq 11$: exit; insert $a[2] \leftarrow 11$
3	2	11	11	11	11	$key \leftarrow a[3] = 11; j \leftarrow 2$
	–	11	11	11	11	$a[2] = 11 \leq 11$: exit; insert $a[3] \leftarrow 11$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	Action
–	–	11	11	11	initial state
2	11	11	11	11	11 already in position
3	11	11	11	11	11 already in position

The algorithm performs 2 key comparisons and 0 shifts, and the sorted array is

[11, 11, 11].

Exercise 3.6 — Sorting algorithms (E6)

Consider the following arrays of natural numbers:

1. $a = [9, 3, 7, 2]$
2. $a = [12, 4, 18, 6, 10]$
3. $a = [5, 4, 3, 2, 1]$
4. $a = [100, 50, 25]$
5. $a = [8]$

Questions

1. Apply the **InsertionSort** algorithm to each array, to sort it in non-decreasing order. Show the execution step by step: for every iteration of the outer loop, give the key being inserted, every shift performed by the inner loop and the state of the array after the insertion.
2. For each array, count the key comparisons $a[j] > key$ and the shifts performed by the algorithm.

Solution

The trace tables follow the conventions stated in the introduction to the exercises of this chapter. Each item answers both questions.

1. Array $a = [9, 3, 7, 2]$ ($n = 4$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	—	9	3	7	2	initial state
2	1	3	9	3	7	2	$key \leftarrow a[2] = 3; j \leftarrow 1$
	0	3	9	9	7	2	$a[1] = 9 > 3$: shift $a[2] \leftarrow 9; j \leftarrow 0$
	—	3	3	9	7	2	$j = 0$: exit; insert $a[1] \leftarrow 3$
3	2	7	3	9	7	2	$key \leftarrow a[3] = 7; j \leftarrow 2$
	1	7	3	9	9	2	$a[2] = 9 > 7$: shift $a[3] \leftarrow 9; j \leftarrow 1$
	—	7	3	7	9	2	$a[1] = 3 \leq 7$: exit; insert $a[2] \leftarrow 7$
4	3	2	3	7	9	2	$key \leftarrow a[4] = 2; j \leftarrow 3$
	2	2	3	7	9	9	$a[3] = 9 > 2$: shift $a[4] \leftarrow 9; j \leftarrow 2$
	1	2	3	7	7	9	$a[2] = 7 > 2$: shift $a[3] \leftarrow 7; j \leftarrow 1$
	0	2	3	3	7	9	$a[1] = 3 > 2$: shift $a[2] \leftarrow 3; j \leftarrow 0$
	—	2	2	3	7	9	$j = 0$: exit; insert $a[1] \leftarrow 2$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
–	–	9	3	7	2	initial state
2	3	3	9	7	2	insert 3 at the beginning
3	7	3	7	9	2	insert 7 between 3 and 9
4	2	2	3	7	9	insert 2 at the beginning

The algorithm performs 6 key comparisons and 5 shifts, and the sorted array is

$[2, 3, 7, 9]$.

2. Array $a = [12, 4, 18, 6, 10]$ ($n = 5$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
–	–	–	12	4	18	6	10	initial state
2	1	4	12	4	18	6	10	$key \leftarrow a[2] = 4; j \leftarrow 1$
	0	4	12	12	18	6	10	$a[1] = 12 > 4$: shift $a[2] \leftarrow 12; j \leftarrow 0$
	–	4	4	12	18	6	10	$j = 0$: exit; insert $a[1] \leftarrow 4$
3	2	18	4	12	18	6	10	$key \leftarrow a[3] = 18; j \leftarrow 2$
	–	18	4	12	18	6	10	$a[2] = 12 \leq 18$: exit; insert $a[3] \leftarrow 18$
4	3	6	4	12	18	6	10	$key \leftarrow a[4] = 6; j \leftarrow 3$
	2	6	4	12	18	18	10	$a[3] = 18 > 6$: shift $a[4] \leftarrow 18; j \leftarrow 2$
	1	6	4	12	12	18	10	$a[2] = 12 > 6$: shift $a[3] \leftarrow 12; j \leftarrow 1$
	–	6	4	6	12	18	10	$a[1] = 4 \leq 6$: exit; insert $a[2] \leftarrow 6$
5	4	10	4	6	12	18	10	$key \leftarrow a[5] = 10; j \leftarrow 4$
	3	10	4	6	12	18	18	$a[4] = 18 > 10$: shift $a[5] \leftarrow 18; j \leftarrow 3$
	2	10	4	6	12	12	18	$a[3] = 12 > 10$: shift $a[4] \leftarrow 12; j \leftarrow 2$
	–	10	4	6	10	12	18	$a[2] = 6 \leq 10$: exit; insert $a[3] \leftarrow 10$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
–	–	12	4	18	6	10	initial state
2	4	4	12	18	6	10	insert 4 at the beginning
3	18	4	12	18	6	10	18 already in position
4	6	4	6	12	18	10	insert 6 between 4 and 12
5	10	4	6	10	12	18	insert 10 between 6 and 12

The algorithm performs 8 key comparisons and 5 shifts, and the sorted array is

$[4, 6, 10, 12, 18]$.

3. Array $a = [5, 4, 3, 2, 1]$ ($n = 5$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
—	—	—	5	4	3	2	1	initial state
2	1	4	5	4	3	2	1	$key \leftarrow a[2] = 4; j \leftarrow 1$
	0	4	5	5	3	2	1	$a[1] = 5 > 4$: shift $a[2] \leftarrow 5; j \leftarrow 0$
	—	4	4	5	3	2	1	$j = 0$: exit; insert $a[1] \leftarrow 4$
3	2	3	4	5	3	2	1	$key \leftarrow a[3] = 3; j \leftarrow 2$
	1	3	4	5	5	2	1	$a[2] = 5 > 3$: shift $a[3] \leftarrow 5; j \leftarrow 1$
	0	3	4	4	5	2	1	$a[1] = 4 > 3$: shift $a[2] \leftarrow 4; j \leftarrow 0$
	—	3	3	4	5	2	1	$j = 0$: exit; insert $a[1] \leftarrow 3$
4	3	2	3	4	5	2	1	$key \leftarrow a[4] = 2; j \leftarrow 3$
	2	2	3	4	5	5	1	$a[3] = 5 > 2$: shift $a[4] \leftarrow 5; j \leftarrow 2$
	1	2	3	4	4	5	1	$a[2] = 4 > 2$: shift $a[3] \leftarrow 4; j \leftarrow 1$
	0	2	3	3	4	5	1	$a[1] = 3 > 2$: shift $a[2] \leftarrow 3; j \leftarrow 0$
	—	2	2	3	4	5	1	$j = 0$: exit; insert $a[1] \leftarrow 2$
5	4	1	2	3	4	5	1	$key \leftarrow a[5] = 1; j \leftarrow 4$
	3	1	2	3	4	5	5	$a[4] = 5 > 1$: shift $a[5] \leftarrow 5; j \leftarrow 3$
	2	1	2	3	4	4	5	$a[3] = 4 > 1$: shift $a[4] \leftarrow 4; j \leftarrow 2$
	1	1	2	3	3	4	5	$a[2] = 3 > 1$: shift $a[3] \leftarrow 3; j \leftarrow 1$
	0	1	2	2	3	4	5	$a[1] = 2 > 1$: shift $a[2] \leftarrow 2; j \leftarrow 0$
	—	1	1	2	3	4	5	$j = 0$: exit; insert $a[1] \leftarrow 1$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
—	—	5	4	3	2	1	initial state
2	4	4	5	3	2	1	insert 4 at the beginning
3	3	3	4	5	2	1	insert 3 at the beginning
4	2	2	3	4	5	1	insert 2 at the beginning
5	1	1	2	3	4	5	insert 1 at the beginning

The algorithm performs 10 key comparisons and 10 shifts, and the sorted array is

$$[1, 2, 3, 4, 5].$$

4. Array $a = [100, 50, 25]$ ($n = 3$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	Action
—	—	—	100	50	25	initial state
2	1	50	100	50	25	$key \leftarrow a[2] = 50; j \leftarrow 1$
	0	50	100	100	25	$a[1] = 100 > 50$: shift $a[2] \leftarrow 100; j \leftarrow 0$
	—	50	50	100	25	$j = 0$: exit; insert $a[1] \leftarrow 50$
3	2	25	50	100	25	$key \leftarrow a[3] = 25; j \leftarrow 2$
	1	25	50	100	100	$a[2] = 100 > 25$: shift $a[3] \leftarrow 100; j \leftarrow 1$
	0	25	50	50	100	$a[1] = 50 > 25$: shift $a[2] \leftarrow 50; j \leftarrow 0$
	—	25	25	50	100	$j = 0$: exit; insert $a[1] \leftarrow 25$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	Action
—	—	100	50	25	initial state
2	50	50	100	25	insert 50 at the beginning
3	25	25	50	100	insert 25 at the beginning

The algorithm performs 3 key comparisons and 3 shifts, and the sorted array is

[25, 50, 100].

5. Array $a = [8]$ ($n = 1$).

The array has a single element: the outer loop is not executed and the array is already sorted.

$a[1]$	Action
8	initial state: $n = 1$, the outer loop is not executed

The sorted array is [8], obtained with no key comparison and no shift.

Exercise 3.7 — Sorting algorithms (E7)

Consider the following arrays of natural numbers:

1. $a = [6, 1, 8, 2, 5]$
2. $a = [13, 11, 9, 7]$
3. $a = [2, 8, 4, 6]$
4. $a = [30, 20, 10]$
5. $a = [5, 10, 15, 20]$

Questions

1. Apply the **InsertionSort** algorithm to each array, to sort it in non-decreasing order. Show the execution step by step: for every iteration of the outer loop, give the key being inserted, every shift performed by the inner loop and the state of the array after the insertion.
2. For each array, count the key comparisons $a[j] > key$ and the shifts performed by the algorithm.

Solution

The trace tables follow the conventions stated in the introduction to the exercises of this chapter. Each item answers both questions.

1. Array $a = [6, 1, 8, 2, 5]$ ($n = 5$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
–	–	–	6	1	8	2	5	initial state
2	1	1	6	1	8	2	5	$key \leftarrow a[2] = 1; j \leftarrow 1$
	0	1	6	6	8	2	5	$a[1] = 6 > 1$: shift $a[2] \leftarrow 6; j \leftarrow 0$
	–	1	1	6	8	2	5	$j = 0$: exit; insert $a[1] \leftarrow 1$
3	2	8	1	6	8	2	5	$key \leftarrow a[3] = 8; j \leftarrow 2$
	–	8	1	6	8	2	5	$a[2] = 6 \leq 8$: exit; insert $a[3] \leftarrow 8$
4	3	2	1	6	8	2	5	$key \leftarrow a[4] = 2; j \leftarrow 3$
	2	2	1	6	8	8	5	$a[3] = 8 > 2$: shift $a[4] \leftarrow 8; j \leftarrow 2$
	1	2	1	6	6	8	5	$a[2] = 6 > 2$: shift $a[3] \leftarrow 6; j \leftarrow 1$
	–	2	1	2	6	8	5	$a[1] = 1 \leq 2$: exit; insert $a[2] \leftarrow 2$
5	4	5	1	2	6	8	5	$key \leftarrow a[5] = 5; j \leftarrow 4$
	3	5	1	2	6	8	8	$a[4] = 8 > 5$: shift $a[5] \leftarrow 8; j \leftarrow 3$
	2	5	1	2	6	6	8	$a[3] = 6 > 5$: shift $a[4] \leftarrow 6; j \leftarrow 2$
	–	5	1	2	5	6	8	$a[2] = 2 \leq 5$: exit; insert $a[3] \leftarrow 5$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
–	–	6	1	8	2	5	initial state
2	1	1	6	8	2	5	insert 1 at the beginning
3	8	1	6	8	2	5	8 already in position
4	2	1	2	6	8	5	insert 2 between 1 and 6
5	5	1	2	5	6	8	insert 5 between 2 and 6

The algorithm performs 8 key comparisons and 5 shifts, and the sorted array is

$[1, 2, 5, 6, 8]$.

2. Array $a = [13, 11, 9, 7]$ ($n = 4$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	—	13	11	9	7	initial state
2	1	11	13	11	9	7	$key \leftarrow a[2] = 11; j \leftarrow 1$
	0	11	13	13	9	7	$a[1] = 13 > 11$: shift $a[2] \leftarrow 13; j \leftarrow 0$
	—	11	11	13	9	7	$j = 0$: exit; insert $a[1] \leftarrow 11$
3	2	9	11	13	9	7	$key \leftarrow a[3] = 9; j \leftarrow 2$
	1	9	11	13	13	7	$a[2] = 13 > 9$: shift $a[3] \leftarrow 13; j \leftarrow 1$
	0	9	11	11	13	7	$a[1] = 11 > 9$: shift $a[2] \leftarrow 11; j \leftarrow 0$
	—	9	9	11	13	7	$j = 0$: exit; insert $a[1] \leftarrow 9$
4	3	7	9	11	13	7	$key \leftarrow a[4] = 7; j \leftarrow 3$
	2	7	9	11	13	13	$a[3] = 13 > 7$: shift $a[4] \leftarrow 13; j \leftarrow 2$
	1	7	9	11	11	13	$a[2] = 11 > 7$: shift $a[3] \leftarrow 11; j \leftarrow 1$
	0	7	9	9	11	13	$a[1] = 9 > 7$: shift $a[2] \leftarrow 9; j \leftarrow 0$
	—	7	7	9	11	13	$j = 0$: exit; insert $a[1] \leftarrow 7$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	13	11	9	7	initial state
2	11	11	13	9	7	insert 11 at the beginning
3	9	9	11	13	7	insert 9 at the beginning
4	7	7	9	11	13	insert 7 at the beginning

The algorithm performs 6 key comparisons and 6 shifts, and the sorted array is

[7, 9, 11, 13].

3. Array $a = [2, 8, 4, 6]$ ($n = 4$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	—	2	8	4	6	initial state
2	1	8	2	8	4	6	$key \leftarrow a[2] = 8; j \leftarrow 1$
	—	8	2	8	4	6	$a[1] = 2 \leq 8$: exit; insert $a[2] \leftarrow 8$
3	2	4	2	8	4	6	$key \leftarrow a[3] = 4; j \leftarrow 2$
	1	4	2	8	8	6	$a[2] = 8 > 4$: shift $a[3] \leftarrow 8; j \leftarrow 1$
	—	4	2	4	8	6	$a[1] = 2 \leq 4$: exit; insert $a[2] \leftarrow 4$
4	3	6	2	4	8	6	$key \leftarrow a[4] = 6; j \leftarrow 3$
	2	6	2	4	8	8	$a[3] = 8 > 6$: shift $a[4] \leftarrow 8; j \leftarrow 2$
	—	6	2	4	6	8	$a[2] = 4 \leq 6$: exit; insert $a[3] \leftarrow 6$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	2	8	4	6	initial state
2	8	2	8	4	6	8 already in position
3	4	2	4	8	6	insert 4 between 2 and 8
4	6	2	4	6	8	insert 6 between 4 and 8

The algorithm performs 5 key comparisons and 2 shifts, and the sorted array is
 $[2, 4, 6, 8]$.

4. Array $a = [30, 20, 10]$ ($n = 3$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	Action
—	—	—	30	20	10	initial state
2	1	20	30	20	10	$key \leftarrow a[2] = 20; j \leftarrow 1$
	0	20	30	30	10	$a[1] = 30 > 20$: shift $a[2] \leftarrow 30; j \leftarrow 0$
	—	20	20	30	10	$j = 0$: exit; insert $a[1] \leftarrow 20$
3	2	10	20	30	10	$key \leftarrow a[3] = 10; j \leftarrow 2$
	1	10	20	30	30	$a[2] = 30 > 10$: shift $a[3] \leftarrow 30; j \leftarrow 1$
	0	10	20	20	30	$a[1] = 20 > 10$: shift $a[2] \leftarrow 20; j \leftarrow 0$
	—	10	10	20	30	$j = 0$: exit; insert $a[1] \leftarrow 10$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	Action
—	—	30	20	10	initial state
2	20	20	30	10	insert 20 at the beginning
3	10	10	20	30	insert 10 at the beginning

The algorithm performs 3 key comparisons and 3 shifts, and the sorted array is
 $[10, 20, 30]$.

5. Array $a = [5, 10, 15, 20]$ ($n = 4$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	—	5	10	15	20	initial state
2	1	10	5	10	15	20	$key \leftarrow a[2] = 10; j \leftarrow 1$
	—	10	5	10	15	20	$a[1] = 5 \leq 10$: exit; insert $a[2] \leftarrow 10$
3	2	15	5	10	15	20	$key \leftarrow a[3] = 15; j \leftarrow 2$
	—	15	5	10	15	20	$a[2] = 10 \leq 15$: exit; insert $a[3] \leftarrow 15$
4	3	20	5	10	15	20	$key \leftarrow a[4] = 20; j \leftarrow 3$
	—	20	5	10	15	20	$a[3] = 15 \leq 20$: exit; insert $a[4] \leftarrow 20$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	5	10	15	20	initial state
2	10	5	10	15	20	10 already in position
3	15	5	10	15	20	15 already in position
4	20	5	10	15	20	20 already in position

The algorithm performs 3 key comparisons and 0 shifts, and the sorted array is
[5, 10, 15, 20].

Exercise 3.8 — Sorting algorithms (E8)

Consider the following arrays of natural numbers:

1. $a = [14, 7, 21, 3, 18]$
2. $a = [40, 30, 20, 10]$
3. $a = [3, 6, 1, 9, 2]$
4. $a = [25, 12, 50]$
5. $a = [17, 17, 17, 17]$

Questions

1. Apply the **InsertionSort** algorithm to each array, to sort it in non-decreasing order. Show the execution step by step: for every iteration of the outer loop, give the key being inserted, every shift performed by the inner loop and the state of the array after the insertion.
2. For each array, count the key comparisons $a[j] > key$ and the shifts performed by the algorithm.

Solution

The trace tables follow the conventions stated in the introduction to the exercises of this chapter. Each item answers both questions.

1. Array $a = [14, 7, 21, 3, 18]$ ($n = 5$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
—	—	—	14	7	21	3	18	initial state
2	1	7	14	7	21	3	18	$key \leftarrow a[2] = 7; j \leftarrow 1$
	0	7	14	14	21	3	18	$a[1] = 14 > 7$: shift $a[2] \leftarrow 14; j \leftarrow 0$
	—	7	7	14	21	3	18	$j = 0$: exit; insert $a[1] \leftarrow 7$
3	2	21	7	14	21	3	18	$key \leftarrow a[3] = 21; j \leftarrow 2$
	—	21	7	14	21	3	18	$a[2] = 14 \leq 21$: exit; insert $a[3] \leftarrow 21$
4	3	3	7	14	21	3	18	$key \leftarrow a[4] = 3; j \leftarrow 3$
	2	3	7	14	21	21	18	$a[3] = 21 > 3$: shift $a[4] \leftarrow 21; j \leftarrow 2$
	1	3	7	14	14	21	18	$a[2] = 14 > 3$: shift $a[3] \leftarrow 14; j \leftarrow 1$
	0	3	7	7	14	21	18	$a[1] = 7 > 3$: shift $a[2] \leftarrow 7; j \leftarrow 0$
	—	3	3	7	14	21	18	$j = 0$: exit; insert $a[1] \leftarrow 3$
5	4	18	3	7	14	21	18	$key \leftarrow a[5] = 18; j \leftarrow 4$
	3	18	3	7	14	21	21	$a[4] = 21 > 18$: shift $a[5] \leftarrow 21; j \leftarrow 3$
	—	18	3	7	14	18	21	$a[3] = 14 \leq 18$: exit; insert $a[4] \leftarrow 18$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
—	—	14	7	21	3	18	initial state
2	7	7	14	21	3	18	insert 7 at the beginning
3	21	7	14	21	3	18	21 already in position
4	3	3	7	14	21	18	insert 3 at the beginning
5	18	3	7	14	18	21	insert 18 between 14 and 21

The algorithm performs 7 key comparisons and 5 shifts, and the sorted array is
[3, 7, 14, 18, 21].

2. Array $a = [40, 30, 20, 10]$ ($n = 4$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	—	40	30	20	10	initial state
2	1	30	40	30	20	10	$key \leftarrow a[2] = 30; j \leftarrow 1$
	0	30	40	40	20	10	$a[1] = 40 > 30$: shift $a[2] \leftarrow 40; j \leftarrow 0$
	—	30	30	40	20	10	$j = 0$: exit; insert $a[1] \leftarrow 30$
3	2	20	30	40	20	10	$key \leftarrow a[3] = 20; j \leftarrow 2$
	1	20	30	40	40	10	$a[2] = 40 > 20$: shift $a[3] \leftarrow 40; j \leftarrow 1$
	0	20	30	30	40	10	$a[1] = 30 > 20$: shift $a[2] \leftarrow 30; j \leftarrow 0$
	—	20	20	30	40	10	$j = 0$: exit; insert $a[1] \leftarrow 20$
4	3	10	20	30	40	10	$key \leftarrow a[4] = 10; j \leftarrow 3$
	2	10	20	30	40	40	$a[3] = 40 > 10$: shift $a[4] \leftarrow 40; j \leftarrow 2$
	1	10	20	30	30	40	$a[2] = 30 > 10$: shift $a[3] \leftarrow 30; j \leftarrow 1$
	0	10	20	20	30	40	$a[1] = 20 > 10$: shift $a[2] \leftarrow 20; j \leftarrow 0$
	—	10	10	20	30	40	$j = 0$: exit; insert $a[1] \leftarrow 10$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	40	30	20	10	initial state
2	30	30	40	20	10	insert 30 at the beginning
3	20	20	30	40	10	insert 20 at the beginning
4	10	10	20	30	40	insert 10 at the beginning

The algorithm performs 6 key comparisons and 6 shifts, and the sorted array is

[10, 20, 30, 40].

3. Array $a = [3, 6, 1, 9, 2]$ ($n = 5$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
—	—	—	3	6	1	9	2	initial state
2	1	6	3	6	1	9	2	$key \leftarrow a[2] = 6; j \leftarrow 1$
—	—	6	3	6	1	9	2	$a[1] = 3 \leq 6$: exit; insert $a[2] \leftarrow 6$
3	2	1	3	6	1	9	2	$key \leftarrow a[3] = 1; j \leftarrow 2$
1	1	1	3	6	6	9	2	$a[2] = 6 > 1$: shift $a[3] \leftarrow 6; j \leftarrow 1$
0	1	1	3	3	6	9	2	$a[1] = 3 > 1$: shift $a[2] \leftarrow 3; j \leftarrow 0$
—	—	1	1	3	6	9	2	$j = 0$: exit; insert $a[1] \leftarrow 1$
4	3	9	1	3	6	9	2	$key \leftarrow a[4] = 9; j \leftarrow 3$
—	—	9	1	3	6	9	2	$a[3] = 6 \leq 9$: exit; insert $a[4] \leftarrow 9$
5	4	2	1	3	6	9	2	$key \leftarrow a[5] = 2; j \leftarrow 4$
3	2	2	1	3	6	9	9	$a[4] = 9 > 2$: shift $a[5] \leftarrow 9; j \leftarrow 3$
2	2	2	1	3	6	6	9	$a[3] = 6 > 2$: shift $a[4] \leftarrow 6; j \leftarrow 2$
1	2	2	1	3	3	6	9	$a[2] = 3 > 2$: shift $a[3] \leftarrow 3; j \leftarrow 1$
—	—	2	1	2	3	6	9	$a[1] = 1 \leq 2$: exit; insert $a[2] \leftarrow 2$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
—	—	3	6	1	9	2	initial state
2	6	3	6	1	9	2	6 already in position
3	1	1	3	6	9	2	insert 1 at the beginning
4	9	1	3	6	9	2	9 already in position
5	2	1	2	3	6	9	insert 2 between 1 and 3

The algorithm performs 8 key comparisons and 5 shifts, and the sorted array is

[1, 2, 3, 6, 9].

4. Array $a = [25, 12, 50]$ ($n = 3$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	Action
—	—	—	25	12	50	initial state
2	1	12	25	12	50	$key \leftarrow a[2] = 12; j \leftarrow 1$
	0	12	25	25	50	$a[1] = 25 > 12$: shift $a[2] \leftarrow 25; j \leftarrow 0$
	—	12	12	25	50	$j = 0$: exit; insert $a[1] \leftarrow 12$
3	2	50	12	25	50	$key \leftarrow a[3] = 50; j \leftarrow 2$
	—	50	12	25	50	$a[2] = 25 \leq 50$: exit; insert $a[3] \leftarrow 50$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	Action
—	—	25	12	50	initial state
2	12	12	25	50	insert 12 at the beginning
3	50	12	25	50	50 already in position

The algorithm performs 2 key comparisons and 1 shift, and the sorted array is

$[12, 25, 50]$.

5. Array $a = [17, 17, 17, 17]$ ($n = 4$).

Detailed trace:

i	j	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	—	17	17	17	17	initial state
2	1	17	17	17	17	17	$key \leftarrow a[2] = 17; j \leftarrow 1$
	—	17	17	17	17	17	$a[1] = 17 \leq 17$: exit; insert $a[2] \leftarrow 17$
3	2	17	17	17	17	17	$key \leftarrow a[3] = 17; j \leftarrow 2$
	—	17	17	17	17	17	$a[2] = 17 \leq 17$: exit; insert $a[3] \leftarrow 17$
4	3	17	17	17	17	17	$key \leftarrow a[4] = 17; j \leftarrow 3$
	—	17	17	17	17	17	$a[3] = 17 \leq 17$: exit; insert $a[4] \leftarrow 17$

Array after each iteration of the outer loop:

i	key	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	17	17	17	17	initial state
2	17	17	17	17	17	17 already in position
3	17	17	17	17	17	17 already in position
4	17	17	17	17	17	17 already in position

The algorithm performs 3 key comparisons and 0 shifts, and the sorted array is

$[17, 17, 17, 17]$.

Exercise 3.9 — Sorting algorithms (E9)

Consider the following arrays of natural numbers:

1. $a = [5, 2, 4, 6, 1, 3]$
2. $a = [8, 3, 7, 4, 9, 2]$
3. $a = [6, 5, 4, 3, 2, 1]$
4. $a = [1, 2, 3, 4, 5, 6]$
5. $a = [10, 5, 15, 3]$

Questions

1. Apply the `ExchangeSort` algorithm to each array, to sort it in non-decreasing order. Show the execution step by step: for every comparison, say whether a swap is performed and give the state of the array after the comparison.
2. For each array, count the key comparisons and the swaps performed by the algorithm.

Solution

The trace tables follow the conventions stated in the introduction to the exercises of this chapter. Each item answers both questions.

1. Array $a = [5, 2, 4, 6, 1, 3]$ ($n = 6$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	—	5	2	4	6	1	3	initial state
1	2	2	5	4	6	1	3	$a[1] = 5 > a[2] = 2$: swap $a[1] \leftrightarrow a[2]$
	3	2	5	4	6	1	3	$a[1] = 2 \leq a[3] = 4$: no swap
	4	2	5	4	6	1	3	$a[1] = 2 \leq a[4] = 6$: no swap
	5	1	5	4	6	2	3	$a[1] = 2 > a[5] = 1$: swap $a[1] \leftrightarrow a[5]$
	6	1	5	4	6	2	3	$a[1] = 1 \leq a[6] = 3$: no swap
2	3	1	4	5	6	2	3	$a[2] = 5 > a[3] = 4$: swap $a[2] \leftrightarrow a[3]$
	4	1	4	5	6	2	3	$a[2] = 4 \leq a[4] = 6$: no swap
	5	1	2	5	6	4	3	$a[2] = 4 > a[5] = 2$: swap $a[2] \leftrightarrow a[5]$
	6	1	2	5	6	4	3	$a[2] = 2 \leq a[6] = 3$: no swap
3	4	1	2	5	6	4	3	$a[3] = 5 \leq a[4] = 6$: no swap
	5	1	2	4	6	5	3	$a[3] = 5 > a[5] = 4$: swap $a[3] \leftrightarrow a[5]$
	6	1	2	3	6	5	4	$a[3] = 4 > a[6] = 3$: swap $a[3] \leftrightarrow a[6]$
4	5	1	2	3	5	6	4	$a[4] = 6 > a[5] = 5$: swap $a[4] \leftrightarrow a[5]$
	6	1	2	3	4	6	5	$a[4] = 5 > a[6] = 4$: swap $a[4] \leftrightarrow a[6]$
5	6	1	2	3	4	5	6	$a[5] = 6 > a[6] = 5$: swap $a[5] \leftrightarrow a[6]$

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	5	2	4	6	1	3	initial state
1	1	5	4	6	2	3	$a[1] = \min(a[1..6]) = 1$ (2 swaps)
2	1	2	5	6	4	3	$a[2] = \min(a[2..6]) = 2$ (2 swaps)
3	1	2	3	6	5	4	$a[3] = \min(a[3..6]) = 3$ (2 swaps)
4	1	2	3	4	6	5	$a[4] = \min(a[4..6]) = 4$ (2 swaps)
5	1	2	3	4	5	6	$a[5] = \min(a[5..6]) = 5$ (1 swap)

The algorithm performs 15 key comparisons and 9 swaps, and the sorted array is
 $[1, 2, 3, 4, 5, 6]$.

2. Array $a = [8, 3, 7, 4, 9, 2]$ ($n = 6$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	—	8	3	7	4	9	2	initial state
1	2	3	8	7	4	9	2	$a[1] = 8 > a[2] = 3$: swap $a[1] \leftrightarrow a[2]$
	3	3	8	7	4	9	2	$a[1] = 3 \leq a[3] = 7$: no swap
	4	3	8	7	4	9	2	$a[1] = 3 \leq a[4] = 4$: no swap
	5	3	8	7	4	9	2	$a[1] = 3 \leq a[5] = 9$: no swap
	6	2	8	7	4	9	3	$a[1] = 3 > a[6] = 2$: swap $a[1] \leftrightarrow a[6]$
2	3	2	7	8	4	9	3	$a[2] = 8 > a[3] = 7$: swap $a[2] \leftrightarrow a[3]$
	4	2	4	8	7	9	3	$a[2] = 7 > a[4] = 4$: swap $a[2] \leftrightarrow a[4]$
	5	2	4	8	7	9	3	$a[2] = 4 \leq a[5] = 9$: no swap
	6	2	3	8	7	9	4	$a[2] = 4 > a[6] = 3$: swap $a[2] \leftrightarrow a[6]$
3	4	2	3	7	8	9	4	$a[3] = 8 > a[4] = 7$: swap $a[3] \leftrightarrow a[4]$
	5	2	3	7	8	9	4	$a[3] = 7 \leq a[5] = 9$: no swap
	6	2	3	4	8	9	7	$a[3] = 7 > a[6] = 4$: swap $a[3] \leftrightarrow a[6]$
4	5	2	3	4	8	9	7	$a[4] = 8 \leq a[5] = 9$: no swap
	6	2	3	4	7	9	8	$a[4] = 8 > a[6] = 7$: swap $a[4] \leftrightarrow a[6]$
5	6	2	3	4	7	8	9	$a[5] = 9 > a[6] = 8$: swap $a[5] \leftrightarrow a[6]$

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	8	3	7	4	9	2	initial state
1	2	8	7	4	9	3	$a[1] = \min(a[1..6]) = 2$ (2 swaps)
2	2	3	8	7	9	4	$a[2] = \min(a[2..6]) = 3$ (3 swaps)
3	2	3	4	8	9	7	$a[3] = \min(a[3..6]) = 4$ (2 swaps)
4	2	3	4	7	9	8	$a[4] = \min(a[4..6]) = 7$ (1 swap)
5	2	3	4	7	8	9	$a[5] = \min(a[5..6]) = 8$ (1 swap)

The algorithm performs 15 key comparisons and 9 swaps, and the sorted array is
 $[2, 3, 4, 7, 8, 9]$.

3. Array $a = [6, 5, 4, 3, 2, 1]$ ($n = 6$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	—	6	5	4	3	2	1	initial state
1	2	5	6	4	3	2	1	$a[1] = 6 > a[2] = 5$: swap $a[1] \leftrightarrow a[2]$
	3	4	6	5	3	2	1	$a[1] = 5 > a[3] = 4$: swap $a[1] \leftrightarrow a[3]$
	4	3	6	5	4	2	1	$a[1] = 4 > a[4] = 3$: swap $a[1] \leftrightarrow a[4]$
	5	2	6	5	4	3	1	$a[1] = 3 > a[5] = 2$: swap $a[1] \leftrightarrow a[5]$
	6	1	6	5	4	3	2	$a[1] = 2 > a[6] = 1$: swap $a[1] \leftrightarrow a[6]$
2	3	1	5	6	4	3	2	$a[2] = 6 > a[3] = 5$: swap $a[2] \leftrightarrow a[3]$
	4	1	4	6	5	3	2	$a[2] = 5 > a[4] = 4$: swap $a[2] \leftrightarrow a[4]$
	5	1	3	6	5	4	2	$a[2] = 4 > a[5] = 3$: swap $a[2] \leftrightarrow a[5]$
	6	1	2	6	5	4	3	$a[2] = 3 > a[6] = 2$: swap $a[2] \leftrightarrow a[6]$
3	4	1	2	5	6	4	3	$a[3] = 6 > a[4] = 5$: swap $a[3] \leftrightarrow a[4]$
	5	1	2	4	6	5	3	$a[3] = 5 > a[5] = 4$: swap $a[3] \leftrightarrow a[5]$
	6	1	2	3	6	5	4	$a[3] = 4 > a[6] = 3$: swap $a[3] \leftrightarrow a[6]$
4	5	1	2	3	5	6	4	$a[4] = 6 > a[5] = 5$: swap $a[4] \leftrightarrow a[5]$
	6	1	2	3	4	6	5	$a[4] = 5 > a[6] = 4$: swap $a[4] \leftrightarrow a[6]$
5	6	1	2	3	4	5	6	$a[5] = 6 > a[6] = 5$: swap $a[5] \leftrightarrow a[6]$

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	6	5	4	3	2	1	initial state
1	1	6	5	4	3	2	$a[1] = \min(a[1..6]) = 1$ (5 swaps)
2	1	2	6	5	4	3	$a[2] = \min(a[2..6]) = 2$ (4 swaps)
3	1	2	3	6	5	4	$a[3] = \min(a[3..6]) = 3$ (3 swaps)
4	1	2	3	4	6	5	$a[4] = \min(a[4..6]) = 4$ (2 swaps)
5	1	2	3	4	5	6	$a[5] = \min(a[5..6]) = 5$ (1 swap)

The algorithm performs 15 key comparisons and 15 swaps, and the sorted array is

$[1, 2, 3, 4, 5, 6]$.

4. Array $a = [1, 2, 3, 4, 5, 6]$ ($n = 6$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	—	1	2	3	4	5	6	initial state
1	2	1	2	3	4	5	6	$a[1] = 1 \leq a[2] = 2$: no swap
	3	1	2	3	4	5	6	$a[1] = 1 \leq a[3] = 3$: no swap
	4	1	2	3	4	5	6	$a[1] = 1 \leq a[4] = 4$: no swap
	5	1	2	3	4	5	6	$a[1] = 1 \leq a[5] = 5$: no swap
	6	1	2	3	4	5	6	$a[1] = 1 \leq a[6] = 6$: no swap
2	3	1	2	3	4	5	6	$a[2] = 2 \leq a[3] = 3$: no swap
	4	1	2	3	4	5	6	$a[2] = 2 \leq a[4] = 4$: no swap
	5	1	2	3	4	5	6	$a[2] = 2 \leq a[5] = 5$: no swap
	6	1	2	3	4	5	6	$a[2] = 2 \leq a[6] = 6$: no swap
3	4	1	2	3	4	5	6	$a[3] = 3 \leq a[4] = 4$: no swap
	5	1	2	3	4	5	6	$a[3] = 3 \leq a[5] = 5$: no swap
	6	1	2	3	4	5	6	$a[3] = 3 \leq a[6] = 6$: no swap
4	5	1	2	3	4	5	6	$a[4] = 4 \leq a[5] = 5$: no swap
	6	1	2	3	4	5	6	$a[4] = 4 \leq a[6] = 6$: no swap
5	6	1	2	3	4	5	6	$a[5] = 5 \leq a[6] = 6$: no swap

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	1	2	3	4	5	6	initial state
1	1	2	3	4	5	6	$a[1] = \min(a[1..6]) = 1$ (0 swaps)
2	1	2	3	4	5	6	$a[2] = \min(a[2..6]) = 2$ (0 swaps)
3	1	2	3	4	5	6	$a[3] = \min(a[3..6]) = 3$ (0 swaps)
4	1	2	3	4	5	6	$a[4] = \min(a[4..6]) = 4$ (0 swaps)
5	1	2	3	4	5	6	$a[5] = \min(a[5..6]) = 5$ (0 swaps)

The algorithm performs 15 key comparisons and 0 swaps, and the sorted array is

$$[1, 2, 3, 4, 5, 6].$$

5. Array $a = [10, 5, 15, 3]$ ($n = 4$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	10	5	15	3	initial state
1	2	5	10	15	3	$a[1] = 10 > a[2] = 5$: swap $a[1] \leftrightarrow a[2]$
	3	5	10	15	3	$a[1] = 5 \leq a[3] = 15$: no swap
	4	3	10	15	5	$a[1] = 5 > a[4] = 3$: swap $a[1] \leftrightarrow a[4]$
2	3	3	10	15	5	$a[2] = 10 \leq a[3] = 15$: no swap
	4	3	5	15	10	$a[2] = 10 > a[4] = 5$: swap $a[2] \leftrightarrow a[4]$
3	4	3	5	10	15	$a[3] = 15 > a[4] = 10$: swap $a[3] \leftrightarrow a[4]$

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
–	10	5	15	3	initial state
1	3	10	15	5	$a[1] = \min(a[1..4]) = 3$ (2 swaps)
2	3	5	15	10	$a[2] = \min(a[2..4]) = 5$ (1 swap)
3	3	5	10	15	$a[3] = \min(a[3..4]) = 10$ (1 swap)

The algorithm performs 6 key comparisons and 4 swaps, and the sorted array is

[3, 5, 10, 15].

Exercise 3.10 — Sorting algorithms (E10)

Consider the following arrays of natural numbers:

1. $a = [7, 2, 9, 4]$
2. $a = [15, 10, 5]$
3. $a = [3, 1, 4, 1, 5, 9, 2, 6]$
4. $a = [20, 18, 16, 14]$
5. $a = [11, 11, 11]$

Questions

1. Apply the **ExchangeSort** algorithm to each array, to sort it in non-decreasing order. Show the execution step by step: for every comparison, say whether a swap is performed and give the state of the array after the comparison.
2. For each array, count the key comparisons and the swaps performed by the algorithm.

Solution

The trace tables follow the conventions stated in the introduction to the exercises of this chapter. Each item answers both questions.

1. Array $a = [7, 2, 9, 4]$ ($n = 4$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	7	2	9	4	initial state
1	2	2	7	9	4	$a[1] = 7 > a[2] = 2$: swap $a[1] \leftrightarrow a[2]$
	3	2	7	9	4	$a[1] = 2 \leq a[3] = 9$: no swap
	4	2	7	9	4	$a[1] = 2 \leq a[4] = 4$: no swap
2	3	2	7	9	4	$a[2] = 7 \leq a[3] = 9$: no swap
	4	2	4	9	7	$a[2] = 7 > a[4] = 4$: swap $a[2] \leftrightarrow a[4]$
3	4	2	4	7	9	$a[3] = 9 > a[4] = 7$: swap $a[3] \leftrightarrow a[4]$

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	7	2	9	4	initial state
1	2	7	9	4	$a[1] = \min(a[1..4]) = 2$ (1 swap)
2	2	4	9	7	$a[2] = \min(a[2..4]) = 4$ (1 swap)
3	2	4	7	9	$a[3] = \min(a[3..4]) = 7$ (1 swap)

The algorithm performs 6 key comparisons and 3 swaps, and the sorted array is

[2, 4, 7, 9].

2. Array $a = [15, 10, 5]$ ($n = 3$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	Action
—	—	15	10	5	initial state
1	2	10	15	5	$a[1] = 15 > a[2] = 10$: swap $a[1] \leftrightarrow a[2]$
	3	5	15	10	$a[1] = 10 > a[3] = 5$: swap $a[1] \leftrightarrow a[3]$
2	3	5	10	15	$a[2] = 15 > a[3] = 10$: swap $a[2] \leftrightarrow a[3]$

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	Action
—	15	10	5	initial state
1	5	15	10	$a[1] = \min(a[1..3]) = 5$ (2 swaps)
2	5	10	15	$a[2] = \min(a[2..3]) = 10$ (1 swap)

The algorithm performs 3 key comparisons and 3 swaps, and the sorted array is

[5, 10, 15].

3. Array $a = [3, 1, 4, 1, 5, 9, 2, 6]$ ($n = 8$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	$a[7]$	$a[8]$	Action
–	–	3	1	4	1	5	9	2	6	initial state
1	2	1	3	4	1	5	9	2	6	$a[1] = 3 > a[2] = 1$: swap $a[1] \leftrightarrow a[2]$
	3	1	3	4	1	5	9	2	6	$a[1] = 1 \leq a[3] = 4$: no swap
	4	1	3	4	1	5	9	2	6	$a[1] = 1 \leq a[4] = 1$: no swap
	5	1	3	4	1	5	9	2	6	$a[1] = 1 \leq a[5] = 5$: no swap
	6	1	3	4	1	5	9	2	6	$a[1] = 1 \leq a[6] = 9$: no swap
	7	1	3	4	1	5	9	2	6	$a[1] = 1 \leq a[7] = 2$: no swap
	8	1	3	4	1	5	9	2	6	$a[1] = 1 \leq a[8] = 6$: no swap
2	3	1	3	4	1	5	9	2	6	$a[2] = 3 \leq a[3] = 4$: no swap
	4	1	1	4	3	5	9	2	6	$a[2] = 3 > a[4] = 1$: swap $a[2] \leftrightarrow a[4]$
	5	1	1	4	3	5	9	2	6	$a[2] = 1 \leq a[5] = 5$: no swap
	6	1	1	4	3	5	9	2	6	$a[2] = 1 \leq a[6] = 9$: no swap
	7	1	1	4	3	5	9	2	6	$a[2] = 1 \leq a[7] = 2$: no swap
	8	1	1	4	3	5	9	2	6	$a[2] = 1 \leq a[8] = 6$: no swap
3	4	1	1	3	4	5	9	2	6	$a[3] = 4 > a[4] = 3$: swap $a[3] \leftrightarrow a[4]$
	5	1	1	3	4	5	9	2	6	$a[3] = 3 \leq a[5] = 5$: no swap
	6	1	1	3	4	5	9	2	6	$a[3] = 3 \leq a[6] = 9$: no swap
	7	1	1	2	4	5	9	3	6	$a[3] = 3 > a[7] = 2$: swap $a[3] \leftrightarrow a[7]$
	8	1	1	2	4	5	9	3	6	$a[3] = 2 \leq a[8] = 6$: no swap
4	5	1	1	2	4	5	9	3	6	$a[4] = 4 \leq a[5] = 5$: no swap
	6	1	1	2	4	5	9	3	6	$a[4] = 4 \leq a[6] = 9$: no swap
	7	1	1	2	3	5	9	4	6	$a[4] = 4 > a[7] = 3$: swap $a[4] \leftrightarrow a[7]$
	8	1	1	2	3	5	9	4	6	$a[4] = 3 \leq a[8] = 6$: no swap
5	6	1	1	2	3	5	9	4	6	$a[5] = 5 \leq a[6] = 9$: no swap
	7	1	1	2	3	4	9	5	6	$a[5] = 5 > a[7] = 4$: swap $a[5] \leftrightarrow a[7]$
	8	1	1	2	3	4	9	5	6	$a[5] = 4 \leq a[8] = 6$: no swap
6	7	1	1	2	3	4	5	9	6	$a[6] = 9 > a[7] = 5$: swap $a[6] \leftrightarrow a[7]$
	8	1	1	2	3	4	5	9	6	$a[6] = 5 \leq a[8] = 6$: no swap
7	8	1	1	2	3	4	5	6	9	$a[7] = 9 > a[8] = 6$: swap $a[7] \leftrightarrow a[8]$

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	$a[7]$	$a[8]$	Action
–	3	1	4	1	5	9	2	6	initial state
1	1	3	4	1	5	9	2	6	$a[1] = \min(a[1..8]) = 1$ (1 swap)
2	1	1	4	3	5	9	2	6	$a[2] = \min(a[2..8]) = 1$ (1 swap)
3	1	1	2	4	5	9	3	6	$a[3] = \min(a[3..8]) = 2$ (2 swaps)
4	1	1	2	3	5	9	4	6	$a[4] = \min(a[4..8]) = 3$ (1 swap)
5	1	1	2	3	4	9	5	6	$a[5] = \min(a[5..8]) = 4$ (1 swap)
6	1	1	2	3	4	5	9	6	$a[6] = \min(a[6..8]) = 5$ (1 swap)
7	1	1	2	3	4	5	6	9	$a[7] = \min(a[7..8]) = 6$ (1 swap)

The algorithm performs 28 key comparisons and 8 swaps, and the sorted array is

[1, 1, 2, 3, 4, 5, 6, 9].

4. Array $a = [20, 18, 16, 14]$ ($n = 4$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	20	18	16	14	initial state
1	2	18	20	16	14	$a[1] = 20 > a[2] = 18$: swap $a[1] \leftrightarrow a[2]$
	3	16	20	18	14	$a[1] = 18 > a[3] = 16$: swap $a[1] \leftrightarrow a[3]$
	4	14	20	18	16	$a[1] = 16 > a[4] = 14$: swap $a[1] \leftrightarrow a[4]$
2	3	14	18	20	16	$a[2] = 20 > a[3] = 18$: swap $a[2] \leftrightarrow a[3]$
	4	14	16	20	18	$a[2] = 18 > a[4] = 16$: swap $a[2] \leftrightarrow a[4]$
3	4	14	16	18	20	$a[3] = 20 > a[4] = 18$: swap $a[3] \leftrightarrow a[4]$

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	20	18	16	14	initial state
1	14	20	18	16	$a[1] = \min(a[1..4]) = 14$ (3 swaps)
2	14	16	20	18	$a[2] = \min(a[2..4]) = 16$ (2 swaps)
3	14	16	18	20	$a[3] = \min(a[3..4]) = 18$ (1 swap)

The algorithm performs 6 key comparisons and 6 swaps, and the sorted array is

[14, 16, 18, 20].

5. Array $a = [11, 11, 11]$ ($n = 3$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	Action
—	—	11	11	11	initial state
1	2	11	11	11	$a[1] = 11 \leq a[2] = 11$: no swap
	3	11	11	11	$a[1] = 11 \leq a[3] = 11$: no swap
2	3	11	11	11	$a[2] = 11 \leq a[3] = 11$: no swap

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	Action
—	11	11	11	initial state
1	11	11	11	$a[1] = \min(a[1..3]) = 11$ (0 swaps)
2	11	11	11	$a[2] = \min(a[2..3]) = 11$ (0 swaps)

The algorithm performs 3 key comparisons and 0 swaps, and the sorted array is

[11, 11, 11].

Exercise 3.11 — Sorting algorithms (E11)

Consider the following arrays of natural numbers:

1. $a = [9, 3, 7, 2]$
2. $a = [12, 4, 18, 6, 10]$
3. $a = [5, 4, 3, 2, 1]$
4. $a = [100, 50, 25]$
5. $a = [8]$

Questions

1. Apply the **ExchangeSort** algorithm to each array, to sort it in non-decreasing order. Show the execution step by step: for every comparison, say whether a swap is performed and give the state of the array after the comparison.
2. For each array, count the key comparisons and the swaps performed by the algorithm.

Solution

The trace tables follow the conventions stated in the introduction to the exercises of this chapter. Each item answers both questions.

1. Array $a = [9, 3, 7, 2]$ ($n = 4$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	9	3	7	2	initial state
1	2	3	9	7	2	$a[1] = 9 > a[2] = 3$: swap $a[1] \leftrightarrow a[2]$
	3	3	9	7	2	$a[1] = 3 \leq a[3] = 7$: no swap
	4	2	9	7	3	$a[1] = 3 > a[4] = 2$: swap $a[1] \leftrightarrow a[4]$
2	3	2	7	9	3	$a[2] = 9 > a[3] = 7$: swap $a[2] \leftrightarrow a[3]$
	4	2	3	9	7	$a[2] = 7 > a[4] = 3$: swap $a[2] \leftrightarrow a[4]$
3	4	2	3	7	9	$a[3] = 9 > a[4] = 7$: swap $a[3] \leftrightarrow a[4]$

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	9	3	7	2	initial state
1	2	9	7	3	$a[1] = \min(a[1..4]) = 2$ (2 swaps)
2	2	3	9	7	$a[2] = \min(a[2..4]) = 3$ (2 swaps)
3	2	3	7	9	$a[3] = \min(a[3..4]) = 7$ (1 swap)

The algorithm performs 6 key comparisons and 5 swaps, and the sorted array is

$[2, 3, 7, 9]$.

2. Array $a = [12, 4, 18, 6, 10]$ ($n = 5$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
—	—	12	4	18	6	10	initial state
1	2	4	12	18	6	10	$a[1] = 12 > a[2] = 4$: swap $a[1] \leftrightarrow a[2]$
	3	4	12	18	6	10	$a[1] = 4 \leq a[3] = 18$: no swap
	4	4	12	18	6	10	$a[1] = 4 \leq a[4] = 6$: no swap
	5	4	12	18	6	10	$a[1] = 4 \leq a[5] = 10$: no swap
2	3	4	12	18	6	10	$a[2] = 12 \leq a[3] = 18$: no swap
	4	4	6	18	12	10	$a[2] = 12 > a[4] = 6$: swap $a[2] \leftrightarrow a[4]$
	5	4	6	18	12	10	$a[2] = 6 \leq a[5] = 10$: no swap
3	4	4	6	12	18	10	$a[3] = 18 > a[4] = 12$: swap $a[3] \leftrightarrow a[4]$
	5	4	6	10	18	12	$a[3] = 12 > a[5] = 10$: swap $a[3] \leftrightarrow a[5]$
4	5	4	6	10	12	18	$a[4] = 18 > a[5] = 12$: swap $a[4] \leftrightarrow a[5]$

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
—	12	4	18	6	10	initial state
1	4	12	18	6	10	$a[1] = \min(a[1..5]) = 4$ (1 swap)
2	4	6	18	12	10	$a[2] = \min(a[2..5]) = 6$ (1 swap)
3	4	6	10	18	12	$a[3] = \min(a[3..5]) = 10$ (2 swaps)
4	4	6	10	12	18	$a[4] = \min(a[4..5]) = 12$ (1 swap)

The algorithm performs 10 key comparisons and 5 swaps, and the sorted array is

$[4, 6, 10, 12, 18]$.

3. Array $a = [5, 4, 3, 2, 1]$ ($n = 5$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
—	—	5	4	3	2	1	initial state
1	2	4	5	3	2	1	$a[1] = 5 > a[2] = 4$: swap $a[1] \leftrightarrow a[2]$
	3	3	5	4	2	1	$a[1] = 4 > a[3] = 3$: swap $a[1] \leftrightarrow a[3]$
	4	2	5	4	3	1	$a[1] = 3 > a[4] = 2$: swap $a[1] \leftrightarrow a[4]$
	5	1	5	4	3	2	$a[1] = 2 > a[5] = 1$: swap $a[1] \leftrightarrow a[5]$
2	3	1	4	5	3	2	$a[2] = 5 > a[3] = 4$: swap $a[2] \leftrightarrow a[3]$
	4	1	3	5	4	2	$a[2] = 4 > a[4] = 3$: swap $a[2] \leftrightarrow a[4]$
	5	1	2	5	4	3	$a[2] = 3 > a[5] = 2$: swap $a[2] \leftrightarrow a[5]$
3	4	1	2	4	5	3	$a[3] = 5 > a[4] = 4$: swap $a[3] \leftrightarrow a[4]$
	5	1	2	3	5	4	$a[3] = 4 > a[5] = 3$: swap $a[3] \leftrightarrow a[5]$
4	5	1	2	3	4	5	$a[4] = 5 > a[5] = 4$: swap $a[4] \leftrightarrow a[5]$

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
–	5	4	3	2	1	initial state
1	1	5	4	3	2	$a[1] = \min(a[1..5]) = 1$ (4 swaps)
2	1	2	5	4	3	$a[2] = \min(a[2..5]) = 2$ (3 swaps)
3	1	2	3	5	4	$a[3] = \min(a[3..5]) = 3$ (2 swaps)
4	1	2	3	4	5	$a[4] = \min(a[4..5]) = 4$ (1 swap)

The algorithm performs 10 key comparisons and 10 swaps, and the sorted array is

[1, 2, 3, 4, 5].

4. Array $a = [100, 50, 25]$ ($n = 3$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	Action
–	–	100	50	25	initial state
1	2	50	100	25	$a[1] = 100 > a[2] = 50$: swap $a[1] \leftrightarrow a[2]$
	3	25	100	50	$a[1] = 50 > a[3] = 25$: swap $a[1] \leftrightarrow a[3]$
2	3	25	50	100	$a[2] = 100 > a[3] = 50$: swap $a[2] \leftrightarrow a[3]$

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	Action
–	100	50	25	initial state
1	25	100	50	$a[1] = \min(a[1..3]) = 25$ (2 swaps)
2	25	50	100	$a[2] = \min(a[2..3]) = 50$ (1 swap)

The algorithm performs 3 key comparisons and 3 swaps, and the sorted array is

[25, 50, 100].

5. Array $a = [8]$ ($n = 1$).

The array has a single element: the outer loop is not executed and the array is already sorted.

$a[1]$	Action
8	initial state: $n = 1$, the outer loop is not executed

The sorted array is [8], obtained with no key comparison and no swap.

Exercise 3.12 — Sorting algorithms (E12)

Consider the following arrays of natural numbers:

1. $a = [6, 1, 8, 2, 5]$
2. $a = [13, 11, 9, 7]$
3. $a = [2, 8, 4, 6]$
4. $a = [30, 20, 10]$
5. $a = [5, 10, 15, 20]$

Questions

1. Apply the **ExchangeSort** algorithm to each array, to sort it in non-decreasing order. Show the execution step by step: for every comparison, say whether a swap is performed and give the state of the array after the comparison.
2. For each array, count the key comparisons and the swaps performed by the algorithm.

Solution

The trace tables follow the conventions stated in the introduction to the exercises of this chapter. Each item answers both questions.

1. Array $a = [6, 1, 8, 2, 5]$ ($n = 5$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
—	—	6	1	8	2	5	initial state
1	2	1	6	8	2	5	$a[1] = 6 > a[2] = 1$: swap $a[1] \leftrightarrow a[2]$
	3	1	6	8	2	5	$a[1] = 1 \leq a[3] = 8$: no swap
	4	1	6	8	2	5	$a[1] = 1 \leq a[4] = 2$: no swap
	5	1	6	8	2	5	$a[1] = 1 \leq a[5] = 5$: no swap
2	3	1	6	8	2	5	$a[2] = 6 \leq a[3] = 8$: no swap
	4	1	2	8	6	5	$a[2] = 6 > a[4] = 2$: swap $a[2] \leftrightarrow a[4]$
	5	1	2	8	6	5	$a[2] = 2 \leq a[5] = 5$: no swap
3	4	1	2	6	8	5	$a[3] = 8 > a[4] = 6$: swap $a[3] \leftrightarrow a[4]$
	5	1	2	5	8	6	$a[3] = 6 > a[5] = 5$: swap $a[3] \leftrightarrow a[5]$
4	5	1	2	5	6	8	$a[4] = 8 > a[5] = 6$: swap $a[4] \leftrightarrow a[5]$

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
–	6	1	8	2	5	initial state
1	1	6	8	2	5	$a[1] = \min(a[1..5]) = 1$ (1 swap)
2	1	2	8	6	5	$a[2] = \min(a[2..5]) = 2$ (1 swap)
3	1	2	5	8	6	$a[3] = \min(a[3..5]) = 5$ (2 swaps)
4	1	2	5	6	8	$a[4] = \min(a[4..5]) = 6$ (1 swap)

The algorithm performs 10 key comparisons and 5 swaps, and the sorted array is
 $[1, 2, 5, 6, 8]$.

2. Array $a = [13, 11, 9, 7]$ ($n = 4$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
–	–	13	11	9	7	initial state
1	2	11	13	9	7	$a[1] = 13 > a[2] = 11$: swap $a[1] \leftrightarrow a[2]$
	3	9	13	11	7	$a[1] = 11 > a[3] = 9$: swap $a[1] \leftrightarrow a[3]$
	4	7	13	11	9	$a[1] = 9 > a[4] = 7$: swap $a[1] \leftrightarrow a[4]$
2	3	7	11	13	9	$a[2] = 13 > a[3] = 11$: swap $a[2] \leftrightarrow a[3]$
	4	7	9	13	11	$a[2] = 11 > a[4] = 9$: swap $a[2] \leftrightarrow a[4]$
3	4	7	9	11	13	$a[3] = 13 > a[4] = 11$: swap $a[3] \leftrightarrow a[4]$

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
–	13	11	9	7	initial state
1	7	13	11	9	$a[1] = \min(a[1..4]) = 7$ (3 swaps)
2	7	9	13	11	$a[2] = \min(a[2..4]) = 9$ (2 swaps)
3	7	9	11	13	$a[3] = \min(a[3..4]) = 11$ (1 swap)

The algorithm performs 6 key comparisons and 6 swaps, and the sorted array is
 $[7, 9, 11, 13]$.

3. Array $a = [2, 8, 4, 6]$ ($n = 4$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
–	–	2	8	4	6	initial state
1	2	2	8	4	6	$a[1] = 2 \leq a[2] = 8$: no swap
	3	2	8	4	6	$a[1] = 2 \leq a[3] = 4$: no swap
	4	2	8	4	6	$a[1] = 2 \leq a[4] = 6$: no swap
2	3	2	4	8	6	$a[2] = 8 > a[3] = 4$: swap $a[2] \leftrightarrow a[3]$
	4	2	4	8	6	$a[2] = 4 \leq a[4] = 6$: no swap
3	4	2	4	6	8	$a[3] = 8 > a[4] = 6$: swap $a[3] \leftrightarrow a[4]$

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
–	2	8	4	6	initial state
1	2	8	4	6	$a[1] = \min(a[1..4]) = 2$ (0 swaps)
2	2	4	8	6	$a[2] = \min(a[2..4]) = 4$ (1 swap)
3	2	4	6	8	$a[3] = \min(a[3..4]) = 6$ (1 swap)

The algorithm performs 6 key comparisons and 2 swaps, and the sorted array is

[2, 4, 6, 8].

4. Array $a = [30, 20, 10]$ ($n = 3$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	Action
–	–	30	20	10	initial state
1	2	20	30	10	$a[1] = 30 > a[2] = 20$: swap $a[1] \leftrightarrow a[2]$
	3	10	30	20	$a[1] = 20 > a[3] = 10$: swap $a[1] \leftrightarrow a[3]$
2	3	10	20	30	$a[2] = 30 > a[3] = 20$: swap $a[2] \leftrightarrow a[3]$

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	Action
–	30	20	10	initial state
1	10	30	20	$a[1] = \min(a[1..3]) = 10$ (2 swaps)
2	10	20	30	$a[2] = \min(a[2..3]) = 20$ (1 swap)

The algorithm performs 3 key comparisons and 3 swaps, and the sorted array is

[10, 20, 30].

5. Array $a = [5, 10, 15, 20]$ ($n = 4$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
–	–	5	10	15	20	initial state
1	2	5	10	15	20	$a[1] = 5 \leq a[2] = 10$: no swap
	3	5	10	15	20	$a[1] = 5 \leq a[3] = 15$: no swap
	4	5	10	15	20	$a[1] = 5 \leq a[4] = 20$: no swap
2	3	5	10	15	20	$a[2] = 10 \leq a[3] = 15$: no swap
	4	5	10	15	20	$a[2] = 10 \leq a[4] = 20$: no swap
3	4	5	10	15	20	$a[3] = 15 \leq a[4] = 20$: no swap

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	5	10	15	20	initial state
1	5	10	15	20	$a[1] = \min(a[1..4]) = 5$ (0 swaps)
2	5	10	15	20	$a[2] = \min(a[2..4]) = 10$ (0 swaps)
3	5	10	15	20	$a[3] = \min(a[3..4]) = 15$ (0 swaps)

The algorithm performs 6 key comparisons and 0 swaps, and the sorted array is

[5, 10, 15, 20].

Exercise 3.13 — Sorting algorithms (E13)

Consider the following arrays of natural numbers:

1. $a = [14, 7, 21, 3, 18]$
2. $a = [40, 30, 20, 10]$
3. $a = [3, 6, 1, 9, 2]$
4. $a = [25, 12, 50]$
5. $a = [17, 17, 17, 17]$

Questions

1. Apply the **ExchangeSort** algorithm to each array, to sort it in non-decreasing order. Show the execution step by step: for every comparison, say whether a swap is performed and give the state of the array after the comparison.
2. For each array, count the key comparisons and the swaps performed by the algorithm.

Solution

The trace tables follow the conventions stated in the introduction to the exercises of this chapter. Each item answers both questions.

1. Array $a = [14, 7, 21, 3, 18]$ ($n = 5$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
–	–	14	7	21	3	18	initial state
1	2	7	14	21	3	18	$a[1] = 14 > a[2] = 7$: swap $a[1] \leftrightarrow a[2]$
	3	7	14	21	3	18	$a[1] = 7 \leq a[3] = 21$: no swap
	4	3	14	21	7	18	$a[1] = 7 > a[4] = 3$: swap $a[1] \leftrightarrow a[4]$
	5	3	14	21	7	18	$a[1] = 3 \leq a[5] = 18$: no swap
2	3	3	14	21	7	18	$a[2] = 14 \leq a[3] = 21$: no swap
	4	3	7	21	14	18	$a[2] = 14 > a[4] = 7$: swap $a[2] \leftrightarrow a[4]$
	5	3	7	21	14	18	$a[2] = 7 \leq a[5] = 18$: no swap
3	4	3	7	14	21	18	$a[3] = 21 > a[4] = 14$: swap $a[3] \leftrightarrow a[4]$
	5	3	7	14	21	18	$a[3] = 14 \leq a[5] = 18$: no swap
4	5	3	7	14	18	21	$a[4] = 21 > a[5] = 18$: swap $a[4] \leftrightarrow a[5]$

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
–	14	7	21	3	18	initial state
1	3	14	21	7	18	$a[1] = \min(a[1..5]) = 3$ (2 swaps)
2	3	7	21	14	18	$a[2] = \min(a[2..5]) = 7$ (1 swap)
3	3	7	14	21	18	$a[3] = \min(a[3..5]) = 14$ (1 swap)
4	3	7	14	18	21	$a[4] = \min(a[4..5]) = 18$ (1 swap)

The algorithm performs 10 key comparisons and 5 swaps, and the sorted array is
 $[3, 7, 14, 18, 21]$.

2. Array $a = [40, 30, 20, 10]$ ($n = 4$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
–	–	40	30	20	10	initial state
1	2	30	40	20	10	$a[1] = 40 > a[2] = 30$: swap $a[1] \leftrightarrow a[2]$
	3	20	40	30	10	$a[1] = 30 > a[3] = 20$: swap $a[1] \leftrightarrow a[3]$
	4	10	40	30	20	$a[1] = 20 > a[4] = 10$: swap $a[1] \leftrightarrow a[4]$
2	3	10	30	40	20	$a[2] = 40 > a[3] = 30$: swap $a[2] \leftrightarrow a[3]$
	4	10	20	40	30	$a[2] = 30 > a[4] = 20$: swap $a[2] \leftrightarrow a[4]$
3	4	10	20	30	40	$a[3] = 40 > a[4] = 30$: swap $a[3] \leftrightarrow a[4]$

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
–	40	30	20	10	initial state
1	10	40	30	20	$a[1] = \min(a[1..4]) = 10$ (3 swaps)
2	10	20	40	30	$a[2] = \min(a[2..4]) = 20$ (2 swaps)
3	10	20	30	40	$a[3] = \min(a[3..4]) = 30$ (1 swap)

The algorithm performs 6 key comparisons and 6 swaps, and the sorted array is

[10, 20, 30, 40].

3. Array $a = [3, 6, 1, 9, 2]$ ($n = 5$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
–	–	3	6	1	9	2	initial state
1	2	3	6	1	9	2	$a[1] = 3 \leq a[2] = 6$: no swap
	3	1	6	3	9	2	$a[1] = 3 > a[3] = 1$: swap $a[1] \leftrightarrow a[3]$
	4	1	6	3	9	2	$a[1] = 1 \leq a[4] = 9$: no swap
	5	1	6	3	9	2	$a[1] = 1 \leq a[5] = 2$: no swap
2	3	1	3	6	9	2	$a[2] = 6 > a[3] = 3$: swap $a[2] \leftrightarrow a[3]$
	4	1	3	6	9	2	$a[2] = 3 \leq a[4] = 9$: no swap
	5	1	2	6	9	3	$a[2] = 3 > a[5] = 2$: swap $a[2] \leftrightarrow a[5]$
3	4	1	2	6	9	3	$a[3] = 6 \leq a[4] = 9$: no swap
	5	1	2	3	9	6	$a[3] = 6 > a[5] = 3$: swap $a[3] \leftrightarrow a[5]$
4	5	1	2	3	6	9	$a[4] = 9 > a[5] = 6$: swap $a[4] \leftrightarrow a[5]$

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
–	3	6	1	9	2	initial state
1	1	6	3	9	2	$a[1] = \min(a[1..5]) = 1$ (1 swap)
2	1	2	6	9	3	$a[2] = \min(a[2..5]) = 2$ (2 swaps)
3	1	2	3	9	6	$a[3] = \min(a[3..5]) = 3$ (1 swap)
4	1	2	3	6	9	$a[4] = \min(a[4..5]) = 6$ (1 swap)

The algorithm performs 10 key comparisons and 5 swaps, and the sorted array is

[1, 2, 3, 6, 9].

4. Array $a = [25, 12, 50]$ ($n = 3$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	Action
–	–	25	12	50	initial state
1	2	12	25	50	$a[1] = 25 > a[2] = 12$: swap $a[1] \leftrightarrow a[2]$
	3	12	25	50	$a[1] = 12 \leq a[3] = 50$: no swap
2	3	12	25	50	$a[2] = 25 \leq a[3] = 50$: no swap

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	Action
—	25	12	50	initial state
1	12	25	50	$a[1] = \min(a[1..3]) = 12$ (1 swap)
2	12	25	50	$a[2] = \min(a[2..3]) = 25$ (0 swaps)

The algorithm performs 3 key comparisons and 1 swap, and the sorted array is
 $[12, 25, 50]$.

5. Array $a = [17, 17, 17, 17]$ ($n = 4$).

Detailed trace:

i	j	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	17	17	17	17	initial state
1	2	17	17	17	17	$a[1] = 17 \leq a[2] = 17$: no swap
	3	17	17	17	17	$a[1] = 17 \leq a[3] = 17$: no swap
	4	17	17	17	17	$a[1] = 17 \leq a[4] = 17$: no swap
2	3	17	17	17	17	$a[2] = 17 \leq a[3] = 17$: no swap
	4	17	17	17	17	$a[2] = 17 \leq a[4] = 17$: no swap
3	4	17	17	17	17	$a[3] = 17 \leq a[4] = 17$: no swap

Array after each iteration of the outer loop:

i	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	17	17	17	17	initial state
1	17	17	17	17	$a[1] = \min(a[1..4]) = 17$ (0 swaps)
2	17	17	17	17	$a[2] = \min(a[2..4]) = 17$ (0 swaps)
3	17	17	17	17	$a[3] = \min(a[3..4]) = 17$ (0 swaps)

The algorithm performs 6 key comparisons and 0 swaps, and the sorted array is
 $[17, 17, 17, 17]$.

Exercise 3.14 — Sorting algorithms (E14)

Consider the following arrays of natural numbers:

1. $a = [5, 2, 4, 6, 1, 3]$
2. $a = [8, 3, 7, 4, 9, 2]$
3. $a = [6, 5, 4, 3, 2, 1]$
4. $a = [1, 2, 3, 4, 5, 6]$

5. $a = [10, 5, 15, 3]$

Questions

1. Apply the **SelectionSort** algorithm to each array, to sort it in non-decreasing order. Show the execution step by step: for every iteration of the outer loop, give the index $minIdx$ of the minimum found by the inner loop and say whether a swap is performed.
2. For each array, count the key comparisons and the swaps performed by the algorithm.

Solution

The trace tables follow the conventions stated in the introduction to the exercises of this chapter. Each item answers both questions.

1. Array $a = [5, 2, 4, 6, 1, 3]$ ($n = 6$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	—	—	5	2	4	6	1	3	initial state
1	—	1	5	2	4	6	1	3	$minIdx := 1$ ($a[1] = 5$)
	2	2	5	2	4	6	1	3	$a[2] = 2 < a[1] = 5$: $minIdx \leftarrow 2$
	3	2	5	2	4	6	1	3	$a[3] = 4 \geq a[2] = 2$: no update
	4	2	5	2	4	6	1	3	$a[4] = 6 \geq a[2] = 2$: no update
	5	5	5	2	4	6	1	3	$a[5] = 1 < a[2] = 2$: $minIdx \leftarrow 5$
	6	5	5	2	4	6	1	3	$a[6] = 3 \geq a[5] = 1$: no update
	—	—	1	2	4	6	5	3	$minIdx = 5 \neq 1$: swap $a[1] \leftrightarrow a[5]$
2	—	2	1	2	4	6	5	3	$minIdx := 2$ ($a[2] = 2$)
	3	2	1	2	4	6	5	3	$a[3] = 4 \geq a[2] = 2$: no update
	4	2	1	2	4	6	5	3	$a[4] = 6 \geq a[2] = 2$: no update
	5	2	1	2	4	6	5	3	$a[5] = 5 \geq a[2] = 2$: no update
	6	2	1	2	4	6	5	3	$a[6] = 3 \geq a[2] = 2$: no update
	—	—	1	2	4	6	5	3	$minIdx = 2$: no swap
	—	—	1	2	4	6	5	3	
3	—	3	1	2	4	6	5	3	$minIdx := 3$ ($a[3] = 4$)
	4	3	1	2	4	6	5	3	$a[4] = 6 \geq a[3] = 4$: no update
	5	3	1	2	4	6	5	3	$a[5] = 5 \geq a[3] = 4$: no update
	6	6	1	2	4	6	5	3	$a[6] = 3 < a[3] = 4$: $minIdx \leftarrow 6$
	—	—	1	2	3	6	5	4	$minIdx = 6 \neq 3$: swap $a[3] \leftrightarrow a[6]$
	—	—	1	2	3	6	5	4	
	—	—	1	2	3	6	5	4	
4	—	4	1	2	3	6	5	4	$minIdx := 4$ ($a[4] = 6$)
	5	5	1	2	3	6	5	4	$a[5] = 5 < a[4] = 6$: $minIdx \leftarrow 5$
	6	6	1	2	3	6	5	4	$a[6] = 4 < a[5] = 5$: $minIdx \leftarrow 6$
	—	—	1	2	3	4	5	6	$minIdx = 6 \neq 4$: swap $a[4] \leftrightarrow a[6]$
	—	—	1	2	3	4	5	6	
	—	—	1	2	3	4	5	6	
	—	—	1	2	3	4	5	6	
5	—	5	1	2	3	4	5	6	$minIdx := 5$ ($a[5] = 5$)
	6	5	1	2	3	4	5	6	$a[6] = 6 \geq a[5] = 5$: no update
	—	—	1	2	3	4	5	6	$minIdx = 5$: no swap

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	—	5	2	4	6	1	3	initial state
1	5	1	2	4	6	5	3	swap $a[1] \leftrightarrow a[5]$: $a[1] = \min(a[1..6]) = 1$
2	2	1	2	4	6	5	3	no swap: $a[2] = \min(a[2..6]) = 2$
3	6	1	2	3	6	5	4	swap $a[3] \leftrightarrow a[6]$: $a[3] = \min(a[3..6]) = 3$
4	6	1	2	3	4	5	6	swap $a[4] \leftrightarrow a[6]$: $a[4] = \min(a[4..6]) = 4$
5	5	1	2	3	4	5	6	no swap: $a[5] = \min(a[5..6]) = 5$

The algorithm performs 15 key comparisons and 3 swaps, and the sorted array is

$$[1, 2, 3, 4, 5, 6].$$

2. Array $a = [8, 3, 7, 4, 9, 2]$ ($n = 6$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	—	—	8	3	7	4	9	2	initial state
1	—	1	8	3	7	4	9	2	$minIdx := 1$ ($a[1] = 8$)
	2	2	8	3	7	4	9	2	$a[2] = 3 < a[1] = 8$: $minIdx \leftarrow 2$
	3	2	8	3	7	4	9	2	$a[3] = 7 \geq a[2] = 3$: no update
	4	2	8	3	7	4	9	2	$a[4] = 4 \geq a[2] = 3$: no update
	5	2	8	3	7	4	9	2	$a[5] = 9 \geq a[2] = 3$: no update
	6	6	8	3	7	4	9	2	$a[6] = 2 < a[2] = 3$: $minIdx \leftarrow 6$
	—	—	2	3	7	4	9	8	$minIdx = 6 \neq 1$: swap $a[1] \leftrightarrow a[6]$
2	—	2	2	3	7	4	9	8	$minIdx := 2$ ($a[2] = 3$)
	3	2	2	3	7	4	9	8	$a[3] = 7 \geq a[2] = 3$: no update
	4	2	2	3	7	4	9	8	$a[4] = 4 \geq a[2] = 3$: no update
	5	2	2	3	7	4	9	8	$a[5] = 9 \geq a[2] = 3$: no update
	6	2	2	3	7	4	9	8	$a[6] = 8 \geq a[2] = 3$: no update
	—	—	2	3	7	4	9	8	$minIdx = 2$: no swap
3	—	3	2	3	7	4	9	8	$minIdx := 3$ ($a[3] = 7$)
	4	4	2	3	7	4	9	8	$a[4] = 4 < a[3] = 7$: $minIdx \leftarrow 4$
	5	4	2	3	7	4	9	8	$a[5] = 9 \geq a[4] = 4$: no update
	6	4	2	3	7	4	9	8	$a[6] = 8 \geq a[4] = 4$: no update
	—	—	2	3	4	7	9	8	$minIdx = 4 \neq 3$: swap $a[3] \leftrightarrow a[4]$
4	—	4	2	3	4	7	9	8	$minIdx := 4$ ($a[4] = 7$)
	5	4	2	3	4	7	9	8	$a[5] = 9 \geq a[4] = 7$: no update
	6	4	2	3	4	7	9	8	$a[6] = 8 \geq a[4] = 7$: no update
	—	—	2	3	4	7	9	8	$minIdx = 4$: no swap
5	—	5	2	3	4	7	9	8	$minIdx := 5$ ($a[5] = 9$)
	6	6	2	3	4	7	9	8	$a[6] = 8 < a[5] = 9$: $minIdx \leftarrow 6$
	—	—	2	3	4	7	8	9	$minIdx = 6 \neq 5$: swap $a[5] \leftrightarrow a[6]$

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	—	8	3	7	4	9	2	initial state
1	6	2	3	7	4	9	8	swap $a[1] \leftrightarrow a[6]$: $a[1] = \min(a[1..6]) = 2$
2	2	2	3	7	4	9	8	no swap: $a[2] = \min(a[2..6]) = 3$
3	4	2	3	4	7	9	8	swap $a[3] \leftrightarrow a[4]$: $a[3] = \min(a[3..6]) = 4$
4	4	2	3	4	7	9	8	no swap: $a[4] = \min(a[4..6]) = 7$
5	6	2	3	4	7	8	9	swap $a[5] \leftrightarrow a[6]$: $a[5] = \min(a[5..6]) = 8$

The algorithm performs 15 key comparisons and 3 swaps, and the sorted array is

$$[2, 3, 4, 7, 8, 9].$$

3. Array $a = [6, 5, 4, 3, 2, 1]$ ($n = 6$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	—	—	6	5	4	3	2	1	initial state
1	—	1	6	5	4	3	2	1	$minIdx := 1$ ($a[1] = 6$)
	2	2	6	5	4	3	2	1	$a[2] = 5 < a[1] = 6$: $minIdx \leftarrow 2$
	3	3	6	5	4	3	2	1	$a[3] = 4 < a[2] = 5$: $minIdx \leftarrow 3$
	4	4	6	5	4	3	2	1	$a[4] = 3 < a[3] = 4$: $minIdx \leftarrow 4$
	5	5	6	5	4	3	2	1	$a[5] = 2 < a[4] = 3$: $minIdx \leftarrow 5$
	6	6	6	5	4	3	2	1	$a[6] = 1 < a[5] = 2$: $minIdx \leftarrow 6$
	—	—	1	5	4	3	2	6	$minIdx = 6 \neq 1$: swap $a[1] \leftrightarrow a[6]$
2	—	2	1	5	4	3	2	6	$minIdx := 2$ ($a[2] = 5$)
	3	3	1	5	4	3	2	6	$a[3] = 4 < a[2] = 5$: $minIdx \leftarrow 3$
	4	4	1	5	4	3	2	6	$a[4] = 3 < a[3] = 4$: $minIdx \leftarrow 4$
	5	5	1	5	4	3	2	6	$a[5] = 2 < a[4] = 3$: $minIdx \leftarrow 5$
	6	5	1	5	4	3	2	6	$a[6] = 6 \geq a[5] = 2$: no update
	—	—	1	2	4	3	5	6	$minIdx = 5 \neq 2$: swap $a[2] \leftrightarrow a[5]$
3	—	3	1	2	4	3	5	6	$minIdx := 3$ ($a[3] = 4$)
	4	4	1	2	4	3	5	6	$a[4] = 3 < a[3] = 4$: $minIdx \leftarrow 4$
	5	4	1	2	4	3	5	6	$a[5] = 5 \geq a[4] = 3$: no update
	6	4	1	2	4	3	5	6	$a[6] = 6 \geq a[4] = 3$: no update
	—	—	1	2	3	4	5	6	$minIdx = 4 \neq 3$: swap $a[3] \leftrightarrow a[4]$
4	—	4	1	2	3	4	5	6	$minIdx := 4$ ($a[4] = 4$)
	5	4	1	2	3	4	5	6	$a[5] = 5 \geq a[4] = 4$: no update
	6	4	1	2	3	4	5	6	$a[6] = 6 \geq a[4] = 4$: no update
	—	—	1	2	3	4	5	6	$minIdx = 4$: no swap
5	—	5	1	2	3	4	5	6	$minIdx := 5$ ($a[5] = 5$)
	6	5	1	2	3	4	5	6	$a[6] = 6 \geq a[5] = 5$: no update
	—	—	1	2	3	4	5	6	$minIdx = 5$: no swap

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	—	6	5	4	3	2	1	initial state
1	6	1	5	4	3	2	6	swap $a[1] \leftrightarrow a[6]$: $a[1] = \min(a[1..6]) = 1$
2	5	1	2	4	3	5	6	swap $a[2] \leftrightarrow a[5]$: $a[2] = \min(a[2..6]) = 2$
3	4	1	2	3	4	5	6	swap $a[3] \leftrightarrow a[4]$: $a[3] = \min(a[3..6]) = 3$
4	4	1	2	3	4	5	6	no swap: $a[4] = \min(a[4..6]) = 4$
5	5	1	2	3	4	5	6	no swap: $a[5] = \min(a[5..6]) = 5$

The algorithm performs 15 key comparisons and 3 swaps, and the sorted array is

$$[1, 2, 3, 4, 5, 6].$$

4. Array $a = [1, 2, 3, 4, 5, 6]$ ($n = 6$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	—	—	1	2	3	4	5	6	initial state
1	—	1	1	2	3	4	5	6	$minIdx := 1$ ($a[1] = 1$)
	2	1	1	2	3	4	5	6	$a[2] = 2 \geq a[1] = 1$: no update
	3	1	1	2	3	4	5	6	$a[3] = 3 \geq a[1] = 1$: no update
	4	1	1	2	3	4	5	6	$a[4] = 4 \geq a[1] = 1$: no update
	5	1	1	2	3	4	5	6	$a[5] = 5 \geq a[1] = 1$: no update
	6	1	1	2	3	4	5	6	$a[6] = 6 \geq a[1] = 1$: no update
	—	—	1	2	3	4	5	6	$minIdx = 1$: no swap
2	—	2	1	2	3	4	5	6	$minIdx := 2$ ($a[2] = 2$)
	3	2	1	2	3	4	5	6	$a[3] = 3 \geq a[2] = 2$: no update
	4	2	1	2	3	4	5	6	$a[4] = 4 \geq a[2] = 2$: no update
	5	2	1	2	3	4	5	6	$a[5] = 5 \geq a[2] = 2$: no update
	6	2	1	2	3	4	5	6	$a[6] = 6 \geq a[2] = 2$: no update
	—	—	1	2	3	4	5	6	$minIdx = 2$: no swap
3	—	3	1	2	3	4	5	6	$minIdx := 3$ ($a[3] = 3$)
	4	3	1	2	3	4	5	6	$a[4] = 4 \geq a[3] = 3$: no update
	5	3	1	2	3	4	5	6	$a[5] = 5 \geq a[3] = 3$: no update
	6	3	1	2	3	4	5	6	$a[6] = 6 \geq a[3] = 3$: no update
	—	—	1	2	3	4	5	6	$minIdx = 3$: no swap
4	—	4	1	2	3	4	5	6	$minIdx := 4$ ($a[4] = 4$)
	5	4	1	2	3	4	5	6	$a[5] = 5 \geq a[4] = 4$: no update
	6	4	1	2	3	4	5	6	$a[6] = 6 \geq a[4] = 4$: no update
	—	—	1	2	3	4	5	6	$minIdx = 4$: no swap
5	—	5	1	2	3	4	5	6	$minIdx := 5$ ($a[5] = 5$)
	6	5	1	2	3	4	5	6	$a[6] = 6 \geq a[5] = 5$: no update
	—	—	1	2	3	4	5	6	$minIdx = 5$: no swap

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	Action
—	—	1	2	3	4	5	6	initial state
1	1	1	2	3	4	5	6	no swap: $a[1] = \min(a[1..6]) = 1$
2	2	1	2	3	4	5	6	no swap: $a[2] = \min(a[2..6]) = 2$
3	3	1	2	3	4	5	6	no swap: $a[3] = \min(a[3..6]) = 3$
4	4	1	2	3	4	5	6	no swap: $a[4] = \min(a[4..6]) = 4$
5	5	1	2	3	4	5	6	no swap: $a[5] = \min(a[5..6]) = 5$

The algorithm performs 15 key comparisons and 0 swaps, and the sorted array is

$$[1, 2, 3, 4, 5, 6].$$

5. Array $a = [10, 5, 15, 3]$ ($n = 4$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	—	10	5	15	3	initial state
1	—	1	10	5	15	3	$minIdx := 1$ ($a[1] = 10$)
	2	2	10	5	15	3	$a[2] = 5 < a[1] = 10$: $minIdx \leftarrow 2$
	3	2	10	5	15	3	$a[3] = 15 \geq a[2] = 5$: no update
	4	4	10	5	15	3	$a[4] = 3 < a[2] = 5$: $minIdx \leftarrow 4$
	—	—	3	5	15	10	$minIdx = 4 \neq 1$: swap $a[1] \leftrightarrow a[4]$
2	—	2	3	5	15	10	$minIdx := 2$ ($a[2] = 5$)
	3	2	3	5	15	10	$a[3] = 15 \geq a[2] = 5$: no update
	4	2	3	5	15	10	$a[4] = 10 \geq a[2] = 5$: no update
	—	—	3	5	15	10	$minIdx = 2$: no swap
3	—	3	3	5	15	10	$minIdx := 3$ ($a[3] = 15$)
	4	4	3	5	15	10	$a[4] = 10 < a[3] = 15$: $minIdx \leftarrow 4$
	—	—	3	5	10	15	$minIdx = 4 \neq 3$: swap $a[3] \leftrightarrow a[4]$

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	10	5	15	3	initial state
1	4	3	5	15	10	swap $a[1] \leftrightarrow a[4]$: $a[1] = \min(a[1..4]) = 3$
2	2	3	5	15	10	no swap: $a[2] = \min(a[2..4]) = 5$
3	4	3	5	10	15	swap $a[3] \leftrightarrow a[4]$: $a[3] = \min(a[3..4]) = 10$

The algorithm performs 6 key comparisons and 2 swaps, and the sorted array is

$$[3, 5, 10, 15].$$

Exercise 3.15 — Sorting algorithms (E15)

Consider the following arrays of natural numbers:

1. $a = [7, 2, 9, 4]$
2. $a = [15, 10, 5]$
3. $a = [3, 1, 4, 1, 5, 9, 2, 6]$
4. $a = [20, 18, 16, 14]$
5. $a = [11, 11, 11]$

Questions

1. Apply the **SelectionSort** algorithm to each array, to sort it in non-decreasing order. Show the execution step by step: for every iteration of the outer loop, give the index $minIdx$ of the minimum found by the inner loop and say whether a swap is performed.
2. For each array, count the key comparisons and the swaps performed by the algorithm.

Solution

The trace tables follow the conventions stated in the introduction to the exercises of this chapter. Each item answers both questions.

1. Array $a = [7, 2, 9, 4]$ ($n = 4$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	—	7	2	9	4	initial state
1	—	1	7	2	9	4	$minIdx := 1$ ($a[1] = 7$)
	2	2	7	2	9	4	$a[2] = 2 < a[1] = 7$: $minIdx \leftarrow 2$
	3	2	7	2	9	4	$a[3] = 9 \geq a[2] = 2$: no update
	4	2	7	2	9	4	$a[4] = 4 \geq a[2] = 2$: no update
	—	—	2	7	9	4	$minIdx = 2 \neq 1$: swap $a[1] \leftrightarrow a[2]$
2	—	2	2	7	9	4	$minIdx := 2$ ($a[2] = 7$)
	3	2	2	7	9	4	$a[3] = 9 \geq a[2] = 7$: no update
	4	4	2	7	9	4	$a[4] = 4 < a[2] = 7$: $minIdx \leftarrow 4$
	—	—	2	4	9	7	$minIdx = 4 \neq 2$: swap $a[2] \leftrightarrow a[4]$
3	—	3	2	4	9	7	$minIdx := 3$ ($a[3] = 9$)
	4	4	2	4	9	7	$a[4] = 7 < a[3] = 9$: $minIdx \leftarrow 4$
	—	—	2	4	7	9	$minIdx = 4 \neq 3$: swap $a[3] \leftrightarrow a[4]$

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	7	2	9	4	initial state
1	2	2	7	9	4	swap $a[1] \leftrightarrow a[2]$: $a[1] = \min(a[1..4]) = 2$
2	4	2	4	9	7	swap $a[2] \leftrightarrow a[4]$: $a[2] = \min(a[2..4]) = 4$
3	4	2	4	7	9	swap $a[3] \leftrightarrow a[4]$: $a[3] = \min(a[3..4]) = 7$

The algorithm performs 6 key comparisons and 3 swaps, and the sorted array is

$$[2, 4, 7, 9].$$

2. Array $a = [15, 10, 5]$ ($n = 3$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	Action
—	—	—	15	10	5	initial state
1	—	1	15	10	5	$minIdx := 1$ ($a[1] = 15$)
	2	2	15	10	5	$a[2] = 10 < a[1] = 15$: $minIdx \leftarrow 2$
	3	3	15	10	5	$a[3] = 5 < a[2] = 10$: $minIdx \leftarrow 3$
	—	—	5	10	15	$minIdx = 3 \neq 1$: swap $a[1] \leftrightarrow a[3]$
2	—	2	5	10	15	$minIdx := 2$ ($a[2] = 10$)
	3	2	5	10	15	$a[3] = 15 \geq a[2] = 10$: no update
	—	—	5	10	15	$minIdx = 2$: no swap

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	Action
—	—	15	10	5	initial state
1	3	5	10	15	swap $a[1] \leftrightarrow a[3]$: $a[1] = \min(a[1..3]) = 5$
2	2	5	10	15	no swap: $a[2] = \min(a[2..3]) = 10$

The algorithm performs 3 key comparisons and 1 swap, and the sorted array is

$$[5, 10, 15].$$

3. Array $a = [3, 1, 4, 1, 5, 9, 2, 6]$ ($n = 8$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	$a[7]$	$a[8]$	Action
—	—	—	3	1	4	1	5	9	2	6	initial state
1	—	1	3	1	4	1	5	9	2	6	$minIdx := 1$ ($a[1] = 3$)
	2	2	3	1	4	1	5	9	2	6	$a[2] = 1 < a[1] = 3$: $minIdx \leftarrow 2$
	3	2	3	1	4	1	5	9	2	6	$a[3] = 4 \geq a[2] = 1$: no update
	4	2	3	1	4	1	5	9	2	6	$a[4] = 1 \geq a[2] = 1$: no update
	5	2	3	1	4	1	5	9	2	6	$a[5] = 5 \geq a[2] = 1$: no update
	6	2	3	1	4	1	5	9	2	6	$a[6] = 9 \geq a[2] = 1$: no update
	7	2	3	1	4	1	5	9	2	6	$a[7] = 2 \geq a[2] = 1$: no update
	8	2	3	1	4	1	5	9	2	6	$a[8] = 6 \geq a[2] = 1$: no update
	—	—	1	3	4	1	5	9	2	6	$minIdx = 2 \neq 1$: swap $a[1] \leftrightarrow a[2]$
2	—	2	1	3	4	1	5	9	2	6	$minIdx := 2$ ($a[2] = 3$)
	3	2	1	3	4	1	5	9	2	6	$a[3] = 4 \geq a[2] = 3$: no update
	4	4	1	3	4	1	5	9	2	6	$a[4] = 1 < a[2] = 3$: $minIdx \leftarrow 4$
	5	4	1	3	4	1	5	9	2	6	$a[5] = 5 \geq a[4] = 1$: no update
	6	4	1	3	4	1	5	9	2	6	$a[6] = 9 \geq a[4] = 1$: no update
	7	4	1	3	4	1	5	9	2	6	$a[7] = 2 \geq a[4] = 1$: no update
	8	4	1	3	4	1	5	9	2	6	$a[8] = 6 \geq a[4] = 1$: no update
	—	—	1	1	4	3	5	9	2	6	$minIdx = 4 \neq 2$: swap $a[2] \leftrightarrow a[4]$
3	—	3	1	1	4	3	5	9	2	6	$minIdx := 3$ ($a[3] = 4$)
	4	4	1	1	4	3	5	9	2	6	$a[4] = 3 < a[3] = 4$: $minIdx \leftarrow 4$
	5	4	1	1	4	3	5	9	2	6	$a[5] = 5 \geq a[4] = 3$: no update
	6	4	1	1	4	3	5	9	2	6	$a[6] = 9 \geq a[4] = 3$: no update
	7	7	1	1	4	3	5	9	2	6	$a[7] = 2 < a[4] = 3$: $minIdx \leftarrow 7$
	8	7	1	1	4	3	5	9	2	6	$a[8] = 6 \geq a[7] = 2$: no update
	—	—	1	1	2	3	5	9	4	6	$minIdx = 7 \neq 3$: swap $a[3] \leftrightarrow a[7]$
4	—	4	1	1	2	3	5	9	4	6	$minIdx := 4$ ($a[4] = 3$)
	5	4	1	1	2	3	5	9	4	6	$a[5] = 5 \geq a[4] = 3$: no update
	6	4	1	1	2	3	5	9	4	6	$a[6] = 9 \geq a[4] = 3$: no update
	7	4	1	1	2	3	5	9	4	6	$a[7] = 4 \geq a[4] = 3$: no update
	8	4	1	1	2	3	5	9	4	6	$a[8] = 6 \geq a[4] = 3$: no update
	—	—	1	1	2	3	5	9	4	6	$minIdx = 4$: no swap
5	—	5	1	1	2	3	5	9	4	6	$minIdx := 5$ ($a[5] = 5$)
	6	5	1	1	2	3	5	9	4	6	$a[6] = 9 \geq a[5] = 5$: no update
	7	7	1	1	2	3	5	9	4	6	$a[7] = 4 < a[5] = 5$: $minIdx \leftarrow 7$
	8	7	1	1	2	3	5	9	4	6	$a[8] = 6 \geq a[7] = 4$: no update
	—	—	1	1	2	3	4	9	5	6	$minIdx = 7 \neq 5$: swap $a[5] \leftrightarrow a[7]$
6	—	6	1	1	2	3	4	9	5	6	$minIdx := 6$ ($a[6] = 9$)
	7	7	1	1	2	3	4	9	5	6	$a[7] = 5 < a[6] = 9$: $minIdx \leftarrow 7$
	8	7	1	1	2	3	4	9	5	6	$a[8] = 6 \geq a[7] = 5$: no update
	—	—	1	1	2	3	4	5	9	6	$minIdx = 7 \neq 6$: swap $a[6] \leftrightarrow a[7]$
7	—	7	1	1	2	3	4	5	9	6	$minIdx := 7$ ($a[7] = 9$)
	8	8	1	1	2	3	4	5	9	6	$a[8] = 6 < a[7] = 9$: $minIdx \leftarrow 8$
	—	—	1	1	2	3	4	5	6	9	$minIdx = 8 \neq 7$: swap $a[7] \leftrightarrow a[8]$

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	$a[7]$	$a[8]$	Action
—	—	3	1	4	1	5	9	2	6	initial state
1	2	1	3	4	1	5	9	2	6	swap $a[1] \leftrightarrow a[2]$: $a[1] = \min(a[1..8]) = 1$
2	4	1	1	4	3	5	9	2	6	swap $a[2] \leftrightarrow a[4]$: $a[2] = \min(a[2..8]) = 1$
3	7	1	1	2	3	5	9	4	6	swap $a[3] \leftrightarrow a[7]$: $a[3] = \min(a[3..8]) = 2$
4	4	1	1	2	3	5	9	4	6	no swap: $a[4] = \min(a[4..8]) = 3$
5	7	1	1	2	3	4	9	5	6	swap $a[5] \leftrightarrow a[7]$: $a[5] = \min(a[5..8]) = 4$
6	7	1	1	2	3	4	5	9	6	swap $a[6] \leftrightarrow a[7]$: $a[6] = \min(a[6..8]) = 5$
7	8	1	1	2	3	4	5	6	9	swap $a[7] \leftrightarrow a[8]$: $a[7] = \min(a[7..8]) = 6$

The algorithm performs 28 key comparisons and 6 swaps, and the sorted array is

$$[1, 1, 2, 3, 4, 5, 6, 9].$$

4. Array $a = [20, 18, 16, 14]$ ($n = 4$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	—	20	18	16	14	initial state
1	—	1	20	18	16	14	$minIdx := 1$ ($a[1] = 20$)
	2	2	20	18	16	14	$a[2] = 18 < a[1] = 20$: $minIdx \leftarrow 2$
	3	3	20	18	16	14	$a[3] = 16 < a[2] = 18$: $minIdx \leftarrow 3$
	4	4	20	18	16	14	$a[4] = 14 < a[3] = 16$: $minIdx \leftarrow 4$
	—	—	14	18	16	20	$minIdx = 4 \neq 1$: swap $a[1] \leftrightarrow a[4]$
2	—	2	14	18	16	20	$minIdx := 2$ ($a[2] = 18$)
	3	3	14	18	16	20	$a[3] = 16 < a[2] = 18$: $minIdx \leftarrow 3$
	4	3	14	18	16	20	$a[4] = 20 \geq a[3] = 16$: no update
	—	—	14	16	18	20	$minIdx = 3 \neq 2$: swap $a[2] \leftrightarrow a[3]$
3	—	3	14	16	18	20	$minIdx := 3$ ($a[3] = 18$)
	4	3	14	16	18	20	$a[4] = 20 \geq a[3] = 18$: no update
	—	—	14	16	18	20	$minIdx = 3$: no swap

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	20	18	16	14	initial state
1	4	14	18	16	20	swap $a[1] \leftrightarrow a[4]$: $a[1] = \min(a[1..4]) = 14$
2	3	14	16	18	20	swap $a[2] \leftrightarrow a[3]$: $a[2] = \min(a[2..4]) = 16$
3	3	14	16	18	20	no swap: $a[3] = \min(a[3..4]) = 18$

The algorithm performs 6 key comparisons and 2 swaps, and the sorted array is

$$[14, 16, 18, 20].$$

5. Array $a = [11, 11, 11]$ ($n = 3$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	Action
–	–	–	11	11	11	initial state
1	–	1	11	11	11	$minIdx := 1$ ($a[1] = 11$)
	2	1	11	11	11	$a[2] = 11 \geq a[1] = 11$: no update
	3	1	11	11	11	$a[3] = 11 \geq a[1] = 11$: no update
	–	–	11	11	11	$minIdx = 1$: no swap
2	–	2	11	11	11	$minIdx := 2$ ($a[2] = 11$)
	3	2	11	11	11	$a[3] = 11 \geq a[2] = 11$: no update
	–	–	11	11	11	$minIdx = 2$: no swap

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	Action
–	–	11	11	11	initial state
1	1	11	11	11	no swap: $a[1] = \min(a[1..3]) = 11$
2	2	11	11	11	no swap: $a[2] = \min(a[2..3]) = 11$

The algorithm performs 3 key comparisons and 0 swaps, and the sorted array is

[11, 11, 11].

Exercise 3.16 — Sorting algorithms (E16)

Consider the following arrays of natural numbers:

1. $a = [9, 3, 7, 2]$
2. $a = [12, 4, 18, 6, 10]$
3. $a = [5, 4, 3, 2, 1]$
4. $a = [100, 50, 25]$
5. $a = [8]$

Questions

1. Apply the **SelectionSort** algorithm to each array, to sort it in non-decreasing order. Show the execution step by step: for every iteration of the outer loop, give the index $minIdx$ of the minimum found by the inner loop and say whether a swap is performed.
2. For each array, count the key comparisons and the swaps performed by the algorithm.

Solution

The trace tables follow the conventions stated in the introduction to the exercises of this chapter. Each item answers both questions.

1. Array $a = [9, 3, 7, 2]$ ($n = 4$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	—	9	3	7	2	initial state
1	—	1	9	3	7	2	$minIdx := 1$ ($a[1] = 9$)
	2	2	9	3	7	2	$a[2] = 3 < a[1] = 9$: $minIdx \leftarrow 2$
	3	2	9	3	7	2	$a[3] = 7 \geq a[2] = 3$: no update
	4	4	9	3	7	2	$a[4] = 2 < a[2] = 3$: $minIdx \leftarrow 4$
	—	—	2	3	7	9	$minIdx = 4 \neq 1$: swap $a[1] \leftrightarrow a[4]$
2	—	2	2	3	7	9	$minIdx := 2$ ($a[2] = 3$)
	3	2	2	3	7	9	$a[3] = 7 \geq a[2] = 3$: no update
	4	2	2	3	7	9	$a[4] = 9 \geq a[2] = 3$: no update
	—	—	2	3	7	9	$minIdx = 2$: no swap
3	—	3	2	3	7	9	$minIdx := 3$ ($a[3] = 7$)
	4	3	2	3	7	9	$a[4] = 9 \geq a[3] = 7$: no update
	—	—	2	3	7	9	$minIdx = 3$: no swap

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	9	3	7	2	initial state
1	4	2	3	7	9	swap $a[1] \leftrightarrow a[4]$: $a[1] = \min(a[1..4]) = 2$
2	2	2	3	7	9	no swap: $a[2] = \min(a[2..4]) = 3$
3	3	2	3	7	9	no swap: $a[3] = \min(a[3..4]) = 7$

The algorithm performs 6 key comparisons and 1 swap, and the sorted array is

$[2, 3, 7, 9]$.

2. Array $a = [12, 4, 18, 6, 10]$ ($n = 5$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
—	—	—	12	4	18	6	10	initial state
1	—	1	12	4	18	6	10	$minIdx := 1$ ($a[1] = 12$)
	2	2	12	4	18	6	10	$a[2] = 4 < a[1] = 12$: $minIdx \leftarrow 2$
	3	2	12	4	18	6	10	$a[3] = 18 \geq a[2] = 4$: no update
	4	2	12	4	18	6	10	$a[4] = 6 \geq a[2] = 4$: no update
	5	2	12	4	18	6	10	$a[5] = 10 \geq a[2] = 4$: no update
	—	—	4	12	18	6	10	$minIdx = 2 \neq 1$: swap $a[1] \leftrightarrow a[2]$
2	—	2	4	12	18	6	10	$minIdx := 2$ ($a[2] = 12$)
	3	2	4	12	18	6	10	$a[3] = 18 \geq a[2] = 12$: no update
	4	4	4	12	18	6	10	$a[4] = 6 < a[2] = 12$: $minIdx \leftarrow 4$
	5	4	4	12	18	6	10	$a[5] = 10 \geq a[4] = 6$: no update
	—	—	4	6	18	12	10	$minIdx = 4 \neq 2$: swap $a[2] \leftrightarrow a[4]$
3	—	3	4	6	18	12	10	$minIdx := 3$ ($a[3] = 18$)
	4	4	4	6	18	12	10	$a[4] = 12 < a[3] = 18$: $minIdx \leftarrow 4$
	5	5	4	6	18	12	10	$a[5] = 10 < a[4] = 12$: $minIdx \leftarrow 5$
	—	—	4	6	10	12	18	$minIdx = 5 \neq 3$: swap $a[3] \leftrightarrow a[5]$
4	—	4	4	6	10	12	18	$minIdx := 4$ ($a[4] = 12$)
	5	4	4	6	10	12	18	$a[5] = 18 \geq a[4] = 12$: no update
	—	—	4	6	10	12	18	$minIdx = 4$: no swap

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
—	—	12	4	18	6	10	initial state
1	2	4	12	18	6	10	swap $a[1] \leftrightarrow a[2]$: $a[1] = \min(a[1..5]) = 4$
2	4	4	6	18	12	10	swap $a[2] \leftrightarrow a[4]$: $a[2] = \min(a[2..5]) = 6$
3	5	4	6	10	12	18	swap $a[3] \leftrightarrow a[5]$: $a[3] = \min(a[3..5]) = 10$
4	4	4	6	10	12	18	no swap: $a[4] = \min(a[4..5]) = 12$

The algorithm performs 10 key comparisons and 3 swaps, and the sorted array is

$$[4, 6, 10, 12, 18].$$

3. Array $a = [5, 4, 3, 2, 1]$ ($n = 5$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
—	—	—	5	4	3	2	1	initial state
1	—	1	5	4	3	2	1	$minIdx := 1$ ($a[1] = 5$)
	2	2	5	4	3	2	1	$a[2] = 4 < a[1] = 5$: $minIdx \leftarrow 2$
	3	3	5	4	3	2	1	$a[3] = 3 < a[2] = 4$: $minIdx \leftarrow 3$
	4	4	5	4	3	2	1	$a[4] = 2 < a[3] = 3$: $minIdx \leftarrow 4$
	5	5	5	4	3	2	1	$a[5] = 1 < a[4] = 2$: $minIdx \leftarrow 5$
	—	—	1	4	3	2	5	$minIdx = 5 \neq 1$: swap $a[1] \leftrightarrow a[5]$
2	—	2	1	4	3	2	5	$minIdx := 2$ ($a[2] = 4$)
	3	3	1	4	3	2	5	$a[3] = 3 < a[2] = 4$: $minIdx \leftarrow 3$
	4	4	1	4	3	2	5	$a[4] = 2 < a[3] = 3$: $minIdx \leftarrow 4$
	5	4	1	4	3	2	5	$a[5] = 5 \geq a[4] = 2$: no update
	—	—	1	2	3	4	5	$minIdx = 4 \neq 2$: swap $a[2] \leftrightarrow a[4]$
	—	—	1	2	3	4	5	
3	—	3	1	2	3	4	5	$minIdx := 3$ ($a[3] = 3$)
	4	3	1	2	3	4	5	$a[4] = 4 \geq a[3] = 3$: no update
	5	3	1	2	3	4	5	$a[5] = 5 \geq a[3] = 3$: no update
	—	—	1	2	3	4	5	$minIdx = 3$: no swap
	—	—	1	2	3	4	5	
4	—	4	1	2	3	4	5	$minIdx := 4$ ($a[4] = 4$)
	5	4	1	2	3	4	5	$a[5] = 5 \geq a[4] = 4$: no update
	—	—	1	2	3	4	5	$minIdx = 4$: no swap

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
—	—	5	4	3	2	1	initial state
1	5	1	4	3	2	5	swap $a[1] \leftrightarrow a[5]$: $a[1] = \min(a[1..5]) = 1$
2	4	1	2	3	4	5	swap $a[2] \leftrightarrow a[4]$: $a[2] = \min(a[2..5]) = 2$
3	3	1	2	3	4	5	no swap: $a[3] = \min(a[3..5]) = 3$
4	4	1	2	3	4	5	no swap: $a[4] = \min(a[4..5]) = 4$

The algorithm performs 10 key comparisons and 2 swaps, and the sorted array is

$$[1, 2, 3, 4, 5].$$

4. Array $a = [100, 50, 25]$ ($n = 3$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	Action
—	—	—	100	50	25	initial state
1	—	1	100	50	25	$minIdx := 1$ ($a[1] = 100$)
	2	2	100	50	25	$a[2] = 50 < a[1] = 100$: $minIdx \leftarrow 2$
	3	3	100	50	25	$a[3] = 25 < a[2] = 50$: $minIdx \leftarrow 3$
	—	—	25	50	100	$minIdx = 3 \neq 1$: swap $a[1] \leftrightarrow a[3]$
	—	—	25	50	100	
2	—	2	25	50	100	$minIdx := 2$ ($a[2] = 50$)
	3	2	25	50	100	$a[3] = 100 \geq a[2] = 50$: no update
	—	—	25	50	100	$minIdx = 2$: no swap

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	Action
—	—	100	50	25	initial state
1	3	25	50	100	swap $a[1] \leftrightarrow a[3]$: $a[1] = \min(a[1..3]) = 25$
2	2	25	50	100	no swap: $a[2] = \min(a[2..3]) = 50$

The algorithm performs 3 key comparisons and 1 swap, and the sorted array is
[25, 50, 100].

5. Array $a = [8]$ ($n = 1$).

The array has a single element: the outer loop is not executed and the array is already sorted.

$a[1]$	Action
8	initial state: $n = 1$, the outer loop is not executed

The sorted array is [8], obtained with no key comparison and no swap.

Exercise 3.17 — Sorting algorithms (E17)

Consider the following arrays of natural numbers:

1. $a = [6, 1, 8, 2, 5]$
2. $a = [13, 11, 9, 7]$
3. $a = [2, 8, 4, 6]$
4. $a = [30, 20, 10]$
5. $a = [5, 10, 15, 20]$

Questions

1. Apply the **SelectionSort** algorithm to each array, to sort it in non-decreasing order. Show the execution step by step: for every iteration of the outer loop, give the index $minIdx$ of the minimum found by the inner loop and say whether a swap is performed.
2. For each array, count the key comparisons and the swaps performed by the algorithm.

Solution

The trace tables follow the conventions stated in the introduction to the exercises of this chapter. Each item answers both questions.

1. Array $a = [6, 1, 8, 2, 5]$ ($n = 5$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
—	—	—	6	1	8	2	5	initial state
1	—	1	6	1	8	2	5	$minIdx := 1$ ($a[1] = 6$)
	2	2	6	1	8	2	5	$a[2] = 1 < a[1] = 6$: $minIdx \leftarrow 2$
	3	2	6	1	8	2	5	$a[3] = 8 \geq a[2] = 1$: no update
	4	2	6	1	8	2	5	$a[4] = 2 \geq a[2] = 1$: no update
	5	2	6	1	8	2	5	$a[5] = 5 \geq a[2] = 1$: no update
	—	—	1	6	8	2	5	$minIdx = 2 \neq 1$: swap $a[1] \leftrightarrow a[2]$
2	—	2	1	6	8	2	5	$minIdx := 2$ ($a[2] = 6$)
	3	2	1	6	8	2	5	$a[3] = 8 \geq a[2] = 6$: no update
	4	4	1	6	8	2	5	$a[4] = 2 < a[2] = 6$: $minIdx \leftarrow 4$
	5	4	1	6	8	2	5	$a[5] = 5 \geq a[4] = 2$: no update
	—	—	1	2	8	6	5	$minIdx = 4 \neq 2$: swap $a[2] \leftrightarrow a[4]$
	—	—	1	2	8	6	5	
3	—	3	1	2	8	6	5	$minIdx := 3$ ($a[3] = 8$)
	4	4	1	2	8	6	5	$a[4] = 6 < a[3] = 8$: $minIdx \leftarrow 4$
	5	5	1	2	8	6	5	$a[5] = 5 < a[4] = 6$: $minIdx \leftarrow 5$
	—	—	1	2	5	6	8	$minIdx = 5 \neq 3$: swap $a[3] \leftrightarrow a[5]$
4	—	4	1	2	5	6	8	$minIdx := 4$ ($a[4] = 6$)
	5	4	1	2	5	6	8	$a[5] = 8 \geq a[4] = 6$: no update
	—	—	1	2	5	6	8	$minIdx = 4$: no swap

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
—	—	6	1	8	2	5	initial state
1	2	1	6	8	2	5	swap $a[1] \leftrightarrow a[2]$: $a[1] = \min(a[1..5]) = 1$
2	4	1	2	8	6	5	swap $a[2] \leftrightarrow a[4]$: $a[2] = \min(a[2..5]) = 2$
3	5	1	2	5	6	8	swap $a[3] \leftrightarrow a[5]$: $a[3] = \min(a[3..5]) = 5$
4	4	1	2	5	6	8	no swap: $a[4] = \min(a[4..5]) = 6$

The algorithm performs 10 key comparisons and 3 swaps, and the sorted array is

$$[1, 2, 5, 6, 8].$$

2. Array $a = [13, 11, 9, 7]$ ($n = 4$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	—	13	11	9	7	initial state
1	—	1	13	11	9	7	$minIdx := 1$ ($a[1] = 13$)
	2	2	13	11	9	7	$a[2] = 11 < a[1] = 13$: $minIdx \leftarrow 2$
	3	3	13	11	9	7	$a[3] = 9 < a[2] = 11$: $minIdx \leftarrow 3$
	4	4	13	11	9	7	$a[4] = 7 < a[3] = 9$: $minIdx \leftarrow 4$
	—	—	7	11	9	13	$minIdx = 4 \neq 1$: swap $a[1] \leftrightarrow a[4]$
2	—	2	7	11	9	13	$minIdx := 2$ ($a[2] = 11$)
	3	3	7	11	9	13	$a[3] = 9 < a[2] = 11$: $minIdx \leftarrow 3$
	4	3	7	11	9	13	$a[4] = 13 \geq a[3] = 9$: no update
	—	—	7	9	11	13	$minIdx = 3 \neq 2$: swap $a[2] \leftrightarrow a[3]$
3	—	3	7	9	11	13	$minIdx := 3$ ($a[3] = 11$)
	4	3	7	9	11	13	$a[4] = 13 \geq a[3] = 11$: no update
	—	—	7	9	11	13	$minIdx = 3$: no swap

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	13	11	9	7	initial state
1	4	7	11	9	13	swap $a[1] \leftrightarrow a[4]$: $a[1] = \min(a[1..4]) = 7$
2	3	7	9	11	13	swap $a[2] \leftrightarrow a[3]$: $a[2] = \min(a[2..4]) = 9$
3	3	7	9	11	13	no swap: $a[3] = \min(a[3..4]) = 11$

The algorithm performs 6 key comparisons and 2 swaps, and the sorted array is

[7, 9, 11, 13].

3. Array $a = [2, 8, 4, 6]$ ($n = 4$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	—	2	8	4	6	initial state
1	—	1	2	8	4	6	$minIdx := 1$ ($a[1] = 2$)
	2	1	2	8	4	6	$a[2] = 8 \geq a[1] = 2$: no update
	3	1	2	8	4	6	$a[3] = 4 \geq a[1] = 2$: no update
	4	1	2	8	4	6	$a[4] = 6 \geq a[1] = 2$: no update
	—	—	2	8	4	6	$minIdx = 1$: no swap
2	—	2	2	8	4	6	$minIdx := 2$ ($a[2] = 8$)
	3	3	2	8	4	6	$a[3] = 4 < a[2] = 8$: $minIdx \leftarrow 3$
	4	3	2	8	4	6	$a[4] = 6 \geq a[3] = 4$: no update
	—	—	2	4	8	6	$minIdx = 3 \neq 2$: swap $a[2] \leftrightarrow a[3]$
3	—	3	2	4	8	6	$minIdx := 3$ ($a[3] = 8$)
	4	4	2	4	8	6	$a[4] = 6 < a[3] = 8$: $minIdx \leftarrow 4$
	—	—	2	4	6	8	$minIdx = 4 \neq 3$: swap $a[3] \leftrightarrow a[4]$

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	2	8	4	6	initial state
1	1	2	8	4	6	no swap: $a[1] = \min(a[1..4]) = 2$
2	3	2	4	8	6	swap $a[2] \leftrightarrow a[3]$: $a[2] = \min(a[2..4]) = 4$
3	4	2	4	6	8	swap $a[3] \leftrightarrow a[4]$: $a[3] = \min(a[3..4]) = 6$

The algorithm performs 6 key comparisons and 2 swaps, and the sorted array is

[2, 4, 6, 8].

4. Array $a = [30, 20, 10]$ ($n = 3$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	Action
—	—	—	30	20	10	initial state
1	—	1	30	20	10	$minIdx := 1$ ($a[1] = 30$)
	2	2	30	20	10	$a[2] = 20 < a[1] = 30$: $minIdx \leftarrow 2$
	3	3	30	20	10	$a[3] = 10 < a[2] = 20$: $minIdx \leftarrow 3$
	—	—	10	20	30	$minIdx = 3 \neq 1$: swap $a[1] \leftrightarrow a[3]$
2	—	2	10	20	30	$minIdx := 2$ ($a[2] = 20$)
	3	2	10	20	30	$a[3] = 30 \geq a[2] = 20$: no update
	—	—	10	20	30	$minIdx = 2$: no swap

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	Action
—	—	30	20	10	initial state
1	3	10	20	30	swap $a[1] \leftrightarrow a[3]$: $a[1] = \min(a[1..3]) = 10$
2	2	10	20	30	no swap: $a[2] = \min(a[2..3]) = 20$

The algorithm performs 3 key comparisons and 1 swap, and the sorted array is

[10, 20, 30].

5. Array $a = [5, 10, 15, 20]$ ($n = 4$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	—	5	10	15	20	initial state
1	—	1	5	10	15	20	$minIdx := 1$ ($a[1] = 5$)
	2	1	5	10	15	20	$a[2] = 10 \geq a[1] = 5$: no update
	3	1	5	10	15	20	$a[3] = 15 \geq a[1] = 5$: no update
	4	1	5	10	15	20	$a[4] = 20 \geq a[1] = 5$: no update
	—	—	5	10	15	20	$minIdx = 1$: no swap
2	—	2	5	10	15	20	$minIdx := 2$ ($a[2] = 10$)
	3	2	5	10	15	20	$a[3] = 15 \geq a[2] = 10$: no update
	4	2	5	10	15	20	$a[4] = 20 \geq a[2] = 10$: no update
	—	—	5	10	15	20	$minIdx = 2$: no swap
3	—	3	5	10	15	20	$minIdx := 3$ ($a[3] = 15$)
	4	3	5	10	15	20	$a[4] = 20 \geq a[3] = 15$: no update
	—	—	5	10	15	20	$minIdx = 3$: no swap

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	5	10	15	20	initial state
1	1	5	10	15	20	no swap: $a[1] = \min(a[1..4]) = 5$
2	2	5	10	15	20	no swap: $a[2] = \min(a[2..4]) = 10$
3	3	5	10	15	20	no swap: $a[3] = \min(a[3..4]) = 15$

The algorithm performs 6 key comparisons and 0 swaps, and the sorted array is

[5, 10, 15, 20].

Exercise 3.18 — Sorting algorithms (E18)

Consider the following arrays of natural numbers:

1. $a = [14, 7, 21, 3, 18]$
2. $a = [40, 30, 20, 10]$
3. $a = [3, 6, 1, 9, 2]$
4. $a = [25, 12, 50]$
5. $a = [17, 17, 17, 17]$

Questions

1. Apply the **SelectionSort** algorithm to each array, to sort it in non-decreasing order. Show the execution step by step: for every iteration of the outer loop, give the index $minIdx$ of the minimum found by the inner loop and say whether a swap is performed.
2. For each array, count the key comparisons and the swaps performed by the algorithm.

Solution

The trace tables follow the conventions stated in the introduction to the exercises of this chapter. Each item answers both questions.

1. Array $a = [14, 7, 21, 3, 18]$ ($n = 5$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
—	—	—	14	7	21	3	18	initial state
1	—	1	14	7	21	3	18	$minIdx := 1$ ($a[1] = 14$)
	2	2	14	7	21	3	18	$a[2] = 7 < a[1] = 14$: $minIdx \leftarrow 2$
	3	2	14	7	21	3	18	$a[3] = 21 \geq a[2] = 7$: no update
	4	4	14	7	21	3	18	$a[4] = 3 < a[2] = 7$: $minIdx \leftarrow 4$
	5	4	14	7	21	3	18	$a[5] = 18 \geq a[4] = 3$: no update
	—	—	3	7	21	14	18	$minIdx = 4 \neq 1$: swap $a[1] \leftrightarrow a[4]$
2	—	2	3	7	21	14	18	$minIdx := 2$ ($a[2] = 7$)
	3	2	3	7	21	14	18	$a[3] = 21 \geq a[2] = 7$: no update
	4	2	3	7	21	14	18	$a[4] = 14 \geq a[2] = 7$: no update
	5	2	3	7	21	14	18	$a[5] = 18 \geq a[2] = 7$: no update
	—	—	3	7	21	14	18	$minIdx = 2$: no swap
	—	—	3	7	21	14	18	
3	—	3	3	7	21	14	18	$minIdx := 3$ ($a[3] = 21$)
	4	4	3	7	21	14	18	$a[4] = 14 < a[3] = 21$: $minIdx \leftarrow 4$
	5	4	3	7	21	14	18	$a[5] = 18 \geq a[4] = 14$: no update
	—	—	3	7	14	21	18	$minIdx = 4 \neq 3$: swap $a[3] \leftrightarrow a[4]$
	—	—	3	7	14	21	18	
4	—	4	3	7	14	21	18	$minIdx := 4$ ($a[4] = 21$)
	5	5	3	7	14	21	18	$a[5] = 18 < a[4] = 21$: $minIdx \leftarrow 5$
	—	—	3	7	14	18	21	$minIdx = 5 \neq 4$: swap $a[4] \leftrightarrow a[5]$

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
—	—	14	7	21	3	18	initial state
1	4	3	7	21	14	18	swap $a[1] \leftrightarrow a[4]$: $a[1] = \min(a[1..5]) = 3$
2	2	3	7	21	14	18	no swap: $a[2] = \min(a[2..5]) = 7$
3	4	3	7	14	21	18	swap $a[3] \leftrightarrow a[4]$: $a[3] = \min(a[3..5]) = 14$
4	5	3	7	14	18	21	swap $a[4] \leftrightarrow a[5]$: $a[4] = \min(a[4..5]) = 18$

The algorithm performs 10 key comparisons and 3 swaps, and the sorted array is

$[3, 7, 14, 18, 21]$.

2. Array $a = [40, 30, 20, 10]$ ($n = 4$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	—	40	30	20	10	initial state
1	—	1	40	30	20	10	$minIdx := 1$ ($a[1] = 40$)
	2	2	40	30	20	10	$a[2] = 30 < a[1] = 40$: $minIdx \leftarrow 2$
	3	3	40	30	20	10	$a[3] = 20 < a[2] = 30$: $minIdx \leftarrow 3$
	4	4	40	30	20	10	$a[4] = 10 < a[3] = 20$: $minIdx \leftarrow 4$
	—	—	10	30	20	40	$minIdx = 4 \neq 1$: swap $a[1] \leftrightarrow a[4]$
2	—	2	10	30	20	40	$minIdx := 2$ ($a[2] = 30$)
	3	3	10	30	20	40	$a[3] = 20 < a[2] = 30$: $minIdx \leftarrow 3$
	4	3	10	30	20	40	$a[4] = 40 \geq a[3] = 20$: no update
	—	—	10	20	30	40	$minIdx = 3 \neq 2$: swap $a[2] \leftrightarrow a[3]$
	—	—	10	20	30	40	
3	—	3	10	20	30	40	$minIdx := 3$ ($a[3] = 30$)
	4	3	10	20	30	40	$a[4] = 40 \geq a[3] = 30$: no update
	—	—	10	20	30	40	$minIdx = 3$: no swap

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	40	30	20	10	initial state
1	4	10	30	20	40	swap $a[1] \leftrightarrow a[4]$: $a[1] = \min(a[1..4]) = 10$
2	3	10	20	30	40	swap $a[2] \leftrightarrow a[3]$: $a[2] = \min(a[2..4]) = 20$
3	3	10	20	30	40	no swap: $a[3] = \min(a[3..4]) = 30$

The algorithm performs 6 key comparisons and 2 swaps, and the sorted array is

[10, 20, 30, 40].

3. Array $a = [3, 6, 1, 9, 2]$ ($n = 5$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
—	—	—	3	6	1	9	2	initial state
1	—	1	3	6	1	9	2	$minIdx := 1$ ($a[1] = 3$)
	2	1	3	6	1	9	2	$a[2] = 6 \geq a[1] = 3$: no update
	3	3	3	6	1	9	2	$a[3] = 1 < a[1] = 3$: $minIdx \leftarrow 3$
	4	3	3	6	1	9	2	$a[4] = 9 \geq a[3] = 1$: no update
	5	3	3	6	1	9	2	$a[5] = 2 \geq a[3] = 1$: no update
	—	—	1	6	3	9	2	$minIdx = 3 \neq 1$: swap $a[1] \leftrightarrow a[3]$
2	—	2	1	6	3	9	2	$minIdx := 2$ ($a[2] = 6$)
	3	3	1	6	3	9	2	$a[3] = 3 < a[2] = 6$: $minIdx \leftarrow 3$
	4	3	1	6	3	9	2	$a[4] = 9 \geq a[3] = 3$: no update
	5	5	1	6	3	9	2	$a[5] = 2 < a[3] = 3$: $minIdx \leftarrow 5$
	—	—	1	2	3	9	6	$minIdx = 5 \neq 2$: swap $a[2] \leftrightarrow a[5]$
3	—	3	1	2	3	9	6	$minIdx := 3$ ($a[3] = 3$)
	4	3	1	2	3	9	6	$a[4] = 9 \geq a[3] = 3$: no update
	5	3	1	2	3	9	6	$a[5] = 6 \geq a[3] = 3$: no update
	—	—	1	2	3	9	6	$minIdx = 3$: no swap
4	—	4	1	2	3	9	6	$minIdx := 4$ ($a[4] = 9$)
	5	5	1	2	3	9	6	$a[5] = 6 < a[4] = 9$: $minIdx \leftarrow 5$
	—	—	1	2	3	6	9	$minIdx = 5 \neq 4$: swap $a[4] \leftrightarrow a[5]$

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	Action
—	—	3	6	1	9	2	initial state
1	3	1	6	3	9	2	swap $a[1] \leftrightarrow a[3]$: $a[1] = \min(a[1..5]) = 1$
2	5	1	2	3	9	6	swap $a[2] \leftrightarrow a[5]$: $a[2] = \min(a[2..5]) = 2$
3	3	1	2	3	9	6	no swap: $a[3] = \min(a[3..5]) = 3$
4	5	1	2	3	6	9	swap $a[4] \leftrightarrow a[5]$: $a[4] = \min(a[4..5]) = 6$

The algorithm performs 10 key comparisons and 3 swaps, and the sorted array is

$$[1, 2, 3, 6, 9].$$

4. Array $a = [25, 12, 50]$ ($n = 3$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	Action
—	—	—	25	12	50	initial state
1	—	1	25	12	50	$minIdx := 1$ ($a[1] = 25$)
	2	2	25	12	50	$a[2] = 12 < a[1] = 25$: $minIdx \leftarrow 2$
	3	2	25	12	50	$a[3] = 50 \geq a[2] = 12$: no update
	—	—	12	25	50	$minIdx = 2 \neq 1$: swap $a[1] \leftrightarrow a[2]$
2	—	2	12	25	50	$minIdx := 2$ ($a[2] = 25$)
	3	2	12	25	50	$a[3] = 50 \geq a[2] = 25$: no update
	—	—	12	25	50	$minIdx = 2$: no swap

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	Action
—	—	25	12	50	initial state
1	2	12	25	50	swap $a[1] \leftrightarrow a[2]$: $a[1] = \min(a[1..3]) = 12$
2	2	12	25	50	no swap: $a[2] = \min(a[2..3]) = 25$

The algorithm performs 3 key comparisons and 1 swap, and the sorted array is
 $[12, 25, 50]$.

5. Array $a = [17, 17, 17, 17]$ ($n = 4$).

Detailed trace:

i	j	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	—	17	17	17	17	initial state
1	—	1	17	17	17	17	$minIdx := 1$ ($a[1] = 17$)
	2	1	17	17	17	17	$a[2] = 17 \geq a[1] = 17$: no update
	3	1	17	17	17	17	$a[3] = 17 \geq a[1] = 17$: no update
	4	1	17	17	17	17	$a[4] = 17 \geq a[1] = 17$: no update
	—	—	17	17	17	17	$minIdx = 1$: no swap
2	—	2	17	17	17	17	$minIdx := 2$ ($a[2] = 17$)
	3	2	17	17	17	17	$a[3] = 17 \geq a[2] = 17$: no update
	4	2	17	17	17	17	$a[4] = 17 \geq a[2] = 17$: no update
	—	—	17	17	17	17	$minIdx = 2$: no swap
3	—	3	17	17	17	17	$minIdx := 3$ ($a[3] = 17$)
	4	3	17	17	17	17	$a[4] = 17 \geq a[3] = 17$: no update
	—	—	17	17	17	17	$minIdx = 3$: no swap

Array after each iteration of the outer loop:

i	$minIdx$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
—	—	17	17	17	17	initial state
1	1	17	17	17	17	no swap: $a[1] = \min(a[1..4]) = 17$
2	2	17	17	17	17	no swap: $a[2] = \min(a[2..4]) = 17$
3	3	17	17	17	17	no swap: $a[3] = \min(a[3..4]) = 17$

The algorithm performs 6 key comparisons and 0 swaps, and the sorted array is
 $[17, 17, 17, 17]$.

Linear and binary search

In the exercises of this chapter the searches are carried out with the two algorithms of the chapter «Linear and binary search», recalled below. The indices of the arrays go from 1 to n , and the value -1 is returned when the target value m is not in the array.

Algorithm: LinearSearch

Input : An array $a[1..n]$ of n natural numbers and a target value m

Output: Index of m in a , or -1 if not found

```
1 function LinearSearch( $a, n, m$ )
2    $i := 1$ 
3   while  $i \leq n$  do
4     if  $a[i] = m$  then
5       return  $i$                                 // Element found at index  $i$ 
6     end
7      $i \leftarrow i + 1$ 
8   end
9   return  $-1$                                     // Element not found
10 end
```

Algorithm: BinarySearch

Input : An array $a[1..n]$ of n natural numbers sorted in non-decreasing order and a target value m

Output: Index of m in a , or -1 if not found

```

1 function BinarySearch( $a, n, m$ )
2    $left := 1$ 
3    $right := n$ 
4   while  $left \leq right$  do
5      $mid := left + \lfloor (right - left) / 2 \rfloor$ 
6     if  $a[mid] = m$  then
7       return  $mid$                                 // Element found
8     end
9     if  $a[mid] < m$  then
10       $left \leftarrow mid + 1$                         // Search in right half
11    end
12    else
13       $right \leftarrow mid - 1$                       // Search in left half
14    end
15  end
16  return  $-1$                                         // Element not found
17 end

```

Exercise 4.1 — Linear and binary search (E1)

Consider the following array:

$$a = [7, 2, 14, 3, 8, 11, 5, 9]$$

with $n = 8$.

Questions

1. Apply the Linear Search algorithm to the array a for each of the following target values:
 - (a) $m = 11$
 - (b) $m = 7$
 - (c) $m = 9$
 - (d) $m = 4$
 - (e) $m = 14$

For each search, show the execution step by step, indicating at each iteration the values of i , $a[i]$ and the result of the comparison, and give the value returned by the algorithm.

Solution

1. Search for
- $m = 11$
- :

iteration	i	$a[i]$	action
1	1	7	$a[i] \neq m \Rightarrow \text{continue}$
2	2	2	$a[i] \neq m \Rightarrow \text{continue}$
3	3	14	$a[i] \neq m \Rightarrow \text{continue}$
4	4	3	$a[i] \neq m \Rightarrow \text{continue}$
5	5	8	$a[i] \neq m \Rightarrow \text{continue}$
6	6	11	$a[i] = m \Rightarrow \text{Found!}$

Accordingly, element 11 is found at index 6.

2. Search for
- $m = 7$
- :

iteration	i	$a[i]$	action
1	1	7	$a[i] = m \Rightarrow \text{Found!}$

Accordingly, element 7 is found at index 1.

3. Search for
- $m = 9$
- :

iteration	i	$a[i]$	action
1	1	7	$a[i] \neq m \Rightarrow \text{continue}$
2	2	2	$a[i] \neq m \Rightarrow \text{continue}$
3	3	14	$a[i] \neq m \Rightarrow \text{continue}$
4	4	3	$a[i] \neq m \Rightarrow \text{continue}$
5	5	8	$a[i] \neq m \Rightarrow \text{continue}$
6	6	11	$a[i] \neq m \Rightarrow \text{continue}$
7	7	5	$a[i] \neq m \Rightarrow \text{continue}$
8	8	9	$a[i] = m \Rightarrow \text{Found!}$

Accordingly, element 9 is found at index 8.

4. Search for
- $m = 4$
- :

iteration	i	$a[i]$	action
1	1	7	$a[i] \neq m \Rightarrow \text{continue}$
2	2	2	$a[i] \neq m \Rightarrow \text{continue}$
3	3	14	$a[i] \neq m \Rightarrow \text{continue}$
4	4	3	$a[i] \neq m \Rightarrow \text{continue}$
5	5	8	$a[i] \neq m \Rightarrow \text{continue}$
6	6	11	$a[i] \neq m \Rightarrow \text{continue}$
7	7	5	$a[i] \neq m \Rightarrow \text{continue}$
8	8	9	$a[i] \neq m \Rightarrow \text{continue}$
—	9	—	$i > n \Rightarrow \text{loop ends, Not found!}$

Accordingly, element 4 is not in the array (return -1).

5. Search for $m = 14$:

iteration	i	$a[i]$	action
1	1	7	$a[i] \neq m \Rightarrow$ continue
2	2	2	$a[i] \neq m \Rightarrow$ continue
3	3	14	$a[i] = m \Rightarrow$ Found!

Accordingly, element 14 is found at index 3.

Exercise 4.2 — Linear and binary search (E2)

Consider the following array:

$$a = [15, 4, 22, 8, 1, 30, 17]$$

with $n = 7$.

Questions

1. Apply the Linear Search algorithm to the array a for each of the following target values:
 - (a) $m = 22$
 - (b) $m = 15$
 - (c) $m = 17$
 - (d) $m = 10$
 - (e) $m = 1$

For each search, show the execution step by step, indicating at each iteration the values of i , $a[i]$ and the result of the comparison, and give the value returned by the algorithm.

Solution

1. Search for $m = 22$:

iteration	i	$a[i]$	action
1	1	15	$a[i] \neq m \Rightarrow$ continue
2	2	4	$a[i] \neq m \Rightarrow$ continue
3	3	22	$a[i] = m \Rightarrow$ Found!

Accordingly, element 22 is found at index 3.

2. Search for $m = 15$:

iteration	i	$a[i]$	action
1	1	15	$a[i] = m \Rightarrow$ Found!

Accordingly, element 15 is found at index 1.

3. Search for $m = 17$:

iteration	i	$a[i]$	action
1	1	15	$a[i] \neq m \Rightarrow$ continue
2	2	4	$a[i] \neq m \Rightarrow$ continue
3	3	22	$a[i] \neq m \Rightarrow$ continue
4	4	8	$a[i] \neq m \Rightarrow$ continue
5	5	1	$a[i] \neq m \Rightarrow$ continue
6	6	30	$a[i] \neq m \Rightarrow$ continue
7	7	17	$a[i] = m \Rightarrow$ Found!

Accordingly, element 17 is found at index 7.

4. Search for $m = 10$:

iteration	i	$a[i]$	action
1	1	15	$a[i] \neq m \Rightarrow$ continue
2	2	4	$a[i] \neq m \Rightarrow$ continue
3	3	22	$a[i] \neq m \Rightarrow$ continue
4	4	8	$a[i] \neq m \Rightarrow$ continue
5	5	1	$a[i] \neq m \Rightarrow$ continue
6	6	30	$a[i] \neq m \Rightarrow$ continue
7	7	17	$a[i] \neq m \Rightarrow$ continue
—	8	—	$i > n \Rightarrow$ loop ends, Not found!

Accordingly, element 10 is not in the array (return -1).

5. Search for $m = 1$:

iteration	i	$a[i]$	action
1	1	15	$a[i] \neq m \Rightarrow$ continue
2	2	4	$a[i] \neq m \Rightarrow$ continue
3	3	22	$a[i] \neq m \Rightarrow$ continue
4	4	8	$a[i] \neq m \Rightarrow$ continue
5	5	1	$a[i] = m \Rightarrow$ Found!

Accordingly, element 1 is found at index 5.

Exercise 4.3 — Linear and binary search (E3)

Consider the following array:

$$a = [6, 19, 3, 27, 12, 5, 33]$$

with $n = 7$.

Questions

1. Apply the Linear Search algorithm to the array a for each of the following target values:

- (a) $m = 27$
- (b) $m = 6$
- (c) $m = 33$
- (d) $m = 15$
- (e) $m = 3$

For each search, show the execution step by step, indicating at each iteration the values of i , $a[i]$ and the result of the comparison, and give the value returned by the algorithm.

Solution

1. Search for $m = 27$:

iteration	i	$a[i]$	action
1	1	6	$a[i] \neq m \Rightarrow \text{continue}$
2	2	19	$a[i] \neq m \Rightarrow \text{continue}$
3	3	3	$a[i] \neq m \Rightarrow \text{continue}$
4	4	27	$a[i] = m \Rightarrow \text{Found!}$

Accordingly, element 27 is found at index 4.

2. Search for $m = 6$:

iteration	i	$a[i]$	action
1	1	6	$a[i] = m \Rightarrow \text{Found!}$

Accordingly, element 6 is found at index 1.

3. Search for $m = 33$:

iteration	i	$a[i]$	action
1	1	6	$a[i] \neq m \Rightarrow \text{continue}$
2	2	19	$a[i] \neq m \Rightarrow \text{continue}$
3	3	3	$a[i] \neq m \Rightarrow \text{continue}$
4	4	27	$a[i] \neq m \Rightarrow \text{continue}$
5	5	12	$a[i] \neq m \Rightarrow \text{continue}$
6	6	5	$a[i] \neq m \Rightarrow \text{continue}$
7	7	33	$a[i] = m \Rightarrow \text{Found!}$

Accordingly, element 33 is found at index 7.

4. Search for $m = 15$:

iteration	i	$a[i]$	action
1	1	6	$a[i] \neq m \Rightarrow \text{continue}$
2	2	19	$a[i] \neq m \Rightarrow \text{continue}$
3	3	3	$a[i] \neq m \Rightarrow \text{continue}$
4	4	27	$a[i] \neq m \Rightarrow \text{continue}$
5	5	12	$a[i] \neq m \Rightarrow \text{continue}$
6	6	5	$a[i] \neq m \Rightarrow \text{continue}$
7	7	33	$a[i] \neq m \Rightarrow \text{continue}$
—	8	—	$i > n \Rightarrow \text{loop ends, Not found!}$

Accordingly, element 15 is not in the array (return -1).

5. Search for $m = 3$:

iteration	i	$a[i]$	action
1	1	6	$a[i] \neq m \Rightarrow \text{continue}$
2	2	19	$a[i] \neq m \Rightarrow \text{continue}$
3	3	3	$a[i] = m \Rightarrow \text{Found!}$

Accordingly, element 3 is found at index 3.

Exercise 4.4 — Linear and binary search (E4)

Consider the following array:

$$a = [50, 25, 75, 10, 40, 60, 5]$$

with $n = 7$.

Questions

1. Apply the Linear Search algorithm to the array a for each of the following target values:

- (a) $m = 10$
- (b) $m = 50$
- (c) $m = 5$
- (d) $m = 30$
- (e) $m = 75$

For each search, show the execution step by step, indicating at each iteration the values of i , $a[i]$ and the result of the comparison, and give the value returned by the algorithm.

Solution

1. Search for $m = 10$:

iteration	i	$a[i]$	action
1	1	50	$a[i] \neq m \Rightarrow$ continue
2	2	25	$a[i] \neq m \Rightarrow$ continue
3	3	75	$a[i] \neq m \Rightarrow$ continue
4	4	10	$a[i] = m \Rightarrow$ Found!

Accordingly, element 10 is found at index 4.

2. Search for $m = 50$:

iteration	i	$a[i]$	action
1	1	50	$a[i] = m \Rightarrow$ Found!

Accordingly, element 50 is found at index 1.

3. Search for $m = 5$:

iteration	i	$a[i]$	action
1	1	50	$a[i] \neq m \Rightarrow$ continue
2	2	25	$a[i] \neq m \Rightarrow$ continue
3	3	75	$a[i] \neq m \Rightarrow$ continue
4	4	10	$a[i] \neq m \Rightarrow$ continue
5	5	40	$a[i] \neq m \Rightarrow$ continue
6	6	60	$a[i] \neq m \Rightarrow$ continue
7	7	5	$a[i] = m \Rightarrow$ Found!

Accordingly, element 5 is found at index 7.

4. Search for $m = 30$:

iteration	i	$a[i]$	action
1	1	50	$a[i] \neq m \Rightarrow \text{continue}$
2	2	25	$a[i] \neq m \Rightarrow \text{continue}$
3	3	75	$a[i] \neq m \Rightarrow \text{continue}$
4	4	10	$a[i] \neq m \Rightarrow \text{continue}$
5	5	40	$a[i] \neq m \Rightarrow \text{continue}$
6	6	60	$a[i] \neq m \Rightarrow \text{continue}$
7	7	5	$a[i] \neq m \Rightarrow \text{continue}$
—	8	—	$i > n \Rightarrow \text{loop ends, Not found!}$

Accordingly, element 30 is not in the array (return -1).

5. Search for $m = 75$:

iteration	i	$a[i]$	action
1	1	50	$a[i] \neq m \Rightarrow \text{continue}$
2	2	25	$a[i] \neq m \Rightarrow \text{continue}$
3	3	75	$a[i] = m \Rightarrow \text{Found!}$

Accordingly, element 75 is found at index 3.

Exercise 4.5 — Linear and binary search (E5)

Consider the following array:

$$a = [11, 7, 3, 18, 24, 9, 16, 2]$$

with $n = 8$.

Questions

1. Apply the Linear Search algorithm to the array a for each of the following target values:
 - (a) $m = 24$
 - (b) $m = 11$
 - (c) $m = 2$
 - (d) $m = 8$

(e) $m = 3$

For each search, show the execution step by step, indicating at each iteration the values of i , $a[i]$ and the result of the comparison, and give the value returned by the algorithm.

Solution

1. Search for $m = 24$:

iteration	i	$a[i]$	action
1	1	11	$a[i] \neq m \Rightarrow \text{continue}$
2	2	7	$a[i] \neq m \Rightarrow \text{continue}$
3	3	3	$a[i] \neq m \Rightarrow \text{continue}$
4	4	18	$a[i] \neq m \Rightarrow \text{continue}$
5	5	24	$a[i] = m \Rightarrow \text{Found!}$

Accordingly, element 24 is found at index 5.

2. Search for $m = 11$:

iteration	i	$a[i]$	action
1	1	11	$a[i] = m \Rightarrow \text{Found!}$

Accordingly, element 11 is found at index 1.

3. Search for $m = 2$:

iteration	i	$a[i]$	action
1	1	11	$a[i] \neq m \Rightarrow \text{continue}$
2	2	7	$a[i] \neq m \Rightarrow \text{continue}$
3	3	3	$a[i] \neq m \Rightarrow \text{continue}$
4	4	18	$a[i] \neq m \Rightarrow \text{continue}$
5	5	24	$a[i] \neq m \Rightarrow \text{continue}$
6	6	9	$a[i] \neq m \Rightarrow \text{continue}$
7	7	16	$a[i] \neq m \Rightarrow \text{continue}$
8	8	2	$a[i] = m \Rightarrow \text{Found!}$

Accordingly, element 2 is found at index 8.

4. Search for $m = 8$:

iteration	i	$a[i]$	action
1	1	11	$a[i] \neq m \Rightarrow \text{continue}$
2	2	7	$a[i] \neq m \Rightarrow \text{continue}$
3	3	3	$a[i] \neq m \Rightarrow \text{continue}$
4	4	18	$a[i] \neq m \Rightarrow \text{continue}$
5	5	24	$a[i] \neq m \Rightarrow \text{continue}$
6	6	9	$a[i] \neq m \Rightarrow \text{continue}$
7	7	16	$a[i] \neq m \Rightarrow \text{continue}$
8	8	2	$a[i] \neq m \Rightarrow \text{continue}$
–	9	–	$i > n \Rightarrow \text{loop ends, Not found!}$

Accordingly, element 8 is not in the array (return -1).

5. Search for $m = 3$:

iteration	i	$a[i]$	action
1	1	11	$a[i] \neq m \Rightarrow \text{continue}$
2	2	7	$a[i] \neq m \Rightarrow \text{continue}$
3	3	3	$a[i] = m \Rightarrow \text{Found!}$

Accordingly, element 3 is found at index 3.

Exercise 4.6 — Linear and binary search (E6)

Consider the following array, sorted in non-decreasing order:

$$a = [2, 5, 8, 12, 16, 23, 38, 45, 56, 67, 78, 89]$$

with $n = 12$.

Questions

1. Apply the Binary Search algorithm to the array a for each of the following target values:
 - (a) $m = 23$
 - (b) $m = 2$
 - (c) $m = 89$
 - (d) $m = 50$
 - (e) $m = 1$

For each search, show the execution step by step, indicating at each iteration the values of $left$, $right$, mid , $a[mid]$ and the result of the comparison, and give the value returned by the algorithm.

Solution

1. Search for
- $m = 23$
- :

iteration	<i>left</i>	<i>right</i>	<i>mid</i>	$a[mid]$	action
1	1	12	6	23	$a[mid] = m \Rightarrow \text{Found!}$

Accordingly, element 23 is found at index 6.

2. Search for
- $m = 2$
- :

iteration	<i>left</i>	<i>right</i>	<i>mid</i>	$a[mid]$	action
1	1	12	6	23	$a[mid] > m \Rightarrow right \leftarrow 5$
2	1	5	3	8	$a[mid] > m \Rightarrow right \leftarrow 2$
3	1	2	1	2	$a[mid] = m \Rightarrow \text{Found!}$

Accordingly, element 2 is found at index 1.

3. Search for
- $m = 89$
- :

iteration	<i>left</i>	<i>right</i>	<i>mid</i>	$a[mid]$	action
1	1	12	6	23	$a[mid] < m \Rightarrow left \leftarrow 7$
2	7	12	9	56	$a[mid] < m \Rightarrow left \leftarrow 10$
3	10	12	11	78	$a[mid] < m \Rightarrow left \leftarrow 12$
4	12	12	12	89	$a[mid] = m \Rightarrow \text{Found!}$

Accordingly, element 89 is found at index 12.

4. Search for
- $m = 50$
- :

iteration	<i>left</i>	<i>right</i>	<i>mid</i>	$a[mid]$	action
1	1	12	6	23	$a[mid] < m \Rightarrow left \leftarrow 7$
2	7	12	9	56	$a[mid] > m \Rightarrow right \leftarrow 8$
3	7	8	7	38	$a[mid] < m \Rightarrow left \leftarrow 8$
4	8	8	8	45	$a[mid] < m \Rightarrow left \leftarrow 9$
—	9	8	—	—	$left > right \Rightarrow \text{Not found!}$

Accordingly, element 50 is not in the array (return -1).

5. Search for
- $m = 1$
- :

iteration	<i>left</i>	<i>right</i>	<i>mid</i>	$a[mid]$	action
1	1	12	6	23	$a[mid] > m \Rightarrow right \leftarrow 5$
2	1	5	3	8	$a[mid] > m \Rightarrow right \leftarrow 2$
3	1	2	1	2	$a[mid] > m \Rightarrow right \leftarrow 0$
—	1	0	—	—	$left > right \Rightarrow \text{Not found!}$

Accordingly, element 1 is not in the array (return -1).

Exercise 4.7 — Linear and binary search (E7)

Consider the following array, sorted in non-decreasing order:

$$a = [1, 3, 5, 7, 9, 11, 13, 15]$$

with $n = 8$.

Questions

1. Apply the Binary Search algorithm to the array a for each of the following target values:

- (a) $m = 7$
- (b) $m = 1$
- (c) $m = 15$
- (d) $m = 10$
- (e) $m = 20$

For each search, show the execution step by step, indicating at each iteration the values of $left$, $right$, mid , $a[mid]$ and the result of the comparison, and give the value returned by the algorithm.

Solution

1. Search for $m = 7$:

iteration	$left$	$right$	mid	$a[mid]$	action
1	1	8	4	7	$a[mid] = m \Rightarrow$ Found!

Accordingly, element 7 is found at index 4.

2. Search for $m = 1$:

iteration	$left$	$right$	mid	$a[mid]$	action
1	1	8	4	7	$a[mid] > m \Rightarrow right \leftarrow 3$
2	1	3	2	3	$a[mid] > m \Rightarrow right \leftarrow 1$
3	1	1	1	1	$a[mid] = m \Rightarrow$ Found!

Accordingly, element 1 is found at index 1.

3. Search for $m = 15$:

iteration	<i>left</i>	<i>right</i>	<i>mid</i>	$a[mid]$	action
1	1	8	4	7	$a[mid] < m \Rightarrow left \leftarrow 5$
2	5	8	6	11	$a[mid] < m \Rightarrow left \leftarrow 7$
3	7	8	7	13	$a[mid] < m \Rightarrow left \leftarrow 8$
4	8	8	8	15	$a[mid] = m \Rightarrow \text{Found!}$

Accordingly, element 15 is found at index 8.

4. Search for $m = 10$:

iteration	<i>left</i>	<i>right</i>	<i>mid</i>	$a[mid]$	action
1	1	8	4	7	$a[mid] < m \Rightarrow left \leftarrow 5$
2	5	8	6	11	$a[mid] > m \Rightarrow right \leftarrow 5$
3	5	5	5	9	$a[mid] < m \Rightarrow left \leftarrow 6$
—	6	5	—	—	$left > right \Rightarrow \text{Not found!}$

Accordingly, element 10 is not in the array (return -1).

5. Search for $m = 20$:

iteration	<i>left</i>	<i>right</i>	<i>mid</i>	$a[mid]$	action
1	1	8	4	7	$a[mid] < m \Rightarrow left \leftarrow 5$
2	5	8	6	11	$a[mid] < m \Rightarrow left \leftarrow 7$
3	7	8	7	13	$a[mid] < m \Rightarrow left \leftarrow 8$
4	8	8	8	15	$a[mid] < m \Rightarrow left \leftarrow 9$
—	9	8	—	—	$left > right \Rightarrow \text{Not found!}$

Accordingly, element 20 is not in the array (return -1).

Exercise 4.8 — Linear and binary search (E8)

Consider the following array, sorted in non-decreasing order:

$$a = [10, 20, 30, 40, 50, 60]$$

with $n = 6$.

Questions

1. Apply the Binary Search algorithm to the array a for each of the following target values:

- (a) $m = 40$
- (b) $m = 10$
- (c) $m = 60$
- (d) $m = 25$
- (e) $m = 70$

For each search, show the execution step by step, indicating at each iteration the values of $left$, $right$, mid , $a[mid]$ and the result of the comparison, and give the value returned by the algorithm.

Solution

1. Search for $m = 40$:

iteration	$left$	$right$	mid	$a[mid]$	action
1	1	6	3	30	$a[mid] < m \Rightarrow left \leftarrow 4$
2	4	6	5	50	$a[mid] > m \Rightarrow right \leftarrow 4$
3	4	4	4	40	$a[mid] = m \Rightarrow \text{Found!}$

Accordingly, element 40 is found at index 4.

2. Search for $m = 10$:

iteration	$left$	$right$	mid	$a[mid]$	action
1	1	6	3	30	$a[mid] > m \Rightarrow right \leftarrow 2$
2	1	2	1	10	$a[mid] = m \Rightarrow \text{Found!}$

Accordingly, element 10 is found at index 1.

3. Search for $m = 60$:

iteration	$left$	$right$	mid	$a[mid]$	action
1	1	6	3	30	$a[mid] < m \Rightarrow left \leftarrow 4$
2	4	6	5	50	$a[mid] < m \Rightarrow left \leftarrow 6$
3	6	6	6	60	$a[mid] = m \Rightarrow \text{Found!}$

Accordingly, element 60 is found at index 6.

4. Search for $m = 25$:

iteration	$left$	$right$	mid	$a[mid]$	action
1	1	6	3	30	$a[mid] > m \Rightarrow right \leftarrow 2$
2	1	2	1	10	$a[mid] < m \Rightarrow left \leftarrow 2$
3	2	2	2	20	$a[mid] < m \Rightarrow left \leftarrow 3$
—	3	2	—	—	$left > right \Rightarrow \text{Not found!}$

Accordingly, element 25 is not in the array (return -1).

5. Search for $m = 70$:

iteration	<i>left</i>	<i>right</i>	<i>mid</i>	$a[mid]$	action
1	1	6	3	30	$a[mid] < m \Rightarrow left \leftarrow 4$
2	4	6	5	50	$a[mid] < m \Rightarrow left \leftarrow 6$
3	6	6	6	60	$a[mid] < m \Rightarrow left \leftarrow 7$
—	7	6	—	—	$left > right \Rightarrow$ Not found!

Accordingly, element 70 is not in the array (return -1).

Exercise 4.9 — Linear and binary search (E9)

Consider the following array, sorted in non-decreasing order:

$$a = [4, 8, 15, 16, 23, 42]$$

with $n = 6$.

Questions

1. Apply the Binary Search algorithm to the array a for each of the following target values:

- (a) $m = 15$
- (b) $m = 4$
- (c) $m = 42$
- (d) $m = 20$
- (e) $m = 2$

For each search, show the execution step by step, indicating at each iteration the values of $left$, $right$, mid , $a[mid]$ and the result of the comparison, and give the value returned by the algorithm.

Solution

1. Search for $m = 15$:

iteration	<i>left</i>	<i>right</i>	<i>mid</i>	$a[mid]$	action
1	1	6	3	15	$a[mid] = m \Rightarrow$ Found!

Accordingly, element 15 is found at index 3.

2. Search for $m = 4$:

iteration	<i>left</i>	<i>right</i>	<i>mid</i>	$a[mid]$	action
1	1	6	3	15	$a[mid] > m \Rightarrow right \leftarrow 2$
2	1	2	1	4	$a[mid] = m \Rightarrow \text{Found!}$

Accordingly, element 4 is found at index 1.

3. Search for $m = 42$:

iteration	<i>left</i>	<i>right</i>	<i>mid</i>	$a[mid]$	action
1	1	6	3	15	$a[mid] < m \Rightarrow left \leftarrow 4$
2	4	6	5	23	$a[mid] < m \Rightarrow left \leftarrow 6$
3	6	6	6	42	$a[mid] = m \Rightarrow \text{Found!}$

Accordingly, element 42 is found at index 6.

4. Search for $m = 20$:

iteration	<i>left</i>	<i>right</i>	<i>mid</i>	$a[mid]$	action
1	1	6	3	15	$a[mid] < m \Rightarrow left \leftarrow 4$
2	4	6	5	23	$a[mid] > m \Rightarrow right \leftarrow 4$
3	4	4	4	16	$a[mid] < m \Rightarrow left \leftarrow 5$
—	5	4	—	—	$left > right \Rightarrow \text{Not found!}$

Accordingly, element 20 is not in the array (return -1).

5. Search for $m = 2$:

iteration	<i>left</i>	<i>right</i>	<i>mid</i>	$a[mid]$	action
1	1	6	3	15	$a[mid] > m \Rightarrow right \leftarrow 2$
2	1	2	1	4	$a[mid] > m \Rightarrow right \leftarrow 0$
—	1	0	—	—	$left > right \Rightarrow \text{Not found!}$

Accordingly, element 2 is not in the array (return -1).

Exercise 4.10 — Linear and binary search (E10)

Consider the following array, sorted in non-decreasing order:

$$a = [3, 6, 9, 12, 15, 18, 21, 24, 27, 30]$$

with $n = 10$.

Questions

1. Apply the Binary Search algorithm to the array a for each of the following target values:

- (a) $m = 18$
- (b) $m = 3$
- (c) $m = 30$
- (d) $m = 10$
- (e) $m = 35$

For each search, show the execution step by step, indicating at each iteration the values of $left$, $right$, mid , $a[mid]$ and the result of the comparison, and give the value returned by the algorithm.

Solution

1. Search for $m = 18$:

iteration	$left$	$right$	mid	$a[mid]$	action
1	1	10	5	15	$a[mid] < m \Rightarrow left \leftarrow 6$
2	6	10	8	24	$a[mid] > m \Rightarrow right \leftarrow 7$
3	6	7	6	18	$a[mid] = m \Rightarrow \text{Found!}$

Accordingly, element 18 is found at index 6.

2. Search for $m = 3$:

iteration	$left$	$right$	mid	$a[mid]$	action
1	1	10	5	15	$a[mid] > m \Rightarrow right \leftarrow 4$
2	1	4	2	6	$a[mid] > m \Rightarrow right \leftarrow 1$
3	1	1	1	3	$a[mid] = m \Rightarrow \text{Found!}$

Accordingly, element 3 is found at index 1.

3. Search for $m = 30$:

iteration	$left$	$right$	mid	$a[mid]$	action
1	1	10	5	15	$a[mid] < m \Rightarrow left \leftarrow 6$
2	6	10	8	24	$a[mid] < m \Rightarrow left \leftarrow 9$
3	9	10	9	27	$a[mid] < m \Rightarrow left \leftarrow 10$
4	10	10	10	30	$a[mid] = m \Rightarrow \text{Found!}$

Accordingly, element 30 is found at index 10.

4. Search for $m = 10$:

iteration	<i>left</i>	<i>right</i>	<i>mid</i>	$a[mid]$	action
1	1	10	5	15	$a[mid] > m \Rightarrow right \leftarrow 4$
2	1	4	2	6	$a[mid] < m \Rightarrow left \leftarrow 3$
3	3	4	3	9	$a[mid] < m \Rightarrow left \leftarrow 4$
4	4	4	4	12	$a[mid] > m \Rightarrow right \leftarrow 3$
–	4	3	–	–	$left > right \Rightarrow$ Not found!

Accordingly, element 10 is not in the array (return -1).

5. Search for $m = 35$:

iteration	<i>left</i>	<i>right</i>	<i>mid</i>	$a[mid]$	action
1	1	10	5	15	$a[mid] < m \Rightarrow left \leftarrow 6$
2	6	10	8	24	$a[mid] < m \Rightarrow left \leftarrow 9$
3	9	10	9	27	$a[mid] < m \Rightarrow left \leftarrow 10$
4	10	10	10	30	$a[mid] < m \Rightarrow left \leftarrow 11$
–	11	10	–	–	$left > right \Rightarrow$ Not found!

Accordingly, element 35 is not in the array (return -1).

Numerical algorithms

The exercises of this chapter apply the four algorithms of the chapter «Numerical algorithms»: `ChangeOfBase`, `BasicPrimalityTest` and `PrimalityTest`, `SieveOfEratosthenes`, `EuclideanAlgorithm`. The following facts of the chapter are used in the solutions.

Division with remainder. For every natural number n and every natural number $b \geq 1$:

$$n = \lfloor n/b \rfloor b + (n \bmod b) \quad \text{with} \quad 0 \leq n \bmod b < b.$$

Change of base. If $n = (d_{k-1} \dots d_1 d_0)_b$, then $n \bmod b = d_0$ and $\lfloor n/b \rfloor = (d_{k-1} \dots d_1)_b$; the number of digits is $\lfloor \log_b n \rfloor + 1$. `ChangeOfBase` stores the digit d_{k-1} in the entry $d[k]$.

Warning

Primality. A natural number $n \geq 2$ is prime if and only if no natural number in $\{2, 3, \dots, \lfloor \sqrt{n} \rfloor\}$ is a divisor of n .

Sieve. The marking of the multiples of a prime p starts at p^2 , and the outer loop stops at $\lfloor \sqrt{n} \rfloor$. In the trace tables, blue cells are the entries just set to 0 and red cells the current value of p .

Greatest common divisor. For every pair of natural numbers $n, m \geq 1$:

$$\gcd(n, m) = \gcd(m, n \bmod m) \quad \text{and} \quad \gcd(n, 0) = n.$$

Two natural numbers are coprime if and only if their greatest common divisor is 1.

Exercise 5.1 — Numerical algorithms (E1)

Consider the following natural number, written as a sequence of digits in base ten:

$$(16)_{10}$$

Questions

1. Convert the number into base $b = 2$ with the algorithm ChangeOfBase, showing the quotient and the remainder of every division.
2. Convert the number into base $b = 3$ with the algorithm ChangeOfBase, showing the quotient and the remainder of every division.
3. Convert the number into base $b = 4$ with the algorithm ChangeOfBase, showing the quotient and the remainder of every division.
4. Convert the number into base $b = 5$ with the algorithm ChangeOfBase, showing the quotient and the remainder of every division.
5. Convert the number into base $b = 6$ with the algorithm ChangeOfBase, showing the quotient and the remainder of every division.

Solution

1. Translation of 16 into base $b = 2$:

iteration k	n	n/b	q	r	digit
1	16	16/2	8	0	$d[1] = d_0$ (LSD)
2	8	8/2	4	0	$d[2] = d_1$
3	4	4/2	2	0	$d[3] = d_2$
4	2	2/2	1	0	$d[4] = d_3$
5	1	1/2	0	1	$d[5] = d_4$ (MSD)

Accordingly, 16 in base 2 is $(10000)_2$, with $nd = \lfloor \log_2 16 \rfloor + 1 = 5$ digits.

2. Translation of 16 into base $b = 3$:

iteration k	n	n/b	q	r	digit
1	16	16/3	5	1	$d[1] = d_0$ (LSD)
2	5	5/3	1	2	$d[2] = d_1$
3	1	1/3	0	1	$d[3] = d_2$ (MSD)

Accordingly, 16 in base 3 is $(121)_3$, with $nd = \lfloor \log_3 16 \rfloor + 1 = 3$ digits.

3. Translation of 16 into base $b = 4$:

iteration k	n	n/b	q	r	digit
1	16	16/4	4	0	$d[1] = d_0$ (LSD)
2	4	4/4	1	0	$d[2] = d_1$
3	1	1/4	0	1	$d[3] = d_2$ (MSD)

Accordingly, 16 in base 4 is $(100)_4$, with $nd = \lfloor \log_4 16 \rfloor + 1 = 3$ digits.

4. Translation of 16 into base $b = 5$:

iteration k	n	n/b	q	r	digit
1	16	16/5	3	1	$d[1] = d_0$ (LSD)
2	3	3/5	0	3	$d[2] = d_1$ (MSD)

Accordingly, 16 in base 5 is $(31)_5$, with $nd = \lfloor \log_5 16 \rfloor + 1 = 2$ digits.

5. Translation of 16 into base $b = 6$:

iteration k	n	n/b	q	r	digit
1	16	16/6	2	4	$d[1] = d_0$ (LSD)
2	2	2/6	0	2	$d[2] = d_1$ (MSD)

Accordingly, 16 in base 6 is $(24)_6$, with $nd = \lfloor \log_6 16 \rfloor + 1 = 2$ digits.

Exercise 5.2 — Numerical algorithms (E2)

Consider the following natural number, written as a sequence of digits in base ten:

$$(25)_{10}$$

Questions

1. Convert the number into base $b = 2$ with the algorithm `ChangeOfBase`, showing the quotient and the remainder of every division.
2. Convert the number into base $b = 3$ with the algorithm `ChangeOfBase`, showing the quotient and the remainder of every division.
3. Convert the number into base $b = 4$ with the algorithm `ChangeOfBase`, showing the quotient and the remainder of every division.
4. Convert the number into base $b = 5$ with the algorithm `ChangeOfBase`, showing the quotient and the remainder of every division.
5. Convert the number into base $b = 7$ with the algorithm `ChangeOfBase`, showing the quotient and the remainder of every division.

Solution

1. Translation of 25 into base
- $b = 2$
- :

iteration k	n	n/b	q	r	digit
1	25	25/2	12	1	$d[1] = d_0$ (LSD)
2	12	12/2	6	0	$d[2] = d_1$
3	6	6/2	3	0	$d[3] = d_2$
4	3	3/2	1	1	$d[4] = d_3$
5	1	1/2	0	1	$d[5] = d_4$ (MSD)

Accordingly, 25 in base 2 is $(11001)_2$, with $nd = \lfloor \log_2 25 \rfloor + 1 = 5$ digits.

2. Translation of 25 into base
- $b = 3$
- :

iteration k	n	n/b	q	r	digit
1	25	25/3	8	1	$d[1] = d_0$ (LSD)
2	8	8/3	2	2	$d[2] = d_1$
3	2	2/3	0	2	$d[3] = d_2$ (MSD)

Accordingly, 25 in base 3 is $(221)_3$, with $nd = \lfloor \log_3 25 \rfloor + 1 = 3$ digits.

3. Translation of 25 into base
- $b = 4$
- :

iteration k	n	n/b	q	r	digit
1	25	25/4	6	1	$d[1] = d_0$ (LSD)
2	6	6/4	1	2	$d[2] = d_1$
3	1	1/4	0	1	$d[3] = d_2$ (MSD)

Accordingly, 25 in base 4 is $(121)_4$, with $nd = \lfloor \log_4 25 \rfloor + 1 = 3$ digits.

4. Translation of 25 into base
- $b = 5$
- :

iteration k	n	n/b	q	r	digit
1	25	25/5	5	0	$d[1] = d_0$ (LSD)
2	5	5/5	1	0	$d[2] = d_1$
3	1	1/5	0	1	$d[3] = d_2$ (MSD)

Accordingly, 25 in base 5 is $(100)_5$, with $nd = \lfloor \log_5 25 \rfloor + 1 = 3$ digits.

5. Translation of 25 into base
- $b = 7$
- :

iteration k	n	n/b	q	r	digit
1	25	25/7	3	4	$d[1] = d_0$ (LSD)
2	3	3/7	0	3	$d[2] = d_1$ (MSD)

Accordingly, 25 in base 7 is $(34)_7$, with $nd = \lfloor \log_7 25 \rfloor + 1 = 2$ digits.

Exercise 5.3 — Numerical algorithms (E3)

Consider the following natural number, written as a sequence of digits in base ten:

$$(42)_{10}$$

Questions

1. Convert the number into base $b = 2$ with the algorithm `ChangeOfBase`, showing the quotient and the remainder of every division.
2. Convert the number into base $b = 3$ with the algorithm `ChangeOfBase`, showing the quotient and the remainder of every division.
3. Convert the number into base $b = 5$ with the algorithm `ChangeOfBase`, showing the quotient and the remainder of every division.
4. Convert the number into base $b = 6$ with the algorithm `ChangeOfBase`, showing the quotient and the remainder of every division.
5. Convert the number into base $b = 8$ with the algorithm `ChangeOfBase`, showing the quotient and the remainder of every division.

Solution

1. Translation of 42 into base $b = 2$:

iteration k	n	n/b	q	r	digit
1	42	42/2	21	0	$d[1] = d_0$ (LSD)
2	21	21/2	10	1	$d[2] = d_1$
3	10	10/2	5	0	$d[3] = d_2$
4	5	5/2	2	1	$d[4] = d_3$
5	2	2/2	1	0	$d[5] = d_4$
6	1	1/2	0	1	$d[6] = d_5$ (MSD)

Accordingly, 42 in base 2 is $(101010)_2$, with $nd = \lfloor \log_2 42 \rfloor + 1 = 6$ digits.

2. Translation of 42 into base $b = 3$:

iteration k	n	n/b	q	r	digit
1	42	42/3	14	0	$d[1] = d_0$ (LSD)
2	14	14/3	4	2	$d[2] = d_1$
3	4	4/3	1	1	$d[3] = d_2$
4	1	1/3	0	1	$d[4] = d_3$ (MSD)

Accordingly, 42 in base 3 is $(1120)_3$, with $nd = \lfloor \log_3 42 \rfloor + 1 = 4$ digits.

3. Translation of 42 into base $b = 5$:

iteration k	n	n/b	q	r	digit
1	42	42/5	8	2	$d[1] = d_0$ (LSD)
2	8	8/5	1	3	$d[2] = d_1$
3	1	1/5	0	1	$d[3] = d_2$ (MSD)

Accordingly, 42 in base 5 is $(132)_5$, with $nd = \lfloor \log_5 42 \rfloor + 1 = 3$ digits.

4. Translation of 42 into base $b = 6$:

iteration k	n	n/b	q	r	digit
1	42	42/6	7	0	$d[1] = d_0$ (LSD)
2	7	7/6	1	1	$d[2] = d_1$
3	1	1/6	0	1	$d[3] = d_2$ (MSD)

Accordingly, 42 in base 6 is $(110)_6$, with $nd = \lfloor \log_6 42 \rfloor + 1 = 3$ digits.

5. Translation of 42 into base $b = 8$:

iteration k	n	n/b	q	r	digit
1	42	42/8	5	2	$d[1] = d_0$ (LSD)
2	5	5/8	0	5	$d[2] = d_1$ (MSD)

Accordingly, 42 in base 8 is $(52)_8$, with $nd = \lfloor \log_8 42 \rfloor + 1 = 2$ digits.

Exercise 5.4 — Numerical algorithms (E4)

Consider the following natural number, written as a sequence of digits in base ten:

$$(63)_{10}$$

Questions

1. Convert the number into base $b = 2$ with the algorithm `ChangeOfBase`, showing the quotient and the remainder of every division.
2. Convert the number into base $b = 4$ with the algorithm `ChangeOfBase`, showing the quotient and the remainder of every division.

3. Convert the number into base $b = 7$ with the algorithm ChangeOfBase, showing the quotient and the remainder of every division.
4. Convert the number into base $b = 8$ with the algorithm ChangeOfBase, showing the quotient and the remainder of every division.
5. Convert the number into base $b = 9$ with the algorithm ChangeOfBase, showing the quotient and the remainder of every division.

Solution

1. Translation of 63 into base $b = 2$:

iteration k	n	n/b	q	r	digit
1	63	63/2	31	1	$d[1] = d_0$ (LSD)
2	31	31/2	15	1	$d[2] = d_1$
3	15	15/2	7	1	$d[3] = d_2$
4	7	7/2	3	1	$d[4] = d_3$
5	3	3/2	1	1	$d[5] = d_4$
6	1	1/2	0	1	$d[6] = d_5$ (MSD)

Accordingly, 63 in base 2 is $(111111)_2$, with $nd = \lfloor \log_2 63 \rfloor + 1 = 6$ digits.

2. Translation of 63 into base $b = 4$:

iteration k	n	n/b	q	r	digit
1	63	63/4	15	3	$d[1] = d_0$ (LSD)
2	15	15/4	3	3	$d[2] = d_1$
3	3	3/4	0	3	$d[3] = d_2$ (MSD)

Accordingly, 63 in base 4 is $(333)_4$, with $nd = \lfloor \log_4 63 \rfloor + 1 = 3$ digits.

3. Translation of 63 into base $b = 7$:

iteration k	n	n/b	q	r	digit
1	63	63/7	9	0	$d[1] = d_0$ (LSD)
2	9	9/7	1	2	$d[2] = d_1$
3	1	1/7	0	1	$d[3] = d_2$ (MSD)

Accordingly, 63 in base 7 is $(120)_7$, with $nd = \lfloor \log_7 63 \rfloor + 1 = 3$ digits.

4. Translation of 63 into base $b = 8$:

iteration k	n	n/b	q	r	digit
1	63	63/8	7	7	$d[1] = d_0$ (LSD)
2	7	7/8	0	7	$d[2] = d_1$ (MSD)

Accordingly, 63 in base 8 is $(77)_8$, with $nd = \lfloor \log_8 63 \rfloor + 1 = 2$ digits.

5. Translation of 63 into base $b = 9$:

iteration k	n	n/b	q	r	digit
1	63	63/9	7	0	$d[1] = d_0$ (LSD)
2	7	7/9	0	7	$d[2] = d_1$ (MSD)

Accordingly, 63 in base 9 is $(70)_9$, with $nd = \lfloor \log_9 63 \rfloor + 1 = 2$ digits.

Exercise 5.5 — Numerical algorithms (E5)

Consider the following natural number, written as a sequence of digits in base ten:

$$(100)_{10}$$

Questions

1. Convert the number into base $b = 2$ with the algorithm `ChangeOfBase`, showing the quotient and the remainder of every division.
2. Convert the number into base $b = 3$ with the algorithm `ChangeOfBase`, showing the quotient and the remainder of every division.
3. Convert the number into base $b = 5$ with the algorithm `ChangeOfBase`, showing the quotient and the remainder of every division.
4. Convert the number into base $b = 8$ with the algorithm `ChangeOfBase`, showing the quotient and the remainder of every division.
5. Convert the number into base $b = 16$ with the algorithm `ChangeOfBase`, showing the quotient and the remainder of every division.

Solution

1. Translation of 100 into base $b = 2$:

iteration k	n	n/b	q	r	digit
1	100	100/2	50	0	$d[1] = d_0$ (LSD)
2	50	50/2	25	0	$d[2] = d_1$
3	25	25/2	12	1	$d[3] = d_2$
4	12	12/2	6	0	$d[4] = d_3$
5	6	6/2	3	0	$d[5] = d_4$
6	3	3/2	1	1	$d[6] = d_5$
7	1	1/2	0	1	$d[7] = d_6$ (MSD)

Accordingly, 100 in base 2 is $(1100100)_2$, with $nd = \lfloor \log_2 100 \rfloor + 1 = 7$ digits.

2. Translation of 100 into base $b = 3$:

iteration k	n	n/b	q	r	digit
1	100	100/3	33	1	$d[1] = d_0$ (LSD)
2	33	33/3	11	0	$d[2] = d_1$
3	11	11/3	3	2	$d[3] = d_2$
4	3	3/3	1	0	$d[4] = d_3$
5	1	1/3	0	1	$d[5] = d_4$ (MSD)

Accordingly, 100 in base 3 is $(10201)_3$, with $nd = \lfloor \log_3 100 \rfloor + 1 = 5$ digits.

3. Translation of 100 into base $b = 5$:

iteration k	n	n/b	q	r	digit
1	100	100/5	20	0	$d[1] = d_0$ (LSD)
2	20	20/5	4	0	$d[2] = d_1$
3	4	4/5	0	4	$d[3] = d_2$ (MSD)

Accordingly, 100 in base 5 is $(400)_5$, with $nd = \lfloor \log_5 100 \rfloor + 1 = 3$ digits.

4. Translation of 100 into base $b = 8$:

iteration k	n	n/b	q	r	digit
1	100	100/8	12	4	$d[1] = d_0$ (LSD)
2	12	12/8	1	4	$d[2] = d_1$
3	1	1/8	0	1	$d[3] = d_2$ (MSD)

Accordingly, 100 in base 8 is $(144)_8$, with $nd = \lfloor \log_8 100 \rfloor + 1 = 3$ digits.

5. Translation of 100 into base $b = 16$:

iteration k	n	n/b	q	r	digit
1	100	100/16	6	4	$d[1] = d_0$ (LSD)
2	6	6/16	0	6	$d[2] = d_1$ (MSD)

Accordingly, 100 in base 16 is $(64)_{16}$, with $nd = \lfloor \log_{16} 100 \rfloor + 1 = 2$ digits.

Exercise 5.6 — Numerical algorithms (E6)

Consider the following natural numbers:

$$n \in \{2, 7, 9, 11, 25\}$$

Questions

1. Apply the algorithm PrimalityTest to each number in the list. For each n , show the execution step by step, indicating the value of $\lfloor \sqrt{n} \rfloor$, the values of d tested, and the result of each test $n \bmod d = 0$.

Solution

1. Execution of PrimalityTest on each number.

Check $n = 2$: $\lfloor \sqrt{2} \rfloor = 1$.

d	$2 \bmod d$	action
The loop is not executed, since $2 > 1$.		
Return true: 2 is prime .		

Check $n = 7$: $\lfloor \sqrt{7} \rfloor = 2$.

d	$7 \bmod d$	action
2	$1 \neq 0$	continue
Loop ends. Return true: 7 is prime .		

Check $n = 9$: $\lfloor \sqrt{9} \rfloor = 3$.

d	$9 \bmod d$	action
2	$1 \neq 0$	continue
3	0	return false
9 is not prime ($9 = 3 \cdot 3$).		

Check $n = 11$: $\lfloor \sqrt{11} \rfloor = 3$.

d	$11 \bmod d$	action
2	$1 \neq 0$	continue
3	$2 \neq 0$	continue
Loop ends. Return true: 11 is prime.		

Check $n = 25$: $\lfloor \sqrt{25} \rfloor = 5$.

d	$25 \bmod d$	action
2	$1 \neq 0$	continue
3	$1 \neq 0$	continue
4	$1 \neq 0$	continue
5	0	return false
25 is not prime ($25 = 5 \cdot 5$).		

Exercise 5.7 — Numerical algorithms (E7)

Consider the following natural numbers:

$$n \in \{4, 13, 17, 49, 97\}$$

Questions

1. Apply the algorithm PrimalityTest to each number in the list. For each n , show the execution step by step, indicating the value of $\lfloor \sqrt{n} \rfloor$, the values of d tested, and the result of each test $n \bmod d = 0$.

Solution

1. Execution of PrimalityTest on each number.

Check $n = 4$: $\lfloor \sqrt{4} \rfloor = 2$.

d	$4 \bmod d$	action
2	0	return false
4 is not prime ($4 = 2 \cdot 2$).		

Check $n = 13$: $\lfloor \sqrt{13} \rfloor = 3$.

d	$13 \bmod d$	action
2	$1 \neq 0$	continue
3	$1 \neq 0$	continue
Loop ends. Return true: 13 is prime.		

Check $n = 17$: $\lfloor \sqrt{17} \rfloor = 4$.

d	$17 \bmod d$	action
2	$1 \neq 0$	continue
3	$2 \neq 0$	continue
4	$1 \neq 0$	continue
Loop ends. Return true: 17 is prime.		

Check $n = 49$: $\lfloor \sqrt{49} \rfloor = 7$.

d	$49 \bmod d$	action
2	$1 \neq 0$	continue
3	$1 \neq 0$	continue
4	$1 \neq 0$	continue
5	$4 \neq 0$	continue
6	$1 \neq 0$	continue
7	0	return false
49 is not prime ($49 = 7 \cdot 7$).		

Check $n = 97$: $\lfloor \sqrt{97} \rfloor = 9$.

d	$97 \bmod d$	action
2	$1 \neq 0$	continue
3	$1 \neq 0$	continue
4	$1 \neq 0$	continue
5	$2 \neq 0$	continue
6	$1 \neq 0$	continue
7	$6 \neq 0$	continue
8	$1 \neq 0$	continue
9	$7 \neq 0$	continue
Loop ends. Return true: 97 is prime.		

Exercise 5.8 — Numerical algorithms (E8)

Consider the following natural numbers:

$$n \in \{1, 3, 15, 23, 36\}$$

Questions

1. Apply the algorithm PrimalityTest to each number in the list. For each n , show the execution step by step, indicating the value of $\lfloor \sqrt{n} \rfloor$, the values of d tested, and the result of each test $n \bmod d = 0$.

Solution

1. Execution of PrimalityTest on each number.

Check $n = 1$: $n < 2$.

d	$1 \bmod d$	action
$n = 1 < 2$: return false immediately.		
1 is not prime.		

Check $n = 3$: $\lfloor \sqrt{3} \rfloor = 1$.

d	$3 \bmod d$	action
The loop is not executed, since $2 > 1$.		
Return true : 3 is prime.		

Check $n = 15$: $\lfloor \sqrt{15} \rfloor = 3$.

d	$15 \bmod d$	action
2	$1 \neq 0$	continue
3	0	return false
15 is not prime ($15 = 3 \cdot 5$).		

Check $n = 23$: $\lfloor \sqrt{23} \rfloor = 4$.

d	$23 \bmod d$	action
2	$1 \neq 0$	continue
3	$2 \neq 0$	continue
4	$3 \neq 0$	continue
Loop ends. Return true : 23 is prime.		

Check $n = 36$: $\lfloor \sqrt{36} \rfloor = 6$.

d	$36 \bmod d$	action
2	0	return false
36 is not prime ($36 = 2 \cdot 18$).		

Exercise 5.9 — Numerical algorithms (E9)

Consider the natural numbers $n = 29$ and $n = 49$.

Questions

- For each number n , apply both the algorithm `BasicPrimalityTest` (loop up to $\lfloor n/2 \rfloor$) and the algorithm `PrimalityTest` (loop up to $\lfloor \sqrt{n} \rfloor$). Show the step-by-step execution of each algorithm.
- For each number, count the number of iterations performed by each algorithm and compare.

Solution

- Step-by-step execution.

Case $n = 29$.

BasicPrimalityTest ($\lfloor 29/2 \rfloor = 14$):

d	$29 \bmod d$	action
2	$1 \neq 0$	continue
3	$2 \neq 0$	continue
4	$1 \neq 0$	continue
5	$4 \neq 0$	continue
6	$5 \neq 0$	continue
7	$1 \neq 0$	continue
8	$5 \neq 0$	continue
9	$2 \neq 0$	continue
10	$9 \neq 0$	continue
11	$7 \neq 0$	continue
12	$5 \neq 0$	continue
13	$3 \neq 0$	continue
14	$1 \neq 0$	continue
Loop ends. Return true : 29 is prime . (13 iterations)		

PrimalityTest ($\lfloor \sqrt{29} \rfloor = 5$):

d	$29 \bmod d$	action
2	$1 \neq 0$	continue
3	$2 \neq 0$	continue
4	$1 \neq 0$	continue
5	$4 \neq 0$	continue
Loop ends. Return true: 29 is prime. (4 iterations)		

Case $n = 49$.

BasicPrimalityTest ($\lfloor 49/2 \rfloor = 24$):

d	$49 \bmod d$	action
2	$1 \neq 0$	continue
3	$1 \neq 0$	continue
4	$1 \neq 0$	continue
5	$4 \neq 0$	continue
6	$1 \neq 0$	continue
7	0	return false
49 is not prime ($49 = 7 \cdot 7$). (6 iterations)		

PrimalityTest ($\lfloor \sqrt{49} \rfloor = 7$):

d	$49 \bmod d$	action
2	$1 \neq 0$	continue
3	$1 \neq 0$	continue
4	$1 \neq 0$	continue
5	$4 \neq 0$	continue
6	$1 \neq 0$	continue
7	0	return false
49 is not prime ($49 = 7 \cdot 7$). (6 iterations)		

2. Comparison of the iterations:

n	$\lfloor n/2 \rfloor$	$\lfloor \sqrt{n} \rfloor$	BasicPrimalityTest	PrimalityTest
29	14	5	13	4
49	24	7	6	6

For $n = 29$ (prime), *PrimalityTest* performs only 4 iterations instead of 13: both algorithms must test all their candidates, and *PrimalityTest* has far fewer. For $n = 49$ (composite, with smallest divisor $7 = \sqrt{49}$), both algorithms stop at the same divisor $d = 7$, so the number of iterations is the same.

Exercise 5.10 — Numerical algorithms (E10)

Consider the natural numbers $n \in \{2, 3, 4, \dots, 30\}$.

Questions

1. Apply the algorithm `PrimalityTest` to each natural number $n \in \{2, 3, \dots, 30\}$ and determine which numbers are prime. For each n , report the value of $\lfloor \sqrt{n} \rfloor$, whether a divisor was found (and which one), and the result.
2. List all the prime numbers in $\{2, 3, \dots, 30\}$ and count them.

Solution

1. Summary of the executions of `PrimalityTest` (the divisor reported is the first one found, that is, the value of d for which the function returns **false**):

n	$\lfloor \sqrt{n} \rfloor$	first divisor found	result
2	1	none (loop not executed)	prime
3	1	none (loop not executed)	prime
4	2	$d = 2$	not prime ($4 = 2 \cdot 2$)
5	2	none	prime
6	2	$d = 2$	not prime ($6 = 2 \cdot 3$)
7	2	none	prime
8	2	$d = 2$	not prime ($8 = 2 \cdot 4$)
9	3	$d = 3$	not prime ($9 = 3 \cdot 3$)
10	3	$d = 2$	not prime ($10 = 2 \cdot 5$)
11	3	none	prime
12	3	$d = 2$	not prime ($12 = 2 \cdot 6$)
13	3	none	prime
14	3	$d = 2$	not prime ($14 = 2 \cdot 7$)
15	3	$d = 3$	not prime ($15 = 3 \cdot 5$)
16	4	$d = 2$	not prime ($16 = 2 \cdot 8$)
17	4	none	prime
18	4	$d = 2$	not prime ($18 = 2 \cdot 9$)
19	4	none	prime
20	4	$d = 2$	not prime ($20 = 2 \cdot 10$)
21	4	$d = 3$	not prime ($21 = 3 \cdot 7$)
22	4	$d = 2$	not prime ($22 = 2 \cdot 11$)
23	4	none	prime
24	4	$d = 2$	not prime ($24 = 2 \cdot 12$)
25	5	$d = 5$	not prime ($25 = 5 \cdot 5$)
26	5	$d = 2$	not prime ($26 = 2 \cdot 13$)
27	5	$d = 3$	not prime ($27 = 3 \cdot 9$)
28	5	$d = 2$	not prime ($28 = 2 \cdot 14$)
29	5	none	prime
30	5	$d = 2$	not prime ($30 = 2 \cdot 15$)

2. The prime numbers in $\{2, 3, \dots, 30\}$ are

2, 3, 5, 7, 11, 13, 17, 19, 23, 29

There are 10 prime numbers in $\{2, 3, \dots, 30\}$.

Exercise 5.11 — Numerical algorithms (E11)

Consider the following natural numbers:

$$n \in \{6, 19, 49, 53, 91\}$$

Questions

1. Apply the algorithm `PrimalityTest` to each number in the list. For each n , show the execution step by step, indicating the value of $\lfloor \sqrt{n} \rfloor$, the values of d tested, and the result of each test $n \bmod d = 0$.

Solution

1. Execution of `PrimalityTest` on each number.

Check $n = 6$: $\lfloor \sqrt{6} \rfloor = 2$.

d	$6 \bmod d$	action
2	0	return false
6 is not prime ($6 = 2 \cdot 3$).		

Check $n = 19$: $\lfloor \sqrt{19} \rfloor = 4$.

d	$19 \bmod d$	action
2	$1 \neq 0$	continue
3	$1 \neq 0$	continue
4	$3 \neq 0$	continue
Loop ends. Return true : 19 is prime .		

Check $n = 49$: $\lfloor \sqrt{49} \rfloor = 7$.

d	$49 \bmod d$	action
2	$1 \neq 0$	continue
3	$1 \neq 0$	continue
4	$1 \neq 0$	continue
5	$4 \neq 0$	continue
6	$1 \neq 0$	continue
7	0	return false
49 is not prime ($49 = 7 \cdot 7$).		

Check $n = 53$: $\lfloor \sqrt{53} \rfloor = 7$.

d	$53 \bmod d$	action
2	$1 \neq 0$	continue
3	$2 \neq 0$	continue
4	$1 \neq 0$	continue
5	$3 \neq 0$	continue
6	$5 \neq 0$	continue
7	$4 \neq 0$	continue
Loop ends. Return true : 53 is prime .		

Check $n = 91$: $\lfloor \sqrt{91} \rfloor = 9$.

d	$91 \bmod d$	action
2	$1 \neq 0$	continue
3	$1 \neq 0$	continue
4	$3 \neq 0$	continue
5	$1 \neq 0$	continue
6	$1 \neq 0$	continue
7	0	return false
91 is not prime ($91 = 7 \cdot 13$).		

Exercise 5.12 — Numerical algorithms (E12)

Consider the natural number $n = 15$.

Questions

1. Apply the algorithm SieveOfEratosthenes for $n = 15$. Show the execution step by step: indicate the value of $\lfloor \sqrt{15} \rfloor$, and for each prime p found, list the values of j in the inner loop and the corresponding assignments $prime[j] \leftarrow 0$.

2. List all the prime numbers ≤ 15 .

Solution

1. We have $n = 15$ and $\lfloor \sqrt{15} \rfloor = 3$. The array $prime[1..15]$ is initialized to 1 and then $prime[1] \leftarrow 0$. Each row of the table shows the state of the array after the assignment $prime[j] \leftarrow 0$ (blue: entry just set to 0; red: current value of p).

p	j	$prime[1..15]$															action
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
—	—	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	initialization
2	4	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	$prime[4] \leftarrow 0$
	6	0	1	1	0	1	0	1	1	1	1	1	1	1	1	1	$prime[6] \leftarrow 0$
	8	0	1	1	0	1	0	1	0	1	1	1	1	1	1	1	$prime[8] \leftarrow 0$
	10	0	1	1	0	1	0	1	0	1	0	1	1	1	1	1	$prime[10] \leftarrow 0$
	12	0	1	1	0	1	0	1	0	1	0	1	0	1	1	1	$prime[12] \leftarrow 0$
	14	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	$prime[14] \leftarrow 0$
	16																$j = 16 > 15$: stop
3	9	0	1	1	0	1	0	1	0	0	0	1	0	1	0	1	$prime[9] \leftarrow 0$
	12	0	1	1	0	1	0	1	0	0	0	1	0	1	0	1	$prime[12] \leftarrow 0$ (already 0)
	15	0	1	1	0	1	0	1	0	0	0	1	0	1	0	0	$prime[15] \leftarrow 0$
	18																$j = 18 > 15$: stop

Outer loop ends: $p = 4 > \lfloor \sqrt{15} \rfloor = 3$.

2. The prime numbers ≤ 15 are 2, 3, 5, 7, 11, 13.

Exercise 5.13 — Numerical algorithms (E13)

Consider the natural number $n = 30$.

Questions

- Apply the algorithm SieveOfEratosthenes for $n = 30$. Show the execution step by step: indicate the value of $\lfloor \sqrt{30} \rfloor$, and for each prime p found, list the values of j in the inner loop and the corresponding assignments $prime[j] \leftarrow 0$.
- List all the prime numbers ≤ 30 .

Solution

1. We have $n = 30$ and $\lfloor \sqrt{30} \rfloor = 5$. The array $prime[1..30]$ is initialized to 1 and then $prime[1] \leftarrow 0$.

- Iteration $p = 2$:** $prime[2] = 1$, so 2 is prime. Inner loop from $j = p^2 = 4$ with step 2:

j	assignment	note
4	$prime[4] \leftarrow 0$	
6	$prime[6] \leftarrow 0$	
8	$prime[8] \leftarrow 0$	
10	$prime[10] \leftarrow 0$	
12	$prime[12] \leftarrow 0$	
14	$prime[14] \leftarrow 0$	
16	$prime[16] \leftarrow 0$	
18	$prime[18] \leftarrow 0$	
20	$prime[20] \leftarrow 0$	
22	$prime[22] \leftarrow 0$	
24	$prime[24] \leftarrow 0$	
26	$prime[26] \leftarrow 0$	
28	$prime[28] \leftarrow 0$	
30	$prime[30] \leftarrow 0$	
32		$j = 32 > 30$: stop

- **Iteration** $p = 3$: $prime[3] = 1$, so 3 is prime. Inner loop from $j = p^2 = 9$ with step 3:

j	assignment	note
9	$prime[9] \leftarrow 0$	
12	$prime[12] \leftarrow 0$	already 0
15	$prime[15] \leftarrow 0$	
18	$prime[18] \leftarrow 0$	already 0
21	$prime[21] \leftarrow 0$	
24	$prime[24] \leftarrow 0$	already 0
27	$prime[27] \leftarrow 0$	
30	$prime[30] \leftarrow 0$	already 0
33		$j = 33 > 30$: stop

- **Iteration** $p = 4$: $prime[4] = 0$, so 4 is not prime: skip.
- **Iteration** $p = 5$: $prime[5] = 1$, so 5 is prime. Inner loop from $j = p^2 = 25$ with step 5:

j	assignment	note
25	$prime[25] \leftarrow 0$	
30	$prime[30] \leftarrow 0$	already 0
35		$j = 35 > 30$: stop

- Outer loop ends: $p = 6 > \lfloor \sqrt{30} \rfloor = 5$.

Final state of the array $prime[1..30]$ (the entries equal to 1 are in blue):

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$prime[i]$	0	1	1	0	1	0	1	0	0	0	1	0	1	0	0

i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
$prime[i]$	0	1	0	1	0	0	0	1	0	0	0	0	0	1	0

2. The prime numbers ≤ 30 are 2, 3, 5, 7, 11, 13, 17, 19, 23, 29.

Exercise 5.14 — Numerical algorithms (E14)

Consider the following execution trace of the algorithm SieveOfEratosthenes for $n = 12$, where some entries have been replaced by a question mark.

p	j	$prime[1..12]$												action
		1	2	3	4	5	6	7	8	9	10	11	12	
–	–	0	1	1	1	1	1	1	1	1	1	1	1	initialization
2	4	0	1	1	?	1	1	1	1	1	1	1	1	$prime[4] \leftarrow 0$
	6	0	1	1	0	1	?	1	1	1	1	1	1	$prime[6] \leftarrow 0$
	8	0	1	1	0	1	0	1	?	1	1	1	1	$prime[8] \leftarrow 0$
	10	0	1	1	0	1	0	1	0	1	?	1	1	$prime[10] \leftarrow 0$
	12	0	1	1	0	1	0	1	0	1	0	1	?	$prime[12] \leftarrow 0$
	14													$j = 14 > 12$: stop
3	9	0	1	1	0	1	0	1	0	?	0	1	0	$prime[9] \leftarrow 0$
	12	0	1	1	0	1	0	1	0	0	0	1	?	$prime[12] \leftarrow 0$ (already 0)
	15													$j = 15 > 12$: stop

Outer loop ends: $p = 4 > \lfloor \sqrt{12} \rfloor = ?$.

Questions

1. Fill in all the entries marked with a question mark in the table.
2. What is the value of $\lfloor \sqrt{12} \rfloor$? Why can the outer loop stop when $p = 4$?
3. List all the prime numbers ≤ 12 .

Solution

1. Completed table:

p	j	$prime[1..12]$												action
		1	2	3	4	5	6	7	8	9	10	11	12	
–	–	0	1	1	1	1	1	1	1	1	1	1	1	initialization
2	4	0	1	1	0	1	1	1	1	1	1	1	1	$prime[4] \leftarrow 0$
	6	0	1	1	0	1	0	1	1	1	1	1	1	$prime[6] \leftarrow 0$
	8	0	1	1	0	1	0	1	0	1	1	1	1	$prime[8] \leftarrow 0$
	10	0	1	1	0	1	0	1	0	1	0	1	1	$prime[10] \leftarrow 0$
	12	0	1	1	0	1	0	1	0	1	0	1	0	$prime[12] \leftarrow 0$
	14													$j = 14 > 12$: stop
3	9	0	1	1	0	1	0	1	0	0	0	1	0	$prime[9] \leftarrow 0$
	12	0	1	1	0	1	0	1	0	0	0	1	0	$prime[12] \leftarrow 0$ (already 0)
	15													$j = 15 > 12$: stop

Outer loop ends: $p = 4 > \lfloor \sqrt{12} \rfloor = 3$.

Every question mark is the entry $prime[j]$ of the row with index j : it is 0 after the

assignment $prime[j] \leftarrow 0$ (in the row $p = 3$, $j = 12$, the entry was already 0).

2. $\lfloor \sqrt{12} \rfloor = 3$. The outer loop runs for $p = 2, 3$ and stops when $p = 4 > 3$. Every composite number $m \leq 12$ has a prime divisor not larger than $\lfloor \sqrt{12} \rfloor = 3$ (chapter «Numerical algorithms», section «Sieve of Eratosthenes»), so all the composite numbers have already been marked: 4, 6, 8, 10, 12 by $p = 2$ and 9 by $p = 3$.
3. The prime numbers ≤ 12 are 2, 3, 5, 7, 11.

Exercise 5.15 — Numerical algorithms (E15)

Consider the algorithm SieveOfEratosthenes applied to $n = 25$.

Questions

1. Apply the algorithm SieveOfEratosthenes for $n = 25$. Show the execution step by step, indicating the value of $\lfloor \sqrt{25} \rfloor$, and for each prime p the values of j tested in the inner loop and the corresponding assignments $prime[j] \leftarrow 0$.
2. For $p = 5$: what is the first value of j processed by the inner loop? Why is $p^2 = 25$ the starting point and not $2p = 10$?
3. List all the prime numbers ≤ 25 .

Solution

1. We have $n = 25$ and $\lfloor \sqrt{25} \rfloor = 5$. The array $prime[1..25]$ is initialized to 1 and then $prime[1] \leftarrow 0$.

- **Iteration** $p = 2$: $prime[2] = 1$, so 2 is prime. Inner loop from $j = p^2 = 4$ with step 2:

j	assignment	note
4	$prime[4] \leftarrow 0$	
6	$prime[6] \leftarrow 0$	
8	$prime[8] \leftarrow 0$	
10	$prime[10] \leftarrow 0$	
12	$prime[12] \leftarrow 0$	
14	$prime[14] \leftarrow 0$	
16	$prime[16] \leftarrow 0$	
18	$prime[18] \leftarrow 0$	
20	$prime[20] \leftarrow 0$	
22	$prime[22] \leftarrow 0$	
24	$prime[24] \leftarrow 0$	
26		$j = 26 > 25$: stop

- **Iteration** $p = 3$: $prime[3] = 1$, so 3 is prime. Inner loop from $j = p^2 = 9$ with step 3:

j	assignment	note
9	$prime[9] \leftarrow 0$	
12	$prime[12] \leftarrow 0$	already 0
15	$prime[15] \leftarrow 0$	
18	$prime[18] \leftarrow 0$	already 0
21	$prime[21] \leftarrow 0$	
24	$prime[24] \leftarrow 0$	already 0
27		$j = 27 > 25$: stop

- **Iteration** $p = 4$: $prime[4] = 0$, so 4 is not prime: skip.
- **Iteration** $p = 5$: $prime[5] = 1$, so 5 is prime. Inner loop from $j = p^2 = 25$ with step 5:

j	assignment	note
25	$prime[25] \leftarrow 0$	
30		$j = 30 > 25$: stop

- Outer loop ends: $p = 6 > \lfloor \sqrt{25} \rfloor = 5$.

Final state of the array $prime[1..25]$ (the entries equal to 1 are in blue):

i	1	2	3	4	5	6	7	8	9	10	11	12	13
$prime[i]$	0	1	1	0	1	0	1	0	0	0	1	0	1

i	14	15	16	17	18	19	20	21	22	23	24	25
$prime[i]$	0	0	0	1	0	1	0	0	0	1	0	0

- For $p = 5$, the inner loop starts at $j = p^2 = 25$. All the smaller multiples of 5 larger than 5, namely 10, 15, 20, have already been set to 0: $10 = 2 \cdot 5$ and $20 = 2 \cdot 10$ when $p = 2$, and $15 = 3 \cdot 5$ when $p = 3$. In general, every multiple m of p with $p < m < p^2$ has a divisor d with $2 \leq d < p$, hence a prime divisor smaller than p , which has already marked it.
- The prime numbers ≤ 25 are 2, 3, 5, 7, 11, 13, 17, 19, 23.

Exercise 5.16 — Numerical algorithms (E16)

Consider the natural number $n = 50$.

Questions

- Apply the algorithm SieveOfEratosthenes for $n = 50$. For each value of p of the outer loop, report the value of $prime[p]$ and, when p is prime, all the values of j processed by the inner loop (the multiples of p from p^2 to 50), distinguishing the entries newly set to 0 from those already equal to 0.
- List all the prime numbers ≤ 50 and count them.

Solution

1. We have $n = 50$ and $\lfloor \sqrt{50} \rfloor = 7$. The array $prime[1..50]$ is initialized to 1 and then $prime[1] \leftarrow 0$.

p	$prime[p]$	values of j	newly set to 0	already 0
2	1 (prime)	4, 6, ..., 50	4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50	—
3	1 (prime)	9, 12, ..., 48	9, 15, 21, 27, 33, 39, 45	12, 18, 24, 30, 36, 42, 48
4	0 (skip)	—	—	—
5	1 (prime)	25, 30, ..., 50	25, 35	30, 40, 45, 50
6	0 (skip)	—	—	—
7	1 (prime)	49	49	—

Outer loop ends: $p = 8 > \lfloor \sqrt{50} \rfloor = 7$.

Final state of the array $prime[1..50]$ (the entries equal to 1 are in blue):

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$prime[i]$	0	1	1	0	1	0	1	0	0	0	1	0	1	0	0	0	1

i	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
$prime[i]$	0	1	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0

i	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
$prime[i]$	0	0	1	0	0	0	1	0	1	0	0	0	1	0	0	0

2. The prime numbers ≤ 50 are

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47

There are 15 prime numbers ≤ 50 .

Exercise 5.17 — Numerical algorithms (E17)

Consider the following pairs of natural numbers (n, m) :

- (1071, 462)
- (100, 75)
- (17, 5)
- (89, 55)
- (30, 84)

Questions

1. Apply the algorithm EuclideanAlgorithm to compute $\gcd(n, m)$ for all the pairs. For each pair, show the execution step by step in a table, indicating at each step the values of n , m and $r = n \bmod m$, and give the number of iterations of the while loop.
2. Which pairs are coprime?
3. In pair 5 we have $n < m$. What happens in the first iteration?

Solution

1. Executions of the Euclidean algorithm.

(a) Compute $\gcd(1071, 462)$:

step	n	m	$r = n \bmod m$
1	1071	462	$1071 \bmod 462 = 147$
2	462	147	$462 \bmod 147 = 21$
3	147	21	$147 \bmod 21 = 0$
4	21	0	— (loop ends)

Result: $\gcd(1071, 462) = 21$, with 3 iterations of the while loop.

(b) Compute $\gcd(100, 75)$:

step	n	m	$r = n \bmod m$
1	100	75	$100 \bmod 75 = 25$
2	75	25	$75 \bmod 25 = 0$
3	25	0	— (loop ends)

Result: $\gcd(100, 75) = 25$, with 2 iterations of the while loop.

(c) Compute $\gcd(17, 5)$:

step	n	m	$r = n \bmod m$
1	17	5	$17 \bmod 5 = 2$
2	5	2	$5 \bmod 2 = 1$
3	2	1	$2 \bmod 1 = 0$
4	1	0	— (loop ends)

Result: $\gcd(17, 5) = 1$, with 3 iterations of the while loop.

(d) Compute $\gcd(89, 55)$:

step	n	m	$r = n \bmod m$
1	89	55	$89 \bmod 55 = 34$
2	55	34	$55 \bmod 34 = 21$
3	34	21	$34 \bmod 21 = 13$
4	21	13	$21 \bmod 13 = 8$
5	13	8	$13 \bmod 8 = 5$
6	8	5	$8 \bmod 5 = 3$
7	5	3	$5 \bmod 3 = 2$
8	3	2	$3 \bmod 2 = 1$
9	2	1	$2 \bmod 1 = 0$
10	1	0	— (loop ends)

Result: $\gcd(89, 55) = 1$, with 9 iterations of the while loop.

The numbers 89, 55, 34, 21, 13, 8, 5, 3, 2, 1 are consecutive Fibonacci numbers: at every step the quotient is 1, and the remainders decrease as slowly as possible.

(e) Compute $\gcd(30, 84)$:

step	n	m	$r = n \bmod m$
1	30	84	$30 \bmod 84 = 30$
2	84	30	$84 \bmod 30 = 24$
3	30	24	$30 \bmod 24 = 6$
4	24	6	$24 \bmod 6 = 0$
5	6	0	— (loop ends)

Result: $\gcd(30, 84) = 6$, with 4 iterations of the while loop.

- Two numbers are coprime if and only if their greatest common divisor is 1. Hence the coprime pairs are (17, 5) and (89, 55). The pairs (1071, 462), (100, 75) and (30, 84) are not coprime.
- Since $30 < 84$, we have $30 \bmod 84 = 30$, so the first iteration replaces (30, 84) with (84, 30): it simply **swaps** the two numbers. From the second iteration on, the execution is the same as for (84, 30), which needs 3 iterations. Therefore the assumption $n \geq m$ is not needed for correctness: if $n < m$, the algorithm only executes one extra iteration.

Exercise 5.18 — Numerical algorithms (E18)

Consider the following **naïve** algorithm for the greatest common divisor, which tries all the candidate divisors from $\min\{n, m\}$ down to 1.

Algorithm: NaiveGCD**Input** : Two natural numbers $n, m \geq 1$ **Output:** $\gcd(n, m)$

```

1 function NaiveGCD( $n, m$ )
2    $d := \min\{n, m\}$ 
3   while  $n \bmod d \neq 0$  or  $m \bmod d \neq 0$  do
4      $d \leftarrow d - 1$ 
5   end
6   return  $d$ 
7 end

```

Questions Questions 2–4 use the notions of loop invariant, number of steps and input size $|I|$ (chapters «Input size», «Running time» and «Correctness»).

1. Apply NaiveGCD to $(48, 18)$ and $(35, 12)$. How many iterations of the while loop are executed? Compare with the number of iterations of the Euclidean algorithm on the same pairs.
2. Prove that NaiveGCD is correct, using a loop invariant.
3. Compute the number of steps S of NaiveGCD as a function of n , m and $\gcd(n, m)$, counting one step for each execution of a line. Which inputs give the worst case? Compare the two algorithms on $(1000, 999)$.
4. Show that NaiveGCD is exponential in the input size $|I|$.

Solution

1. For $(48, 18)$: d starts from $\min\{48, 18\} = 18$, and the candidates are checked in the order

$$18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6.$$

The first common divisor is $d = 6$ ($48 = 8 \cdot 6$, $18 = 3 \cdot 6$). The while loop body is executed $18 - 6 = 12$ times. The Euclidean algorithm needs only 3 iterations ($48 \bmod 18 = 12$, $18 \bmod 12 = 6$, $12 \bmod 6 = 0$).

For $(35, 12)$: d goes from 12 down to 1, since 35 and 12 are coprime: $12 - 1 = 11$ iterations. The Euclidean algorithm needs 3 iterations ($35 \bmod 12 = 11$, $12 \bmod 11 = 1$, $11 \bmod 1 = 0$).

2. Loop invariant:

Every time the condition at line 3 is evaluated, $1 \leq \gcd(n, m) \leq d \leq \min\{n, m\}$, i.e., no number d' with $d < d' \leq \min\{n, m\}$ is a common divisor of n and m .

- **Initialization.** $d = \min\{n, m\}$, and every common divisor is at most $\min\{n, m\}$.

- **Maintenance.** If the condition is true, d is not a common divisor; since no larger number up to $\min\{n, m\}$ is a common divisor, $\gcd(n, m) < d$, i.e., $\gcd(n, m) \leq d - 1$. Line 4 sets $d \leftarrow d - 1$, and the invariant holds.
- **Termination.** d decreases by 1 at every iteration and, by the invariant, it can never become smaller than $\gcd(n, m) \geq 1$; when $d = \gcd(n, m)$ the condition is false. Hence the loop terminates, and when it terminates d is a common divisor with $d \geq \gcd(n, m)$, hence $d = \gcd(n, m)$, which is returned.

3. Let $k = \min\{n, m\} - \gcd(n, m)$ be the number of iterations. Lines 2 and 6 are executed once, line 3 $k + 1$ times, line 4 k times:

$$S = 1 + (k + 1) + k + 1 = 2k + 3 = 2(\min\{n, m\} - \gcd(n, m)) + 3.$$

The worst case is when n and m are coprime: $S = 2\min\{n, m\} + 1$. For $(1000, 999)$ (consecutive numbers are always coprime) NaiveGCD executes $999 - 1 = 998$ iterations, while the Euclidean algorithm executes only 2 ($1000 \bmod 999 = 1$, $999 \bmod 1 = 0$).

4. Consider the pairs $(n, m) = (M, M - 1)$, with $M \geq 3$: they are coprime, so $S = 2(M - 1) + 1 = 2M - 1 = \Theta(M)$. The input consists of two natural numbers, so $|I| = \Theta(\log_2 M)$ with $M = \max\{n, m\}$, i.e., there exist $c_1, c_2 > 0$ such that

$$2^{\frac{1}{c_2}|I|} \leq M \leq 2^{\frac{1}{c_1}|I|}.$$

Hence, on these inputs, $S = 2^{\Theta(|I|)}$, and the worst-case running time satisfies $T(|I|) = 2^{\Theta(|I|)}$ (it is also at most $2M + 1$ on every input).

NaiveGCD is exponential in the input size, while the Euclidean algorithm executes $O(\log_2 m)$ iterations (see the next exercise): for numbers with hundreds of digits, only the Euclidean algorithm can be used.

Exercise 5.19 — Numerical algorithms (E19)

Consider the Euclidean algorithm.

Algorithm: EuclideanAlgorithm**Input** : Two natural numbers $n, m \geq 1$ **Output:** $\gcd(n, m)$

```

1 function EuclideanAlgorithm( $n, m$ )
2   while  $m \neq 0$  do
3      $r := n \bmod m$ 
4      $n \leftarrow m$ 
5      $m \leftarrow r$ 
6   end
7   return  $n$ 
8 end

```

The Fibonacci numbers are $F_1 = F_2 = 1$ and $F_k = F_{k-1} + F_{k-2}$ for $k \geq 3$:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...

Questions Question 4 uses the notions of running time and input size $|I|$ (chapters «Input size» and «Running time»).

1. Count the number of iterations of the while loop for the pairs of consecutive Fibonacci numbers (5, 3), (8, 5), (13, 8), (21, 13), (34, 21), (55, 34), (89, 55), (144, 89), (233, 144).
2. Prove that, for every $n \geq m \geq 1$,

$$n \bmod m < \frac{n}{2}.$$

3. Using point 2, prove that the number t of iterations of the while loop on input (n_0, m_0) satisfies

$$t < 2 \log_2 m_0 + 2.$$

4. Assuming that each arithmetic operation is a step, determine the worst-case running time $T(|I|)$ of the Euclidean algorithm. Is it polynomial?

Solution

1. For a pair (F_{k+1}, F_k) with $k \geq 3$ we have $F_k < F_{k+1} < 2F_k$, hence $F_{k+1} \bmod F_k = F_{k+1} - F_k = F_{k-1}$: each iteration moves to the previous pair of consecutive Fibonacci numbers, until $(2, 1) \rightarrow (1, 0)$. The number of iterations is:

(n, m)	iterations t	$2\log_2 m + 2$
(5, 3)	3	≈ 5.17
(8, 5)	4	≈ 6.64
(13, 8)	5	8
(21, 13)	6	≈ 9.40
(34, 21)	7	≈ 10.78
(55, 34)	8	≈ 12.17
(89, 55)	9	≈ 13.56
(144, 89)	10	≈ 14.95
(233, 144)	11	≈ 16.34

For example, $(89, 55) \rightarrow (55, 34) \rightarrow (34, 21) \rightarrow (21, 13) \rightarrow (13, 8) \rightarrow (8, 5) \rightarrow (5, 3) \rightarrow (3, 2) \rightarrow (2, 1) \rightarrow (1, 0)$: 9 iterations. The number of iterations grows by 1 each time m is multiplied by about 1.618: it grows logarithmically in m . (Consecutive Fibonacci numbers are, in fact, the worst case of the Euclidean algorithm.)

2. Let $n \geq m \geq 1$. Two cases:

- if $m \leq \frac{n}{2}$, then $n \bmod m < m \leq \frac{n}{2}$;
- if $m > \frac{n}{2}$, then $m \leq n < 2m$, so the quotient of n divided by m is 1 and $n \bmod m = n - m < n - \frac{n}{2} = \frac{n}{2}$.

3. Denote by m_s the value of m after s iterations (m_0 is the input). In iteration $s + 1$ the pair (n, m) becomes $(m, n \bmod m)$, so after iteration $s + 1$ the variable n is equal to m_s and $m_{s+1} < m_s$. Therefore, if iteration $s + 2$ is executed,

$$m_{s+2} = m_s \bmod m_{s+1} \quad \text{with} \quad m_s > m_{s+1} \geq 1,$$

and by point 2, $m_{s+2} < \frac{m_s}{2}$: every two iterations the value of m is at least halved. By induction on k ,

$$m_{2k} < \frac{m_0}{2^k} \quad \text{whenever iteration } 2k \text{ is executed, } k \geq 1.$$

If $t \leq 2$ the bound holds: $t = 1 < 2 \leq 2\log_2 m_0 + 2$, and $t = 2$ is possible only if $m_0 \geq 2$ (for $m_0 = 1$ the first remainder is 0), so $2 < 2\log_2 m_0 + 2$. Let $t \geq 3$ and let $k = \lfloor (t-1)/2 \rfloor \geq 1$. Since $2k \leq t-1$, iteration $2k+1$ is executed, hence $m_{2k} \geq 1$. Therefore

$$1 \leq m_{2k} < \frac{m_0}{2^k} \implies 2^k < m_0 \implies k < \log_2 m_0.$$

Since $k \geq \frac{t-2}{2}$, we get $\frac{t-2}{2} < \log_2 m_0$, i.e.,

$$t < 2\log_2 m_0 + 2.$$

The values in the table of point 1 respect this bound.

4. Each iteration executes a constant number of steps (lines 2–5), so the number of steps satisfies $S = O(t) = O(\log_2 m_0)$. The input consists of two natural numbers,

hence $|I| = \Theta(\log_2 M)$ with $M = \max\{n_0, m_0\} \geq m_0$: there is $c > 0$ such that $\log_2 m_0 \leq \log_2 M \leq \frac{1}{c}|I|$ for all sufficiently large inputs. Therefore

$$T(|I|) = O(|I|).$$

The Euclidean algorithm executes $O(\log_2 m)$ iterations, and its worst-case running time is $T(|I|) = O(|I|)$: it is polynomial (linear) in the input size.

Remark: if the numbers are very large, an operation $n \bmod m$ is not a single step, but it can be executed in a number of steps polynomial in $|I|$; hence the algorithm remains polynomial.

Exercise 5.20 — Numerical algorithms (E20)

Consider the following variant of the Euclidean algorithm, which uses only subtractions (this is the version originally described by Euclid).

Algorithm: SubtractiveGCD

Input : Two natural numbers $n, m \geq 1$

Output: $\gcd(n, m)$

```

1 function SubtractiveGCD( $n, m$ )
2   while  $n \neq m$  do
3     if  $n > m$  then
4        $n \leftarrow n - m$ 
5     end
6     else
7        $m \leftarrow m - n$ 
8     end
9   end
10  return  $n$ 
11 end
```

Questions Questions 3 and 4 use the notions of loop invariant, running time and input size $|I|$ (chapters «Input size», «Running time» and «Correctness»).

1. Apply SubtractiveGCD to $(48, 18)$, showing the values of n and m at every iteration. Compare with the Euclidean algorithm.
2. Prove that, for every $n > m \geq 1$, $\gcd(n, m) = \gcd(n - m, m)$.
3. Prove the correctness of SubtractiveGCD using a loop invariant.
4. How many iterations are executed on the input $(n, 1)$? Is SubtractiveGCD polynomial in the input size?

Solution

1. Execution on $(48, 18)$:

iteration	n	m	action
1	48	18	$n > m$: $n \leftarrow 48 - 18 = 30$
2	30	18	$n > m$: $n \leftarrow 30 - 18 = 12$
3	12	18	$n < m$: $m \leftarrow 18 - 12 = 6$
4	12	6	$n > m$: $n \leftarrow 12 - 6 = 6$
—	6	6	$n = m$: loop ends, return 6

Result: $\gcd(48, 18) = 6$, with 4 iterations. The Euclidean algorithm computes $48 \bmod 18 = 12$ in a single step: subtracting 18 from 48 as long as possible (iterations 1–2) gives exactly the remainder 12. In general, a sequence of consecutive subtractions of the same number is one step of the Euclidean algorithm.

2. Let $n > m \geq 1$. If d is a common divisor of n and m , then $n = k_1 d$ and $m = k_2 d$, so $n - m = (k_1 - k_2)d$, with $k_1 - k_2 \geq 1$ since $n > m$: d is a common divisor of $n - m$ and m . Conversely, if $n - m = k_1 d$ and $m = k_2 d$, then $n = (n - m) + m = (k_1 + k_2)d$: d is a common divisor of n and m . The two pairs have the same common divisors, hence the same greatest common divisor.
3. Denote by n_0, m_0 the input values. Loop invariant:

Every time the condition at line 2 is evaluated, $n \geq 1$, $m \geq 1$ and $\gcd(n, m) = \gcd(n_0, m_0)$.

- **Initialization.** $n = n_0 \geq 1$, $m = m_0 \geq 1$.
- **Maintenance.** Assume the invariant and $n \neq m$. If $n > m$, line 4 replaces n with $n - m \geq 1$ and, by point 2, $\gcd(n - m, m) = \gcd(n, m)$. If $n < m$, line 7 replaces m with $m - n \geq 1$ and, by point 2 with the roles exchanged, $\gcd(n, m - n) = \gcd(n, m)$. In both cases the invariant still holds.
- **Termination.** The natural number $n + m$ decreases by at least 1 at every iteration (we subtract a number ≥ 1), and it is always at least 2. Hence the loop terminates, and it terminates when $n = m$. Then

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

which is returned at line 10.

4. On $(n, 1)$ with $n \geq 2$, every iteration executes $n \leftarrow n - 1$, until $n = 1$: the loop executes $n - 1$ iterations (e.g., $(7, 1)$ needs 6 iterations), while the Euclidean algorithm needs only 1 iteration ($n \bmod 1 = 0$). Each iteration executes a constant number of steps, so on the inputs $(M, 1)$ we have $S = \Theta(M)$. The input size is $|I| = (\lfloor \log_2 M \rfloor + 1) + 1 = \Theta(\log_2 M)$, hence $M = 2^{\Theta(|I|)}$ and

$$T(|I|) = 2^{\Theta(|I|)}.$$

SubtractiveGCD is correct, but it is exponential in the input size. Replacing the repeated subtractions with a single `mod` operation gives the Euclidean algorithm, which is polynomial.