



SAPIENZA  
UNIVERSITÀ DI ROMA

---

# Algorithms

## Exercises 4 — Analysis of algorithms

---

Worked exercises on chapters 11–13: input size, running time, correctness. Every exercise has a complete solution.

**Fabio Furini**

[fabiofurini.github.io](https://fabiofurini.github.io)

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

Online version and downloadable material:

[fabiofurini.github.io/algorithms/](https://fabiofurini.github.io/algorithms/)  
[github.com/fabiofurini/algorithms](https://github.com/fabiofurini/algorithms)

Academic year 2026–2027

---

# Contents

---

|    |              |    |
|----|--------------|----|
| 11 | Input size   | 2  |
| 12 | Running time | 12 |
| 13 | Correctness  | 31 |



## Input size

In the exercises of this chapter the input size is computed with the formulas of the chapter «Input size», summarized below.

The input size  $|I|$  of an instance  $I$  is the total number of bits needed to represent  $I$  in binary:

- one natural number  $n \geq 1$ :

$$|I| = \lfloor \log_2 n \rfloor + 1 = \Theta(\log_2 n);$$

- two natural numbers  $n, m \geq 1$ :

$$|I| = (\lfloor \log_2 n \rfloor + 1) + (\lfloor \log_2 m \rfloor + 1) = \Theta(\log_2 \max\{n, m\});$$

- an array  $a[1..n]$  of natural numbers (an entry equal to 0 uses 1 bit):

$$|I| = \sum_{i=1}^n (\lfloor \log_2(\max\{1, a[i]\}) \rfloor + 1), \quad n \leq |I| \leq n(1 + \log_2 M),$$

where  $M = \max\{1, a[1], a[2], \dots, a[n]\}$ .

A natural number is written with exactly  $k$  bits if and only if  $2^{k-1} \leq n \leq 2^k - 1$ .

### Exercise 11.1 — Input size (E1)

Consider the following input instances:

- (a) the natural numbers  $n = 1, n = 13, n = 255, n = 256, n = 1000, n = 10^6$ ;
- (b) the pairs of natural numbers  $(n, m) = (48, 18)$  and  $(n, m) = (1000, 7)$ ;
- (c) the arrays  $[5, 2, 4, 1, 3], [1, 1, 1, 1], [1024, 1, 1], [0, 7, 8]$ .

### Questions

1. Compute the input size  $|I|$  of every instance in (a), and write the binary representation of 13, 255 and 256.
2. Compute the input size of the two instances in (b).
3. Compute the input size of the four arrays in (c). For each array compare  $|I|$  with the number  $n$  of entries and with the upper bound  $n(1 + \log_2 M)$ , where  $M = \max\{1, a[1], a[2], \dots, a[n]\}$ .
4. What is the largest natural number that can be represented with  $|I| = k$  bits? And the smallest natural number  $n \geq 1$  whose input size is  $k$ ?

### Solution

1. Using  $|I| = \lfloor \log_2 n \rfloor + 1$ :

| $n$    | powers of 2                 | $\lfloor \log_2 n \rfloor$ | $ I $ |
|--------|-----------------------------|----------------------------|-------|
| 1      | $2^0 \leq 1 < 2^1$          | 0                          | 1     |
| 13     | $2^3 \leq 13 < 2^4$         | 3                          | 4     |
| 255    | $2^7 \leq 255 < 2^8$        | 7                          | 8     |
| 256    | $2^8 \leq 256 < 2^9$        | 8                          | 9     |
| 1000   | $2^9 \leq 1000 < 2^{10}$    | 9                          | 10    |
| $10^6$ | $2^{19} \leq 10^6 < 2^{20}$ | 19                         | 20    |

Binary representations:  $13 = (1101)_2$  (4 bits),  $255 = (11111111)_2$  (8 bits),  $256 = (100000000)_2$  (9 bits). Note that 255 and 256 differ by 1, but their input sizes differ by one bit, because  $256 = 2^8$  is a power of 2.

2. For  $(48, 18)$ :  $2^5 \leq 48 < 2^6$  and  $2^4 \leq 18 < 2^5$ , so

$$|I| = (5 + 1) + (4 + 1) = 6 + 5 = 11.$$

For  $(1000, 7)$ :  $2^9 \leq 1000 < 2^{10}$  and  $2^2 \leq 7 < 2^3$ , so

$$|I| = (9 + 1) + (2 + 1) = 10 + 3 = 13.$$

3. Bits of each entry and total:

| array             | bits per entry      | $ I $ | $n$ | $n(1 + \log_2 M)$               |
|-------------------|---------------------|-------|-----|---------------------------------|
| $[5, 2, 4, 1, 3]$ | $3 + 2 + 3 + 1 + 2$ | 11    | 5   | $5(1 + \log_2 5) \approx 16.61$ |
| $[1, 1, 1, 1]$    | $1 + 1 + 1 + 1$     | 4     | 4   | $4(1 + 0) = 4$                  |
| $[1024, 1, 1]$    | $11 + 1 + 1$        | 13    | 3   | $3(1 + 10) = 33$                |
| $[0, 7, 8]$       | $1 + 3 + 4$         | 8     | 3   | $3(1 + 3) = 12$                 |

In all cases  $n \leq |I| \leq n(1 + \log_2 M)$ , in agreement with  $|I| = \Omega(n)$  and  $|I| = O(n(1 + \log_2 M))$ . The upper bound is tight for  $[1, 1, 1, 1]$  (all entries equal), while it is far from  $|I|$  for  $[1024, 1, 1]$ , where only one entry is large.

4. With  $k$  bits the largest representable number is

$$\underbrace{(11 \dots 1)}_k_2 = 2^k - 1,$$

and the smallest number  $n \geq 1$  with  $\lfloor \log_2 n \rfloor + 1 = k$  is  $n = 2^{k-1} = \underbrace{(10 \dots 0)}_{k-1}_2$ . Hence the numbers with input size exactly  $k$  are those with

$$2^{k-1} \leq n \leq 2^k - 1.$$

For example, for  $k = 8$  these are the numbers from 128 to 255.

### Exercise 11.2 — Input size (E2)

Consider the following families of input instances, parametrized by a natural number  $n \geq 1$  (or  $k \geq 1$ ):

- (a) the single natural number  $2^k$ , and the single natural number  $2^k - 1$ ;
- (b) the array  $a[1..n]$  with  $a[i] = i$  for every  $i \in \{1, 2, \dots, n\}$ ;
- (c) the array  $a[1..n]$  with  $a[i] = 2^i$  for every  $i \in \{1, 2, \dots, n\}$ ;
- (d) the array  $a[1..n]$  with  $a[i] = n$  for every  $i \in \{1, 2, \dots, n\}$ ;
- (e) an array  $a[1..n]$  whose entries are natural numbers smaller than  $2^{32}$ , so that each of them can be written with at most 32 bits.

### Questions

1. Compute exactly the input size of the instances in (a), as a function of  $k$ .
2. Compute exactly the input size of the arrays in (c), and express it asymptotically as a function of  $n$ .
3. Express asymptotically the input size of the arrays in (d) and (e).
4. For the arrays in (b), compute  $|I|$  for  $n = 1, 2, 3, 4, 8, 16$ , and prove that  $|I| = \Theta(n \log_2 n)$ .

### Solution

1. Since  $\log_2 2^k = k$ , the number  $2^k$  has input size

$$|I| = \lfloor \log_2 2^k \rfloor + 1 = k + 1.$$

Since  $2^{k-1} \leq 2^k - 1 < 2^k$ , we have  $\lfloor \log_2(2^k - 1) \rfloor = k - 1$  and

$$|I| = k.$$

In both cases  $|I| = \Theta(k)$ , while the numerical value is  $\Theta(2^k)$ : the value is exponential in the input size.

2. The entry  $a[i] = 2^i$  requires  $\lfloor \log_2 2^i \rfloor + 1 = i + 1$  bits, so

$$|I| = \sum_{i=1}^n (i + 1) = \frac{n(n+1)}{2} + n = \frac{n(n+3)}{2}.$$

For example, for  $n = 4$ :  $2 + 3 + 4 + 5 = 14 = \frac{4 \cdot 7}{2}$ . Since

$$\lim_{n \rightarrow \infty} \frac{n(n+3)/2}{n^2} = \lim_{n \rightarrow \infty} \frac{1}{2} \left( 1 + \frac{3}{n} \right) = \frac{1}{2} \in \mathbb{R}_{>0},$$

we have  $|I| = \Theta(n^2)$  by the limit criterion. Directly, since  $n^2 \leq n(n+3) \leq 4n^2$  for every  $n \geq 1$ , the constants  $c_1 = \frac{1}{2}$ ,  $c_2 = 2$  and  $n_1 = 1$  give  $\frac{1}{2}n^2 \leq |I| \leq 2n^2$ . Here the entries do **not** use a constant number of bits, and the input size is quadratic in the number of entries.

3. (d) Every entry is equal to  $n$  and uses  $\lfloor \log_2 n \rfloor + 1$  bits, so

$$|I| = n(\lfloor \log_2 n \rfloor + 1) = \Theta(n \log_2 n),$$

because  $\lfloor \log_2 n \rfloor + 1 = \Theta(\log_2 n)$  (Observation on one natural number), multiplied by  $n$ .

- (e) Every entry uses at least 1 and at most 32 bits, so

$$n \leq |I| \leq 32n,$$

and  $|I| = \Theta(n)$ . This is the hypothesis “every entry uses a constant number of bits” of the corollary on entries of constant size. (If every entry is stored in a memory word of exactly 32 bits, the number of stored bits is  $32n$ , again  $\Theta(n)$ .)

4. The entry  $a[i] = i$  uses  $\lfloor \log_2 i \rfloor + 1$  bits:

| $n$ | $ I $                                |
|-----|--------------------------------------|
| 1   | 1                                    |
| 2   | $1 + 2 = 3$                          |
| 3   | $1 + 2 + 2 = 5$                      |
| 4   | $1 + 2 + 2 + 3 = 8$                  |
| 8   | $1 + 2 + 2 + 3 + 3 + 3 + 3 + 4 = 21$ |
| 16  | $21 + 7 \cdot 4 + 5 = 54$            |

**Upper bound.** Since  $i \leq n$ , each entry uses at most  $\lfloor \log_2 n \rfloor + 1 \leq \log_2 n + 1 \leq 2 \log_2 n$  bits for  $n \geq 2$ , so

$$|I| \leq 2n \log_2 n, \quad \forall n \geq 2.$$

**Lower bound.** Each entry uses at least  $\log_2 i$  bits (since  $\lfloor \log_2 i \rfloor + 1 \geq \log_2 i$ ). Keeping only the indices  $i \geq \lceil n/2 \rceil$ , which are at least  $n/2$  and are at least  $n/2$  in number,

$$|I| \geq \sum_{i=\lceil n/2 \rceil}^n \log_2 i \geq \frac{n}{2} \log_2 \frac{n}{2} = \frac{n}{2} (\log_2 n - 1) \geq \frac{n}{4} \log_2 n,$$

where the last inequality holds when  $\log_2 n - 1 \geq \frac{1}{2} \log_2 n$ , i.e., for  $n \geq 4$ .

Therefore, with  $c_1 = \frac{1}{4}$ ,  $c_2 = 2$  and  $n_1 = 4$ ,

$$0 \leq \frac{1}{4} n \log_2 n \leq |I| \leq 2 n \log_2 n, \quad \forall n \geq 4,$$

i.e.,  $|I| = \Theta(n \log_2 n)$ . (Check: for  $n = 16$ ,  $\frac{1}{4} \cdot 16 \cdot 4 = 16 \leq 54 \leq 2 \cdot 16 \cdot 4 = 128$ .)

### Exercise 11.3 — Input size (E3)

Consider the following algorithm, and recall the **PrimalityTest** algorithm (trial division by  $d = 2, 3, \dots, \lfloor \sqrt{n} \rfloor$ ), whose number of steps in the worst case ( $n$  prime) is  $S(n) = 3\lfloor \sqrt{n} \rfloor + 1$ , with the body of its while loop executed  $\lfloor \sqrt{n} \rfloor - 1$  times.

---

#### Algorithm: SumUpTo

---

**Input :** A natural number  $n \geq 1$

**Output:**  $1 + 2 + \dots + n$

```

1 function SumUpTo(n)
2   s := 0
3   i := 1
4   while i ≤ n do
5     s ← s + i
6     i ← i + 1
7   end
8   return s
9 end
```

---

#### Questions

1. Compute the number of steps  $S(n)$  of **SumUpTo** and determine its worst-case running time  $T(|I|)$ . Is the algorithm polynomial?
2. Propose an algorithm that computes the same output with a constant number of steps (assume that arithmetic operations take a constant number of steps). Is it polynomial?
3. Prove that **PrimalityTest** is exponential in  $|I|$ , even though  $S(n) = \Theta(\sqrt{n})$  is “sub-linear” in the value  $n$ .
4. Give an upper bound on the number of iterations of the while loop of **PrimalityTest** for a prime number  $n$  of 32 bits and of 64 bits. What happens to the number of iterations when the input size doubles?

## Solution

1. The number of times each line of code is executed is:

- **Line 2** is executed 1 time
- **Line 3** is executed 1 time
- **Line 4** is executed  $n + 1$  times
- **Line 5** is executed  $n$  times
- **Line 6** is executed  $n$  times
- **Line 8** is executed 1 time

Therefore

$$S(n) = 1 + 1 + (n + 1) + n + n + 1 = 3n + 4 = \Theta(n).$$

The input is a single natural number, so  $|I| = \lfloor \log_2 n \rfloor + 1 = \Theta(\log_2 n)$ : there exist  $c_1, c_2 > 0$  such that, for all sufficiently large  $n$ ,

$$c_1 \log_2 n \leq |I| \leq c_2 \log_2 n \iff 2^{\frac{1}{c_2}|I|} \leq n \leq 2^{\frac{1}{c_1}|I|}.$$

Since  $a_1 n \leq S(n) \leq a_2 n$  for some  $a_1, a_2 > 0$ ,

$$a_1 2^{\frac{1}{c_2}|I|} \leq S(n) \leq a_2 2^{\frac{1}{c_1}|I|}, \quad \text{hence} \quad T(|I|) = 2^{\Theta(|I|)},$$

since the constant factors  $a_1$  and  $a_2$  are absorbed in the exponent (Observation «constant factors in the exponent» of the chapter). The worst case can also be computed exactly. The inputs of size  $|I|$  are the numbers  $n$  with  $2^{|I|-1} \leq n \leq 2^{|I|} - 1$ , and  $S(n) = 3n + 4$  is largest for  $n = 2^{|I|} - 1$ , so

$$T(|I|) = 3(2^{|I|} - 1) + 4 = 3 \cdot 2^{|I|} + 1.$$

**SumUpTo** is linear in the value  $n$  but **exponential** in the input size: it is not polynomial.

2. By Gauss' formula,  $1 + 2 + \dots + n = \frac{n(n+1)}{2}$ . The algorithm “**return**  $n(n+1)/2$ ” executes a constant number of steps (one sum, one product, one division, one return), so  $T(|I|) = \Theta(1) = O(|I|^0)$ : it is polynomial.

3. The input is a single natural number, so as above  $2^{\frac{1}{c_2}|I|} \leq n \leq 2^{\frac{1}{c_1}|I|}$ . Taking square roots (all terms are positive),

$$2^{\frac{1}{2c_2}|I|} \leq \sqrt{n} \leq 2^{\frac{1}{2c_1}|I|}.$$

Since  $S(n) = \Theta(\sqrt{n})$ , there exist  $a_1, a_2 > 0$  with  $a_1 \sqrt{n} \leq S(n) \leq a_2 \sqrt{n}$ , hence

$$a_1 2^{\frac{1}{2c_2}|I|} \leq S(n) \leq a_2 2^{\frac{1}{2c_1}|I|}, \quad \text{hence} \quad T(|I|) = 2^{\Theta(|I|)},$$

since the constant factors  $a_1$  and  $a_2$  are absorbed in the exponent (Observation «constant factors in the exponent» of the chapter). Being sub-linear in  $n$  does not help:  $\sqrt{n} = 2^{\frac{1}{2} \log_2 n}$  is still exponential in the number of bits  $\log_2 n$ . The algorithm is not polynomial: for every constant  $k$ ,  $\lim_{|I| \rightarrow \infty} 2^{c|I|}/|I|^k = +\infty$  for any  $c > 0$ .

4. The while loop is executed  $\lfloor \sqrt{n} \rfloor - 1$  times for a prime  $n$ .

- 32 bits:  $n < 2^{32}$ , so  $\sqrt{n} < 2^{16} = 65\,536$  and the loop executes fewer than 65 536 iterations.
- 64 bits:  $n < 2^{64}$ , so  $\sqrt{n} < 2^{32} = 4\,294\,967\,296$  and the loop executes up to about  $4.3 \cdot 10^9$  iterations.

When the input size doubles (from 32 to 64 bits) the bound on the number of iterations is **squared** ( $2^{16} \rightarrow 2^{32}$ ), while for a polynomial algorithm with  $T(|I|) = \Theta(|I|^k)$  it would only be multiplied by the constant  $2^k$ .

#### Exercise 11.4 — Input size (E4)

Consider the following two algorithms. Both contain two nested loops.

---

##### Algorithm: CountPairs

---

**Input** : A natural number  $n \geq 1$

**Output**: The number of pairs  $(i, j)$  with  $i, j \in \{1, 2, \dots, n\}$

---

```

1 function CountPairs(n)
2   c := 0
3   i := 1
4   while i ≤ n do
5     j := 1
6     while j ≤ n do
7       c ← c + 1
8       j ← j + 1
9     end
10    i ← i + 1
11  end
12  return c
13 end
```

---

**Algorithm:** CountInversions**Input** : An array  $a[1..n]$  of  $n \geq 1$  natural numbers**Output**: The number of pairs  $i < j$  with  $a[i] > a[j]$ 

```

1 function CountInversions( $a, n$ )
2    $c := 0$ 
3    $i := 1$ 
4   while  $i \leq n - 1$  do
5      $j := i + 1$ 
6     while  $j \leq n$  do
7       if  $a[i] > a[j]$  then
8          $c \leftarrow c + 1$ 
9       end
10       $j \leftarrow j + 1$ 
11    end
12     $i \leftarrow i + 1$ 
13  end
14  return  $c$ 
15 end

```

**Questions**

1. Compute the number of steps  $S(n)$  of **CountPairs** and its worst-case running time  $T(|I|)$ .
2. Compute the number of steps  $S(n)$  of **CountInversions** in the worst case and its worst-case running time  $T(|I|)$ , in general and under the hypothesis that every entry uses a constant number of bits.
3. Both algorithms have  $S(n) = \Theta(n^2)$ . Explain why one is polynomial and the other is exponential in the input size.
4. Suppose that **CountInversions** also receives the number  $n$  as part of the input. Compute  $|I|$  for  $n = 5$ ,  $a = [5, 2, 4, 1, 3]$ , and explain why the conclusion of question 2 does not change.

**Solution**

1. The number of times each line of code is executed is:

- Lines 2, 3 and 12 are executed 1 time each
- Line 4 is executed  $n + 1$  times
- Lines 5 and 10 are executed  $n$  times each
- Line 6 is executed  $n(n + 1)$  times
- Lines 7 and 8 are executed  $n^2$  times each

Therefore

$$S(n) = 3 + (n + 1) + 2n + n(n + 1) + 2n^2 = 3n^2 + 4n + 4 = \Theta(n^2).$$

The input is a single natural number, so  $|I| = \Theta(\log_2 n)$  and, as for every single-number input,  $2^{\frac{1}{c_2}|I|} \leq n \leq 2^{\frac{1}{c_1}|I|}$  for some  $c_1, c_2 > 0$ . Squaring,

$$2^{\frac{2}{c_2}|I|} \leq n^2 \leq 2^{\frac{2}{c_1}|I|},$$

and since  $a_1 n^2 \leq S(n) \leq a_2 n^2$  for some  $a_1, a_2 > 0$ , absorbing the constant factors in the exponent (Observation «constant factors in the exponent» of the chapter) we obtain

$$T(|I|) = 2^{\Theta(|I|)} :$$

**CountPairs** is exponential in the input size.

2. In the worst case (array in strictly decreasing order) line 8 is executed at every iteration of the inner loop. For a fixed  $i$ , the inner loop executes  $n - i$  iterations, for a total of  $\sum_{i=1}^{n-1} (n - i) = \frac{n(n-1)}{2}$ . The number of times each line is executed is:

- **Lines** 2, 3 and 14 are executed 1 time each
- **Line** 4 is executed  $n$  times
- **Lines** 5 and 12 are executed  $n - 1$  times each
- **Line** 6 is executed  $\frac{n(n-1)}{2} + (n - 1)$  times
- **Lines** 7, 8 and 10 are executed  $\frac{n(n-1)}{2}$  times each

Therefore

$$S(n) = 3 + n + 3(n - 1) + 4 \cdot \frac{n(n - 1)}{2} = 2n^2 + 2n = \Theta(n^2).$$

The input is an array, so  $|I| = \Omega(n)$ , i.e.,  $n \leq \frac{1}{c}|I|$ , and

$$S(n) \leq a n^2 \leq \frac{a}{c^2} |I|^2 \implies T(|I|) = O(|I|^2).$$

If every entry uses at most  $c$  bits, with  $c$  a constant, then  $|I| = \Theta(n)$ , and more precisely  $n \geq \frac{1}{c}|I|$ . For a lower bound, note that only line 8 depends on the values: even if it is never executed, the algorithm executes

$$2n^2 + 2n - \frac{n(n - 1)}{2} = \frac{3n^2 + 5n}{2} \geq \frac{3}{2} n^2 \geq \frac{3}{2c^2} |I|^2$$

steps. Every input of size  $|I|$  requires at least this number of steps, so  $T(|I|) = \Omega(|I|^2)$  and, together with the upper bound,

$$T(|I|) = \Theta(|I|^2).$$

In both cases **CountInversions** is polynomial (quadratic).

3. The number of steps is the same function of  $n$ , but  $n$  has a different relation with the input size:
- in **CountPairs**,  $n$  is the **value** of the input, which is written with only  $\Theta(\log_2 n)$  bits, so  $n = 2^{\Theta(|I|)}$ ;

- in `CountInversions`,  $n$  is the **number of entries** of the array, and each entry needs at least one bit, so  $n = O(|I|)$ .

The same  $S(n)$  can correspond to a polynomial or to an exponential running time: it depends on how the parameter  $n$  is related to the input size  $|I|$ .

4. The bits of  $n = 5$  are  $\lfloor \log_2 5 \rfloor + 1 = 3$ , and the bits of the array are  $3+2+3+1+2 = 11$ , so

$$|I| = 3 + 11 = 14.$$

In general  $|I| = (\lfloor \log_2 n \rfloor + 1) + \sum_{i=1}^n (\lfloor \log_2 a[i] \rfloor + 1) \geq n$ , so still  $|I| = \Omega(n)$  and  $T(|I|) = O(|I|^2)$ . With entries of constant size,  $|I| = \Theta(\log_2 n) + \Theta(n) = \Theta(n)$ , so  $T(|I|) = \Theta(|I|^2)$  as before: adding  $n$  to the input adds only  $\Theta(\log_2 n)$  bits, which is negligible with respect to the  $\Omega(n)$  bits of the array.

## Running time

In the exercises of this chapter the steps are counted with the conventions of the chapter «Running time»: every execution of a numbered line of pseudo-code (including the lines `return`) is one step, the condition of a `while` loop whose body is executed  $k$  times is evaluated  $k + 1$  times, and the lines of a branch that is not taken are not executed. After the number of steps  $S(n)$ , the running time is expressed as a function of the input size  $|I|$  with one of the following two rules.

**Rule (A): the input is a natural number  $n$ .** Then  $|I| = \Theta(\log_2 n)$ : there exist constants  $c_1, c_2 > 0$  such that, for all sufficiently large  $n$ ,

$$c_1 \log_2 n \leq |I| \leq c_2 \log_2 n \quad \Longleftrightarrow \quad 2^{\frac{1}{c_2}|I|} \leq n \leq 2^{\frac{1}{c_1}|I|},$$

since  $x \mapsto 2^x$  is strictly increasing and  $2^{\log_2 n} = n$ . If  $S(n) = \Theta(n)$ , there exist constants  $a_1, a_2 > 0$  such that  $a_1 n \leq S(n) \leq a_2 n$  for all sufficiently large  $n$ , and therefore

$$a_1 2^{\frac{1}{c_2}|I|} \leq S(n) \leq a_2 2^{\frac{1}{c_1}|I|}, \quad \text{that is,} \quad T(|I|) = 2^{\Theta(|I|)}.$$

**Rule (B): the input is an array  $a[1..n]$  whose entries use a fixed constant number of bits.** Then  $|I| = \Theta(n)$ : there exist constants  $c_1, c_2 > 0$  such that, for all sufficiently large  $n$ ,

$$c_1 n \leq |I| \leq c_2 n \quad \Longleftrightarrow \quad \frac{1}{c_2}|I| \leq n \leq \frac{1}{c_1}|I|.$$

If  $S(n) = \Theta(n^k)$  for a constant  $k \geq 1$ , there exist constants  $a_1, a_2 > 0$  such that  $a_1 n^k \leq S(n) \leq a_2 n^k$  for all sufficiently large  $n$ , and therefore

$$\frac{a_1}{c_2^k} |I|^k \leq S(n) \leq \frac{a_2}{c_1^k} |I|^k, \quad \text{that is,} \quad T(|I|) = \Theta(|I|^k).$$

### Exercise 12.1 — Running time (E1)

Consider the following algorithm, whose input is a natural number  $n$ .

**Algorithm:** Algorithm 1**Input** : A natural number  $n$ **Output:** A natural number  $m$ 

```

1 function Algorithm1( $n$ )
2    $m := 0$ 
3    $i := 1$ 
4   while  $i \leq n$  do
5      $m \leftarrow m + 2$ 
6      $i \leftarrow i + 1$ 
7   end
8   return  $m$ 
9 end

```

**Questions**

1. What does the algorithm determine?
2. Determine the worst-case running time of the algorithm.

**Solution**

1. The algorithm adds 2 to  $m$  exactly  $n$  times, starting from  $m = 0$ : it determines the double of  $n$ , that is  $2n$ .
2. Let  $S(n)$  denote the number of steps executed by **Algorithm1**, where  $n$  is the natural number given in input. There is only one input for each value of  $n$ , so this is also the worst case. The body of the loop is executed for  $i = 1, 2, \dots, n$ , that is  $n$  times, and the condition of line 4 is evaluated  $n + 1$  times:

| line | number of executions |
|------|----------------------|
| 2    | 1                    |
| 3    | 1                    |
| 4    | $n + 1$              |
| 5    | $n$                  |
| 6    | $n$                  |
| 8    | 1                    |

Therefore, the total number of steps is

$$S(n) = 1 + 1 + (n + 1) + n + n + 1 = 3n + 4 = \Theta(n).$$

We now express the running time as a function of the input size. The input is the natural number  $n$ , hence  $|I| = \Theta(\log_2 n)$ , and  $S(n) = 3n + 4 = \Theta(n)$ . By rule (A) of

the introduction to the exercises, there exist constants  $a_1, a_2, c_1, c_2 > 0$  such that, for all sufficiently large  $n$ ,

$$a_1 2^{\frac{1}{c_2}|I|} \leq S(n) \leq a_2 2^{\frac{1}{c_1}|I|}, \quad \text{hence} \quad T(|I|) = 2^{\Theta(|I|)}.$$

The worst-case running time of **Algorithm1**, as a function of the input size  $|I|$ , is  $T(|I|) = 2^{\Theta(|I|)}$ : it is **exponential** in  $|I|$ , although the number of steps is linear in the value  $n$ .

### Exercise 12.2 — Running time (E2)

Consider the following algorithm, whose input is a natural number  $n$ .

#### Algorithm: Algorithm 2

**Input** : A natural number  $n$

**Output**: A natural number  $m$

```

1 function Algorithm2( $n$ )
2   if  $n \bmod 2 = 0$  then
3      $m := 0$ 
4      $i := 1$ 
5     while  $i \leq n$  do
6        $m \leftarrow m + 2$ 
7        $i \leftarrow i + 1$ 
8     end
9   else
10     $m := n$ 
11  end
12  return  $m$ 
13 end
```

#### Questions

1. What does the algorithm determine?
2. Determine the worst-case running time of the algorithm.

#### Solution

1. The algorithm returns the double  $2n$  of  $n$  if  $n$  is even. If  $n$  is odd, it returns  $n$  unchanged.

2. Let  $S(n)$  denote the number of steps executed by **Algorithm2**, where  $n$  is the natural number given in input. The execution of lines 3–7 depends on the specific input value  $n$ : before executing the algorithm, we do not know whether  $n$  is even or odd. We define a function

$$t : \mathbb{N} \rightarrow \{0, 1\},$$

where  $t(n) = 1$  if  $n \bmod 2 = 0$ , and  $t(n) = 0$  otherwise. In other words,  $t(n)$  is an oracle that tells us whether the input number  $n$  is even. The number of times each line is executed is:

| line | number of executions |
|------|----------------------|
| 2    | 1                    |
| 3    | $t(n)$               |
| 4    | $t(n)$               |
| 5    | $t(n)(n+1)$          |
| 6    | $t(n)n$              |
| 7    | $t(n)n$              |
| 10   | $1 - t(n)$           |
| 12   | 1                    |

Therefore, the total number of steps is

$$1 + t(n) + t(n) + t(n)(n+1) + t(n)n + t(n)n + (1 - t(n)) + 1 = 3nt(n) + 2t(n) + 3.$$

The worst possible case is when  $n$  is even: then  $t(n) = 1$  and the algorithm executes  $3n + 5$  steps, while for  $n$  odd it executes only 3 steps. Every input size of at least two bits is the size of some even numbers, so in the worst case

$$S(n) = 3n + 5 = \Theta(n).$$

We now express the running time as a function of the input size. The input is the natural number  $n$ , hence  $|I| = \Theta(\log_2 n)$ , and  $S(n) = 3n + 5 = \Theta(n)$ . By rule (A) of the introduction to the exercises, there exist constants  $a_1, a_2, c_1, c_2 > 0$  such that, for all sufficiently large  $n$ ,

$$a_1 2^{\frac{1}{c_2}|I|} \leq S(n) \leq a_2 2^{\frac{1}{c_1}|I|}, \quad \text{hence} \quad T(|I|) = 2^{\Theta(|I|)}.$$

The worst-case running time of **Algorithm2**, as a function of the input size  $|I|$ , is  $T(|I|) = 2^{\Theta(|I|)}$ : it is **exponential** in  $|I|$ , although the number of steps is linear in the value  $n$ .

### Exercise 12.3 — Running time (E3)

Consider the following algorithm, whose input is a natural number  $n$ .

**Algorithm:** Algorithm 3**Input** : A natural number  $n$ **Output:** A natural number  $s$ 

```

1 function Algorithm3( $n$ )
2    $s := 0$ 
3    $i := 1$ 
4   while  $i \leq n$  do
5      $s \leftarrow s + i$ 
6      $i \leftarrow i + 1$ 
7   end
8   return  $s$ 
9 end

```

**Questions**

1. What does the algorithm determine?
2. Determine the worst-case running time of the algorithm.

**Solution**

1. The algorithm determines the sum of the first  $n$  positive natural numbers:

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}.$$

2. Let  $S(n)$  denote the number of steps executed by **Algorithm3**, where  $n$  is the natural number given in input (there is only one input for each value of  $n$ ). The body of the loop is executed  $n$  times and its condition  $n + 1$  times:

| line | number of executions |
|------|----------------------|
| 2    | 1                    |
| 3    | 1                    |
| 4    | $n + 1$              |
| 5    | $n$                  |
| 6    | $n$                  |
| 8    | 1                    |

Therefore, the total number of steps is

$$S(n) = 1 + 1 + (n + 1) + n + n + 1 = 3n + 4 = \Theta(n).$$

We now express the running time as a function of the input size. The input is the natural number  $n$ , hence  $|I| = \Theta(\log_2 n)$ , and  $S(n) = 3n + 4 = \Theta(n)$ . By rule (A) of the introduction to the exercises, there exist constants  $a_1, a_2, c_1, c_2 > 0$  such that, for all sufficiently large  $n$ ,

$$a_1 2^{\frac{1}{c_2}|I|} \leq S(n) \leq a_2 2^{\frac{1}{c_1}|I|}, \quad \text{hence} \quad T(|I|) = 2^{\Theta(|I|)}.$$

The worst-case running time of **Algorithm3**, as a function of the input size  $|I|$ , is  $T(|I|) = 2^{\Theta(|I|)}$ : it is **exponential** in  $|I|$ , although the number of steps is linear in the value  $n$ .

### Exercise 12.4 — Running time (E4)

Consider the following algorithm, whose input is a natural number  $n$ .

**Algorithm:** Algorithm 4

**Input :** A natural number  $n$

**Output:** A natural number  $p$

```

1 function Algorithm4( $n$ )
2    $p := 1$ 
3    $i := 1$ 
4   while  $i \leq n$  do
5      $p \leftarrow p \cdot i$ 
6      $i \leftarrow i + 1$ 
7   end
8   return  $p$ 
9 end
```

### Questions

1. What does the algorithm determine?
2. Determine the worst-case running time of the algorithm.

### Solution

1. The algorithm determines the product

$$1 \cdot 2 \cdot \dots \cdot n = n!$$

(with  $0! = 1$ , returned for  $n = 0$ ).

2. Let  $S(n)$  denote the number of steps executed by **Algorithm4**, where  $n$  is the natural number given in input (there is only one input for each value of  $n$ ). The body of the loop is executed  $n$  times and its condition  $n + 1$  times:

| line | number of executions |
|------|----------------------|
| 2    | 1                    |
| 3    | 1                    |
| 4    | $n + 1$              |
| 5    | $n$                  |
| 6    | $n$                  |
| 8    | 1                    |

Therefore, the total number of steps is

$$S(n) = 1 + 1 + (n + 1) + n + n + 1 = 3n + 4 = \Theta(n).$$

We now express the running time as a function of the input size. The input is the natural number  $n$ , hence  $|I| = \Theta(\log_2 n)$ , and  $S(n) = 3n + 4 = \Theta(n)$ . By rule (A) of the introduction to the exercises, there exist constants  $a_1, a_2, c_1, c_2 > 0$  such that, for all sufficiently large  $n$ ,

$$a_1 2^{\frac{1}{c_2}|I|} \leq S(n) \leq a_2 2^{\frac{1}{c_1}|I|}, \quad \text{hence} \quad T(|I|) = 2^{\Theta(|I|)}.$$

The worst-case running time of **Algorithm4**, as a function of the input size  $|I|$ , is  $T(|I|) = 2^{\Theta(|I|)}$ : it is **exponential** in  $|I|$ , although the number of steps is linear in the value  $n$ .

### Exercise 12.5 — Running time (E5)

Consider the following algorithm, whose input is a natural number  $n$ .

**Algorithm:** Algorithm 5**Input** : A natural number  $n$ **Output:** A natural number  $c$ 

```

1 function Algorithm5( $n$ )
2    $c := 0$ 
3    $i := 1$ 
4   while  $i \leq n$  do
5     if  $i \bmod 2 = 0$  then
6        $c \leftarrow c + 1$ 
7     end
8      $i \leftarrow i + 1$ 
9   end
10  return  $c$ 
11 end

```

**Questions**

1. What does the algorithm determine?
2. Determine the worst-case running time of the algorithm.

**Solution**

1. The algorithm determines how many natural numbers between 1 and  $n$  are even, that is  $\lfloor n/2 \rfloor$ .
2. Let  $S(n)$  denote the number of steps executed by **Algorithm5**, where  $n$  is the natural number given in input. Line 6 is executed only when  $i$  is even, that is  $\lfloor n/2 \rfloor$  times. The number of times each line is executed is:

| line | number of executions  |
|------|-----------------------|
| 2    | 1                     |
| 3    | 1                     |
| 4    | $n + 1$               |
| 5    | $n$                   |
| 6    | $\lfloor n/2 \rfloor$ |
| 8    | $n$                   |
| 10   | 1                     |

Therefore, the total number of steps is

$$S(n) = 1 + 1 + (n + 1) + n + \lfloor n/2 \rfloor + n + 1 = 3n + \lfloor n/2 \rfloor + 4.$$

Since  $\frac{n-1}{2} \leq \lfloor n/2 \rfloor \leq \frac{n}{2}$ , we have  $\frac{7}{2}n + \frac{7}{2} \leq S(n) \leq \frac{7}{2}n + 4$ , hence  $S(n) = \Theta(n)$ .

We now express the running time as a function of the input size. The input is the natural number  $n$ , hence  $|I| = \Theta(\log_2 n)$ , and  $S(n) = 3n + \lfloor n/2 \rfloor + 4 = \Theta(n)$ . By rule (A) of the introduction to the exercises, there exist constants  $a_1, a_2, c_1, c_2 > 0$  such that, for all sufficiently large  $n$ ,

$$a_1 2^{\frac{1}{c_2}|I|} \leq S(n) \leq a_2 2^{\frac{1}{c_1}|I|}, \quad \text{hence} \quad T(|I|) = 2^{\Theta(|I|)}.$$

The worst-case running time of **Algorithm5**, as a function of the input size  $|I|$ , is  $T(|I|) = 2^{\Theta(|I|)}$ : it is **exponential** in  $|I|$ , although the number of steps is linear in the value  $n$ .

### Exercise 12.6 — Running time (E6)

Consider the following algorithm, whose input is a natural number  $n$ .

#### Algorithm: Algorithm 6

**Input** : A natural number  $n$

**Output**: A natural number  $q$

```

1 function Algorithm6( $n$ )
2    $q := 0$ 
3    $i := 1$ 
4   while  $i \leq n$  do
5      $q \leftarrow q + n$ 
6      $i \leftarrow i + 1$ 
7   end
8   return  $q$ 
9 end
```

#### Questions

1. What does the algorithm determine?
2. Determine the worst-case running time of the algorithm.

#### Solution

1. The algorithm adds  $n$  to  $q$  exactly  $n$  times, starting from  $q = 0$ : it determines  $n \cdot n = n^2$ .
2. Let  $S(n)$  denote the number of steps executed by **Algorithm6**, where  $n$  is the natural

number given in input (there is only one input for each value of  $n$ ). The body of the loop is executed  $n$  times and its condition  $n + 1$  times:

| line | number of executions |
|------|----------------------|
| 2    | 1                    |
| 3    | 1                    |
| 4    | $n + 1$              |
| 5    | $n$                  |
| 6    | $n$                  |
| 8    | 1                    |

Therefore, the total number of steps is

$$S(n) = 1 + 1 + (n + 1) + n + n + 1 = 3n + 4 = \Theta(n).$$

We now express the running time as a function of the input size. The input is the natural number  $n$ , hence  $|I| = \Theta(\log_2 n)$ , and  $S(n) = 3n + 4 = \Theta(n)$ . By rule (A) of the introduction to the exercises, there exist constants  $a_1, a_2, c_1, c_2 > 0$  such that, for all sufficiently large  $n$ ,

$$a_1 2^{\frac{1}{c_2}|I|} \leq S(n) \leq a_2 2^{\frac{1}{c_1}|I|}, \quad \text{hence} \quad T(|I|) = 2^{\Theta(|I|)}.$$

The worst-case running time of **Algorithm6**, as a function of the input size  $|I|$ , is  $T(|I|) = 2^{\Theta(|I|)}$ : it is **exponential** in  $|I|$ , although the number of steps is linear in the value  $n$ .

### Exercise 12.7 — Running time (E7)

Consider the following algorithm, whose input is an array  $a[1..n]$  of  $n$  integer numbers.

**Algorithm: Algorithm 7****Input** : An array  $a[1..n]$  of  $n$  integer numbers**Output**: An integer number  $s$ 

```

1 function Algorithm7( $a, n$ )
2    $s := 0$ 
3    $i := 1$ 
4   while  $i \leq n$  do
5     if  $a[i] > 0$  then
6        $s \leftarrow s + a[i]$ 
7     end
8      $i \leftarrow i + 1$ 
9   end
10  return  $s$ 
11 end

```

**Questions**

1. What does the algorithm determine?
2. Determine the worst-case running time of the algorithm.

**Solution**

1. The algorithm determines the sum of the positive values in the array  $a[1..n]$ .
2. Let  $S(n)$  denote the number of steps executed by **Algorithm7** in the worst case, where  $n$  is the size of the array. The execution of line 6 depends on the input array. We define a function

$$t : \{1, 2, \dots, n\} \rightarrow \{0, 1\},$$

where  $t(i) = 1$  if  $a[i] > 0$ , and  $t(i) = 0$  otherwise. The number of times each line is executed is:

| line | number of executions |
|------|----------------------|
| 2    | 1                    |
| 3    | 1                    |
| 4    | $n + 1$              |
| 5    | $n$                  |
| 6    | $\sum_{i=1}^n t(i)$  |
| 8    | $n$                  |
| 10   | 1                    |

Therefore, the total number of steps is

$$1 + 1 + (n + 1) + n + \sum_{i=1}^n t(i) + n + 1 = 3n + 4 + \sum_{i=1}^n t(i).$$

The worst possible case is when all entries are positive. Then  $t(i) = 1$  for every  $i \in \{1, 2, \dots, n\}$ , and

$$S(n) = 3n + 4 + \sum_{i=1}^n 1 = 4n + 4 = \Theta(n).$$

We now express the running time as a function of the input size. If all input values use a fixed constant number of bits, then  $|I| = \Theta(n)$ , and  $S(n) = \Theta(n)$ . By rule (B) of the introduction to the exercises (with  $k = 1$ ), there exist constants  $a_1, a_2, c_1, c_2 > 0$  such that, for all sufficiently large  $n$ ,

$$\frac{a_1}{c_2} |I| \leq S(n) \leq \frac{a_2}{c_1} |I|, \quad \text{hence} \quad T(|I|) = \Theta(|I|).$$

The worst-case running time of **Algorithm7**, as a function of the input size  $|I|$ , is  $T(|I|) = \Theta(|I|)$ : it is **linear** in  $|I|$ .

### Exercise 12.8 — Running time (E8)

Consider the following algorithm, whose input is an array  $a[1..n]$  of  $n$  integer numbers.

#### Algorithm: Algorithm 8

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

**Output**: A natural number  $c$

```

1 function Algorithm8( $a, n$ )
2    $c := 0$ 
3    $i := 1$ 
4   while  $i \leq n$  do
5     if  $a[i] \bmod 2 = 0$  then
6        $c \leftarrow c + 1$ 
7     end
8      $i \leftarrow i + 1$ 
9   end
10  return  $c$ 
11 end
```

#### Questions

1. What does the algorithm determine?
2. Determine the worst-case running time of the algorithm.

### Solution

1. The algorithm determines the number of even values in the array  $a[1..n]$ .
2. Let  $S(n)$  denote the number of steps executed by **Algorithm8** in the worst case, where  $n$  is the size of the array. The execution of line 6 depends on the input array. We define a function

$$t : \{1, 2, \dots, n\} \rightarrow \{0, 1\},$$

where  $t(i) = 1$  if  $a[i] \bmod 2 = 0$ , and  $t(i) = 0$  otherwise. The number of times each line is executed is:

| line | number of executions |
|------|----------------------|
| 2    | 1                    |
| 3    | 1                    |
| 4    | $n + 1$              |
| 5    | $n$                  |
| 6    | $\sum_{i=1}^n t(i)$  |
| 8    | $n$                  |
| 10   | 1                    |

Therefore, the total number of steps is

$$1 + 1 + (n + 1) + n + \sum_{i=1}^n t(i) + n + 1 = 3n + 4 + \sum_{i=1}^n t(i).$$

The worst possible case is when all entries are even. Then  $t(i) = 1$  for every  $i \in \{1, 2, \dots, n\}$ , and

$$S(n) = 3n + 4 + \sum_{i=1}^n 1 = 4n + 4 = \Theta(n).$$

We now express the running time as a function of the input size. If all input values use a fixed constant number of bits, then  $|I| = \Theta(n)$ , and  $S(n) = \Theta(n)$ . By rule (B) of the introduction to the exercises (with  $k = 1$ ), there exist constants  $a_1, a_2, c_1, c_2 > 0$  such that, for all sufficiently large  $n$ ,

$$\frac{a_1}{c_2} |I| \leq S(n) \leq \frac{a_2}{c_1} |I|, \quad \text{hence} \quad T(|I|) = \Theta(|I|).$$

The worst-case running time of **Algorithm8**, as a function of the input size  $|I|$ , is  $T(|I|) = \Theta(|I|)$ : it is **linear** in  $|I|$ .

**Exercise 12.9 — Running time (E9)**

Consider the following algorithm, whose input is an array  $a[1..n]$  of  $n \geq 1$  integer numbers.

**Algorithm:** Algorithm 9

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

**Output:** An integer number  $m$

```

1 function Algorithm9( $a, n$ )
2    $m := a[1]$ 
3    $i := 2$ 
4   while  $i \leq n$  do
5     if  $a[i] < m$  then
6        $m \leftarrow a[i]$ 
7     end
8      $i \leftarrow i + 1$ 
9   end
10  return  $m$ 
11 end

```

**Questions**

1. What does the algorithm determine?
2. Determine the worst-case running time of the algorithm.

**Solution**

1. The algorithm determines the minimum value in the array  $a[1..n]$ .
2. Let  $S(n)$  denote the number of steps executed by **Algorithm9** in the worst case, where  $n$  is the size of the array. The body of the loop is executed for  $i = 2, 3, \dots, n$ , that is  $n - 1$  times. We define

$$t : \{2, 3, \dots, n\} \rightarrow \{0, 1\},$$

where  $t(i) = 1$  if line 6 is executed at iteration  $i$ , and  $t(i) = 0$  otherwise. The number of times each line is executed is:

| line | number of executions |
|------|----------------------|
| 2    | 1                    |
| 3    | 1                    |
| 4    | $n$                  |
| 5    | $n - 1$              |
| 6    | $\sum_{i=2}^n t(i)$  |
| 8    | $n - 1$              |
| 10   | 1                    |

Therefore, the total number of steps is

$$1 + 1 + n + (n - 1) + \sum_{i=2}^n t(i) + (n - 1) + 1 = 3n + 1 + \sum_{i=2}^n t(i).$$

The worst possible case is when

$$a[1] > a[2] > \dots > a[n].$$

Then  $t(i) = 1$  for every  $i \in \{2, 3, \dots, n\}$ , and

$$S(n) = 3n + 1 + \sum_{i=2}^n 1 = 3n + 1 + (n - 1) = 4n = \Theta(n).$$

We now express the running time as a function of the input size. If all input values use a fixed constant number of bits, then  $|I| = \Theta(n)$ , and  $S(n) = \Theta(n)$ . By rule (B) of the introduction to the exercises (with  $k = 1$ ), there exist constants  $a_1, a_2, c_1, c_2 > 0$  such that, for all sufficiently large  $n$ ,

$$\frac{a_1}{c_2} |I| \leq S(n) \leq \frac{a_2}{c_1} |I|, \quad \text{hence} \quad T(|I|) = \Theta(|I|).$$

The worst-case running time of **Algorithm9**, as a function of the input size  $|I|$ , is  $T(|I|) = \Theta(|I|)$ : it is **linear** in  $|I|$ .

### Exercise 12.10 — Running time (E10)

Consider the following algorithm, whose input is an array  $a[1..n]$  of  $n \geq 1$  integer numbers.

**Algorithm:** Algorithm 10**Input** : An array  $a[1..n]$  of  $n \geq 1$  integer numbers**Output:** true or false

```

1 function Algorithm10( $a, n$ )
2    $i := 1$ 
3   while  $i < n$  do
4     if  $a[i] = a[i + 1]$  then
5       return true
6     end
7      $i \leftarrow i + 1$ 
8   end
9   return false
10 end

```

**Questions**

1. What does the algorithm determine?
2. Determine the worst-case running time of the algorithm.

**Solution**

1. The algorithm determines whether the array contains two consecutive equal values: it returns **true** if  $a[i] = a[i + 1]$  for some  $i \in \{1, 2, \dots, n - 1\}$ , and **false** otherwise.
2. Let  $S(n)$  denote the number of steps executed by Algorithm10 in the worst case, where  $n$  is the size of the array. If the first pair of consecutive equal values is found at iteration  $k$ , the algorithm stops at line 5 after  $1 + k + k + (k - 1) + 1 = 3k + 1 \leq 3(n - 1) + 1$  steps. The worst possible case is therefore when no two consecutive values are equal: line 5 is never executed, and the algorithm checks all the  $n - 1$  consecutive pairs. The number of times each line is executed in this case is:

| line | number of executions |
|------|----------------------|
| 2    | 1                    |
| 3    | $n$                  |
| 4    | $n - 1$              |
| 5    | 0                    |
| 7    | $n - 1$              |
| 9    | 1                    |

Therefore, the total number of steps is

$$S(n) = 1 + n + (n - 1) + (n - 1) + 1 = 3n = \Theta(n).$$

(and  $3n > 3(n-1) + 1$ , so this is indeed the worst case).

We now express the running time as a function of the input size. If all input values use a fixed constant number of bits, then  $|I| = \Theta(n)$ , and  $S(n) = \Theta(n)$ . By rule (B) of the introduction to the exercises (with  $k = 1$ ), there exist constants  $a_1, a_2, c_1, c_2 > 0$  such that, for all sufficiently large  $n$ ,

$$\frac{a_1}{c_2} |I| \leq S(n) \leq \frac{a_2}{c_1} |I|, \quad \text{hence} \quad T(|I|) = \Theta(|I|).$$

The worst-case running time of **Algorithm10**, as a function of the input size  $|I|$ , is  $T(|I|) = \Theta(|I|)$ : it is **linear** in  $|I|$ .

### Exercise 12.11 — Running time (E11)

Consider the following algorithm, whose input is an array  $a[1..n]$  of  $n \geq 1$  integer numbers.

#### Algorithm: Algorithm 11

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

**Output**: A natural number  $c$

```

1 function Algorithm11( $a, n$ )
2    $c := 0$ 
3    $i := 1$ 
4   while  $i \leq n - 1$  do
5      $j := i + 1$ 
6     while  $j \leq n$  do
7       if  $a[i] < a[j]$  then
8          $c \leftarrow c + 1$ 
9       end
10       $j \leftarrow j + 1$ 
11    end
12     $i \leftarrow i + 1$ 
13  end
14  return  $c$ 
15 end
```

#### Questions

1. What does the algorithm determine?
2. Determine the worst-case running time of the algorithm.

## Solution

1. The algorithm determines the number of pairs of positions  $(i, j)$ , with  $1 \leq i < j \leq n$ , such that

$$a[i] < a[j].$$

2. Let  $S(n)$  denote the number of steps executed by **Algorithm11** in the worst case, where  $n$  is the size of the array. The outer loop is executed for  $i = 1, 2, \dots, n-1$  (its condition is evaluated  $n$  times), and for each  $i$  the inner loop is executed for  $j = i+1, \dots, n$ , that is  $n-i$  times (its condition is evaluated  $n-i+1$  times). Hence the body of the inner loop is executed once for every pair  $(i, j)$  with  $1 \leq i < j \leq n$ , and the number of such pairs is

$$P = (n-1) + (n-2) + \dots + 1 = \frac{n(n-1)}{2}.$$

We define  $t(i, j) = 1$  if line 8 is executed for the pair  $(i, j)$ , and  $t(i, j) = 0$  otherwise. The number of times each line is executed is:

| line | number of executions                   |
|------|----------------------------------------|
| 2    | 1                                      |
| 3    | 1                                      |
| 4    | $n$                                    |
| 5    | $n-1$                                  |
| 6    | $\sum_{i=1}^{n-1} (n-i+1) = P + (n-1)$ |
| 7    | $P$                                    |
| 8    | $\sum_{1 \leq i < j \leq n} t(i, j)$   |
| 10   | $P$                                    |
| 12   | $n-1$                                  |
| 14   | 1                                      |

Therefore, the total number of steps is

$$\begin{aligned} & 1 + 1 + n + (n-1) + (P + (n-1)) + P + \sum_{1 \leq i < j \leq n} t(i, j) + P + (n-1) + 1 \\ &= 4n + 3P + \sum_{1 \leq i < j \leq n} t(i, j). \end{aligned}$$

The worst possible case for line 8 is when

$$a[1] < a[2] < \dots < a[n] :$$

then  $t(i, j) = 1$  for every pair, and

$$S(n) = 4n + 4P = 4n + 2n(n-1) = 2n^2 + 2n = \Theta(n^2).$$

For example, for  $n = 4$  and  $a[1..4] = (1, 2, 3, 4)$  the algorithm returns  $c = 6$  after  $S(4) = 2 \cdot 16 + 8 = 40$  steps.

We now express the running time as a function of the input size. If all input values use a fixed constant number of bits, then  $|I| = \Theta(n)$ , and  $S(n) = \Theta(n^2)$ . By rule (B) of the introduction to the exercises (with  $k = 2$ ), there exist constants  $a_1, a_2, c_1, c_2 > 0$  such that, for all sufficiently large  $n$ ,

$$\frac{a_1}{c_2} |I|^2 \leq S(n) \leq \frac{a_2}{c_1} |I|^2, \quad \text{hence} \quad T(|I|) = \Theta(|I|^2).$$

The worst-case running time of **Algorithm11**, as a function of the input size  $|I|$ , is  $T(|I|) = \Theta(|I|^2)$ : it is **quadratic** in  $|I|$ .

---

## Correctness

---

In the exercises of this chapter the correctness of a loop is proved with a loop invariant, as in the chapter «Correctness», following the scheme below.

### Warning

A loop invariant is a property of the variables that is true every time the condition of the loop is evaluated. To prove the correctness of the loop:

1. **Initialization:** the invariant is true before the first iteration;
2. **Maintenance:** if the invariant is true before an iteration, it is true before the next one;
3. **Termination:** the loop terminates after a finite number of iterations (a natural number strictly decreases at each iteration), and the invariant, together with the negated condition of the loop, gives the correct output.

A sub-array  $a[l..r]$  with  $l > r$  is empty, and an empty sum is equal to 0. To show that an algorithm is **wrong**, one counterexample is enough.

### Exercise 13.1 — Correctness (E1)

Consider the following algorithm, which computes the minimum value of an array  $a[1..n]$  of  $n \geq 1$  natural numbers.

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

```

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

```

**Questions**

1. State a loop invariant for the while loop of FindMinimum.
2. Verify the loop invariant on the array  $a = [7, 3, 9, 2, 5, 2]$ , showing the values of  $i$  and  $min$  every time the condition at line 4 is evaluated.
3. Prove the correctness of FindMinimum (initialization, maintenance, termination).
4. A student replaces line 2 with  $min := 0$ . Show, with a counterexample, that the new algorithm is wrong, and indicate which property of the loop invariant fails.

**Solution**

1. Loop invariant:

Every time the condition at line 4 is evaluated,

$$min = \min\{a[1], a[2], \dots, a[i-1]\}.$$

2. For  $a = [7, 3, 9, 2, 5, 2]$  ( $n = 6$ ):

| $i$ | $min$ | $a[1..i-1]$        | $\min\{a[1], \dots, a[i-1]\}$ |
|-----|-------|--------------------|-------------------------------|
| 2   | 7     | [7]                | 7                             |
| 3   | 3     | [7, 3]             | 3                             |
| 4   | 3     | [7, 3, 9]          | 3                             |
| 5   | 2     | [7, 3, 9, 2]       | 2                             |
| 6   | 2     | [7, 3, 9, 2, 5]    | 2                             |
| 7   | 2     | [7, 3, 9, 2, 5, 2] | 2 (loop ends)                 |

The invariant holds at every evaluation, and the algorithm returns 2.

## 3. Proof of correctness.

- **Initialization.** Before the first iteration  $\min = a[1]$  and  $i = 2$ . The sub-array  $a[1..1]$  has the single element  $a[1]$ , whose minimum is  $a[1] = \min$ .
- **Maintenance.** Assume  $\min = \min\{a[1], \dots, a[i-1]\}$  and  $i \leq n$ . If  $a[i] < \min$ , line 6 sets  $\min \leftarrow a[i]$ , which is smaller than all the elements of  $a[1..i-1]$ ; otherwise  $\min \leq a[i]$  and  $\min$  is not changed. In both cases  $\min = \min\{a[1], \dots, a[i]\}$ . Line 8 sets  $i \leftarrow i + 1$ , so the invariant holds with the new value of  $i$ .
- **Termination.** The variable  $i$  starts from  $2 \leq n + 1$  and increases by 1 at every iteration, so the loop terminates after  $n - 1$  iterations, when  $i = n + 1$ . Substituting  $i = n + 1$  in the invariant:

$$\min = \min\{a[1], \dots, a[n]\},$$

which is returned at line 10. Hence the algorithm is correct.

4. With  $\min := 0$ , consider  $a = [3, 5]$ . Since no element is smaller than 0, line 6 is never executed and the algorithm returns 0, while the correct output is 3. Hence the algorithm is wrong.

The property that fails is the **initialization**: before the first iteration  $i = 2$  and  $\min = 0$ , while  $\min\{a[1]\} = a[1] = 3 \neq 0$ . Maintenance is still true (if the invariant held, it would continue to hold), but since it is false at the beginning, the termination argument cannot be applied. In fact, since the entries are natural numbers, the algorithm always returns  $\min\{0, a[2], \dots, a[n]\} = 0$ , which is correct only if at least one entry of the array is equal to 0.

## Exercise 13.2 — Correctness (E2)

Consider the following algorithm.

**Algorithm:** ArraySum

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

**Output**: A natural number  $s$

```

1 function ArraySum( $a, n$ )
2    $s := 0$ 
3    $i := 1$ 
4   while  $i \leq n$  do
5      $s \leftarrow s + a[i]$ 
6      $i \leftarrow i + 1$ 
7   end
8   return  $s$ 
9 end
```

### Questions

1. What does the algorithm compute? State a loop invariant for its while loop.
2. Verify the loop invariant on the array  $a = [4, 1, 7, 3]$ , showing the values of  $i$  and  $s$  every time the condition at line 4 is evaluated.
3. Prove the correctness of **ArraySum**.
4. A student modifies line 3 into  $i := 0$  and line 4 into “**while**  $i < n$ ”, keeping  $a[i]$  at line 5. Is the new algorithm correct?

### Solution

1. The algorithm computes the sum of the elements of the array:

$$s = \sum_{k=1}^n a[k].$$

Loop invariant:

Every time the condition at line 4 is evaluated,

$$s = \sum_{k=1}^{i-1} a[k],$$

where an empty sum ( $i = 1$ ) is equal to 0.

2. For  $a = [4, 1, 7, 3]$  ( $n = 4$ ):

| $i$ | $s$ | $\sum_{k=1}^{i-1} a[k]$          |
|-----|-----|----------------------------------|
| 1   | 0   | 0 (empty sum)                    |
| 2   | 4   | 4                                |
| 3   | 5   | $4 + 1 = 5$                      |
| 4   | 12  | $4 + 1 + 7 = 12$                 |
| 5   | 15  | $4 + 1 + 7 + 3 = 15$ (loop ends) |

The algorithm returns 15.

3. Proof of correctness.

- **Initialization.** Before the first iteration  $s = 0$  and  $i = 1$ ; the sum  $\sum_{k=1}^0 a[k]$  is empty, hence equal to  $0 = s$ .
- **Maintenance.** Assume  $s = \sum_{k=1}^{i-1} a[k]$  and  $i \leq n$ . Line 5 gives

$$s + a[i] = \sum_{k=1}^{i-1} a[k] + a[i] = \sum_{k=1}^i a[k],$$

and line 6 increases  $i$  by 1, so the invariant holds with the new value of  $i$ .

- **Termination.** The variable  $i$  starts from 1 and increases by 1 at every iteration, so the loop terminates after  $n$  iterations, when  $i = n + 1$ . Substituting  $i = n + 1$  in the invariant gives  $s = \sum_{k=1}^n a[k]$ , which is returned at line 8. Hence the algorithm is correct.
4. No. With  $i := 0$  the first iteration accesses  $a[0]$ , which does not exist in an array  $a[1..n]$ ; moreover, the condition  $i < n$  stops the loop when  $i = n$ , so  $a[n]$  is never added. In terms of the invariant, the natural invariant of the new loop is  $s = \sum_{k=0}^{i-1} a[k]$ , and at termination ( $i = n$ ) it gives  $s = \sum_{k=0}^{n-1} a[k] \neq \sum_{k=1}^n a[k]$ : the termination step does not give the required property. For example, on  $a = [4, 1, 7, 3]$  the element  $a[4] = 3$  is never added, so even ignoring the invalid access the result cannot be 15. This is a typical error obtained by mixing 0-indexed code (as in C++) with 1-indexed pseudo-code.

### Exercise 13.3 — Correctness (E3)

Consider the **SelectionSort** algorithm, which sorts an array  $a[1..n]$  of  $n \geq 1$  natural numbers in non-decreasing order.

---

#### Algorithm: SelectionSort

---

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

**Output**: The array  $a$  sorted in non-decreasing order

---

```

1 function SelectionSort( $a, n$ )
2    $i := 1$ 
3   while  $i \leq n - 1$  do
4      $minIdx := i$                                 // index of the minimum in  $a[i..n]$ 
5      $j := i + 1$ 
6     while  $j \leq n$  do
7       if  $a[j] < a[minIdx]$  then
8          $minIdx \leftarrow j$                         // new minimum found
9       end
10       $j \leftarrow j + 1$ 
11    end
12    if  $minIdx \neq i$  then
13       $tmp \leftarrow a[i]$                             // swap  $a[i]$  and  $a[minIdx]$ 
14       $a[i] \leftarrow a[minIdx]$ 
15       $a[minIdx] \leftarrow tmp$ 
16    end
17     $i \leftarrow i + 1$ 
18  end
19 end

```

---

#### Questions

1. State a loop invariant for the inner while loop (line 6) and prove it.

2. State a loop invariant for the outer while loop (line 3).
3. Verify the invariant of the outer loop on the array  $a = [29, 10, 14, 37, 13]$ .
4. Prove the correctness of **SelectionSort**.

### Solution

1. Consider an iteration of the outer loop with a given value of  $i$ . The inner loop does not modify the array. Its invariant is:

Every time the condition at line 6 is evaluated,  $i \leq \text{minIdx} \leq j - 1$  and

$$a[\text{minIdx}] = \min\{a[i], a[i+1], \dots, a[j-1]\}.$$

- **Initialization.** Lines 4–5 set  $\text{minIdx} = i$  and  $j = i+1$ : the sub-array  $a[i..j-1] = a[i..i]$  has the single element  $a[i] = a[\text{minIdx}]$ .
- **Maintenance.** Assume the invariant and  $j \leq n$ . If  $a[j] < a[\text{minIdx}]$ , then  $a[j]$  is smaller than all the elements of  $a[i..j-1]$  and line 8 sets  $\text{minIdx} \leftarrow j$ ; otherwise  $a[\text{minIdx}] \leq a[j]$ . In both cases  $a[\text{minIdx}] = \min\{a[i], \dots, a[j]\}$  and  $i \leq \text{minIdx} \leq j$ . Line 10 sets  $j \leftarrow j+1$ , so the invariant holds with the new  $j$ .
- **Termination.**  $j$  increases by 1 from  $i+1 \leq n$ , so the loop stops when  $j = n+1$ . Then  $a[\text{minIdx}] = \min\{a[i], \dots, a[n]\}$ :  $\text{minIdx}$  is the index of a minimum element of  $a[i..n]$ .

2. Invariant of the outer loop:

Every time the condition at line 3 is evaluated:

- (P1) the array  $a[1..n]$  contains the original elements (it is a rearrangement of the input);
- (P2) the sub-array  $a[1..i-1]$  is sorted in non-decreasing order;
- (P3) every element of  $a[1..i-1]$  is  $\leq$  every element of  $a[i..n]$ .

3. For  $a = [29, 10, 14, 37, 13]$  ( $n = 5$ ); the sorted prefix  $a[1..i-1]$  is highlighted in **blue**:

| $i$ | $a[1]$ | $a[2]$ | $a[3]$ | $a[4]$ | $a[5]$ | minimum of $a[i..n]$            |
|-----|--------|--------|--------|--------|--------|---------------------------------|
| 1   | 29     | 10     | 14     | 37     | 13     | $a[2] = 10$ : swap $a[1], a[2]$ |
| 2   | 10     | 29     | 14     | 37     | 13     | $a[5] = 13$ : swap $a[2], a[5]$ |
| 3   | 10     | 13     | 14     | 37     | 29     | $a[3] = 14$ : no swap           |
| 4   | 10     | 13     | 14     | 37     | 29     | $a[5] = 29$ : swap $a[4], a[5]$ |
| 5   | 10     | 13     | 14     | 29     | 37     | loop ends ( $i = n$ )           |

At every row the blue prefix is sorted, and each blue element is  $\leq$  every non-blue element: e.g. for  $i = 4$ ,  $\max\{10, 13, 14\} = 14 \leq \min\{37, 29\} = 29$ .

4. Proof of correctness.

- **Initialization.** Before the first iteration  $i = 1$  and nothing has been modified: (P1) holds, and  $a[1..0]$  is empty, so (P2) and (P3) hold trivially.
- **Maintenance.** Assume (P1)–(P3) and  $i \leq n - 1$ . By point 1, after the inner loop

$$a[\text{minIdx}] = \min\{a[i], \dots, a[n]\}.$$

Lines 12–15 swap  $a[i]$  and  $a[\text{minIdx}]$  (if they are different positions). A swap only exchanges two elements, so (P1) still holds. Now  $a[i]$  is the minimum of  $a[i..n]$ , so:

- by (P3),  $a[i] \geq$  every element of  $a[1..i - 1]$ , and since  $a[1..i - 1]$  is sorted (P2),  $a[1..i]$  is sorted;
- $a[i] \leq$  every element of  $a[i + 1..n]$ , because  $a[i]$  is the minimum of  $a[i..n]$  and the swap only rearranges the values in positions  $i..n$ ; for the same reason, by (P3) every element of  $a[1..i - 1]$  is  $\leq$  every element of  $a[i + 1..n]$ . Hence every element of  $a[1..i]$  is  $\leq$  every element of  $a[i + 1..n]$ .

Line 17 sets  $i \leftarrow i + 1$ , so (P1)–(P3) hold with the new value of  $i$ .

- **Termination.** The inner loop always terminates (point 1), and  $i$  increases by 1 from 1; the outer loop stops when  $i = n$  for the first time (after  $n - 1$  iterations; if  $n = 1$  it is never executed and  $i = 1 = n$ ). Substituting  $i = n$ :  $a[1..n - 1]$  is sorted (P2) and every element of  $a[1..n - 1]$  is  $\leq a[n]$  (P3); hence the whole array  $a[1..n]$  is sorted and, by (P1), it contains the original elements. Hence the algorithm is correct.

Note that the outer loop needs only  $n - 1$  iterations: when the first  $n - 1$  positions are correct, the last element is automatically in its correct position.

#### Exercise 13.4 — Correctness (E4)

Consider the `LinearSearch` algorithm, which determines if a target value  $m$  is one of the elements of an array  $a[1..n]$  of  $n$  natural numbers (not necessarily sorted).

**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

```

**Questions**

1. State a loop invariant for the while loop of LinearSearch.
2. Verify the loop invariant on the array  $a = [8, 3, 6, 3, 9]$  for  $m = 6$  and for  $m = 5$ .
3. Prove that LinearSearch is correct. Prove also that, when  $m$  is in the array, the returned index is the **smallest** index  $k$  such that  $a[k] = m$ .
4. A student swaps lines 4–5 and line 7, i.e., the body of the loop first increases  $i$  and then checks whether  $a[i] = m$ . Find a counterexample showing that the new algorithm is wrong, and indicate which property of the loop invariant fails.

**Solution**

1. Loop invariant:

Every time the condition at line 3 is evaluated,  $1 \leq i \leq n + 1$  and  $m$  is **not** in the sub-array  $a[1..i - 1]$ :

$$a[k] \neq m \quad \text{for every } k \in \{1, 2, \dots, i - 1\}.$$

2. For  $a = [8, 3, 6, 3, 9]$ :

| $m$ | $i$ | $a[1..i-1]$     | action                                      |
|-----|-----|-----------------|---------------------------------------------|
| 6   | 1   | [ ]             | $a[1] = 8 \neq 6$                           |
|     | 2   | [8]             | $a[2] = 3 \neq 6$                           |
|     | 3   | [8, 3]          | $a[3] = 6 = m \Rightarrow \text{return } 3$ |
| 5   | 1   | [ ]             | $a[1] = 8 \neq 5$                           |
|     | 2   | [8]             | $a[2] = 3 \neq 5$                           |
|     | 3   | [8, 3]          | $a[3] = 6 \neq 5$                           |
|     | 4   | [8, 3, 6]       | $a[4] = 3 \neq 5$                           |
|     | 5   | [8, 3, 6, 3]    | $a[5] = 9 \neq 5$                           |
|     | 6   | [8, 3, 6, 3, 9] | $i > n \Rightarrow \text{return } -1$       |

At every evaluation,  $m$  does not appear in  $a[1..i-1]$ .

### 3. Proof of correctness.

- **Initialization.** Before the first iteration  $i = 1$  and  $a[1..0]$  is empty, so it does not contain  $m$ .
- **Maintenance.** Assume the invariant and  $i \leq n$ . If  $a[i] = m$ , the algorithm returns  $i$  (line 5), which is a correct output. Otherwise  $a[i] \neq m$ , so  $m$  is not in  $a[1..i]$ ; line 7 sets  $i \leftarrow i + 1$  and the invariant holds with the new value of  $i$ .
- **Termination.**  $i$  increases by 1 from 1, so the loop executes at most  $n$  iterations. The algorithm ends in one of two ways:
  - it returns  $i$  with  $a[i] = m$ : correct. Moreover, by the invariant  $a[k] \neq m$  for all  $k < i$ , so  $i$  is the smallest index with  $a[i] = m$ ;
  - the condition at line 3 is false, i.e.,  $i = n + 1$ : by the invariant  $m$  is not in  $a[1..n]$ , and the algorithm correctly returns  $-1$ .

Hence the algorithm is correct.

4. The new body is:  $i \leftarrow i + 1$ ; **if**  $a[i] = m$  **then return**  $i$ . Consider  $a = [6, 3]$  and  $m = 6$ . In the first iteration  $i$  becomes 2 and  $a[2] = 3 \neq 6$ ; in the second iteration  $i$  becomes 3 and the algorithm accesses  $a[3]$ , which is outside the array; in any case  $a[1] = 6$  is **never** checked, so the algorithm cannot return 1.

The property that fails is the **maintenance**: if  $m$  is not in  $a[1..i-1]$  at the beginning of the iteration, after the body we only know that  $m$  is not in  $a[1..i-1]$  and that  $a[i+1] \neq m$ , but nothing is known about  $a[i]$ . With the new value  $i + 1$ , the invariant would require  $m$  not in  $a[1..i]$ , which is not guaranteed: in the example, before the second evaluation of the condition  $i = 2$  and  $a[1] = 6 = m$ .

### Exercise 13.5 — Correctness (E5)

Consider the following two algorithms, which should compute  $x^n$  for two natural numbers  $x \geq 1$  and  $n \geq 0$  (recall that  $x^0 = 1$ ).

**Algorithm: Power****Input** : Two natural numbers  $x \geq 1$  and  $n \geq 0$ **Output**:  $x^n$ 

```

1 function Power( $x, n$ )
2    $p := 1$ 
3    $i := 1$ 
4   while  $i \leq n$  do
5      $p \leftarrow p \cdot x$ 
6      $i \leftarrow i + 1$ 
7   end
8   return  $p$ 
9 end

```

**Algorithm: WrongPower****Input** : Two natural numbers  $x \geq 1$  and  $n \geq 0$ **Output**:  $x^n$  (?)

```

1 function WrongPower( $x, n$ )
2    $p := x$ 
3    $i := 1$ 
4   while  $i \leq n$  do
5      $p \leftarrow p \cdot x$ 
6      $i \leftarrow i + 1$ 
7   end
8   return  $p$ 
9 end

```

**Questions**

1. State a loop invariant for the while loop of **Power** and verify it for  $x = 3, n = 4$ .
2. Prove the correctness of **Power**, including the case  $n = 0$ .
3. Show that the same invariant is **not** a loop invariant of **WrongPower**: which of the three properties fails? Give a counterexample showing that **WrongPower** is wrong, with a trace for  $x = 2, n = 3$ .
4. Find the correct loop invariant of **WrongPower** and use it to determine exactly what **WrongPower** computes. For which inputs is its output correct?

**Solution**

1. Loop invariant of **Power**:

Every time the condition at line 4 is evaluated,  $1 \leq i \leq n + 1$  and

$$p = x^{i-1}.$$

For  $x = 3$ ,  $n = 4$ :

| $i$ | $p$ | $x^{i-1}$              |
|-----|-----|------------------------|
| 1   | 1   | $3^0 = 1$              |
| 2   | 3   | $3^1 = 3$              |
| 3   | 9   | $3^2 = 9$              |
| 4   | 27  | $3^3 = 27$             |
| 5   | 81  | $3^4 = 81$ (loop ends) |

The algorithm returns  $81 = 3^4$ .

2. Proof of correctness of **Power**.

- **Initialization.** Before the first iteration  $p = 1$  and  $i = 1$ , so  $p = x^0 = x^{i-1}$ ; moreover  $1 \leq i \leq n + 1$  since  $n \geq 0$ .
- **Maintenance.** Assume  $p = x^{i-1}$  and  $i \leq n$ . Line 5 gives  $p \cdot x = x^{i-1} \cdot x = x^i$ , and line 6 sets  $i \leftarrow i + 1 \leq n + 1$ . With the new value of  $i$ ,  $p = x^{i-1}$ .
- **Termination.**  $i$  increases by 1 from  $1 \leq n + 1$ , so the loop terminates after exactly  $n$  iterations, when  $i = n + 1$ . Substituting  $i = n + 1$ :  $p = x^n$ , which is returned at line 8.

If  $n = 0$ , the condition  $1 \leq 0$  is false at the first evaluation, the loop is never executed and the algorithm returns  $p = 1 = x^0$ , in agreement with the proof (initialization and termination coincide, with  $i = 1 = n + 1$ ).

3. For **WrongPower**, the **initialization** fails: before the first iteration  $i = 1$  and  $p = x$ , while  $x^{i-1} = x^0 = 1$ . Hence  $p \neq x^{i-1}$  whenever  $x \geq 2$ . (Maintenance is still true: if  $p = x^{i-1}$  held, the body would preserve it; but the “induction” has no valid base case.)

Counterexample:  $x = 2$ ,  $n = 3$ , for which  $x^n = 8$ .

| $i$ | $p$ | $x^{i-1}$ | $x^i$          |
|-----|-----|-----------|----------------|
| 1   | 2   | 1         | 2              |
| 2   | 4   | 2         | 4              |
| 3   | 8   | 4         | 8              |
| 4   | 16  | 8         | 16 (loop ends) |

**WrongPower** returns  $16 \neq 8 = 2^3$ , so it is wrong.

4. The table suggests the invariant  $p = x^i$ , which is a correct loop invariant of **WrongPower**:

- **Initialization:**  $p = x = x^1 = x^i$  with  $i = 1$ ;
- **Maintenance:** if  $p = x^i$  and  $i \leq n$ , after the body  $p = x^{i+1}$  and  $i$  is increased by 1;
- **Termination:** the loop ends with  $i = n + 1$ , so  $p = x^{n+1}$ .

Therefore **WrongPower** computes  $x^{n+1}$ . Its output is correct, i.e.,  $x^{n+1} = x^n$ , if and only if  $x^n(x - 1) = 0$ , that is (since  $x \geq 1$ ) if and only if  $x = 1$ . For every  $x \geq 2$  and every  $n \geq 0$  it is wrong: one counterexample is enough to show that an algorithm is not correct.