

**Aufgabe 1.1**

Ein Algorithmus ist eine Problemlösungsbeschreibung mit diesen Eigenschaften:

- endlich
- deterministisch (eindeutig bestimmt)
- effektiv (ausführbar)

**Aufgabe 1.2**

- (1) Input: Zahl  $n \in \mathbb{N}$
- (2) Gilt  $n < 100$ ?  
Falls ja: gehe zu (3)  
Sonst: gehe zu (5)
- (3) Setze  $(n + 1) \rightarrow n$
- (4) Gehe zu (2)
- (5) Ende

Die Beschreibung ist endlich, deterministisch und effektiv. Daher ist es ein Algorithmus.

Falls die Zahl  $n$  kleiner als 100 ist, wird sie so lange um 1 vergrößert, bis  $n = 100$  gilt und dann ist man fertig. Ist die Zahl  $n$  nicht kleiner als 100, ist man sofort fertig.

**Aufgabe 1.3**

- (1) Input: Liste L mit 9 natürlichen Zahlen
- (2) Wähle ein Zahl  $a$  aus der Liste L
- (3) Output( $a$ )
- (4) Ende

Die Beschreibung ist endlich und effektiv aber nicht deterministisch, da unklar ist, *wie* die Zahl  $a$  gewählt wird. Daher ist es kein Algorithmus

**Aufgabe 1.4**

- (1) Input: Zahl  $n \in \mathbb{N}$
- (2)  $0 \rightarrow m$
- (3) Ist  $m < n$ ?  
Falls ja: gehe zu (4)  
Sonst: gehe zu (7)
- (4) Output( $n/m$ )
- (5)  $(m + 1) \rightarrow m$
- (6) Gehe zu (2)
- (7) Ende

Die Beschreibung ist endlich und eindeutig aber nicht effektiv, da zu Beginn  $n$  durch  $m = 0$  dividiert wird. Daher ist es kein Algorithmus.

## Aufgabe 1.5

- (1) Input: Zahl  $n \in \mathbb{N}$
- (2) Output( $n$ ) (d. h. schreibe  $n$  auf)
- (3) Gilt  $n = 1$ ?  
Falls ja: gehe zu (6)  
Sonst: gehe zu (4)
- (4) Ist die Zahl  $n$  gerade?  
(4.1) Falls ja, setze  $n/2 \rightarrow n$   
(4.2) Sonst setze  $(3n + 1) \rightarrow n$
- (5) Gehe zu (3)
- (6) Ende

(a) Input:  $n = 10$     Output: 10, 5, 16, 8, 4, 2, 1

(b) Input:  $n = 7$     Output: 7, 22, 11, 32, 16, 8, 4, 1

(c) Hier handelt sich um das COLLATZ- oder  $(3n+1)$ -Problem: Bisher hat noch niemand eine Zahl  $n$  gefunden, für welche die obige Anweisungsfolge nicht terminiert aber es konnte auch noch niemand einen Beweis dafür angeben, dass dies *immer* so ist. Somit wissen wir nicht, ob das obige Verfahren ein Algorithmus gemäss unserer Definition ist.

## Aufgabe 2.1

grösster gemeinsamer Teiler (ggT)     $\Leftrightarrow$     greatest common divisor (gcd)

## Aufgabe 2.2

klassischer euklidischer Algorithmus:

$$\begin{aligned} \text{ggT}(28, 16) &= (12, 16) \stackrel{S}{=} (16, 12) = (4, 12) \stackrel{S}{=} (12, 4) = (8, 4) \\ &= (4, 4) = (0, 4) \stackrel{S}{=} (4, 0) = (4, 0) = 4 \end{aligned}$$

## Aufgabe 2.3

klassischer euklidischer Algorithmus:  $\text{ggT}(6, -4) = (10, -4) = (14, -4) = \dots$

Da der erste Operand bei jedem Schritt grösser wird, terminiert der Algorithmus nicht.

## Aufgabe 2.4

Der zentrale Nachteil besteht darin, dass bei (positiven) Operanden  $a$  und  $b$  mit grossem Unterschied das klassische Verfahren viel Zeit mit Subtraktionen verbringt, während die moderne Berechnungsmethode dies durch die Berechnung des Rests mit dem Modulo-Operator abkürzt. *Beispiel:*

- *klassisch:*  $\text{ggT}(99, 1) = (98, 1) = \dots = (0, 1) = (1, 0) = 1$
- *modern:*  $\text{ggT}(99, 1) = (1, 0) = 1$

## Aufgabe 2.5

Python-Funktion, die  $\text{ggt}(a,b)$  mit dem klassischen Algorithmus von Euklid berechnet und zurückgibt.

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| d | e | f |   | g | g | t | _ | c | l | a | s | s | i | c | ( | a | , |   | b | ) | : |   |
|   |   | a | , |   | b |   | = |   | a | b | s | ( | a | ) | , |   | a | b | s | ( | b | ) |
|   |   | w | h | i | l | e |   | b |   | ! | = |   | 0 | : |   |   |   |   |   |   |   |   |
|   |   |   |   | i | f |   | a |   | < |   | b | : |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   | a | , |   | b |   | = |   | b | , |   | a |   |   |   |   |   |   |   |
|   |   |   |   |   | a | , |   | b |   | = |   | a | - | b | , |   | b |   |   |   |   |   |
|   |   | r | e | t | u | r | n |   | a |   |   |   |   |   |   |   |   |   |   |   |   |   |

## Aufgabe 2.6

moderner euklidischer Algorithmus:

$$\text{ggT}(33, 12) = (12, 9) = (9, 3) = (3, 0) = 3$$

## Aufgabe 2.7

Python-Funktion, die  $\text{ggt}(a,b)$  mit dem modernen Algorithmus von Euklid berechnet und zurückgibt.

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|--|
| d | e | f |   | g | g | t | _ | m | o | d | e | r | n | ( | a | , |   | b | ) | : |  |
|   |   | w | h | i | l | e |   | b |   | ! | = |   | 0 | : |   |   |   |   |   |   |  |
|   |   |   |   | a | , |   | b |   | = |   | b | , |   | a | % |   | b |   |   |   |  |
|   |   | r | e | t | u | r | n |   | a |   |   |   |   |   |   |   |   |   |   |   |  |

## Aufgabe 2.8

Beim beim (modernen) Algorithmus von Euklid tritt der Worst Case auf, wenn  $a$  und  $b$  zwei aufeinanderfolgende Fibonacci-Zahlen sind.

## Aufgabe 2.9

Die ersten 10 Werte der Fibonacci-Folge:

ältere Schreibweise: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

moderne Schreibweise: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

### Aufgabe 3.1

$$B < E < C < A < D$$

Da es sich um *asymptotische Laufzeiten*, d. h. Laufzeiten für *grosse*  $n$  handelt, kann es sein, dass die Dominanz einer der Funktionen erst ab einem ausreichend grossen  $n$  erkennbar wird.

### Aufgabe 3.2

(a)  $O(2^{n+1}) = O(2 \cdot 2^n) = O(2^n)$

(b)  $O(\sqrt{4n}) = 2\sqrt{n} = O(\sqrt{n})$

(c)  $O((n^2 + n)(n + 2)(n + 3)) = O(n^4 + \dots) = O(n^4)$

(d)  $O(3) = O(1)$

### Aufgabe 3.3

```
1 def funktion(n):  
2     s = 0  
3     for i in range(0, n):  
4         s = s + i  
5     return s
```

| Zeile | Kosten | Anzahl |
|-------|--------|--------|
| 2     | $c$    | 1      |
| 3     | $c$    | $n$    |
| 4     | $c$    | $n$    |
| 5     | $c$    | 1      |

$$T(n) = c + c \cdot n + c \cdot n + c = 2c + 2cn \in O(n)$$

### Aufgabe 3.4

```
1 def funktion(n):  
2     s = 0  
3     for i in range(0, n):  
4         for j in range(0, i):  
5             s = s + j  
6     return s
```

| Zeile | Kosten | Anzahl |
|-------|--------|--------|
| 2     | $c$    | 1      |
| 3     | $c$    | $n$    |
| 4     | $c$    | $n^2$  |
| 5     | $c$    | $n^2$  |
| 6     | $c$    | 1      |

$$T(n) = 2c + cn + 2cn^2 \in O(n^2)$$

### Aufgabe 3.5

```
1 def funktion(n):  
2     s = 0  
3     i = 1  
4     while i < n:  
5         s = s + i;  
6         i = 3*i  
7     return n
```

| Zeile | Kosten | Anzahl     |
|-------|--------|------------|
| 2     | $c$    | 1          |
| 3     | $c$    | 1          |
| 4     | $c$    | $\log_3 n$ |
| 5     | $c$    | $\log_3 n$ |
| 6     | $c$    | $\log_3 n$ |
| 6     | $c$    | 1          |

$$T(n) = 3c + 3c \log_3(n) \in O(\log_3 n)$$

### Aufgabe 3.6

```
1 def funktion(n):  
2     s = 0  
3     for i in range(0, n):  
4         s = s + i  
5     for j in range(0, n):  
6         s = s * j  
7     return s
```

| Zeile | Kosten | Anzahl |
|-------|--------|--------|
| 2     | $c$    | 1      |
| 3     | $c$    | $n$    |
| 4     | $c$    | $n$    |
| 5     | $c$    | $n$    |
| 6     | $c$    | $n$    |
| 7     | $c$    | 1      |

$$T(n) = 2c + 4cn \in O(n)$$

### Aufgabe 3.7

$$T(n) \in O(n^2)$$

$$T(10\,000) = c \cdot 10\,000^2 \stackrel{*}{=} 40\text{ s}$$

$$T(30\,000) = c \cdot 30\,000^2 = c \cdot (3 \cdot 10\,000)^2 = 9 \cdot c \cdot 10\,000^2 \stackrel{*}{=} 9 \cdot 40\text{ s} = 360\text{ s} = 6\text{ Min}$$

### Aufgabe 3.8

$$T(n) \in O(\log n)$$

$$T(10^4) = c \cdot \log 10^4 \stackrel{*}{=} 8 \text{ Min.}$$

$$\begin{aligned} T(10^{12}) &= c \cdot \log(10^{12}) = c \cdot \log(10^{4 \cdot 3}) = c \cdot \log((10^4)^3) \\ &= 3 \cdot c \cdot \log(10^4) \stackrel{*}{=} 3 \cdot 8 \text{ Min.} = 32 \text{ Min} \end{aligned}$$

### Aufgabe 3.9

$$T(n) \in O(n)$$

$$T(6 \cdot 10^7) = c \cdot 6 \cdot 10^7 \stackrel{*}{=} 4 \text{ s}$$

$$T(9 \cdot 10^7) = c \cdot 9 \cdot 10^7 = 1.5 \cdot c \cdot 6 \cdot 10^7 \stackrel{*}{=} 1.5 \cdot 4 \text{ s} = 6 \text{ s}$$

### Aufgabe 3.10

$$T(n) \in O(\textcolor{red}{n} \log(n))$$

$$T(200) = c \cdot 200 \log_2(200) \stackrel{*}{=} 3 \text{ ms}$$

$$\begin{aligned} T(40\,000) &= c \cdot 40\,000 \cdot \log_2(40\,000) = c \cdot 200^2 \cdot \log_2(200^2) \\ &= c \cdot 200 \cdot 200 \cdot \textcolor{red}{2} \cdot \log_2(200) = 400 \cdot c \cdot 200 \log_2(200) \\ &\stackrel{*}{=} 400 \cdot 3 \text{ ms} = 1200 \text{ ms} = 1.2 \text{ s} \end{aligned}$$

### Aufgabe 3.11

$$T(n) \in O(2^n)$$

$$T(40) = c \cdot 2^{\textcolor{red}{40}} \stackrel{*}{=} 6 \text{ h}$$

$$\begin{aligned} T(45) &= c \cdot 2^{45} = c \cdot 2^{40} \cdot 2^5 = 32 \cdot c \cdot 2^{40} \\ &\stackrel{*}{=} 32 \cdot 6 \text{ h} = 8 \cdot 4 \cdot 6 \text{ h} = 8 \cdot 24 \text{ h} = 8 \text{ d} \end{aligned}$$

### Aufgabe 3.12

$$T(n) \in O(n!)$$

$$T(7) = c \cdot 7! \stackrel{*}{=} 10 \text{ s}$$

$$\begin{aligned} T(9) &= c \cdot 9! = c \cdot 9 \cdot 8 \cdot 7! = 72 \cdot c \cdot 7! \\ &\stackrel{*}{=} 72 \cdot 10 \text{ s} = 12 \cdot 6 \cdot 10 \text{ s} = 12 \cdot 60 \text{ s} = 12 \text{ Min.} \end{aligned}$$

### Aufgabe 3.13

- (a) Gnomesort:  $O(n^2)$
- (b) Suche in unsortierter Liste:  $O(n)$
- (c) Mergesort:  $O(n \log_2 n)$
- (d) Lesen eines Elements aus Python-Liste:  $O(1)$

### Aufgabe 3.14

Nicht handhabbare Probleme sind Probleme, für die es keine effizienten Algorithmen gibt, mit denen sie gelöst werden können. Diese liegen, im Allgemeinen in den Komplexitätsklassen  $O(k^n)$ ,  $O(n!)$  oder  $O(n^n)$ .