

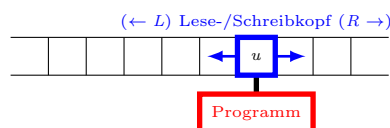
## Aufgabe 1

Ein Algorithmus ist eine Vorschrift zur *Lösung eines Problems*, die *endlich*, *deterministisch* (eindeutig) und *effektiv* (ausführbar) ist.

*Wer es genau wissen will:* Ein *Algorithmus* ist eine Berechnungsvorschrift zur Lösung eines Problems, wenn eine zu dieser Berechnungsvorschrift äquivalente Turingmaschine existiert, die für jede Eingabe, die eine Lösung hat, stoppt

Eine Turingmaschine (TM) ist ein 7-Tupel  $T = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$  mit:

- $Q$ : endliche Zustandsmenge
- $\Sigma$ : endliches Eingabealphabet
- $\Gamma$ : endliches Bandalphabet
- $\delta: (Q \setminus F) \times \Gamma \rightarrow Q \times \Gamma \times \{L, N, R\}$  die Übergangsfunktion
- $q_0 \in Q$ : Anfangszustand
- $\square \in \Gamma \setminus \Sigma$ : das leere Feld
- $F \subset Q$ : Menge der akzeptierenden Zustände



Die TM liest das Zeichen über dem Lesekopf, geht abhängig vom aktuellen Zustand  $q$  und dem gelesenen Zeichen  $u \in \Gamma$  in den Zustand  $q'$ , schreibt je nach Programm ein Zeichen aufs Band und verschiebt den Lese-/Schreibkopf um eine Position nach links ( $L$ ), rechts ( $R$ ) oder bleibt stehen ( $N$ ). Die TM stoppt, wenn ein akzeptierender Zustand erreicht wird.

## Aufgabe 2

Die Laufzeitkomplexität beschreibt, wie sich die Laufzeit eines Algorithmus mit zunehmender Größe der Eingabedaten verändert. Die „Big-Oh“-Notation gibt eine obere Schranke für das Wachstum der Laufzeit an.

- $O(1)$  (konstante Laufzeit): Lese- oder Schreibzugriff auf ein Array, Wertzuweisung einer Variablen, Addition von ganzen Zahlen
- $O(\log n)$  (logarithmische Laufzeit): Binäre Suche in einem sortierten Array
- $O(n)$  (lineare Laufzeit): Sequentielle Suche in einem unsortierten Array
- $O(n \log n)$  (linear-logarithmische Laufzeit): Effizienten Sortieralgorithmen wie Quick- oder Mergesort
- $O(n^2)$  (quadratische Laufzeit): „Naive“ Sortieralgorithmen wie Insertionsort, Selectionsort oder Gnomesort
- $O(n^3)$  (kubische Laufzeit): Die Multiplikation von  $(n \times n)$ -Matrizen.
- $O(2^n)$  (exponentielle Laufzeit): Erzeugen aller Binärzahlen mit  $n$  Stellen.

## Aufgabe 3

Algorithmen mit exponentiellen (und höheren) Laufzeiten.

## Aufgabe 4

- (a)  $T(n) = 3n + 4n^2 + 2 \in O(n^2)$
- (b)  $T(n) = (3n + 2)(5n^2 + 1) \in O(n^3)$
- (c)  $T(n) = 2^{n+3} = 2^n \cdot 2^3 \in O(2^n)$
- (d)  $T(n) = \log_2(4n^3) = \log_2(4) + 3 \log_2(n) \in \log_2(n)$

## Aufgabe 5

$$T(n) = C \cdot n^2$$

$$T(100) = C \cdot 100^2 = 20\text{s} (*)$$

$$T(200) = C \cdot 200^2 = C \cdot (2 \cdot 100)^2 = 2^2 \cdot C \cdot 100^2 = 4 \cdot 20\text{s} = 80\text{s}$$

Allgemein: In  $O(n^2)$  bewirkt das Verdoppeln der Problemgrösse eine Vervierfachung der Laufzeit.

Man hätte auch die erste Gleichung nach  $C$  auflösen und diesen Wert in die zweite Gleichung einsetzen können. Meist lässt sich die Rechnung jedoch in der oben beschriebenen Weise „kurzschliessen“.

## Aufgabe 6

$$T(n) = C \cdot 2^n$$

$$T(10) = C \cdot 2^{10} = 3\text{s}$$

$$T(12) = C \cdot 2^{12} = C \cdot 2^{10} \cdot 2^2 = 3\text{s} \cdot 4 = 12\text{s}$$

## Aufgabe 7

$$T(n) = C \cdot \log_2(n)$$

$$T(100) = C \cdot \log_2(100) = 3\text{ms}$$

$$T(10\,000) = C \cdot \log_2(10\,000) = C \cdot \log_2(100^2) = 2 \cdot C \cdot \log_2(100) = 2 \cdot 3\text{ms} = 6\text{ms}$$

## Aufgabe 8

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

Der Aufruf der Funktion, die Zuweisung  $s=0$  und die Rückgabe des Wertes benötigen jeweils konstante Zeit. Die Ausführungsdauer der Schleife ist proportional zu  $n$ .

Somit  $T(n) = c_1 + c_2 \cdot n \in O(n)$

## Aufgabe 9

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

Der Aufruf der Funktion, die Zuweisung  $s=0$  und die Rückgabe des Wertes benötigen jeweils konstante Zeit. Da für jede der  $n$  Iterationen der äusseren Schleife  $n$  Iterationen der inneren Schleife ausgeführt werden, hat der Schleifenkomplex eine Ausführungsdauer, die proportional zu  $n^2$  ist.

Somit:  $T(n) = c_1 + c_2 \cdot n^2 \in O(n^2)$

## Aufgabe 10

```
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
```

Der Aufruf der Funktion, die Zuweisung  $s=0$  und die Rückgabe des Wertes benötigen jeweils konstante Zeit. Da für jede der  $n$  Iterationen der ersten Schleife  $n$  Iterationen ausgeführt werden und danach nochmals  $n$  Iterationen der zweiten ausgeführt werden, hat der Schleifenblock eine Ausführungsdauer, die proportional zu  $n$  ist.

Somit:  $T(n) = c_1 + c_2 \cdot n \in O(n)$

## Aufgabe 11

```
1 def funktion(n):
2     s = 0
3     while n > 0:
4         s = s + n
5         n = n // 2
6     return s
```

Der Aufruf der Funktion, die Zuweisung  $s=0$  und die Rückgabe des Wertes benötigen jeweils konstante Zeit. Da sich der Wert von  $n$  bei jedem Durchlauf der while-Schleife halbiert, benötigt die Schleife  $\log_2(n)$  Iterationen.

Somit:  $T(n) = c_1 + c_2 \cdot \log_2(n) \in O(\log_2(n))$

## Aufgabe 12

- *natürliche Sprache*

Vorteil: Kann auch von Menschen ohne Programmierkenntnissen verstanden werden

Nachteil: Natürliche Sprache ist nicht immer präzise

- *Pseudocode*

Vorteil: Klare und strukturierte Darstellung von Algorithmen

Nachteil: Keine standardisierte Syntax

- *Struktogramm*

Vorteil: Die Visuelle Darstellung kann das Verständnis erleichtern.

Nachteil: Kann bei komplexen Algorithmen unübersichtlich werden

- *Quellcode einer Programmiersprache*

Vorteil: Exakte und ausführbare Darstellung des Algorithmus

Nachteil: Erfordert Kenntnisse der spezifischen Programmiersprache

### Aufgabe 13

Eingabe: $x$
$y \leftarrow 3$
$x \leftarrow x + y$
$y \leftarrow x \bmod 3$
Ausgabe: $y$

Ausgabe für  $x = 10$ : 1

### Aufgabe 14

Eingabe: $x$	
$x > 0$	
ja	nein
$y \leftarrow x + 1$	$y \leftarrow x - 1$
Ausgabe: $y$	

Ausgabe für  $x = 17$ : 18

### Aufgabe 15

Eingabe: $n$
$n < 10$
$n \leftarrow n + 3$
Ausgabe: $n$

Ausgabe für  $n = 5$ : 11

### Aufgabe 16

Hinweis: Annehmende Schleifen testen die Bedingung am Ende des Schleifenkörpers (Schleifenblocks) und verlassen die Schleife in der Regel dann, wenn die Bedingung *wahr* ist.

Eingabe: $n$
$s \leftarrow 0$
$i \leftarrow 1$
$i \leftarrow i + 1$
$s \leftarrow s + i$
$i > n$
Ausgabe: $s$

Ausgabe für  $n = 3$ : 9

### Aufgabe 17

```
def gcd(a, b):
    b = abs(b)
    while b > 0:
        a, b = b, a % b
    return a
```

### Aufgabe 18

- (a) Erkläre, was eine *rekursive Funktion* ist.

Eine rekursive Funktion ist eine Funktion, die sich in ihrer Definition selber aufruft. Da dies zu einem nicht endenden Prozess führt, benötigt es eine Bedingung, welche diesen Prozess stoppt (Base Case).

- (b) Nenne zwei Anwendungsbeispiele der Rekursion.

Binäre Suche, Quicksort

- (c) Nenne einen Vor- und einen Nachteil der Rekursion.

Vorteil: Rekursive Vorgänge haben oft eine kurze Beschreibung

Nachteil: Die wiederholten Aufrufe der Funktion legen viele Werte auf einem Stack ab, was zu einem Stack Overflow führen kann.

### Aufgabe 19

```
def f(n):
    if n == 0:
        return 3
    else:
        return 2 + f(n-1)

print(f(4))
```

$$\begin{aligned}
f(4) &= 2 + f(3) \\
&= 2 + (2 + f(2)) \\
&= 2 + (2 + (2 + f(1))) \\
&= 2 + (2 + (2 + (2 + f(0)))) \\
&\stackrel{*}{=} 2 + (2 + (2 + (2 + 3))) \\
&= 2 + (2 + (2 + 5)) \\
&= 2 + (2 + 7) \\
&= 2 + 9 \\
&= 11
\end{aligned}$$

### Aufgabe 20

```

def f(n):
    if n > 0:
        return n + f(n-2)
    else:
        return 7

print(f(5))

```

$$\begin{aligned}
f(5) &= 5 + f(3) \\
&= 5 + (3 + f(1)) \\
&= 5 + (3 + (1 + f(-1))) \\
&\stackrel{*}{=} 5 + (3 + (1 + 7)) \\
&= 5 + (3 + 8) \\
&= 5 + 11 \\
&= 16
\end{aligned}$$

### Aufgabe 21

Ein Einkauf mit Liste ist ein *Lösungsverfahren*.

- Es ist *endlich*, da Einkaufslisten nicht unendlich lange sind.
- Es ist nicht *deterministisch*, da nicht gesagt wird, wie bei der Suche vorgegangen werden soll.
- Es ist möglicherweise nicht effektiv, wenn der Artikel (derzeit) im Sortiment fehlt.

### Aufgabe 22

$$\text{ggT}(32, 18) = (18, 14) = (14, 4) = (4, 2) = (2, 0) = 2$$