

2.10.24

Agenda

- Algorithmus
- Arbeitsauftrag erledigen → Rechnen
- Laufzeitkomplexität (Wiederholen)
- 2 Arbeitsaufträge
 - Wertetabelle: theoretische Laufzeit
Datenmenge
 - Pseudoprogramme → Laufzeitkomplexität
bestimmen (Qualitativ)

20 min → 17:50 Uhr

Algorithmus

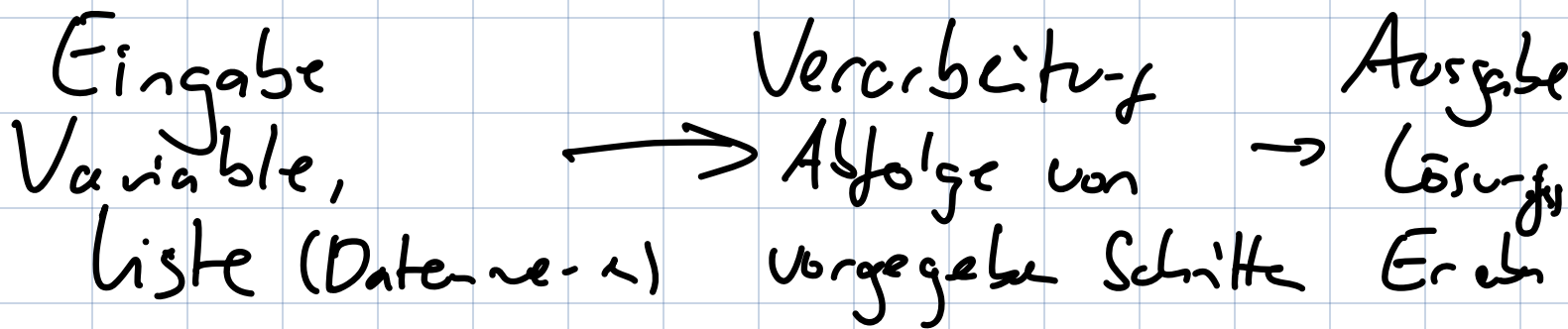
Was ist ein Algorithmus?

- Eindeutig & Schrittweise von Schema
- Abfolge von Anweisungen (Kochrezept oder Programmiersprache (Navigation),

Bauanleitungen

- Bestimmte Aufgabe (Eindeutig)
- Eindeutige Handlungsanweisung zur Lösung eines Problems
- Klare Eingabe meist klare Ausgabe
- Feste Reihenfolge von Schritten

Schematische Darstellung mittels EVA-Prin^p



Arten von Algorithmen

- **Deterministisch**: Gleiche Eingabe \rightarrow
Gleiche Ausgabe/
Ergebnis
- **Nicht-Deterministisch**: Gleiche Eingabe \rightarrow
Unterschiedliche
Ergebnisse
- Neuronale Netzwerk
- Gewichtungen

- Stochastische & Optimierungsparameter

Beispiel: Würfel

Beispiel: Bildererkennung



95%



96%



93%

→ Gleiche Eingabe

→ Unterschiedliche Ergebnisse

→ WAHRSCHEINLICHKEITEN

Eigenschaften von Algorithmen

- Eindeutig (Determiniertheit)
 - Endlichkeit (Finitheit)
 - Terminiertheit
 - **Effizienz** → Laufzeitkomplexität
 - Eingaben & Ausgaben
- Einzelne Schritte sind fest definiert

Unterschied zwischen Algorithmus & Programm

Algorithmus: Reine Anleitung

Von bestimmten Schritten nach eine
vorgegebenen Struktur
• Programmiersprache unabhängig

O -Notation Laufzeitkomplexität

Ziel: Das grobe Abschätzen der Laufzeit
sein über die von einem Code \rightarrow
Wirtschaftlich & Effizient
• Erste Beurteilung der Effizienz zweier
Algorithmen mit der gleichen Aufgabe

Wiederholung Laufzeit Treffen 18:50 Uhr

$O(1)$	n	4	16	64	256	1024
	+	1	1	1	1	1

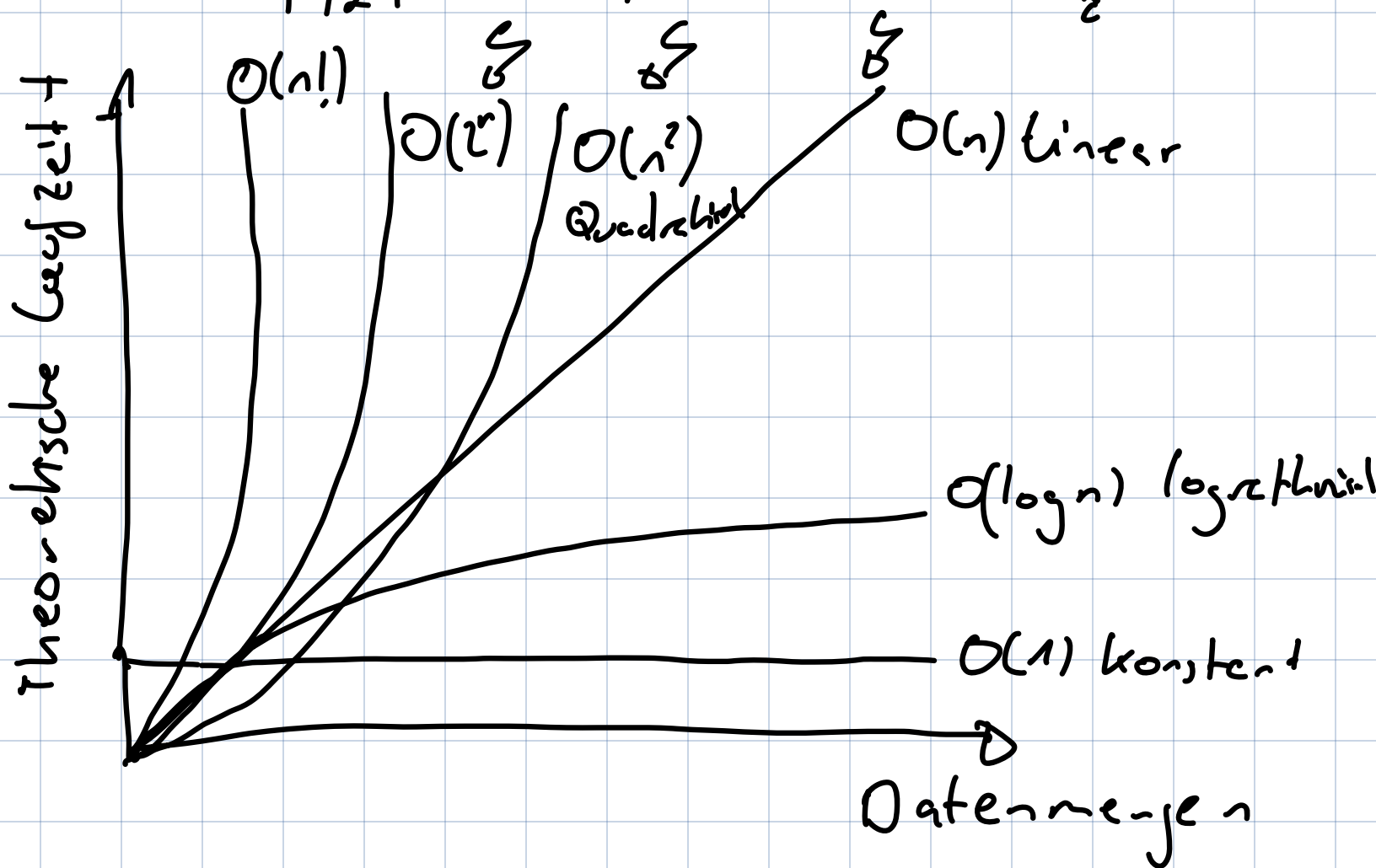
$O(\log_2 n)$	n	4	16	64	256	1024
	+	2	4	6	8	10

$O(n)$	n	4	16	64	256	1024
	+	4	16	64	256	1024

$O(n^2)$	n	4	16	64	256	1024
+		16	256	4096	65.536	1.048.576

$O(2^n)$	n	4	16	64	256	1024
+		16	65.536	$1,8 \cdot 10^{13}$	$1,5 \cdot 10^{77}$	∞

$O(n!)$	n	4	16	64	256	1024
+		24	$2,1 \cdot 10^3$	$1,3 \cdot 10^{25}$	$9,5 \cdot 10^{506}$	∞





Erkennen von Laufzeitkomplexität

in Code

- If-Anweisung $O(1)$
 - keine Schleifen $O(1)$
 - Eine Schleife $O(n)$
 - Verkettete Schleifen
 $O(n^2) \rightarrow 2$ verkettete Schleifen
 $O(n^3) \rightarrow 3$ verkettete Schleifen
- ```
For....
 For....

For....
 For....
 For....
```
- Liste werden um ein vielfaches  
Reduziert  $O(\log(n))$   
Liste halbieren  $\log_2(n)$
  - Permutation (Mögliche Anzahl  
von Anordnungen)  $O(n!)$

Bsp: 10 Personen  ...   
 $10! = 3.628.000$

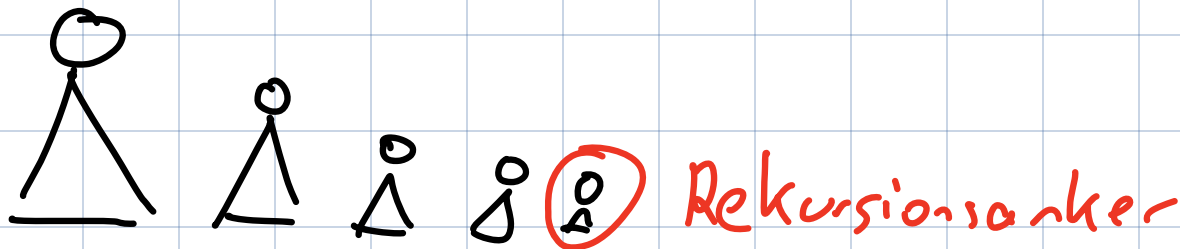
Bsp: Liste mit 3 Elementen  $[3, 4, 6]$  1  
 $3! = 1 \cdot 2 \cdot 3 = 6$   $[3, 6, 4]$  2

$[4, 3, 6] \rightarrow 3$   
 $[4, 6, 3] \rightarrow 4$   
 $[6, 3, 4] \rightarrow 5$   
 $[6, 4, 3] \rightarrow 6$

- Rekursion: Funktion / Algorithmus der sich selber bis Rekursionsanker wiederholt

$O(n) \rightarrow$  Anzahl der Durchläufe oder  $O(n!)$

Bsp. Matroschka Puppe  $O(n)$



Funktion

Fakultät  $4! = 4 \cdot 3! \quad O(n!)$

$$= 4 \cdot 3 \cdot 2!$$

$$= 4 \cdot 3 \cdot 2 \cdot 1!$$

$$= 4 \cdot 3 \cdot 2 \cdot 1 \cdot 1 \quad \text{Rekursionsanker}$$

Rekursionsanker ist nicht 0 da

$$4! = 4 \cdot 3 \cdot 2 \cdot 1 \cdot 0 = 0 \quad 4! = 0 \quad \text{falsch}$$

# Schleifenarten

|                     |                         |        |
|---------------------|-------------------------|--------|
| Zählerschleife      | FOR ... DO              | $O(n)$ |
| Kopfschleife        | WHILE .... Befehl       | $O(n)$ |
| Fußgesteuert        | DO..., WHILE            | $O(n)$ |
| Per Element / Liste | FOREACH Element OF list | $O(n)$ |

1)  
For Anweisung 1 DO  $n$   
    For Anweisung 2 DO  $n$   
        Befehl  $1$

$$n \cdot n \cdot 1 = n^2 \rightarrow O(n^2)$$

2) FOREACH Element OF list DO  $n$   
    FOR Anweisung DO  $n$   
        Befehl  $1$

$$n \cdot n \cdot 1 = n^2 \rightarrow O(n^2)$$



3) FOR Bedingung DO  $n$   
     IF Anweisung DO  $1$   
         Befehl  $1$

$$n \cdot 1 \cdot 1 = n \rightarrow O(n)$$

4) FOR Bedingung 1 DO  $n$   
     Befehl 1  $1$   
     FOR Bedingung 2 DO  $n$   
         Befehl 2  $1$

$$n \cdot 1 + n \cdot 1 = 2n \rightarrow O(n)$$

6) WHILE Bedingung 1  $n$   
     FOREACH Element OF List DO  $n$   
         FOR Anweisung DO  $n$   
             IF Bedingung DO  $1$   
                 Befehl 1  $1$   
             ELSE  $1$   
                 Befehl 2  $1$

FOR Bedingung 3 DO  $n$   
Bedingung 3 1

$$n \cdot n^3 + n \cdot 1 + n \cdot 1 + n \cdot 1 + n \cdot 1 + n \cdot 1 \rightarrow O(n^3)$$

7) FOR Anweisung DO  $n$   
Binäre Suche  $\log(n)$

$$n \cdot \log(n) \rightarrow O(n \cdot \log(n))$$

8) WHILE Bedingung 1  $n$   
Heapsort  $n \cdot \log(n)$

$$n \cdot n \cdot \log(n) = n^2 \cdot \log(n) = O(n^2 \cdot \log(n))$$

Regel / Hinweise

1. Verschachtelung  $\rightarrow$  Multiplikation

2. Abfolge  $\rightarrow$  Addition
3. Vorfaktoren werden vernachlässigt  
 $2n^3 \rightarrow O(n^3)$
4. Bei Addition von mehreren Laufzeitkomplexitäten schauen wir uns "schlimmste" Funktion an

$$\underline{3} \cdot \underline{2^n} + 4n^2 + \frac{1}{2}n \rightarrow O(2^n)$$

vernachlässigen

$$\underline{O(1)} < O(\log n) < O(n) < O(n \cdot \log n) < O(n!) < O(2^n) < \underline{O(n!)}$$