

3

Mathematische Grundlagen

Algorithmen & Datenstrukturen · Sommersemester 2026

Prof. Dr. Sebastian Wild

3 Mathematische Grundlagen

- 3.1 Beispiel: 3SUM
- 3.2 Folgen & Grenzwerte
- 3.3 Asymptotische Notation
- 3.4 Rechenregeln für Asymptotiken
- 3.5 Rekursiv definierte Folgen
- 3.6 Das Master-Theorem
- 3.7 Grundlagen Stochastik

Recap: Skalierbarkeit

Analogie: Architektur

- ▶ 1m Brücke: Holzplanke genügt!
- ▶ 10x \rightsquigarrow Konstruktion nötig
- ▶ 1000x \rightsquigarrow Meisterleistung der Bauingenieurskunst

Eine 1km Brücke ist nicht einfach ein 1000-fach größeres Brett.

Recap: Skalierbarkeit

Analogie: Architektur

- ▶ 1m Brücke: Holzplanke genügt!
- ▶ 10x \rightsquigarrow Konstruktion nötig
- ▶ 1000x \rightsquigarrow Meisterleistung der Bauingenieurskunst

Eine 1km Brücke ist nicht einfach ein 1000-fach größeres Brett.



Recap: Skalierbarkeit

Analogie: Architektur

- ▶ 1m Brücke: Holzplanke genügt!
- ▶ 10x \rightsquigarrow Konstruktion nötig
- ▶ 1000x \rightsquigarrow Meisterleistung der Bauingenieurskunst

Eine 1km Brücke ist nicht einfach ein 1000-fach größeres Brett.



2m

$\sim 10x$
 \longrightarrow



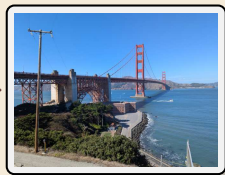
12m

$\sim 10x$
 \longrightarrow



170m

$\sim 10x$
 \longrightarrow



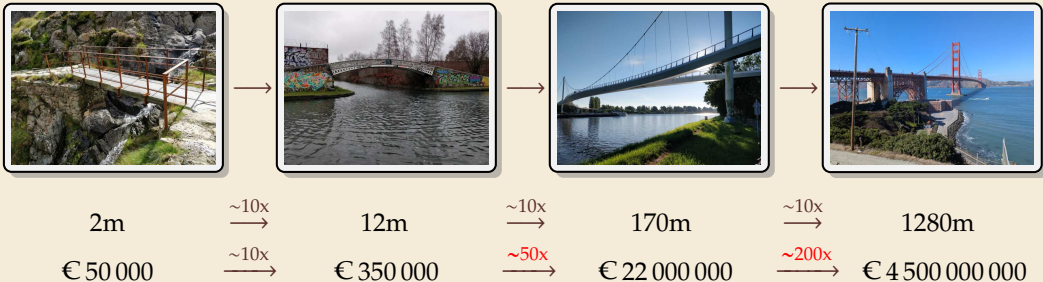
1280m

Recap: Skalierbarkeit

Analogie: Architektur

- ▶ 1m Brücke: Holzplanke genügt!
- ▶ 10x \rightsquigarrow Konstruktion nötig
- ▶ 1000x \rightsquigarrow Meisterleistung der Bauingenieurskunst

Eine 1km Brücke ist nicht einfach ein 1000-fach größeres Brett.



(Kosten sind Schätzungen der heutigen Kosten)

Recap: Skalierbarkeit

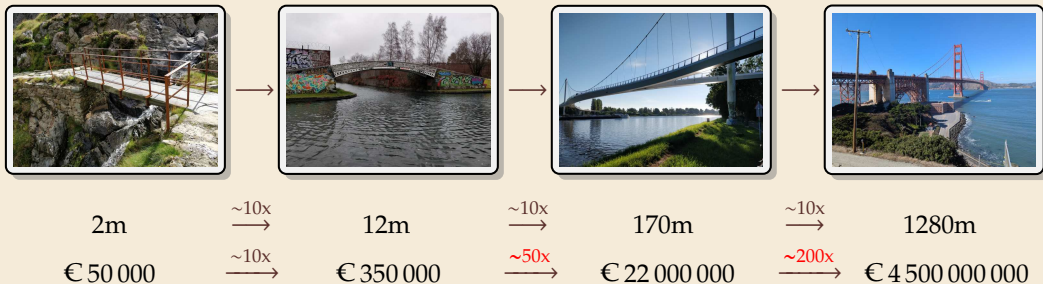
Analogie: Architektur

- ▶ 1m Brücke: Holzplanke genügt!
- ▶ 10x \rightsquigarrow Konstruktion nötig
- ▶ 1000x \rightsquigarrow Meisterleistung der Bauingenieurskunst

Informatik

- ▶ Eingabegrößen in Informatiksystemen erreichen oft Faktor 1 000 000 000!
- 1 Byte \rightarrow 1 GB
- \rightsquigarrow Können uns keine Holzbretter leisten.

Eine 1km Brücke ist nicht einfach ein 1000-fach größeres Brett.



(Kosten sind Schätzungen der heutigen Kosten)

Wie quantifizieren wir “Skalierbarkeit”?

- ↪ Für Algorithmen (in erster Näherung) **Wachstum** der Kosten in Eingabegröße entscheidend.
 - ▶ Tricks in der Implementierung und bessere Hardware retten uns nicht
 - ▶ tragen aber natürlich zu den Gesamtkosten bei
- ↪ für Performance-Bottlenecks durch Experten noch etwas rauszuholen
- ↪ den deutlich größeren Impact hat aber die Wahl der richtigen Algorithmen und Datenstrukturen

Wie quantifizieren wir “Skalierbarkeit”?

- ↪ Für Algorithmen (in erster Näherung) Wachstum der Kosten in Eingabegröße entscheidend.
 - ▶ Tricks in der Implementierung und bessere Hardware retten uns nicht
 - ▶ tragen aber natürlich zu den Gesamtkosten bei
- ↪ für Performance-Bottlenecks durch Experten noch etwas rauszuholen
- ↪ den deutlich größeren Impact hat aber die Wahl der richtigen Algorithmen und Datenstrukturen

Wie können wir also Algorithmen bezüglich des **Wachstums** der Kosten *vergleichen*?

- ▶ Sprache für Performancecharakteristika: **Asymptotische Approximationen**
O-Notation (plus Varianten und Verfeinerungen)
- ↪ Damit müssen Sie souverän umgehen lernen.

Wie quantifizieren wir “Skalierbarkeit”?

- ↪ Für Algorithmen (in erster Näherung) **Wachstum** der Kosten in Eingabegröße entscheidend.
 - ▶ Tricks in der Implementierung und bessere Hardware retten uns nicht
 - ▶ tragen aber natürlich zu den Gesamtkosten bei
- ↪ für Performance-Bottlenecks durch Experten noch etwas rauszuholen
- ↪ den deutlich größeren Impact hat aber die Wahl der richtigen Algorithmen und Datenstrukturen

Wie können wir also Algorithmen bezüglich des **Wachstums** der Kosten *vergleichen*?

- ▶ Sprache für Performancecharakteristika: **Asymptotische Approximationen**
O-Notation (plus Varianten und Verfeinerungen)
- ↪ Damit müssen Sie souverän umgehen lernen.

Dazu legen wir hier die Grundlagen und geben die formalen Definitionen.

3.1 Beispiel: 3SUM

3SUM

3SUM-Problem

► **Gegeben:** $A[0..n)$ with $A[i] \in \mathbb{Z}$ for $0 \leq i < n$.

► **Ziel:** Anzahl c von Tripeln aus A mit Summe 0

$$c = \left| \left\{ (i, j, k) : 0 \leq i < j < k < n \wedge A[i] + A[j] + A[k] = 0 \right\} \right|$$

Parameter:

► Eingabegröße: # Zahlen n

3SUM

3SUM-Problem

▶ **Gegeben:** $A[0..n)$ with $A[i] \in \mathbb{Z}$ for $0 \leq i < n$.

▶ **Ziel:** Anzahl c von Tripeln aus A mit Summe 0

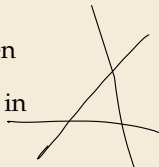
$$c = \left| \left\{ (i, j, k) : 0 \leq i < j < k < n \wedge A[i] + A[j] + A[k] = 0 \right\} \right|$$

Parameter:

▶ Eingabegröße: # Zahlen n

Anwendungen: 3 Punkte auf einer Geraden

- ▶ kollineare Tripel von Punkten finden
- ▶ Tripel von Geraden finden, die sich in einem Punkt schneiden
- ▶ Dreiecke aus Punkten formen mit kleinem Flächeninhalt
- ▶ ...



Clicker Question

$$[a, b) = \{x \in \mathbb{R} : a \leq x < b\}$$

$$\{a, a+1, \dots, b-1\}$$

"

$$[a..b) = \{n \in \mathbb{Z} : a \leq n < b\}$$

Was ist die Ausgabe für 3SUM mit folgender Eingabe?

A[0..6) = [1, 0, -1, 3, 2, -1] (0,1,2)
 0 1 2 3 4 5 (0,1,5)
 (2,4,5)

A[i]



- | | |
|---|--------------|
| A ArrayIndexOutOfBoundsException | F 4 |
| B 0 | G 4.5 |
| C 1 | H 5 |
| D 2 | I 6 |
| E 3 | |



→ sli.do/cs210

Clicker Question

Was ist die Ausgabe für 3SUM mit folgender Eingabe?

$A[0..6] = [1, 0, -1, 3, 2, -1]$



- | | | | |
|----------|---|----------|----------------|
| A | ArrayIndexOutOfBoundsException | F | 4 |
| B | 0 | G | 4.5 |
| C | 1 | H | 5 |
| D | 2 | I | 6 |
| E | 3 ✓ | | |



→ sli.do/cs210

Einschub: Template für Entwurf von Algorithmen

1. 💡 **Algorithmische Idee**

Kurze abstrakte Idee, die dem Algorithmus zu Grunde liegt (Freitext)
(ein Experte könnte den Rest von hier ergänzen)

2. </> **Pseudocode**

strukturierte Beschreibung der Methode inklusive Rand- und Basisfälle
eindeutig, präzise; nah an echtem Code

3. © **Korrektheitsbeweis**

Argument, warum der Algorithmus das korrekte Ergebnis liefert
oft per Induktion unter Verwendung von Invarianten

4. 🏔️ **Analyse**

Analyse der Kosten des Algorithmus
i. d. R. Θ -Klasse für den Worst Case;
wo interessant auch Speicherbedarf

Brute-force 3SUM

▶ allgemeiner Ratschlag: erstmal die denkbar einfachste Lösung versuchen

1. 💡 Algorithmische Idee

Alle Tripel durchprobieren

Brute-force 3SUM

- ▶ allgemeiner Ratschlag: erstmal die denkbar einfachste Lösung versuchen

1. 💡 Algorithmische Idee

Alle Tripel durchprobieren

2. </> Pseudocode

```
1  $c = 0$ 
2 for  $i = 0, \dots, n - 3$ 
3   for  $j = i + 1, \dots, n - 2$ 
4     for  $k = j + 1, \dots, n - 1$ 
5        $t = A[i] + A[j] + A[k]$ 
6       if  $t == 0$  then  $c := c + 1$ 
7     end for
8   end for
9 end for
10 return  $c$ 
```

Brute-force 3SUM

- ▶ allgemeiner Ratschlag: erstmal die denkbar einfachste Lösung versuchen

1. 💡 Algorithmische Idee

Alle Tripel durchprobieren

2. </> Pseudocode

```
1 c = 0
2 for i = 0, ..., n - 3
3     for j = i + 1, ..., n - 2
4         for k = j + 1, ..., n - 1
5             t = A[i] + A[j] + A[k]
6             if t == 0 then c := c + 1
7         end for
8     end for
9 end for
10 return c
```

3. ☉ Korrektheitsbeweis

direkt nach Definition von c

$$c = \left| \{(i, j, k) : 0 \leq i < j < k < n \wedge A[i] + A[j] + A[k] = 0\} \right|$$

Brute-force 3SUM

- ▶ allgemeiner Ratschlag: erstmal die denkbar einfachste Lösung versuchen

1. 💡 **Algorithmische Idee**
Alle Tripel durchprobieren

2. </> **Pseudocode**

```
1 c = 0
2 for i = 0, ..., n - 3
3     for j = i + 1, ..., n - 2
4         for k = j + 1, ..., n - 1
5             t = A[i] + A[j] + A[k]
6             if t == 0 then c := c + 1
7         end for
8     end for
9 end for
10 return c
```

3. ☉ **Korrektheitsbeweis**
direkt nach Definition von c

Analysen später nochmal im Detail

4. 🏔 **Analyse**

Ausführungen von Zeilen 5–6

Brute-force 3SUM

- ▶ allgemeiner Ratschlag: erstmal die denkbar einfachste Lösung versuchen

1. 💡 Algorithmische Idee

Alle Tripel durchprobieren

2. </> Pseudocode

```
1 c = 0
2 for i = 0, ..., n - 3
3     for j = i + 1, ..., n - 2
4         for k = j + 1, ..., n - 1
5             t = A[i] + A[j] + A[k]
6             if t == 0 then c := c + 1
7         end for
8     end for
9 end for
10 return c
```

3. © Korrektheitsbeweis

direkt nach Definition von c

Analysen später nochmal im Detail

4. 🏠 Analyse

Ausführungen von Zeilen 5–6

$$= \sum_{i=0}^{n-3} \sum_{j=i+1}^{n-2} \sum_{k=j+1}^{n-1} 1$$

Brute-force 3SUM

- ▶ allgemeiner Ratschlag: erstmal die denkbar einfachste Lösung versuchen

1. 💡 Algorithmische Idee

Alle Tripel durchprobieren

2. </> Pseudocode

```
1 c = 0
2 for i = 0, ..., n - 3
3     for j = i + 1, ..., n - 2
4         for k = j + 1, ..., n - 1
5             t = A[i] + A[j] + A[k]
6             if t == 0 then c := c + 1
7         end for
8     end for
9 end for
10 return c
```

3. ☉ Korrektheitsbeweis

direkt nach Definition von c

Analysen später nochmal im Detail

4. 🏠 Analyse

Ausführungen von Zeilen 5–6

$$= \sum_{i=0}^{n-3} \sum_{j=i+1}^{n-2} \left(\sum_{k=j+1}^{n-1} 1 \right) = \sum_{i=0}^{n-3} \sum_{j=i+1}^{n-2} (n - (j + 1))$$

$$|[a..b]| = b - a + 1$$

$$n - (i + 1 + 1)$$

$$= n - i - 2$$

$$n - (n - 2 + i)$$

$$= 1$$

Brute-force 3SUM

- ▶ allgemeiner Ratschlag: erstmal die denkbar einfachste Lösung versuchen

1. 💡 Algorithmische Idee

Alle Tripel durchprobieren

2. </> Pseudocode

```
1 c = 0
2 for i = 0, ..., n - 3
3     for j = i + 1, ..., n - 2
4         for k = j + 1, ..., n - 1
5             t = A[i] + A[j] + A[k]
6             if t == 0 then c := c + 1
7         end for
8     end for
9 end for
10 return c
```

Analysen später nochmal im Detail

4. 🏠 Analyse

Ausführungen von Zeilen 5–6

$$= \sum_{i=0}^{n-3} \sum_{j=i+1}^{n-2} \sum_{k=j+1}^{n-1} 1 = \sum_{i=0}^{n-3} \sum_{j=i+1}^{n-2} (n - (j + 1))$$

$$= \sum_{i=0}^{n-3} \sum_{j=1}^{n-i-2} j$$

$$1 + 2 + 3 + 4$$

$$+ 4 + 3 + 2 + 1$$

$$\hline 5 + 5 \dots 5$$

$$\sum_{i=1}^n i = \frac{n(n+1)}{2},$$

3. ☉ Korrektheitsbeweis

direkt nach Definition von c

<https://www.wild-inter.net/tscs.pdf>

Brute-force 3SUM

- ▶ allgemeiner Ratschlag: erstmal die denkbar einfachste Lösung versuchen

1. 💡 Algorithmische Idee

Alle Tripel durchprobieren

2. </> Pseudocode

```
1 c = 0
2 for i = 0, ..., n - 3
3     for j = i + 1, ..., n - 2
4         for k = j + 1, ..., n - 1
5             t = A[i] + A[j] + A[k]
6             if t == 0 then c := c + 1
7         end for
8     end for
9 end for
10 return c
```

3. ☉ Korrektheitsbeweis

direkt nach Definition von c

Analysen später nochmal im Detail

4. 🏠 Analyse

Ausführungen von Zeilen 5–6

$$\begin{aligned} &= \sum_{i=0}^{n-3} \sum_{j=i+1}^{n-2} \sum_{k=j+1}^{n-1} 1 = \sum_{i=0}^{n-3} \sum_{j=i+1}^{n-2} (n - (j + 1)) \\ &= \sum_{i=0}^{n-3} \sum_{j=1}^{n-i-2} j = \sum_{i=0}^{n-3} \frac{(n-i-2)(n-i-1)}{2} \end{aligned}$$

$$\frac{1-2}{2} \quad \frac{(n-2)(n-1)}{2}$$

Brute-force 3SUM

- ▶ allgemeiner Ratschlag: erstmal die denkbar einfachste Lösung versuchen

1. 💡 Algorithmische Idee

Alle Tripel durchprobieren

2. </> Pseudocode

```
1 c = 0
2 for i = 0, ..., n - 3
3     for j = i + 1, ..., n - 2
4         for k = j + 1, ..., n - 1
5             t = A[i] + A[j] + A[k]
6             if t == 0 then c := c + 1
7         end for
8     end for
9 end for
10 return c
```

3. ☉ Korrektheitsbeweis

direkt nach Definition von c

Analysen später nochmal im Detail

4. 🏠 Analyse

Ausführungen von Zeilen 5–6

$$\begin{aligned} &= \sum_{i=0}^{n-3} \sum_{j=i+1}^{n-2} \sum_{k=j+1}^{n-1} 1 = \sum_{i=0}^{n-3} \sum_{j=i+1}^{n-2} (n - (j + 1)) \\ &= \sum_{i=0}^{n-3} \sum_{j=1}^{n-i-2} j = \sum_{i=0}^{n-3} \frac{(n - i - 2)(n - i - 1)}{2} \\ &= \frac{1}{2} \sum_{i=1}^{n-2} i(i + 1) = \frac{1}{2} \sum_{i=1}^{n-2} i^2 + \frac{1}{2} \sum_{i=1}^{n-2} i \end{aligned}$$

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

Brute-force 3SUM

- ▶ allgemeiner Ratschlag: erstmal die denkbar einfachste Lösung versuchen

1. 💡 Algorithmische Idee

Alle Tripel durchprobieren

2. </> Pseudocode

```
1 c = 0
2 for i = 0, ..., n - 3
3     for j = i + 1, ..., n - 2
4         for k = j + 1, ..., n - 1
5             t = A[i] + A[j] + A[k]
6             if t == 0 then c := c + 1
7         end for
8     end for
9 end for
10 return c
```

3. ☉ Korrektheitsbeweis

direkt nach Definition von c

Analysen später nochmal im Detail

4. 🏠 Analyse

Ausführungen von Zeilen 5–6

$$\begin{aligned} &= \sum_{i=0}^{n-3} \sum_{j=i+1}^{n-2} \sum_{k=j+1}^{n-1} 1 = \sum_{i=0}^{n-3} \sum_{j=i+1}^{n-2} (n - (j + 1)) \\ &= \sum_{i=0}^{n-3} \sum_{j=1}^{n-i-2} j = \sum_{i=0}^{n-3} \frac{(n - i - 2)(n - i - 1)}{2} \\ &= \frac{1}{2} \sum_{i=1}^{n-2} i(i + 1) = \frac{1}{2} \sum_{i=1}^{n-2} i^2 + \frac{1}{2} \sum_{i=1}^{n-2} i \\ &= \frac{(n - 2)(n - 1)(2(n - 2) + 1)}{12} + \frac{(n - 2)(n - 1)}{4} \end{aligned}$$

Brute-force 3SUM

- ▶ allgemeiner Ratschlag: erstmal die denkbar einfachste Lösung versuchen

1. 💡 Algorithmische Idee

Alle Tripel durchprobieren

2. </> Pseudocode

```
1 c = 0
2 for i = 0, ..., n - 3
3     for j = i + 1, ..., n - 2
4         for k = j + 1, ..., n - 1
5             t = A[i] + A[j] + A[k]
6             if t == 0 then c := c + 1
7         end for
8     end for
9 end for
10 return c
```

3. © Korrektheitsbeweis

direkt nach Definition von c

Analysen später nochmal im Detail

4. 🏠 Analyse

Ausführungen von Zeilen 5–6

$$\begin{aligned} &= \sum_{i=0}^{n-3} \sum_{j=i+1}^{n-2} \sum_{k=j+1}^{n-1} 1 = \sum_{i=0}^{n-3} \sum_{j=i+1}^{n-2} (n - (j + 1)) \\ &= \sum_{i=0}^{n-3} \sum_{j=1}^{n-i-2} j = \sum_{i=0}^{n-3} \frac{(n - i - 2)(n - i - 1)}{2} \\ &= \frac{1}{2} \sum_{i=1}^{n-2} i(i + 1) = \frac{1}{2} \sum_{i=1}^{n-2} i^2 + \frac{1}{2} \sum_{i=1}^{n-2} i \\ &= \frac{(n - 2)(n - 1)(2(n - 2) + 1)}{12} + \frac{(n - 2)(n - 1)}{4} \\ &= \frac{(n - 2)(n - 1)n}{6} = \binom{n}{3} \end{aligned}$$

Brute-force 3SUM

- ▶ allgemeiner Ratschlag: erstmal die denkbar einfachste Lösung versuchen

1. 💡 Algorithmische Idee

Alle Tripel durchprobieren

2. </> Pseudocode

```
1 c = 0
2 for i = 0, ..., n - 3
3     for j = i + 1, ..., n - 2
4         for k = j + 1, ..., n - 1
5             t = A[i] + A[j] + A[k]
6             if t == 0 then c := c + 1
7         end for
8     end for
9 end for
10 return c
```

3. ☉ Korrektheitsbeweis

direkt nach Definition von c

Analysen später nochmal im Detail

4. 🏠 Analyse

Ausführungen von Zeilen 5–6

$$\begin{aligned} &= \sum_{i=0}^{n-3} \sum_{j=i+1}^{n-2} \sum_{k=j+1}^{n-1} 1 = \sum_{i=0}^{n-3} \sum_{j=i+1}^{n-2} (n - (j + 1)) \\ &= \sum_{i=0}^{n-3} \sum_{j=1}^{n-i-2} j = \sum_{i=0}^{n-3} \frac{(n - i - 2)(n - i - 1)}{2} \\ &= \frac{1}{2} \sum_{i=1}^{n-2} i(i + 1) = \frac{1}{2} \sum_{i=1}^{n-2} i^2 + \frac{1}{2} \sum_{i=1}^{n-2} i \\ &= \frac{(n - 2)(n - 1)(2(n - 2) + 1)}{12} + \frac{(n - 2)(n - 1)}{4} \\ &= \frac{(n - 2)(n - 1)n}{6} \quad \text{Ist das jetzt gut??} \end{aligned}$$

Einschub: Überschlagsrechnungen (*back-of-the-envelope math*)

Ob $T(n) = \frac{(n-2)(n-1)n}{6}$ Operationen machbar sind, hängt von n ab!

Ganz grobe Überschlagsrechnungen sind hier hilfreich!

n	$T(n)$	Zeit bei 1GHz / 1ns pro Operation
10	120	instant
100	161 700	0.1 ms
1 000	166 167 000	0.16 s
10 000	166 616 670 000	2:40 min
100 000	166 661 666 700 000	46 h (!)

↪ Arrays mit 10 000 Zahlen grade noch machbar, 100 000 eher nicht.

Einschub: Überschlagsrechnungen (*back-of-the-envelope math*)

Ob $T(n) = \frac{(n-2)(n-1)n}{6}$ Operationen machbar sind, hängt von n ab!

Ganz grobe Überschlagsrechnungen sind hier hilfreich!

n	$T(n)$	Zeit bei 1GHz / 1ns pro Operation
10	120	instant
100	161 700	0.1 ms
1 000	166 167 000	0.16 s
10 000	166 616 670 000	2:40 min
100 000	166 661 666 700 000	46 h (!)

↪ Arrays mit 10 000 Zahlen grade noch machbar, 100 000 eher nicht.

Übrigens: Man kann 3SUM deutlich schneller lösen . . .
werden wir noch einmal wiedersehen 😊

3.2 Folgen & Grenzwerte

Definitionen zur Erinnerung

Ein bisschen Notation

- ▶ $\mathbb{N}_0 = \{0, 1, 2, 3, \dots, \}$ natürliche Zahlen mit 0
- ▶ $\mathbb{N}_{\geq 1} = \{1, 2, 3, \dots, \}$ natürliche Zahlen ab 1
- ▶ ~~\mathbb{N}~~ : (Uneindeutigkeit vermeiden)
- ▶ \mathbb{R} : reelle Zahlen

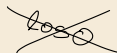
\mathbb{Z}

Definitionen zur Erinnerung

Ein bisschen Notation

- ▶ $\mathbb{N}_0 = \{0, 1, 2, 3, \dots, \}$ natürliche Zahlen mit 0
- ▶ $\mathbb{N}_{\geq 1} = \{1, 2, 3, \dots, \}$ natürliche Zahlen ab 1
- ▶ ~~\mathbb{N}~~ : (Uneindeutigkeit vermeiden)
- ▶ \mathbb{R} : reelle Zahlen
- ▶ $\lg = \log_2$; $\ln = \log_e$ $e \approx 2.71$

ed



Definitionen zur Erinnerung

Ein bisschen Notation

- ▶ $\mathbb{N}_0 = \{0, 1, 2, 3, \dots, \}$ natürliche Zahlen mit 0
- ▶ $\mathbb{N}_{\geq 1} = \{1, 2, 3, \dots, \}$ natürliche Zahlen ab 1
- ▶ ~~\mathbb{N}~~ : (Uneindeutigkeit vermeiden)
- ▶ \mathbb{R} : reelle Zahlen
- ▶ $\lg = \log_2$; $\ln = \log_e$

Folgen

- ▶ (Reelle) *Zahlenfolge* $(a_n)_{n \in \mathbb{N}_{\geq 1}}$
formal eine Funktion $a : \mathbb{N}_{\geq 1} \rightarrow \mathbb{R}$
schreiben auch " a_1, a_2, a_3, \dots "

Definitionen zur Erinnerung

Ein bisschen Notation

- ▶ $\mathbb{N}_0 = \{0, 1, 2, 3, \dots\}$ natürliche Zahlen mit 0
- ▶ $\mathbb{N}_{\geq 1} = \{1, 2, 3, \dots\}$ natürliche Zahlen ab 1
- ▶ ~~\mathbb{N}~~ : (Uneindeutigkeit vermeiden)
- ▶ \mathbb{R} : reelle Zahlen
- ▶ $\lg = \log_2$; $\ln = \log_e$

Folgen

- ▶ (Reelle) *Zahlenfolge* $(a_n)_{n \in \mathbb{N}_{\geq 1}}$
formal eine Funktion $a : \mathbb{N}_{\geq 1} \rightarrow \mathbb{R}$
schreiben auch " a_1, a_2, a_3, \dots "

" $a_n = n^2$ " $(a : n \mapsto n^2)$

- ▶ *Grenzwert* einer Folge

$$\lim_{n \rightarrow \infty} a_n = a^*$$

genau dann, wenn

forall existiert

$$\forall \varepsilon > 0 \exists n_0 \in \mathbb{N}_{\geq 1} \forall n \geq n_0 : |a_n - a^*| < \varepsilon$$

Definitionen zur Erinnerung

Ein bisschen Notation

- ▶ $\mathbb{N}_0 = \{0, 1, 2, 3, \dots\}$ natürliche Zahlen mit 0
- ▶ $\mathbb{N}_{\geq 1} = \{1, 2, 3, \dots\}$ natürliche Zahlen ab 1
- ▶ ~~\mathbb{N}~~ : (Uneindeutigkeit vermeiden)
- ▶ \mathbb{R} : reelle Zahlen
- ▶ $\lg = \log_2$; $\ln = \log_e$

Folgen

$$a_n = (-1)^n$$

- ▶ (Reelle) *Zahlenfolge* $(a_n)_{n \in \mathbb{N}_{\geq 1}}$
formal eine Funktion $a : \mathbb{N}_{\geq 1} \rightarrow \mathbb{R}$
schreiben auch " a_1, a_2, a_3, \dots "

- ▶ *Grenzwert* einer Folge

$$\lim_{n \rightarrow \infty} a_n = a^* \text{ genau dann, wenn } \boxed{\forall \varepsilon > 0 \exists n_0 \in \mathbb{N}_{\geq 1} \forall n \geq n_0 : |a_n - a^*| < \varepsilon}$$

- ▶ Nicht jede Folge hat einen Grenzwert, aber wenn er existiert, dann ist er eindeutig.

Folgen – Beispiel

$$\forall \varepsilon > 0 \exists n_0 \in \mathbb{N}_{\geq 1} \forall n \geq n_0 : |a_n - a^*| < \varepsilon$$

Es sei die Folge $(a_n)_{n \in \mathbb{N}_{\geq 1}}$ mit $a_n = \frac{1}{n}$ gegeben.

Es gibt: $\lim_{n \rightarrow \infty} a_n = 0$

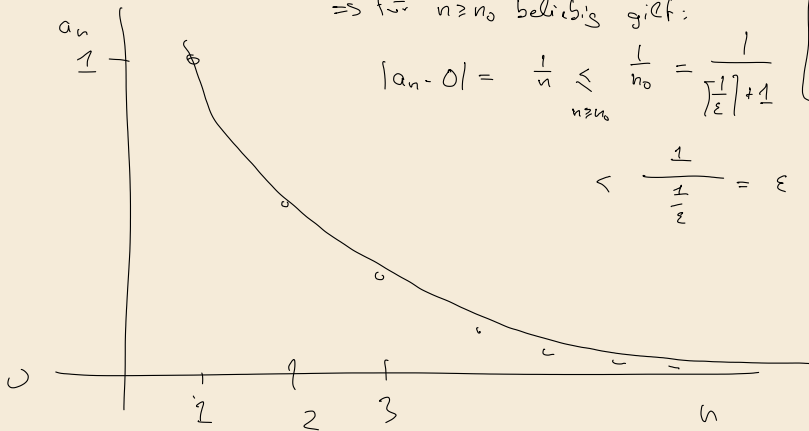
$\varepsilon > 0$ beliebig, Wähle $n_0 := \left\lceil \frac{1}{\varepsilon} \right\rceil + 1$

\Rightarrow für $n \geq n_0$ beliebig gilt:

$$|a_n - 0| = \frac{1}{n} \stackrel{n \geq n_0}{\leq} \frac{1}{n_0} = \frac{1}{\left\lceil \frac{1}{\varepsilon} \right\rceil + 1}$$

$$< \frac{1}{\frac{1}{\varepsilon}} = \varepsilon \quad \square$$

$$\begin{aligned} \text{N.R.} \\ |a_n - 0| < \varepsilon \\ &= \frac{1}{n} < \varepsilon \\ n &> \frac{1}{\varepsilon} \text{ tut's} \end{aligned}$$



Grenzwerte von Funktionen

$$f: \mathbb{R} \rightarrow \mathbb{R}$$

$$\lim_{x \rightarrow \xi} f(x) = f^* \text{ gdw } \forall \varepsilon > 0 \exists \delta > 0 \forall x \in \mathbb{R} : |x - \xi| < \delta \implies |f(x) - f^*| < \varepsilon$$

" ε - δ -Kriterium"

Limes superior, Limes inferior

- Supremum ist die kleinste obere Schranke

Für $M \subseteq \mathbb{R}$ ist $\sup M = \min\{b \in \mathbb{R} : \forall x \in M : x \leq b\}$

$$M = \left\{ 1 - \frac{1}{n} : n \in \mathbb{N}_0 \right\}$$

$$x \in M \Rightarrow x \leq 1$$

$$\sup M = 1$$

Limes superior, Limes inferior

- ▶ Supremum ist die kleinste obere Schranke
Für $M \subseteq \mathbb{R}$ ist $\sup M = \min\{b \in \mathbb{R} : \forall x \in M : x \leq b\}$

- ▶ Analog ist Infimum die größte untere Schranke

- ▶ Beide müssen nicht existieren ($+\infty$ bzw. $-\infty$); wenn sie existieren, sind sie eindeutig.

$$\inf \left\{ \frac{1}{n} ; n \in \mathbb{N} \right\} = 0$$

Limes superior, Limes inferior

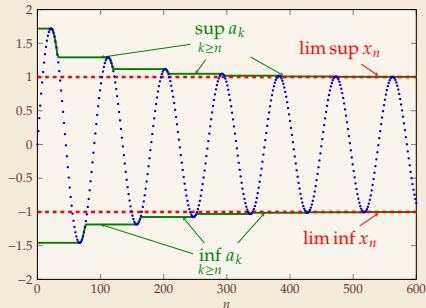
- ▶ Supremum ist die kleinste obere Schranke
Für $M \subseteq \mathbb{R}$ ist $\sup M = \min\{b \in \mathbb{R} : \forall x \in M : x \leq b\}$
- ▶ Analog ist Infimum die größte untere Schranke
- ▶ Beide müssen nicht existieren ($+\infty$ bzw. $-\infty$); wenn sie existieren, sind sie eindeutig.

- ▶ Für Folge $(a_n)_{n \in \mathbb{N}_{\geq 1}}$ ist der Limes superior:

$$\limsup_{n \rightarrow \infty} a_n = \inf \left\{ \sup \{ a_k : k \geq n \} : n \in \mathbb{N}_{\geq 1} \right\}$$

Äquivalent: größter Häufungspunkt von (a_n) ;
beste obere Schranke für a_n
für genügend große n

- ▶ Analog Limes inferior: $\liminf_{n \rightarrow \infty} a_n = \sup \left\{ \inf \{ a_k : k \geq n \} : n \in \mathbb{N}_{\geq 1} \right\}$



3.3 Asymptotische Notation

Clicker Question



Ist $n^{0.1} = O((\log(n))^{10})$?

- | | | | |
|----------|------------------------|----------|--------------|
| A | O?? nie gesehen | D | Denke Ja |
| B | Ja (bin mir sicher). | E | Denke Nein |
| C | Nein (bin mir sicher). | F | keine Ahnung |



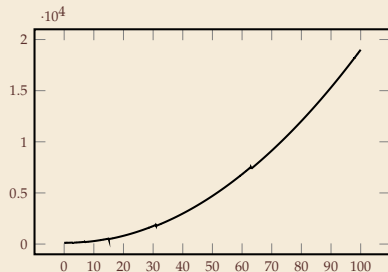
→ sli.do/cs210

Asymptotische Vergleiche

Asymptotik = Approximation um ∞

Beispiel: Betrachte Funktion $f(n)$ mit

$$2n^2 - 3n\lfloor\log_2(n+1)\rfloor + 7n - 3\lfloor\log_2(n+1)\rfloor + 120$$

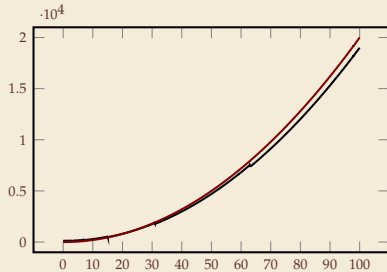


Asymptotische Vergleiche

Asymptotik = Approximation um ∞

Beispiel: Betrachte Funktion $f(n)$ mit

$$2n^2 - 3n\lceil\log_2(n+1)\rceil + 7n - 3\lfloor\log_2(n+1)\rfloor + 120 \sim 2n^2$$



Asymptotische Approximationen in der Analyse von Algorithmen

- ▶ abstrahiert von unwichtigen Details
- ▶ vereinfacht die Analyse selbst
- ▶ ermöglicht oft erst konklusiven Vergleich

Asymptotische Notation – Formale Definition

► “Tilde Notation”: $f(n) \sim g(n)$ genau dann, wenn (iff) \downarrow **gdw** $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$
„ f und g sind *asymptotisch äquivalent*“

Asymptotische Notation – Formale Definition

▶ **“Tilde Notation”:** $f(n) \sim g(n)$ genau dann, wenn (iff) $\text{gdw } \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$

„ f und g sind *asymptotisch äquivalent*“

▶ **“Big-Oh Notation”:** $f(n) \in O(g(n))$ schreiben auch ‘=’ stattdessen $\text{gdw } \left| \frac{f(n)}{g(n)} \right|$ beschränkt für $n \geq n_0$

brauchen Supremum da Limes evtl. nicht existiert!

$$\text{gdw } \limsup_{n \rightarrow \infty} \left| \frac{f(n)}{g(n)} \right| < \infty$$

Varianten: “Big-Omega”

▶ $f(n) \in \Omega(g(n))$ “Big-Omega” $\text{gdw } g(n) \in O(f(n))$

▶ $f(n) \in \Theta(g(n))$ $\text{gdw } f(n) \in O(g(n))$ **und** $f(n) \in \Omega(g(n))$

“Big-Theta”

Asymptotische Notation – Formale Definition

- ▶ **“Tilde Notation”:** $f(n) \sim g(n)$ genau dann, wenn (iff) gdw $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$.
„ f und g sind *asymptotisch äquivalent*“

- ▶ **“Big-Oh Notation”:** $f(n) \in O(g(n))$ schreiben auch ‘=’ stattdessen gdw $\left| \frac{f(n)}{g(n)} \right|$ beschränkt für $n \geq n_0$

brauchen Supremum da Limes evtl. nicht existiert!

$$\text{gdw } \lim_{n \rightarrow \infty} \sup \left| \frac{f(n)}{g(n)} \right| < \infty$$

Varianten: “Big-Omega”

- ▶ $f(n) \in \Omega(g(n))$ gdw $g(n) \in O(f(n))$
▶ $f(n) \in \Theta(g(n))$ gdw $f(n) \in O(g(n))$ **und** $f(n) \in \Omega(g(n))$

“Big-Theta”

- ▶ **“Little-Oh Notation”:** $f(n) \in o(g(n))$ gdw $\lim_{n \rightarrow \infty} \left| \frac{f(n)}{g(n)} \right| = 0$

analog: $f(n) \in \omega(g(n))$ falls $|f(n)/g(n)| \rightarrow \infty$

(Vorteil dieser Definition: Funktioniert genauso für $f, g : \mathbb{R} \rightarrow \mathbb{R}$ und für andere Grenzprozesse als $n \rightarrow \infty$)

Asymptotische Notation – Intuition

\leq

- $f(n) = O(g(n))$: $f(n)$ ist **höchstens** $g(n)$
bis auf konstante Faktoren
und
für hinreichend große n

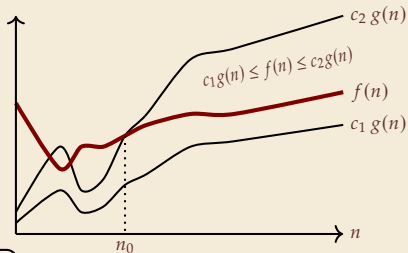
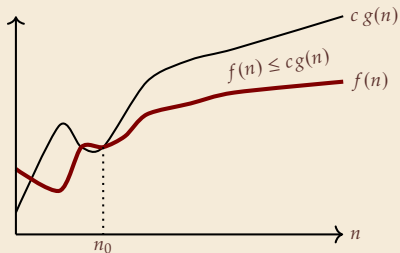
\geq

$$f(n) = \Omega(g(n))$$

\ll

$$f(n) = o(g(n))$$

- $f(n) = \Theta(g(n))$: $f(n)$ ist **gleich** $g(n)$
bis auf konstante Faktoren
und
für hinreichend große n



Plots können irreführend sein!

Beispiel ↗

O-Notation – Beispiel

$$f(n) = 3n^3 + 2n^2, \quad g(n) = n^3$$

$$\frac{f(n)}{g(n)} = \frac{3n^3 + 2n^2}{n^3} = 3 + \frac{2}{\frac{n}{n}} \rightarrow 0$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 3$$

$f \neq g$

$f(n) = O(g(n))$ da 3 beschränkt

$$g(n) \stackrel{?}{=} O(f(n)) \quad \checkmark \quad \frac{g(n)}{f(n)} \rightarrow \frac{1}{3}$$

$$\Rightarrow f(n) = \Omega(g(n))$$

$$\Rightarrow f(n) = \Theta(g(n))$$

3.4 Rechenregeln für Asymptotiken

O-Notation – Beispiel 2

$$f(n) \in o(g(n)) \quad \text{gdw} \quad \lim_{n \rightarrow \infty} \left| \frac{f(n)}{g(n)} \right| = 0$$

$$n \ln(n) = o(n^2) \quad \checkmark$$

$$\lim_{n \rightarrow \infty} \frac{n \ln(n)}{n^2} = \lim_{n \rightarrow \infty} \frac{\ln(n)}{n} = \frac{\infty}{\infty}$$

$$\begin{aligned} f(u) &= \ln(u) & f'(u) &= \frac{1}{u} \quad \neq 0 \\ g(u) &= u & g'(u) &= 1 \quad \neq \end{aligned}$$

$$\lim_{u \rightarrow \infty} \frac{f'(u)}{g'(u)} = \lim_{u \rightarrow \infty} \frac{1}{u} = 0$$

$$\Rightarrow \text{l'Hospital} \quad \lim_{n \rightarrow \infty} \frac{n \ln(n)}{n^2} = \lim_{u \rightarrow \infty} \frac{f'(u)}{g'(u)} = \lim_{u \rightarrow \infty} \frac{1}{u} = 0$$

Regel von de l'Hospital

Bei Grenzwerten der Form $\frac{\infty}{\infty}$ bzw. $\frac{0}{0}$ kann man oft folgende Regel anwenden.

Theorem 3.1 (de l'Hospital)

Gegeben Funktionen $f, g : \mathbb{R}_{>n_0} \rightarrow \mathbb{R}$, die

1. *differenzierbar* auf $\mathbb{R}_{>n_0}$ sind,
2. $\forall x \in \mathbb{R}_{>n_0} : g'(x) \neq 0$,
3. $\lim_{x \rightarrow \infty} |f(x)| = \lim_{x \rightarrow \infty} |g(x)| \in \{0, \infty\}$, und
4. $\lim_{x \rightarrow \infty} \frac{f'(x)}{g'(x)}$ existiert

Dann ist $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{f'(n)}{g'(n)}$.

Ausnahme von der Regel „log wie ohne Basis“

$$\text{in } \Theta(\log u) = \Theta(\log_2 u) = \Theta(\log_e u)$$

$$\log_b u = \frac{\ln u}{\ln(b)=c} = c \cdot \ln u$$

$$\frac{\log_b u}{\ln u} = \frac{1}{c} \Rightarrow \log_b u = \Theta(\ln u)$$

Clicker Question

$$\lim_{n \rightarrow \infty} \frac{n^{0.1}}{\ln^{10}(n)} = \frac{\infty}{\infty}$$

$$\frac{d}{dn} n^{0.1} = 0.1 \cdot n^{-0.9}$$

$$\frac{d}{dn} \ln^{10}(n) = 10 \ln^9(n) \cdot \frac{1}{n}$$

$$\frac{\frac{d}{dn} n^{0.1}}{\frac{d}{dn} \ln^{10}(n)} = \frac{n^{0.1}}{100 \ln^9(n)}$$

Ist $n^{0.1} = O((\log(n))^{10})$?



- | | |
|---------------------------------|-----------------------|
| A O?? nie gesehen | D Denke Ja |
| B Ja (bin mir sicher). | E Denke Nein |
| C Nein (bin mir sicher). | F keine Ahnung |

∴
10 mal
l'Hospital
→ ∞

$$\forall c, d > 0 \quad n^c = \omega(\log^d(n))$$



→ sli.do/cs210

Clicker Question

Ist $n^{0.1} = O((\log(n))^{10})$?



- A** ~~0?? nie gesehen~~
- B** ~~Ja (bin mir sicher).~~
- C** Nein (bin mir sicher). ✓
- D** ~~Denke Ja~~
- E** ~~Denke Nein~~
- F** ~~keine Ahnung~~



→ sli.do/cs210

Rechenregeln

Lemma 3.2 (Rechenregeln für asymptotische Notationen)

Es gilt:

- (a) $f = O(f)$, $f = \Theta(f)$, $f = \Omega(f)$ (Reflexivität). $f \notin o(f)$
- (b) $c \cdot O(f) = O(f)$ für c konstant. $f \notin \omega(f)$

Rechenregeln

Lemma 3.2 (Rechenregeln für asymptotische Notationen)

Es gilt:

(a) $f = O(f)$, $f = \Theta(f)$, $f = \Omega(f)$ (Reflexivität).

(b) $c \cdot O(f) = O(f)$ für c konstant.

(c) $O(f) + O(g) = O(f + g) = O(\max(f, g))$.

(d) $O(O(f)) = O(f)$.

(e) $O(f) \cdot O(g) = O(f \cdot g)$.

Rechenregeln

Lemma 3.2 (Rechenregeln für asymptotische Notationen)

Es gilt:

(a) $f = O(f), f = \Theta(f), f = \Omega(f)$ (Reflexivität).

(b) $c \cdot O(f) = O(f)$ für c konstant.

(c) $O(f) + O(g) = O(f + g) = O(\max(f, g))$.

(d) $O(O(f)) = O(f)$.

(e) $O(f) \cdot O(g) = O(f \cdot g)$.

(f) Falls $f \sim g$, dann gilt $f = g \cdot (1 \pm o(1))$.

(g) $f = O(g) \leftrightarrow g = \Omega(f)$ (Schiefsymmetrie),

$f = \Theta(g) \leftrightarrow g = \Theta(f)$ (Symmetrie).

(h) $\Omega(f) + \Omega(g) = \Omega(\min(f, g))$.

$$O(f) = \left\{ h : \limsup_{n \rightarrow \infty} \left| \frac{h(n)}{f(n)} \right| < \infty \right\}$$

$$O(g) = \left\{ h : \limsup_{n \rightarrow \infty} \left| \frac{h(n)}{g(n)} \right| < \infty \right\}$$

$$O(f) + O(g) =$$

$$1 \pm o(1) = \left\{ h : h(n) = 1 + h_2(n), h_2 \in o(1) \vee \right. \\ \left. h(n) = 1 - h_2(n), h_2 \in o(1) \right\}$$

$$\left\{ h : h(n) = h_1(n) + h_2(n) \wedge h_1 \in O(f) \wedge h_2 \in O(g) \right\}$$

Rechenregeln

Lemma 3.2 (Rechenregeln für asymptotische Notationen)

Es gilt:

- (a) $f = O(f)$, $f = \Theta(f)$, $f = \Omega(f)$ (Reflexivität).
- (b) $c \cdot O(f) = O(f)$ für c konstant.
- (c) $O(f) + O(g) = O(f + g) = O(\max(f, g))$.
- (d) $O(O(f)) = O(f)$.
- (e) $O(f) \cdot O(g) = O(f \cdot g)$.
- (f) Falls $f \sim g$, dann gilt $f = g \cdot (1 \pm o(1))$.
- (g) $f = O(g) \leftrightarrow g = \Omega(f)$ (Schiefsymmetrie),
 $f = \Theta(g) \leftrightarrow g = \Theta(f)$ (Symmetrie).
- (h) $\Omega(f) + \Omega(g) = \Omega(\min(f, g))$.
- (i) $f = o(g) \wedge g = o(h) \rightarrow f = o(h)$ für $o \in \{O, o, \Omega, \omega, \Theta\}$ (Transitivität).



O-Notation – Quantoren-Definition

Für alle asymptotischen Notationen $f(n) = O(g(n))$: Grenzwert des Quotienten $\frac{f(n)}{g(n)}$

Alternative Definition (EAA Buch) (äquivalent für $f : \mathbb{N}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$)

Definition 1.16 (Asymptotische Notation [Landau-Klassen]):

Sei $\mathcal{F} := \text{Abb}(\mathbb{N}_0, \mathbb{R}_0^+)$ die Menge der Abbildungen von der Menge der natürlichen Zahlen inklusive der Null in die Menge der nicht-negativen reellen Zahlen. Sei ferner $g \in \mathcal{F}$. Wir definieren die folgenden Mengen von Funktionen:

- 1.) $\mathcal{O}(g) := \{f \in \mathcal{F} \mid (\exists c > 0)(\exists n_0 \in \mathbb{N}_0)(\forall n \geq n_0) (f(n) \leq c \cdot g(n))\}$,
- 2.) $\Omega(g) := \{f \in \mathcal{F} \mid (\exists c > 0)(\exists n_0 \in \mathbb{N}_0)(\forall n \geq n_0) (f(n) \geq c \cdot g(n))\}$,
- 3.) $\Theta(g) := \mathcal{O}(g) \cap \Omega(g)$,
- 4.) $o(g) := \{f \in \mathcal{F} \mid (\forall c > 0)(\exists n_0 \in \mathbb{N}_0)(\forall n \geq n_0) (f(n) \leq c \cdot g(n))\}$,
- 5.) $\omega(g) := \{f \in \mathcal{F} \mid (\forall c > 0)(\exists n_0 \in \mathbb{N}_0)(\forall n \geq n_0) (f(n) \geq c \cdot g(n))\}$,
- 6.) $\sim(g) := \{f \in \mathcal{F} \mid (\forall \varepsilon > 0)(\exists n_0 \in \mathbb{N}_0)(\forall n \geq n_0) (|\frac{f(n)}{g(n)} - 1| \leq \varepsilon)\}$.

Hier nehmen wir an, dass $g(n) = 0$ höchstens für endlich viele n gilt.

(Beachte die unterschiedlichen Quantoren bei c bzw. ε !) ◀

3.5 Rekursiv definierte Folgen

Clicker Question



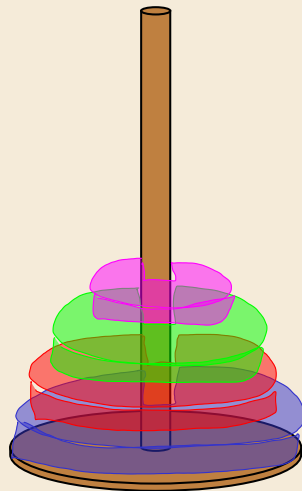
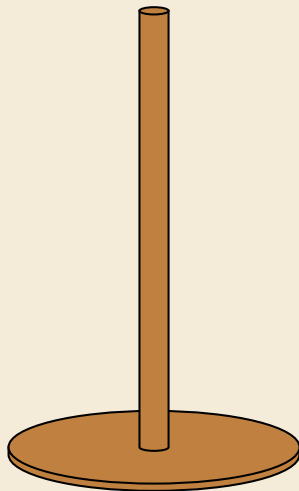
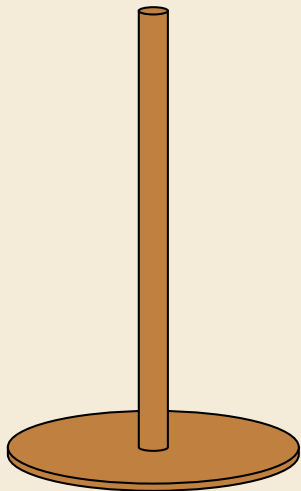
Kennen Sie die Türme von Hanoi?

- A** Absolut, könnte ich erklären.
- B** Kam wo vor, aber ich erinnere mich nicht so ganz.
- C** Ziemlich sicher noch nie gehört.
- D** War noch nie in Vietnam.



→ sli.do/cs210

Die "Türme von Hanoi"



Rekursive Lösung

```
1 procedure towersOfHanoi( $s, t, b, n$ )
2   // Move topmost  $n$  pieces from  $s$  to  $t$ 
3   // Assumes the topmost  $n$  disks are on pile  $s$ 
4   if  $n == 0$  return
5   towersOfHanoi( $s, b, t, n - 1$ )
6   move disk  $n$  from  $s$  to  $t$ 
7   towersOfHanoi( $b, t, s, n - 1$ )
```

s : Quellstab (source)

t : Zielstab (target)

b : Hilfsstab (buffer)

towersOfHanoi(1, 3, 2, n)

Rekursive Lösung

```
1 procedure towersOfHanoi(s, t, b, n)
2   // Move topmost n pieces from s to t
3   // Assumes the topmost n disks are on pile s
4   if n == 0 return
5   towersOfHanoi(s, b, t, n - 1)
6   move disk n from s to t
7   towersOfHanoi(b, t, s, n - 1)
```

s: Quellstab (source)

t: Zielstab (target)

b: Hilfsstab (buffer)

► einfacher Code

► landet gezielt auf korrektem Stab!

https://upload.wikimedia.org/wikipedia/commons/2/20/Tower_of_Hanoi_recursion_SMIL.svg

Rekursive Lösung

```
1 procedure towersOfHanoi(s, t, b, n)
2   // Move topmost n pieces from s to t
3   // Assumes the topmost n disks are on pile s
4   if n == 0 return
5   towersOfHanoi(s, b, t, n - 1)
6   move disk n from s to t
7   towersOfHanoi(b, t, s, n - 1)
```

s: Quellstab (source)

t: Zielstab (target)

b: Hilfsstab (buffer)

Anzahl Bewegungen

- ▶ Nur Zeile 6 bewegt eine Scheibe!

- ▶ einfacher Code

- ▶ landet gezielt auf korrektem Stab!

https://upload.wikimedia.org/wikipedia/commons/2/20/Tower_of_Hanoi_recursion_SMIL.svg

- ▶ aber wie lange braucht der Code denn?

Rekursive Lösung

```
1 procedure towersOfHanoi(s, t, b,  $n$ )
2   // Move topmost  $n$  pieces from  $s$  to  $t$ 
3   // Assumes the topmost  $n$  disks are on pile  $s$ 
4   if  $n == 0$  return
5   towersOfHanoi(s, b, t,  $n - 1$ )
6   move disk  $n$  from  $s$  to  $t$ 
7   towersOfHanoi(b, t, s,  $n - 1$ )
```

s : Quellstab (source)

t : Zielstab (target)

b : Hilfsstab (buffer)

Anzahl Bewegungen

- ▶ Nur Zeile 6 bewegt eine Scheibe! ... aber natürlich auch rekursiv aufgerufen.
 - ▶ Für $n = 0$ keine weitere Bewegung; sonst eine Bewegung und 2 mal die Bewegungen für $n - 1$

▶ einfacher Code

▶ landet gezielt auf korrektem Stab!

https://upload.wikimedia.org/wikipedia/commons/2/20/Tower_of_Hanoi_recursion_SMIL.svg

▶ aber wie lange braucht der Code denn?

Rekursive Lösung

```
1 procedure towersOfHanoi(s, t, b, n)
2   // Move topmost n pieces from s to t
3   // Assumes the topmost n disks are on pile s
4   if n == 0 return
5   towersOfHanoi(s, b, t, n - 1)
6   move disk n from s to t
7   towersOfHanoi(b, t, s, n - 1)
```

► einfacher Code

► landet gezielt auf korrektem Stab!

https://upload.wikimedia.org/wikipedia/commons/2/20/Tower_of_Hanoi_recursion_SMIL.svg

s: Quellstab (source)

t: Zielstab (target)

b: Hilfsstab (buffer)

► aber wie lange braucht der Code denn?

Anzahl Bewegungen

► Nur Zeile 6 bewegt eine Scheibe! ... aber natürlich auch rekursiv aufgerufen.

► Für $n = 0$ keine weitere Bewegung; sonst eine Bewegung und 2 mal die Bewegungen für $n - 1$

↪ *Rekursionsgleichung / Rekurrenz*: $T : \mathbb{N}_{\geq 0} \rightarrow \mathbb{N}_{\geq 0}$ $T(n) = \#$ Bewegungen für n Scheiben

$$T(0) = 0,$$

Rekursive Lösung

```
1 procedure towersOfHanoi(s, t, b, n)
2   // Move topmost n pieces from s to t
3   // Assumes the topmost n disks are on pile s
4   if n == 0 return
5   towersOfHanoi(s, b, t, n - 1)
6   move disk n from s to t
7   towersOfHanoi(b, t, s, n - 1)
```

► einfacher Code

► landet gezielt auf korrektem Stab!

https://upload.wikimedia.org/wikipedia/commons/2/20/Tower_of_Hanoi_recursion_SMIL.svg

s: Quellstab (source)

t: Zielstab (target)

b: Hilfsstab (buffer)

► aber wie lange braucht der Code denn?

Anzahl Bewegungen

► Nur Zeile 6 bewegt eine Scheibe! ... aber natürlich auch rekursiv aufgerufen.

► Für $n = 0$ keine weitere Bewegung; sonst eine Bewegung und 2 mal die Bewegungen für $n - 1$

↪ *Rekursionsgleichung / Rekurrenz*: $T : \mathbb{N}_{\geq 0} \rightarrow \mathbb{N}_{> 0}$ $T(n) = \#$ Bewegungen für n Scheiben

$$T(0) = 0, \quad T(n) = T(n - 1) + 1 + T(n - 1) = 2T(n - 1) + 1 \quad (n \geq 1)$$

Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0

Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0
1	$2 \cdot 0 + 1 = 1$

Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0
1	$2 \cdot 0 + 1 = 1$
2	$2 \cdot 1 + 1 = 3$

Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0
1	$2 \cdot 0 + 1 = 1$
2	$2 \cdot 1 + 1 = 3$
3	$2 \cdot 3 + 1 = 7$
4	$2 \cdot 7 + 1 = 15$
5	$2 \cdot 15 + 1 = 31$
6	$2 \cdot 31 + 1 = 63$

(Informatiker

erkennen

Zweierpotenzen 😊)

Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0
1	$2 \cdot 0 + 1 = 1$
2	$2 \cdot 1 + 1 = 3$
3	$2 \cdot 3 + 1 = 7$
4	$2 \cdot 7 + 1 = 15$
5	$2 \cdot 15 + 1 = 31$
6	$2 \cdot 31 + 1 = 63$

(Informatiker

erkennen

Zweierpotenzen 😊)

2. Lösung raten

(manchmal schwierig!!)

Oft hilfreich: einsetzen!

$T(n)$

Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0
1	$2 \cdot 0 + 1 = 1$
2	$2 \cdot 1 + 1 = 3$
3	$2 \cdot 3 + 1 = 7$
4	$2 \cdot 7 + 1 = 15$
5	$2 \cdot 15 + 1 = 31$
6	$2 \cdot 31 + 1 = 63$

(Informatiker

erkennen

Zweierpotenzen 😊)

2. Lösung raten

(manchmal schwierig!!)

Oft hilfreich: einsetzen!

$$T(n) = 2T(n-1) + 1$$

Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0
1	$2 \cdot 0 + 1 = 1$
2	$2 \cdot 1 + 1 = 3$
3	$2 \cdot 3 + 1 = 7$
4	$2 \cdot 7 + 1 = 15$
5	$2 \cdot 15 + 1 = 31$
6	$2 \cdot 31 + 1 = 63$

(Informatiker

erkennen

Zweierpotenzen 😊)

2. Lösung raten

(manchmal schwierig!!)

Oft hilfreich: einsetzen!

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \end{aligned}$$

Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0
1	$2 \cdot 0 + 1 = 1$
2	$2 \cdot 1 + 1 = 3$
3	$2 \cdot 3 + 1 = 7$
4	$2 \cdot 7 + 1 = 15$
5	$2 \cdot 15 + 1 = 31$
6	$2 \cdot 31 + 1 = 63$

(Informatiker

erkennen

Zweierpotenzen 😊)

2. Lösung raten

(manchmal schwierig!!)

Oft hilfreich: einsetzen!

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \\ &= 2^2 T(n-2) + 2 + 1 \end{aligned}$$

Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0
1	$2 \cdot 0 + 1 = 1$
2	$2 \cdot 1 + 1 = 3$
3	$2 \cdot 3 + 1 = 7$
4	$2 \cdot 7 + 1 = 15$
5	$2 \cdot 15 + 1 = 31$
6	$2 \cdot 31 + 1 = 63$

(Informatiker

erkennen

Zweierpotenzen 😊)

2. Lösung raten

(manchmal schwierig!!)

Oft hilfreich: einsetzen!

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \\ &= 2^2 T(n-2) + 2 + 1 \\ &= 2^3 T(n-3) + 2^2 + 2 + 1 \end{aligned}$$

Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0
1	$2 \cdot 0 + 1 = 1$
2	$2 \cdot 1 + 1 = 3$
3	$2 \cdot 3 + 1 = 7$
4	$2 \cdot 7 + 1 = 15$
5	$2 \cdot 15 + 1 = 31$
6	$2 \cdot 31 + 1 = 63$

(Informatiker

erkennen

Zweierpotenzen 😊)

2. Lösung raten

(manchmal schwierig!!)

Oft hilfreich: einsetzen!

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \\ &= 2^2 T(n-2) + 2 + 1 \\ &= 2^3 T(n-3) + 2^2 + 2 + 1 \\ &= \cancel{2^n T(n-n)} + \\ &\quad 2^{n-1} + \dots + 2 + 1 \end{aligned}$$

Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0
1	$2 \cdot 0 + 1 = 1$
2	$2 \cdot 1 + 1 = 3$
3	$2 \cdot 3 + 1 = 7$
4	$2 \cdot 7 + 1 = 15$
5	$2 \cdot 15 + 1 = 31$
6	$2 \cdot 31 + 1 = 63$

(Informatiker
erkennen
Zweierpotenzen 😊)

2. Lösung raten

(manchmal schwierig!!)

Oft hilfreich: einsetzen!

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \\ &= 2^2 T(n-2) + 2 + 1 \\ &= 2^3 T(n-3) + 2^2 + 2 + 1 \\ &= 2^n T(n-n) + \\ &\quad 2^{n-1} + \dots + 2 + 1 \\ &= \sum_{i=0}^{n-1} 2^i \end{aligned}$$

Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0
1	$2 \cdot 0 + 1 = 1$
2	$2 \cdot 1 + 1 = 3$
3	$2 \cdot 3 + 1 = 7$
4	$2 \cdot 7 + 1 = 15$
5	$2 \cdot 15 + 1 = 31$
6	$2 \cdot 31 + 1 = 63$

(Informatiker
erkennen
Zweierpotenzen 😊)

2. Lösung raten

(manchmal schwierig!!)

Oft hilfreich: einsetzen!

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \\ &= 2^2 T(n-2) + 2 + 1 \\ &= 2^3 T(n-3) + 2^2 + 2 + 1 \\ &\hookrightarrow = 2^n T(n-n) + \\ &\quad 2^{n-1} + \dots + 2 + 1 \\ &= \sum_{i=0}^{n-1} 2^i = 2^n - 1 \end{aligned}$$

Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0
1	$2 \cdot 0 + 1 = 1$
2	$2 \cdot 1 + 1 = 3$
3	$2 \cdot 3 + 1 = 7$
4	$2 \cdot 7 + 1 = 15$
5	$2 \cdot 15 + 1 = 31$
6	$2 \cdot 31 + 1 = 63$

(Informatiker
erkennen
Zweierpotenzen 😊)

2. Lösung raten

(manchmal schwierig!!)

Oft hilfreich: einsetzen!

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \\ &= 2^2 T(n-2) + 2 + 1 \\ &= 2^3 T(n-3) + 2^2 + 2 + 1 \\ &= 2^n T(n-n) + \\ &\quad 2^{n-1} + \dots + 2 + 1 \\ &= \sum_{i=0}^{n-1} 2^i = 2^n - 1 \end{aligned}$$

↪ "educated guess":

$$T(n) = 2^n - 1$$

Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0
1	$2 \cdot 0 + 1 = 1$
2	$2 \cdot 1 + 1 = 3$
3	$2 \cdot 3 + 1 = 7$
4	$2 \cdot 7 + 1 = 15$
5	$2 \cdot 15 + 1 = 31$
6	$2 \cdot 31 + 1 = 63$

(Informatiker
erkennen
Zweierpotenzen 😊)

2. Lösung raten

(manchmal schwierig!!)

Oft hilfreich: einsetzen!

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \\ &= 2^2 T(n-2) + 2 + 1 \\ &= 2^3 T(n-3) + 2^2 + 2 + 1 \\ &= 2^n T(n-n) + \\ &\quad 2^{n-1} + \dots + 2 + 1 \\ &= \sum_{i=0}^{n-1} 2^i = 2^n - 1 \end{aligned}$$

↪ "educated guess":

$$T(n) = 2^n - 1$$

3. Per Induktion beweisen

Lemma 3.3 (Türme von Hanoi)

$$T(n) = 2^n - 1.$$

Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0
1	$2 \cdot 0 + 1 = 1$
2	$2 \cdot 1 + 1 = 3$
3	$2 \cdot 3 + 1 = 7$
4	$2 \cdot 7 + 1 = 15$
5	$2 \cdot 15 + 1 = 31$
6	$2 \cdot 31 + 1 = 63$

(Informatiker
erkennen
Zweierpotenzen 😊)

2. Lösung raten

(manchmal schwierig!!)

Oft hilfreich: einsetzen!

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \\ &= 2^2 T(n-2) + 2 + 1 \\ &= 2^3 T(n-3) + 2^2 + 2 + 1 \\ &= 2^n T(n-n) + \\ &\quad 2^{n-1} + \dots + 2 + 1 \\ &= \sum_{i=0}^{n-1} 2^i = 2^n - 1 \end{aligned}$$

↪ "educated guess":

$$T(n) = 2^n - 1$$

3. Per Induktion beweisen

Lemma 3.3 (Türme von Hanoi)

$$T(n) = 2^n - 1.$$

Beweis:

Per Induktion über n .



Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0
1	$2 \cdot 0 + 1 = 1$
2	$2 \cdot 1 + 1 = 3$
3	$2 \cdot 3 + 1 = 7$
4	$2 \cdot 7 + 1 = 15$
5	$2 \cdot 15 + 1 = 31$
6	$2 \cdot 31 + 1 = 63$

(Informatiker
erkennen
Zweierpotenzen 😊)

2. Lösung raten

(manchmal schwierig!!)

Oft hilfreich: einsetzen!

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \\ &= 2^2 T(n-2) + 2 + 1 \\ &= 2^3 T(n-3) + 2^2 + 2 + 1 \\ &= 2^n T(n-n) + \\ &\quad 2^{n-1} + \dots + 2 + 1 \\ &= \sum_{i=0}^{n-1} 2^i = 2^n - 1 \end{aligned}$$

↪ "educated guess":

$$T(n) = 2^n - 1$$

3. Per Induktion beweisen

Lemma 3.3 (Türme von Hanoi)

$$T(n) = 2^n - 1. \quad \text{für alle } n \in \mathbb{N}_0$$

Beweis:

Per Induktion über n .

► Induktionsanfang (IA): $n = 0$

$$T(0) = 0$$



Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0
1	$2 \cdot 0 + 1 = 1$
2	$2 \cdot 1 + 1 = 3$
3	$2 \cdot 3 + 1 = 7$
4	$2 \cdot 7 + 1 = 15$
5	$2 \cdot 15 + 1 = 31$
6	$2 \cdot 31 + 1 = 63$

(Informatiker
erkennen
Zweierpotenzen 😊)

2. Lösung raten

(manchmal schwierig!!)

Oft hilfreich: einsetzen!

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \\ &= 2^2 T(n-2) + 2 + 1 \\ &= 2^3 T(n-3) + 2^2 + 2 + 1 \\ &= 2^n T(n-n) + \\ &\quad 2^{n-1} + \dots + 2 + 1 \\ &= \sum_{i=0}^{n-1} 2^i = 2^n - 1 \end{aligned}$$

↪ "educated guess":

$$T(n) = 2^n - 1$$

3. Per Induktion beweisen

Lemma 3.3 (Türme von Hanoi)

$$T(n) = 2^n - 1.$$

Beweis:

Per Induktion über n .

► Induktionsanfang (IA): $n = 0$

$$T(0) = 0 = 2^0 - 1 \quad \checkmark$$

Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0
1	$2 \cdot 0 + 1 = 1$
2	$2 \cdot 1 + 1 = 3$
3	$2 \cdot 3 + 1 = 7$
4	$2 \cdot 7 + 1 = 15$
5	$2 \cdot 15 + 1 = 31$
6	$2 \cdot 31 + 1 = 63$

(Informatiker
erkennen
Zweierpotenzen 😊)

2. Lösung raten

(manchmal schwierig!!)

Oft hilfreich: einsetzen!

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \\ &= 2^2 T(n-2) + 2 + 1 \\ &= 2^3 T(n-3) + 2^2 + 2 + 1 \\ &= 2^n T(n-n) + \\ &\quad 2^{n-1} + \dots + 2 + 1 \\ &= \sum_{i=0}^{n-1} 2^i = 2^n - 1 \end{aligned}$$

↪ "educated guess":

$$T(n) = 2^n - 1$$

3. Per Induktion beweisen

Lemma 3.3 (Türme von Hanoi)

$$T(n) = 2^n - 1.$$

Beweis:

Per Induktion über n .

- ▶ Induktionsanfang (IA): $n = 0$
 $T(0) = 0 = 2^0 - 1$ ✓
- ▶ Induktionsvoraussetzung (IV):
 $\forall n' \leq n : T(n') = 2^{n'} - 1$

Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0
1	$2 \cdot 0 + 1 = 1$
2	$2 \cdot 1 + 1 = 3$
3	$2 \cdot 3 + 1 = 7$
4	$2 \cdot 7 + 1 = 15$
5	$2 \cdot 15 + 1 = 31$
6	$2 \cdot 31 + 1 = 63$

(Informatiker
erkennen
Zweierpotenzen 😊)

2. Lösung raten

(manchmal schwierig!!)

Oft hilfreich: einsetzen!

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \\ &= 2^2 T(n-2) + 2 + 1 \\ &= 2^3 T(n-3) + 2^2 + 2 + 1 \\ &= 2^n T(n-n) + \\ &\quad 2^{n-1} + \dots + 2 + 1 \\ &= \sum_{i=0}^{n-1} 2^i = 2^n - 1 \end{aligned}$$

↪ "educated guess":

$$T(n) = 2^n - 1$$

3. Per Induktion beweisen

Lemma 3.3 (Türme von Hanoi)

$$T(n) = 2^n - 1.$$

Beweis:

Per Induktion über n .

► Induktionsanfang (IA): $n = 0$

$$T(0) = 0 = 2^0 - 1 \quad \checkmark$$

► Induktionsvoraussetzung (IV):

$$\forall n' \leq n : T(n') = 2^{n'} - 1$$

► Induktionsschritt (IS): $n \rightarrow n + 1$

$$T(n+1) = 2T(n) + 1$$

$$2^{n+1} - 1$$

Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0
1	$2 \cdot 0 + 1 = 1$
2	$2 \cdot 1 + 1 = 3$
3	$2 \cdot 3 + 1 = 7$
4	$2 \cdot 7 + 1 = 15$
5	$2 \cdot 15 + 1 = 31$
6	$2 \cdot 31 + 1 = 63$

(Informatiker
erkennen
Zweierpotenzen 😊)

2. Lösung raten

(manchmal schwierig!!)

Oft hilfreich: einsetzen!

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \\ &= 2^2 T(n-2) + 2 + 1 \\ &= 2^3 T(n-3) + 2^2 + 2 + 1 \\ &= 2^n T(n-n) + \\ &\quad 2^{n-1} + \dots + 2 + 1 \\ &= \sum_{i=0}^{n-1} 2^i = 2^n - 1 \end{aligned}$$

↪ "educated guess":

$$T(n) = 2^n - 1$$

3. Per Induktion beweisen

Lemma 3.3 (Türme von Hanoi)

$$T(n) = 2^n - 1.$$

Beweis:

Per Induktion über n .

- ▶ Induktionsanfang (IA): $n = 0$

$$T(0) = 0 = 2^0 - 1 \quad \checkmark$$

- ▶ Induktionsvoraussetzung (IV):

$$\forall n' \leq n : T(n') = 2^{n'} - 1$$

- ▶ Induktionsschritt (IS): $n \rightarrow n + 1$

$$\begin{aligned} T(n+1) &= 2T(n) + 1 \\ &\stackrel{\text{IV}}{=} 2(2^n - 1) + 1 \end{aligned}$$

Rekursion und Rekurrenzen

$$T(0) = 0, \quad T(n) = 2T(n-1) + 1 \quad (n \geq 1)$$

1. Werte für kleine n

n	$T(n)$
0	0
1	$2 \cdot 0 + 1 = 1$
2	$2 \cdot 1 + 1 = 3$
3	$2 \cdot 3 + 1 = 7$
4	$2 \cdot 7 + 1 = 15$
5	$2 \cdot 15 + 1 = 31$
6	$2 \cdot 31 + 1 = 63$

(Informatiker
erkennen
Zweierpotenzen 😊)

2. Lösung raten

(manchmal schwierig!!)

Oft hilfreich: einsetzen!

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \\ &= 2^2 T(n-2) + 2 + 1 \\ &= 2^3 T(n-3) + 2^2 + 2 + 1 \\ &= 2^n T(n-n) + \\ &\quad 2^{n-1} + \dots + 2 + 1 \\ &= \sum_{i=0}^{n-1} 2^i = 2^n - 1 \end{aligned}$$

↪ "educated guess":

$$T(n) = 2^n - 1$$

3. Per Induktion beweisen

Lemma 3.3 (Türme von Hanoi)

$$T(n) = 2^n - 1.$$

Beweis:

Per Induktion über n .

- ▶ Induktionsanfang (IA): $n = 0$

$$T(0) = 0 = 2^0 - 1 \quad \checkmark$$

- ▶ Induktionsvoraussetzung (IV):

$$\forall n' \leq n : T(n') = 2^{n'} - 1$$

- ▶ Induktionsschritt (IS): $n \rightarrow n + 1$

$$\begin{aligned} T(n+1) &= 2T(n) + 1 \\ &\stackrel{\text{IV}}{=} 2(2^n - 1) + 1 \\ &= 2^{n+1} - 2 + 1 \\ &= 2^{n+1} - 1 \end{aligned}$$

$$2^{64} \geq (2^{10})^6 = 10^{18}$$

3.6 Das Master-Theorem

Clicker Question



Wie funktioniert Mergesort?

- A** Partitioniere Elemente um Median, sortiere dann rekursiv kleinere und größere Elemente.
- B** Sortiere rekursiv linke und rechte Hälfte und kombiniere die sortierten Hälften.
- C** Bilde sortierten Präfix, füge wiederholt nächstes Element an passende Stelle hinzu.
- D** Wähle wiederholt 2 Elemente und vertausche sie, falls sie nicht in der richtigen Reihenfolge stehen.
- E** Was fürn Ding?



→ sli.do/cs210

Clicker Question

Wie funktioniert Mergesort?



- ~~A Partitioniere Elemente um Median, sortiere dann rekursiv kleinere und größere Elemente.~~
- B Sortiere rekursiv linke und rechte Hälfte und kombiniere die sortierten Hälften. ✓
- ~~C Bilde sortierten Präfix, füge wiederholt nächstes Element an passende Stelle hinzu.~~
- ~~D Wähle wiederholt 2 Elemente und vertausche sie, falls sie nicht in der richtigen Reihenfolge stehen.~~
- ~~E Was fürn Ding?~~



→ sli.do/cs210

Divide-and-Conquer Rekursionen

Wir werden eine Reihe von Algorithmen sehen die nach dem *Divide-&-Conquer-Prinzip* („Teile und herrsche“) arbeiten.

Divide-and-Conquer Rekursionen

Wir werden eine Reihe von Algorithmen sehen die nach dem *Divide-&Conquer-Prinzip* („Teile und herrsche“) arbeiten.

Klassisches Beispiel: *Mergesort* (↔ Unit 7)

0. Falls Eingabegröße $n \leq 1$, **fertig**.
1. *Teile* die Eingabe der Größe n in **2** Teile der Größe $\approx \frac{n}{2}$ auf.
2. Löse **rekursiv** die **2** Teilprobleme der Größe $\frac{n}{2}$.
3. **Kombiniere** die Lösungen der Teilprobleme zu einer Gesamtlösung.

(Wir kümmern uns hier (noch) nicht darum, wie/was das algorithmisch tut!)

Divide-and-Conquer Rekursionen

Wir werden eine Reihe von Algorithmen sehen die nach dem *Divide-&Conquer-Prinzip* („Teile und herrsche“) arbeiten.

Klassisches Beispiel: *Mergesort* (↔ Unit 7)

0. Falls Eingabegröße $n \leq 1$, **fertig**.
1. *Teile* die Eingabe der Größe n in **2** Teile der Größe $\approx \frac{n}{2}$ auf.
2. Löse **rekursiv** die **2** Teilprobleme der Größe $\frac{n}{2}$.
3. **Kombiniere** die Lösungen der Teilprobleme zu einer Gesamtlösung.

(Wir kümmern uns hier (noch) nicht darum, wie/was das algorithmisch tut!)

Aber wir wollen Verfahren kennenlernen, wie man solche Algorithmen analysieren kann.

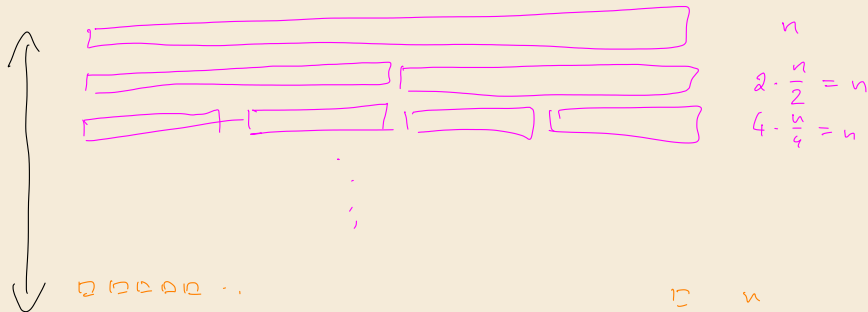
Laufzeit von Mergesort

$$T(n) = \begin{cases} 0 & n \leq 1 \\ 2 \cdot T(n/2) + n & n \geq 2 \end{cases}$$

Für $n = 2^k, k \in \mathbb{N}_0$

Rekursionsbaum-Methode für Mergesort

$$T(n) = \begin{cases} 0 & n \leq 1 \\ 2 \cdot T(n/2) + n & n \geq 2 \end{cases}$$



Die Master-Methode

- ▶ Allgemeines D&C-Verfahren
 - ▶ a rekursive Aufrufe (für Konstante $a > 0$)
 - ▶ Teilprobleme der Größe n/b (für Konstante $b > 1$)
 - ▶ nicht-rekursiver "conquer" Aufwand $f(n)$ (für Funktion $f : \mathbb{R} \rightarrow \mathbb{R}$)
 - ▶ Basisfall-Aufwand d (some constant $d > 0$)

Die Master-Methode

► Allgemeines D&C-Verfahren

- a rekursive Aufrufe (für Konstante $a > 0$)
- Teilprobleme der Größe n/b (für Konstante $b > 1$)
- nicht-rekursiver "conquer" Aufwand $f(n)$ (für Funktion $f : \mathbb{R} \rightarrow \mathbb{R}$)
- Basisfall-Aufwand d (some constant $d > 0$)

$$T(n) = \begin{cases} 0 & n \leq 1 \\ 2 \cdot T(n/2) + n & n \geq 2 \end{cases}$$

↪ Laufzeit $T(n)$ erfüllt Rekursion

$$T(n) = \begin{cases} a \cdot T\left(\frac{n}{b}\right) + f(n) & n > 1 \\ d & n \leq 1 \end{cases}$$

$(n = b^k, k \in \mathbb{N}_0)$

Eine Beispielrekursion

$$f(n) \in O(n^2) \wedge \Omega(n^2)$$

$$\Rightarrow \exists c_1, c_2 \in \mathbb{R}_{>0} : c_1 \cdot n^2 \leq f(n) \leq c_2 \cdot n^2$$

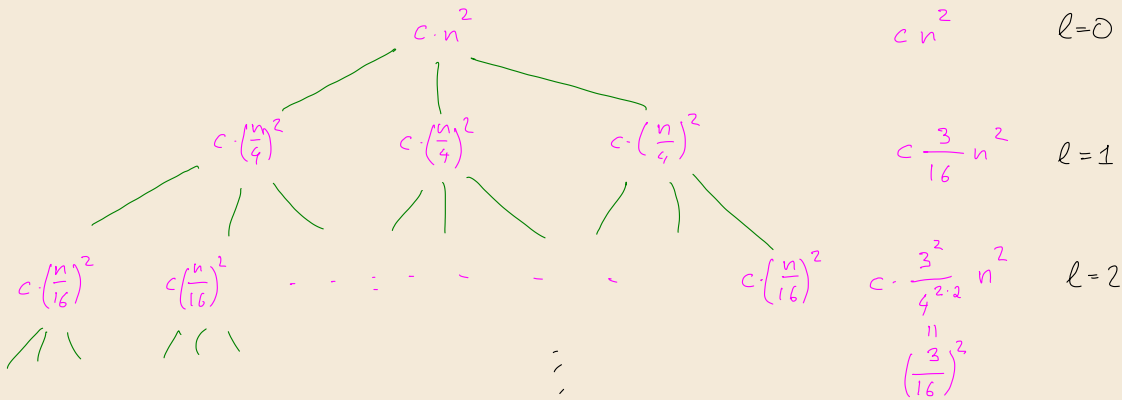
für $n \geq n_0$

$$T(n) = 3T\left(\frac{n}{4}\right) + \Theta(n^2)$$

$$T(1) = 1$$

Rekursionsbaum für $T(n)$ (obere Schranke!)

$$\Theta(n^2) \rightsquigarrow \frac{c \cdot n^2}{c n^2}$$



$$T(n) \leq \sum_{l=0}^{\log_4(n)} \left(\frac{3}{16}\right)^l c n^2 + n^{\log_4(3)}$$

$$= c \cdot n^2 \sum_{l=0}^{\log_4(n)} \left(\frac{3}{16}\right)^l + n^{\log_4(3)}$$

$$< c n^2 \cdot \underbrace{\sum_{l=0}^{\infty} \left(\frac{3}{16}\right)^l}_{\text{"}} + \text{"}$$

$$\frac{1}{1 - \frac{3}{16}} = \frac{16}{13}$$

$$= O(n^2)$$

Die Master-Methode

- ▶ Allgemeines D&C-Verfahren
 - ▶ a rekursive Aufrufe (für Konstante $a > 0$)
 - ▶ Teilprobleme der Größe n/b (für Konstante $b > 1$)
 - ▶ nicht-rekursiver "conquer" Aufwand $f(n)$ (für Funktion $f : \mathbb{R} \rightarrow \mathbb{R}$)
 - ▶ Basisfall-Aufwand d (some constant $d > 0$)

↪ Laufzeit $T(n)$ erfüllt Rekursion

$$T(n) = \begin{cases} a \cdot T\left(\frac{n}{b}\right) + f(n) & n > 1 \\ d & n \leq 1 \end{cases}$$

$(n = b^k, k \in \mathbb{N}_0)$

Theorem 3.4 (Master Theorem)

Mit $c := \log_b(a)$, gilt für die obige Rekursion:

(a) $T(n) = \Theta(n^c)$ falls $f(n) = O(\underline{n^{c-\varepsilon}})$ für konstantes $\varepsilon > 0$. #Brekur dominant

(b) $T(n) = \Theta(n^c \log n)$ falls $f(n) = \Theta(n^c)$.

(c) $T(n) = \Theta(f(n))$ falls $f(n) = \Omega(n^{c+\varepsilon})$ für konstantes $\varepsilon > 0$ **und** f erfüllt die
Regularitätsbedingung $\exists n_0, \alpha < 1 \forall n \geq n_0 : a \cdot f\left(\frac{n}{b}\right) \leq \alpha f(n)$.

erster Aufruf dominant

Regularitätsbedingung

Fall (c) braucht leider die technische Regularitätsbedingung.

Aber, für einfache Funktionen f gilt diese immer.

Lemma 3.5 (Regularitätsbedingung)

Für $f(n) = n^\beta \lg^\gamma(n)$ mit Konstanten $\beta > \log_b(a)$ und γ gilt die Regularitätsbedingung. ◀

Was ist mit Runden?

Für allgemeines n sind die Teilproblem $\lfloor \frac{n}{b} \rfloor$ bzw. $\lceil \frac{n}{b} \rceil$. *Macht das was?*

Was ist mit Runden?

Für allgemeines n sind die Teilproblem $\lfloor \frac{n}{b} \rfloor$ bzw. $\lceil \frac{n}{b} \rceil$. Macht das was?

Im Allgemeinen Ja, aber:

Lemma 3.6 (Polynomial-growth master method)

Falls f die *polynomial-growth condition* erfüllt, ist die Θ -Klasse der Lösung der D&C-Rekursion mit oder ohne Runden gleich. ◀

► $f : \mathbb{R}_{>0} \rightarrow \mathbb{R}$ erfüllt die *polynomial-growth condition*, falls

$$\exists n_0 \forall C \geq 1 \exists D > 1 \quad \forall n \geq n_0 \forall c \in [1, C] : \frac{1}{D} f(n) \leq f(cn) \leq D f(n)$$

Was ist mit Runden?

Für allgemeines n sind die Teilproblem $\lfloor \frac{n}{b} \rfloor$ bzw. $\lceil \frac{n}{b} \rceil$. Macht das was?

Im Allgemeinen Ja, aber:

Lemma 3.6 (Polynomial-growth master method)

Falls f die *polynomial-growth condition* erfüllt, ist die Θ -Klasse der Lösung der D&C-Rekursion mit oder ohne Runden gleich. ◀

- $f : \mathbb{R}_{>0} \rightarrow \mathbb{R}$ erfüllt die *polynomial-growth condition*, falls

✂ exam

$$\exists n_0 \forall C \geq 1 \exists D > 1 \quad \forall n \geq n_0 \forall c \in [1, C] : \frac{1}{D} f(n) \leq f(cn) \leq D f(n)$$

- Intuition n um (bis zu) konstanten Faktor C größer machen
verändert Funktionswert auch höchstens um konstanten Faktor $D = D(C)$
(für hinreichend große n)

Null erlaubt!

- Insbesondere: $f(n) = \Theta(n^\beta \log^\gamma(n) \log^\delta(n))$ für Konstanten β, γ, δ
erfüllt *polynomial-growth condition*

A Rigorous and Stronger Meta Theorem (≠ exam)

Theorem 3.7 (Roura's Discrete Master Theorem)

Let $T(n)$ be recursively defined as

$$T(n) = \begin{cases} b_n & 0 \leq n < n_0, \\ f(n) + \sum_{d=1}^D a_d \cdot T\left(\frac{n}{b_d} + r_{n,d}\right) & n \geq n_0, \end{cases}$$

where $D \in \mathbb{N}$, $a_d > 0$, $b_d > 1$, for $d = 1, \dots, D$ are constants, functions $r_{n,d}$ satisfy $|r_{n,d}| = O(1)$ as $n \rightarrow \infty$, and function $f(n)$ satisfies $f(n) \sim B \cdot n^\alpha (\ln n)^\gamma$ for constants $B > 0$, α, γ .

Set $H = 1 - \sum_{d=1}^D a_d (1/b_d)^\alpha$; then we have:

- (a) If $H < 0$, then $T(n) = O(n^{\tilde{\alpha}})$, for $\tilde{\alpha}$ the unique value of α that would make $H = 0$.
- (b) If $H = 0$ and $\gamma > -1$, then $T(n) \sim f(n) \ln(n) / \tilde{H}$ with constant $\tilde{H} = (\gamma + 1) \sum_{d=1}^D a_d b_d^{-\alpha} \ln(b_d)$.
- (c) If $H = 0$ and $\gamma = -1$, then $T(n) \sim f(n) \ln(n) \ln(\ln(n)) / \hat{H}$ with constant $\hat{H} = \sum_{d=1}^D a_d b_d^{-\alpha} \ln(b_d)$.
- (d) If $H = 0$ and $\gamma < -1$, then $T(n) = O(n^\alpha)$.
- (e) If $H > 0$, then $T(n) \sim f(n)/H$.

3.7 Grundlagen Stochastik

Grundbegriffe der Stochastik

- ▶ Betrachten *Zufallsexperimente*, deren Ergebnisse von zufällig wirkenden Faktoren abhängen (z.B. Würfeln)
- ▶ Ein *Elementarereignis* ω ist ein unmittelbar mögliches Ergebnis des betrachteten Zufallsexperiments
Menge möglicher Elementarereignisse ist der Ereignisraum Ω
- ▶ (für abzählbar unendliche oder endliche Ω)
 $A \subseteq \Omega$ ein Ereignis (z.B. Würfel zeigt gerade Zahl)

$$\text{'Würfel zeigt gerade Zahl'} = \left\{ \begin{array}{|c|} \hline \cdot \\ \hline \cdot \\ \hline \cdot \\ \hline \end{array} , \begin{array}{|c|} \hline \cdot \cdot \\ \hline \cdot \cdot \\ \hline \cdot \cdot \\ \hline \end{array} , \begin{array}{|c|} \hline \cdot \cdot \cdot \\ \hline \cdot \cdot \cdot \\ \hline \cdot \cdot \cdot \\ \hline \end{array} \right\}$$

Grundbegriffe der Stochastik

- ▶ Betrachten *Zufallsexperimente*, deren Ergebnisse von zufällig wirkenden Faktoren abhängen (z.B. Würfeln)
- ▶ Ein *Elementarereignis* ω ist ein unmittelbar mögliches Ergebnis des betrachteten Zufallsexperiments
Menge möglicher Elementarereignisse ist der *Ereignisraum* Ω
- ▶ (für abzählbar unendliche oder endliche Ω)
 $A \subseteq \Omega$ ein *Ereignis* (z.B. Würfel zeigt gerade Zahl)
- ▶ Wir ordnen A die *Wahrscheinlichkeit* $\mathbb{P}[A]$ zu

1. $\mathbb{P}[A] \geq 0$ für alle $A \subset \Omega$.
2. $\mathbb{P}[\Omega] = 1$.
3. Für A_1, A_2, \dots **disjunkte** Ereignisse in Ω , ist

$$\mathbb{P}[A_1 \cup A_2 \cup \dots] = \sum_{i=1}^{\infty} \mathbb{P}[A_i]$$

(*Ver-oder-ung* der die Ereignisse definierenden Bedingungen).

Grundbegriffe der Stochastik [2]

- ▶ Die *bedingte Wahrscheinlichkeit* für Ereignis A , unter der Voraussetzung, dass B bereits eingetreten ist, ist definiert als

$$\mathbb{P}[A | B] := \frac{\mathbb{P}[A \cap B]}{\mathbb{P}[B]}, \quad \mathbb{P}[B] > 0.$$

Grundbegriffe der Stochastik [2]

- ▶ Die *bedingte Wahrscheinlichkeit* für Ereignis A , unter der Voraussetzung, dass B bereits eingetreten ist, ist definiert als

$$\mathbb{P}[A | B] := \frac{\mathbb{P}[A \cap B]}{\mathbb{P}[B]}, \quad \mathbb{P}[B] > 0.$$

- ▶ A und B sind (stochastisch) unabhängig, falls $\mathbb{P}[A | B] = \mathbb{P}[A]$
- ▶ Damit gilt für **unabhängige** Ereignisse $\mathbb{P}[A \cap B] = \mathbb{P}[A] \cdot \mathbb{P}[B]$
(anschaulich: *Ver-und-ung* der Ereignisse entspricht)

Zufallsvariablen

- ▶ Eine *Zufallsvariable* ist eine Funktion $X : \Omega \rightarrow \mathbb{R}$
Schreiben X statt $X(\omega)$.

Ω Ereignisraum (Physik)

$A \subseteq \Omega$ Ereignis (was uns interessiert)

Zufallsvariablen

- ▶ Eine *Zufallsvariable* ist eine Funktion $X : \Omega \rightarrow \mathbb{R}$
Schreiben X statt $X(\omega)$.

- ▶ Ist der Wertebereich von X gleich $\{a_i : i \in \mathcal{J}\}$, dann ist

$$\mathbb{P}[X = a_i] = \sum_{\substack{\omega \in \Omega \\ X(\omega) = a_i}} \mathbb{P}[\omega], \quad i \in \mathcal{J},$$

die zugehörige *Wahrscheinlichkeitsverteilung* von X .

- ▶ Für \mathcal{J} endlich bzw. abzählbar unendlich heißt X *diskrete* Zufallsvariable.

Zufallsvariablen

- ▶ Eine *Zufallsvariable* ist eine Funktion $X : \Omega \rightarrow \mathbb{R}$

Schreiben X statt $X(\omega)$.

- ▶ Ist der Wertebereich von X gleich $\{a_i : i \in \mathcal{J}\}$, dann ist

$$\mathbb{P}[X = a_i] = \sum_{\substack{\omega \in \Omega \\ X(\omega) = a_i}} \mathbb{P}[\omega], \quad i \in \mathcal{J},$$

die zugehörige *Wahrscheinlichkeitsverteilung* von X .

- ▶ Für \mathcal{J} endlich bzw. abzählbar unendlich heißt X *diskrete* Zufallsvariable.

▶ Beispiele

- ▶ **Uniforme Verteilung:** für $|\mathcal{J}| = n < \infty$ fairer Würfel $n=6$

$$\mathbb{P}[X = a_i] = \frac{1}{n}$$

- ▶ **Bernoulli-Verteilung:** $p \in [0, 1]$ Münzwurf (unfair)

$$\mathcal{J} = \{0, 1\}, \quad \mathbb{P}[X = 1] = p, \mathbb{P}[X = 0] = 1 - p,$$

- ▶ **Binomial-Verteilung:**

Summe von n unabhängigen Bernoulli Variablen mit Parameter p

Beispiel Zufallsvariablen

Gegeben ein **roter** und ein **grüner** fairer Würfel.

Beispiel Zufallsvariablen

Gegeben ein **roter** und ein **grüner** fairer Würfel.

$$\rightsquigarrow \text{Ereignisraum } \Omega = \{ \color{red}\blacksquare, \color{red}\blacksquare, \color{red}\blacksquare, \color{red}\blacksquare, \color{red}\blacksquare, \color{red}\blacksquare \} \times \{ \color{green}\blacksquare, \color{green}\blacksquare, \color{green}\blacksquare, \color{green}\blacksquare, \color{green}\blacksquare, \color{green}\blacksquare \}$$

Beispiel Zufallsvariablen

Gegeben ein **roter** und ein **grüner** fairer Würfel.

$$\begin{aligned} \rightsquigarrow \text{Ereignisraum } \Omega &= \{ \text{1}, \text{2}, \text{3}, \text{4}, \text{5}, \text{6} \} \times \{ \text{1}, \text{2}, \text{3}, \text{4}, \text{5}, \text{6} \} \\ &= \{ \text{11}, \text{12}, \text{13}, \text{14}, \text{15}, \text{16}, \text{21}, \text{22}, \text{23}, \text{24}, \text{25}, \text{26}, \text{31}, \text{32}, \text{33}, \text{34}, \text{35}, \text{36}, \\ &\quad \text{41}, \text{42}, \text{43}, \text{44}, \text{45}, \text{46}, \text{51}, \text{52}, \text{53}, \text{54}, \text{55}, \text{56}, \text{61}, \text{62}, \text{63}, \text{64}, \text{65}, \text{66} \} \end{aligned}$$

Beispiel Zufallsvariablen

Gegeben ein **roter** und ein **grüner** fairer Würfel.

$$\begin{aligned}\rightsquigarrow \text{Ereignisraum } \Omega &= \{ \text{1}, \text{2}, \text{3}, \text{4}, \text{5}, \text{6} \} \times \{ \text{1}, \text{2}, \text{3}, \text{4}, \text{5}, \text{6} \} \\ &= \{ \text{11}, \text{12}, \text{13}, \text{14}, \text{15}, \text{16}, \text{21}, \text{22}, \text{23}, \text{24}, \text{25}, \text{26}, \text{31}, \text{32}, \text{33}, \text{34}, \text{35}, \text{36}, \\ &\quad \text{41}, \text{42}, \text{43}, \text{44}, \text{45}, \text{46}, \text{51}, \text{52}, \text{53}, \text{54}, \text{55}, \text{56}, \text{61}, \text{62}, \text{63}, \text{64}, \text{65}, \text{66} \}\end{aligned}$$

- ▶ Zufallsvariable $X =$ Anzahl Augen des roten Würfels
formal $X(\text{12}) = 3$ etc.
- ▶ analog $Y =$ Anzahl Augen des grünen Würfels
 $Y(\text{12}) = 5$

Beispiel Zufallsvariablen

Gegeben ein **roter** und ein **grüner** fairer Würfel.

$$\begin{aligned}\rightsquigarrow \text{Ereignisraum } \Omega &= \{ \text{1}, \text{2}, \text{3}, \text{4}, \text{5}, \text{6} \} \times \{ \text{1}, \text{2}, \text{3}, \text{4}, \text{5}, \text{6} \} \\ &= \{ \text{11}, \text{12}, \text{13}, \text{14}, \text{15}, \text{16}, \text{21}, \text{22}, \text{23}, \text{24}, \text{25}, \text{26}, \text{31}, \text{32}, \text{33}, \text{34}, \text{35}, \text{36}, \\ &\quad \text{41}, \text{42}, \text{43}, \text{44}, \text{45}, \text{46}, \text{51}, \text{52}, \text{53}, \text{54}, \text{55}, \text{56}, \text{61}, \text{62}, \text{63}, \text{64}, \text{65}, \text{66} \}\end{aligned}$$

- ▶ Zufallsvariable X = Anzahl Augen des roten Würfels
formal $X(\text{12}) = 3$ etc.
- ▶ analog Y = Anzahl Augen des grünen Würfels
 $Y(\text{12}) = 5$
- ▶ können mit $X + Y$ die Summe der gewürfelten Zahlen bilden
 $(X + Y)(\text{12}) = 8$

\rightsquigarrow Durch Zufallsvariablen können wir mit Ergebnissen von Zufallsexperimenten *rechnen!*

Beispiel Zufallsvariablen

Gegeben ein **roter** und ein **grüner** fairer Würfel.

$$\begin{aligned}\rightsquigarrow \text{Ereignisraum } \Omega &= \{ \text{red 1, red 2, red 3, red 4, red 5, red 6} \} \times \{ \text{green 1, green 2, green 3, green 4, green 5, green 6} \} \\ &= \{ \text{red 1 green 1, red 1 green 2, red 1 green 3, red 1 green 4, red 1 green 5, red 1 green 6, red 2 green 1, red 2 green 2, red 2 green 3, red 2 green 4, red 2 green 5, red 2 green 6, red 3 green 1, red 3 green 2, red 3 green 3, red 3 green 4, red 3 green 5, red 3 green 6, red 4 green 1, red 4 green 2, red 4 green 3, red 4 green 4, red 4 green 5, red 4 green 6, red 5 green 1, red 5 green 2, red 5 green 3, red 5 green 4, red 5 green 5, red 5 green 6, red 6 green 1, red 6 green 2, red 6 green 3, red 6 green 4, red 6 green 5, red 6 green 6} \}\end{aligned}$$

- ▶ Zufallsvariable X = Anzahl Augen des roten Würfels
formal $X(\text{red 3 green 5}) = 3$ etc.
- ▶ analog Y = Anzahl Augen des grünen Würfels
 $Y(\text{red 3 green 5}) = 5$
- ▶ können mit $X + Y$ die Summe der gewürfelten Zahlen bilden
 $(X + Y)(\text{red 3 green 5}) = 8$

\rightsquigarrow Durch Zufallsvariablen können wir mit Ergebnissen von Zufallsexperimenten *rechnen!*

$$\text{▶ } \mathbb{P}[X + Y = 8] = \frac{|\{ \text{red 2 green 6, red 3 green 5, red 4 green 4, red 5 green 3, red 6 green 2} \}|}{36} = \frac{5}{36}$$

Erwartungswert

Definition 3.8

Ist X eine diskrete Zufallsvariable mit Werten in $\mathcal{X} \subset \mathbb{R}$ dann ist der *Erwartungswert von X*

definiert als $\mathbb{E}[X] = \sum_{x \in \mathcal{X}} x \cdot \mathbb{P}[X = x]$.



Erwartungswert

Definition 3.8

Ist X eine diskrete Zufallsvariable mit Werten in $\mathcal{X} \subset \mathbb{R}$ dann ist der *Erwartungswert von X*

definiert als $\mathbb{E}[X] = \sum_{x \in \mathcal{X}} x \cdot \mathbb{P}[X = x]$. ◀

Lemma 3.9 (Linearität des Erwartungswerts)

Für reellen Zahlen a, b und c und Zufallsvariablen X und Y ist

$$\mathbb{E}[a \cdot X + b \cdot Y + c] = a \cdot \mathbb{E}[X] + b \cdot \mathbb{E}[Y] + c. \quad \blacktriangleleft$$

Linearität gilt für beliebige X und Y , auch stochastisch abhängige.

$$\mathbb{E}[X] = \sum_{x \in \{1..6\}} x \cdot \mathbb{P}[X=x] = \frac{1}{6} \cdot \sum_{x=1}^6 x = \frac{1}{6} \cdot \frac{6 \cdot 7}{2} = 3,5$$

Clicker Question

Was ist der Erwartungswert $\mathbb{E}[X + 1]$ wenn X eine Binomial-Verteilung hat mit Parametern $n \in \mathbb{N}_{\geq 1}$ und $p \in (0, 1)$?



A $X + 1$

B $X^2 + 2X + 1$

C np

D $np + 1$

E $(n + 1)p$

F Erwartungswert?

G Binomialverteilung?



→ sli.do/cs210

Clicker Question

Was ist der Erwartungswert $\mathbb{E}[X + 1]$ wenn X eine Binomial-Verteilung hat mit Parametern $n \in \mathbb{N}_{\geq 1}$ und $p \in (0, 1)$?



A ~~$X + 1$~~

B ~~$X^2 + 2X + 1$~~

C ~~np~~

D $np + 1$ ✓

E ~~$(n + 1)p$~~

F ~~Erwartungswert?~~

G ~~Binomialverteilung?~~



→ sli.do/cs210