

Logik

Aussagenlogik

Aussage Satz/Formel entweder wahr oder falsch; „-form“ bei zu wenig Infos.

Theoreme sind wahre Aussagen.

Junktoren

Negation $\neg A$ „Nicht“ (!, ~, \neg)

Konjunkt. $A \wedge B$ „und“ (&, \sqcap)

Disjunkt. $A \vee B$ „oder“ (||, \sqcup)

Implikat. $A \Rightarrow B$ „Wenn, dann“ / „B“ (\rightarrow , \Rightarrow)

$A \Rightarrow B$ „A hinreichend“

$B \Rightarrow A$ „A notwendig“

Äquiv. $A \Leftrightarrow B$ „Genau dann, wenn“ (\leftrightarrow , \equiv , $=$, \Leftrightarrow)

Wahrheitstabelle mit 2^n Zeilen für n Atome. Konstruktionssystematik: Frequenz pro Atom verdoppeln.

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \Rightarrow B$	$A \Leftrightarrow B$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

Äquivalente Formeln \Leftrightarrow		Bezeichnung
$A \wedge B$	$B \wedge A$	Kommutativ
$A \vee B$	$B \vee A$	
$A \wedge (B \wedge C)$	$(A \wedge B) \wedge C$	Assoziativ
$A \vee (B \vee C)$	$(A \vee B) \vee C$	
$A \wedge (B \vee C)$	$(A \wedge B) \vee (A \wedge C)$	Distributiv
$A \vee (B \wedge C)$	$(A \vee B) \wedge (A \vee C)$	
$A \wedge A$	A	Idempotenz
$A \vee A$	A	
$\neg \neg A$	A	Involution
$\neg(A \wedge B)$	$\neg A \vee \neg B$	
$\neg(A \vee B)$	$\neg A \wedge \neg B$	DE-MORGAN
$A \wedge (A \vee B)$	A	
$A \vee (A \wedge B)$	A	Absorption
$A \Rightarrow B$	$\neg A \vee B$	
$\neg(A \Rightarrow B)$	$A \wedge \neg B$	Elimination
$A \Leftrightarrow B$	$(A \Rightarrow B) \wedge (B \Rightarrow A)$	

Axiomatik

Axiome als wahr angenommene Aussagen; an Nützlichkeit gemessen. Anspruch, aber nach GÖDELS Unvollständigkeitssatz nicht möglich:

- Unabhängig
- Vollständig
- Widerspruchsfrei

Prädikatenlogik

Quantoren Innerhalb eines Universums:

Existenzq. \exists „Mind. eines“

Individuum $\exists!$ „Genau eines“

Allq. \forall „Für alle“

Quantitative Aussagen

Erfüllbar $\exists x F(x)$

Widerlegbar $\exists x \neg F(x)$

Tautologie $\top = \forall x F(x)$ (alle Schlussregeln)

Kontradiktion $\perp = \forall x \neg F(x)$



Klassische Tautologien	Bezeichnung
$A \vee \neg A$	Ausgeschlossenes Drittes
$A \wedge (A \Rightarrow B) \Rightarrow B$	Modus ponens
$(A \wedge B) \Rightarrow A$	Abschwächung
$A \Rightarrow (A \vee B)$	

Negation (DE-MORGAN)

$$\neg \exists x F(x) \Leftrightarrow \forall x \neg F(x)$$

$$\neg \forall x F(x) \Leftrightarrow \exists x \neg F(x)$$

Häufige Fehler

- $U = \emptyset^c$ nicht notwendig
- $\exists x(P(x) \Rightarrow Q(x)) \not\Leftrightarrow \exists x P(x)$
- $\neg \exists x \exists y P(x, y) \Leftrightarrow \forall x \neg \exists y P(x, y)$

Beweistechniken

Achtung: Aus falschen Aussagen können wahre *und* falsche Aussagen folgen.

Direkt $A \Rightarrow B$ Angenommen A , zeige B . Oder: Angenommen $\neg B$, zeige $\neg A$ (**Kontraposition**).

$$(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A)$$

Fallunters. Aufteilen, lösen, zusammenführen. O.B.d.A = „Ohne Beschränkung der Allgemeinheit“

Widerspruch $(\neg A \Rightarrow \perp) \Rightarrow A$ Angenommen $A \wedge \neg B$, zeige Kontradiktion. (Reductio ad absurdum)

Ring (Transitivität der Implikation)

$$A \Leftrightarrow B \Leftrightarrow C \Leftrightarrow \dots$$
$$\equiv A \Rightarrow B \Rightarrow C \Rightarrow \dots \Rightarrow A$$

Induktion $F(n) \quad \forall n \geq n_0 \in \mathbb{N}$

1. **Anfang:** Zeige $F(n_0)$.

2. **Schritt:** Angenommen $F(n)$ (Hypothese), zeige $F(n+1)$ (Behauptung).

Starke Induktion: Angenommen $F(k) \quad \forall n_0 \leq k \leq n \in \mathbb{N}$.

Häufige Fehler

- Nicht voraussetzen, was zu beweisen ist
- Äquiv. von Implikat. unterscheiden (Zweifelsfall immer Implikat.)

Naive Mengenlehre

$$f(1) = 0, r_{11} r_{12} r_{13} r_{14} \dots$$
$$f(2) = 0, r_{21} r_{22} r_{23} r_{24} \dots$$
$$f(3) = 0, r_{31} r_{32} r_{33} r_{34} \dots$$
$$f(4) = 0, r_{41} r_{42} r_{43} r_{44} \dots$$

\vdots

(CANTORS Diagonalargumente)

Mengen Zusammenfassung versch. Objekte „Elemente“.

Element $x \in M$ „enthält“

Leere M. $\emptyset = \{\}$

Universum U

Einschränkung $\{x \mid F(x)\}$

Relationen

Teilmenge $N \subseteq M$
 $\Leftrightarrow \forall n \in N : n \in M$ \odot

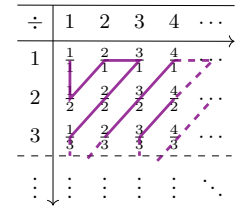
Gleichheit $M = N$
 $\Leftrightarrow M \subseteq N \wedge N \subseteq M$ \odot

Mächtigkeit

$$|M| \begin{cases} = n & \text{endlich} \\ \geq \infty & \text{unendlich} \end{cases}$$
$$= |N| \Leftrightarrow \exists f_{\text{bijekt.}} : M \rightarrow N$$

Abzählbar $\exists f_{\text{surj.}} : \mathbb{N} \rightarrow M$

- Endliche Mengen, $\emptyset, \mathbb{N}, \mathbb{Z}, \mathbb{Q}$
- $M_{\text{abz.}} \wedge N_{\text{abz.}} \Rightarrow (M \cup N)_{\text{abz.}}$ ($= \{m_1, n_1, m_2, n_2, \dots\}$)
- $M_{\text{abz.}} \wedge N \subseteq M \Rightarrow N_{\text{abz.}}$



Operationen

Vereinig. $M \cup N$
 $\Leftrightarrow \{x \mid x \in M \vee x \in N\}$ \odot

Schnitt $M \cap N \Leftrightarrow \{x \mid x \in M \wedge x \in N\}$ ($= \emptyset$ „disjunkt“) \odot

Diff. $M \setminus N \Leftrightarrow \{x \mid x \in M \wedge x \notin N\}$ \odot

Komplement $M^c = \{x \mid x \notin M\}$ \odot

Alle logischen Äquivalenzen gelten auch für die Mengenoperationen.

Häufige Fehler

- $\forall M : \emptyset \subseteq M$, nicht $\forall M : \emptyset \in M$

Quantitative Relationen

Sei Indexmenge I und Mengen $M_i \quad \forall i \in I$.

$$\bigcup_{i \in I} M_i := \{x \mid \exists i \in I : x \in M_i\}$$

$$\bigcap_{i \in I} M_i := \{x \mid \forall i \in I : x \in M_i\}$$

Neutrale Elemente

- $\bigcup_{i \in \emptyset} M_i = \emptyset$ („hinzufügen“)
- $\bigcap_{i \in \emptyset} M_i = U$ („wegnehmen“)

Potenzmenge

$$\mathcal{P}(M) := \{N \mid N \subseteq M\}$$

$$|\mathcal{P}(M)| = 2^{|M|} \quad (\in / \notin \text{ binär})$$

Abbildungen

Abbildung f von X (Definitions- b.) nach Y (Werteb.) ordnet jedem $x \in X$ eindeutig ein $y \in Y$ zu.

$$f : X \rightarrow Y$$

Graph $\text{gr}(f) := \{(x, f(x)) \mid x \in X\}$

Identität

$$\text{id}_A : A \rightarrow A$$

$$\text{id}_A(a) := a \quad \forall a \in A$$

Umkehrfunktion $f^{-1} : Y \rightarrow X$ wenn f bijektiv und $(f \circ f^{-1})(y) = y$

Eigenschaften

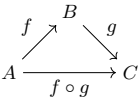
Injektiv $\forall x_1, x_2 \in X :$
 $x_1 \neq x_2 \Leftrightarrow f(x_1) \neq f(x_2)$

Surjektiv $\forall y \in Y \exists x \in X : y = f(x)$

Bijektiv wenn injektiv und surjektiv

Verkettung $f \circ g : A \rightarrow C$

$$(f \circ g)(a) = f(g(a))$$
 (der Reihenfolge nach)



Relationen

Kartesisches Produkt

$$X_1 \times \cdots \times X_n := \{(x_1, \cdots, x_n) \mid x_1 \in X_1, \cdots, x_n \in X_n\}$$

Relation \sim von/auf M nach N ist Teilmenge $R \subseteq M \times N$. ($R' \subseteq N \times P$)

$$m \sim n \Leftrightarrow (m, n) \in R$$

\equiv **Reflexiv** $\forall x \in M : (x, x) \in R$
 $\Leftrightarrow \text{id}_M \subseteq R$

Irreflexiv $\forall x \in M : (x, x) \notin R$
 $\Leftrightarrow \text{id}_M \cap R = \emptyset$

\equiv **Sym.** $\forall (x, y) \in R : (y, x) \in R$
 $\Leftrightarrow R \subseteq R^{-1}$

Antis. $\forall x, y : ((x, y) \in R \wedge (y, x) \in R) \Rightarrow x = y$
 $\Leftrightarrow R \cap R' \subseteq \text{id}_M$

\equiv **Transitiv** $\forall x, y, z : ((x, y) \in R \wedge (y, z) \in R) \Rightarrow (x, z) \in R$
 $\Leftrightarrow R; R \subseteq R$

Vollst. $\forall x, y \in M : (x, y) \in R \vee (y, x) \in R$
 $\Leftrightarrow R \cup R^{-1} = M \times M$

Spezielle Relationen

Inverse Relation R^{-1} mit $R \in M \times N := \{(n, m) \in N \times M \mid (m, n) \in R\}$

Komposition $R; R$ mit $R' \in N \times P := \{(m, p) \in M \times P \mid \exists n \in N : (m, n) \in R \wedge (n, p) \in R'\}$

Leere Relation \emptyset

Identität $\text{id}_M := \{(m, m) \mid m \in M\}$
(=)

Allrelation $M \times M$

Äquivalenzrelation \equiv reflexiv, symmetrisch und transitiv. (Gleichheit***)

Äquivalenzklasse $[m]_{\equiv}$ auf M , Vertreter $m \in M$.

$$[m]_{\equiv} := \{x \in M \mid m \equiv x\}$$

$$\Leftrightarrow [m]_{\equiv} = [x]_{\equiv}$$

Zerlegung $\mathcal{N} \subseteq \mathcal{P}(M)$ von M .

- $\emptyset \notin \mathcal{N}$
- $M = \bigcup \mathcal{N}$
- $N \cap N' = \emptyset$
($N, N' \in \mathcal{N} : N \neq N'$)
- (Korrespondiert zur ÄR.)

Quotient (M / \equiv) Sei \equiv ÄR. auf M . (ist Zerlegung)

$$(M / \equiv) := \{[m]_{\equiv} \mid m \in M\}$$

Analysis

Reelle Zahlen \mathbb{R}

Angeordnete Körper

(Gilt auch für \mathbb{Z} und \mathbb{Q})

Körperaxiome $(\mathbb{R}, +, *)$ $a, b, c \in \mathbb{R}$

Addition $(\mathbb{R}, +)$

Assoziativität
 $a + (b + c) = (a + b) + c$

Kommutativität
 $a + b = b + a$

Neutrales Element Null

$$a + 0 = a \quad 0 \in \mathbb{R}$$

Inverses „Negativ“

$$a + (-a) = 0 \quad (-a) \in \mathbb{R}$$

Multiplikation $(\mathbb{R}, *)$

Assoziativität $a * (b * c) = (a * b) * c$

Kommutativität $a * b = b * a$

Neutrales Element Eins

$$a * 1 = a \quad 1 \in \mathbb{R} \setminus \{0\}$$

Inverses „Kehrwert“

$$a * (a^{-1}) = 1$$

$$a \neq 0, (a^{-1}) \in \mathbb{R}$$

Distributivität

$$a * (b + c) = a * b + a * c$$

Totale Ordnung

Transitivität

$$a < b \wedge b < c \Rightarrow a < c$$

Trichotomie Entweder

$$a < b \text{ oder } a = b \text{ oder } b < a$$

$$\Rightarrow \text{Irreflexivität } (a < b \Rightarrow a \neq b)$$

Addition

$$a < b \Rightarrow a + c < b + c$$

Multiplikation

$$a < b \Rightarrow a * c < b * c \quad 0 < c$$

Bei Additiver oder Multiplikativer Inversion dreht sich die Ungleichung.

ARCHIMEDES Axiom

$$\forall x \in \mathbb{R} \exists n \in \mathbb{N} : n > x$$

$$n > \frac{1}{x}$$

Teilbarkeit

$$a \mid b \Leftrightarrow \exists n \in \mathbb{Z} : b = a * n$$

($\Rightarrow \sqrt{2} \notin \mathbb{Q}$, da mit $\frac{a}{b} = \sqrt{2}$ nicht teilerfremd)

Häufige Fehler

- Nicht durch Null teilen/kürzen
- Nicht $-x < 0$ annehmen
- Multiplikation mit negativen Zahlen kehrt Ungleichungen

Operationen

Brüche

- $\frac{a}{b} * \frac{c}{d} = \frac{a*c}{b*d}$
- $\frac{a}{b} \stackrel{*d}{=} \frac{a*d}{b*d}$
- $\frac{a}{c} + \frac{b}{c} = \frac{a+b}{c}$
- $\frac{a}{b} + \frac{c}{d} = \frac{a*d+c*b}{b*d}$

Wurzeln $b^n = a \Leftrightarrow b = \sqrt[n]{a}$

- $\sqrt[n]{a * b} = \sqrt[n]{a} * \sqrt[n]{b}$
- $\sqrt[n]{\sqrt[m]{a}} = \sqrt[n*m]{a}$
- $\sqrt[n]{a} < \sqrt[n]{b} \quad 0 \leq a < b$
- $\sqrt[n+1]{a} < \sqrt[n]{a} \quad 1 < a$
- $\sqrt[n]{a} < \sqrt[n+1]{b} \quad 0 < a < 1$

$$\sqrt[n]{a^n} = |a| \quad a \in \mathbb{R}$$

Potenzen $a^{\frac{x}{y}} = \sqrt[y]{a^x}$

- $a^x * b^x = (a * b)^x$
- $a^x * a^y = a^{x+y}$
- $(a^x)^y = a^{x*y}$

Intervalle

Sei $A \subseteq \mathbb{R}, A \neq \emptyset, a_0 \in A$.

Geschlossen $[a; b] := \{x \in \mathbb{R} \mid a \leq x \leq b\}$
(„Ecken sind mit enthalten“)

Offen $(a; b) := \{x \in \mathbb{R} \mid a < x < b\}$
(Bei ∞ immer offen, da $\infty \notin \mathbb{R}$)

Kleinstes/Größtes Element

Minimum $\min(A) := a_0$
 $\Leftrightarrow \forall a \in A : a_0 \leq a$

Maximum $\max(A) := a_0$
 $\Leftrightarrow \forall a \in A : a \leq a_0$

$$(\sharp^{\min} / \max(a; b))$$

Beschränktheit A heißt

Ober beschränkt $\exists s \in \mathbb{R} \forall a \in A : a \leq s$

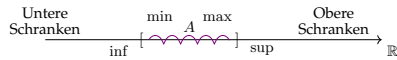
Unten beschränkt $\exists s \in \mathbb{R} \forall a \in A : s \leq a$

Vollständigkeit

Infimum (klein) $\inf(A)$
:= $\max\{s \in \mathbb{R} \mid \forall a \in A : s \leq a\}$

Supremum (groß) $\sup(A)$
:= $\min\{s \in \mathbb{R} \mid \forall a \in A : a \leq s\}$

Vollständigkeitsaxiom $\exists \sup(A).$



Folgen

Folge $(a_n)_{n \in \mathbb{N}}$ in A ist eine Abb. $f : \mathbb{N} \rightarrow A$ mit $a_n = f(n)$.

Arithmetische Folge $a_{n+1} = a_n + d$
 $a_n = a + (n - 1) * d \quad d, a \in \mathbb{R}$

Geometrische Folge $a_{n+1} = a_n * q$
 $a_n = q^n \quad q \in \mathbb{R}$

Rekursion a_n ist auf a_{n-1} definiert.

$$a_{n+1} = F(n, a_n) \quad \forall n \in \mathbb{N}$$
$$F : A \times \mathbb{N} \rightarrow A$$

Primfaktorzerlegung $n \in \mathbb{N}, n \geq 2$

$$\exists p_1, \dots, p_n \in \mathbb{P} : n = p_1 * \dots * p_n$$

Summen und Produkte

Summe $\sum_{i=1}^n i = 1 + 2 + \dots + n$

Produkt $\prod_{i=1}^n i = 1 * 2 * 3 * \dots * n$

Fakultät $n! = \prod_{i=1}^n i \quad (0! = 1)$

GAUSSISCHE SUMME $n \in \mathbb{N}$

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

Geom. Summe $q \in \mathbb{R} \setminus \{0\}, n \in \mathbb{N}_0$

$$\sum_{i=0}^n q^i = \frac{1 - q^{n+1}}{1 - q}$$

BERNOULLI Unglei. $n \in \mathbb{N}_0, x \geq -1$

$$(1 + x)^n \geq 1 + n * x$$

Binom. Koeff. $\binom{n}{k} = \frac{n!}{k! * (n - k)!}$

• Rechnen: $\frac{n > k}{0 < (n - k)}$

• $\binom{n}{0} = \binom{n}{n} = 1$

• $\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}$

Binomischer Satz $n \in \mathbb{N}$

$$(a + b)^n = \sum_{k=0}^n \binom{n}{k} * a^{n-k} * b^k$$

Grenzwerte

Betrag $|x| := \begin{cases} x & 0 \leq x \\ -x & x < 0 \end{cases}$

Lemma $|x * y| = |x| * |y|$

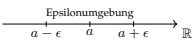
Dreiecksungleichung $|x + y| \leq |x| + |y|$

Umgekehrte Dreiecksungleichung
 $||x| - |y|| \leq |x - y|$

Konvergenz

Sei $(a_n)_{n \in \mathbb{N}} \subseteq \mathbb{R}, a \in \mathbb{R}.$

$$a_n \xrightarrow{n \rightarrow \infty} a \Leftrightarrow \forall \epsilon > 0 \exists n_0 \in \mathbb{N} \forall n \in \mathbb{N} n \geq n_0 : |a_n - a| \leq \epsilon$$
$$(a - \epsilon \leq a_n \leq a + \epsilon)$$



• $a_n \xrightarrow{n \rightarrow \infty} a \Leftrightarrow \lim_{n \rightarrow \infty} a_n = a$

Beschränkt + monoton \Rightarrow konvergent:

$$\lim_{n \rightarrow \infty} a_n = \begin{cases} \inf\{a_n \mid n \in \mathbb{N}\} & (a_n)_{n \in \mathbb{N}} \text{ fall.} \\ \sup\{a_n \mid n \in \mathbb{N}\} & (a_n)_{n \in \mathbb{N}} \text{ steig.} \end{cases}$$

Nullfolgen $\lim_{n \rightarrow \infty} a_n = 0$

• $\lim_{n \rightarrow \infty} \frac{1}{n^k} = 0 \quad k \in \mathbb{N}$

• $\lim_{n \rightarrow \infty} n * q^n = 0$

Folgen gegen 1

• $\lim_{n \rightarrow \infty} \sqrt[n]{a} = 1 \quad a > 0$

• $\lim_{n \rightarrow \infty} \sqrt[n]{n} = 1$

Bestimmt Divergent

$$a_n \xrightarrow{n \rightarrow \infty} \infty \Leftrightarrow \forall R > 0 \exists n \geq n_0 \in \mathbb{N} : a_n \geq R$$
$$a_n \xrightarrow{n \rightarrow \infty} -\infty \Leftrightarrow \forall R < 0 \exists n \geq n_0 \in \mathbb{N} : a_n \leq R$$

$$\lim_{n \rightarrow \infty} q^n = \begin{cases} = 0 & (-1; 1) \\ = 1 & = 1 \\ \geq \infty & > 1 \\ \text{div.} & \leq -1 \end{cases}$$

Monotonie

Monoton fallend

$$a_n \underset{(\text{streng})}{\geq} a_{n+1} \quad \forall n \in \mathbb{N}$$

Monoton steigend

$$a_n \underset{(\text{streng})}{\leq} a_{n+1} \quad \forall n \in \mathbb{N}$$

Beschränktheit

$$\exists k > 0 \forall n \in \mathbb{N} : |a_n| \leq k$$

• Konvergent \Rightarrow beschränkt

• Unbeschränkt \Rightarrow divergent

Grenzwertsätze

$$\lim_{n \rightarrow \infty} a_n = a, \lim_{n \rightarrow \infty} b_n = b$$

• $a_n \xrightarrow{n \rightarrow \infty} a \wedge a_n \xrightarrow{n \rightarrow \infty} b \Rightarrow a = b$ (Max. einen Grenzw.)

• $a = 0 \wedge (b_n)_{n \in \mathbb{N}} \text{ beschr.} \Leftrightarrow \lim_{n \rightarrow \infty} a_n * b_n = 0$

• $a_n \leq b_n \Leftrightarrow a \leq b$ (nicht $<$)

• $\lim_{n \rightarrow \infty} \begin{cases} a_n \pm b_n = a \pm b \\ a_n * b_n = a * b \\ a_n * c = a * c \\ \sqrt[k]{a_n} = \sqrt[k]{a} \\ |a_n| = |a| \end{cases}$

Einschachtelungssatz

$$\lim_{n \rightarrow \infty} a_n = \lim_{n \rightarrow \infty} b_n = a$$
$$\forall n \geq N \in \mathbb{N} : a_n \leq c_n \leq b_n$$
$$(\exists) \lim_{n \rightarrow \infty} c_n = a$$

Spezielle Folgen

Teilfolge *streng mnt.* Folge $(b_k)_{k \in \mathbb{N}}$ mit $(n_k)_{k \in \mathbb{N}}$, sodass $b_k = a_{n_k} \quad \forall k \in \mathbb{N}.$

$$\lim_{n \rightarrow \infty} a_n = a \Rightarrow \lim_{n \rightarrow \infty} a_{n_k} = a$$

(da n_k mnt. steigend)

$$\forall (a_n)_{n \in \mathbb{N}} \exists (a_{n_k})_{k \in \mathbb{N}} \text{ mnt.}$$

(nicht streng!)

Häufungspunkt h mit einer Teilfolge

$$\lim_{n \rightarrow \infty} a_{n_k} = h$$

• $\lim_{n \rightarrow \infty} a_n = a \Leftrightarrow \exists ! : h = a$

BOLZANO-WEIERSTRASS

$$(a_n)_{n \in \mathbb{N}} \text{ beschr.} \Rightarrow \exists h \text{ Häuf.}$$

(Teilfolge + (beschr.) $\Rightarrow \exists$ Häuf.)

CAUCHY-Folge

$$\forall \epsilon > 0 \exists n_0 \in \mathbb{N} \forall n, m \geq n_0 :$$

$$|a_n - a_m| \leq \epsilon$$

(Konv. ohne bekannten Grenzwert)

Vollständigkeit von \mathbb{R}

$$(a_n)_{n \in \mathbb{N}} \text{ CAUCHY} \Leftrightarrow \exists \lim_{n \rightarrow \infty} a_n$$

$$(\exists \lim_{n \rightarrow \infty} a_n \Rightarrow (a_n)_{n \in \mathbb{N}} \text{ CAUCHY}$$
$$\Rightarrow (a_n)_{n \in \mathbb{N}} \text{ beschr.}$$
$$\Rightarrow \exists h \quad (\text{BW})$$
$$\Rightarrow \lim_{n \rightarrow \infty} a_n = h)$$

Reihen

Reihe $(s_n)_{n \in \mathbb{N}} = \sum_{k=1}^{\infty} a_k$ mit den Gliedern $(a_k)_{k \in \mathbb{N}}$.

n-te Partialsumme $s_n = \sum_{k=1}^n a_k$

Grenzwert ebenfalls $\sum_{k=1}^{\infty} a_k$, falls s_n konvergiert

Spezielle Reihen

Geom. $\sum_{k=0}^{\infty} q^k = \frac{1}{1 - q} \quad q \in (-1; 1)$

Harmon. $\sum_{k=1}^{\infty} \frac{1}{k}$ divergent

Allg. Harmon. $\sum_{k=1}^{\infty} \frac{1}{k^\alpha}$ konvergiert $\forall \alpha > 1$

Lemma

- $\sum_{k=1}^{\infty} a_k, \sum_{k=1}^{\infty} b_k$ konvergent
$$- \sum_{k=1}^{\infty} a_k + \sum_{k=1}^{\infty} b_k = \sum_{k=1}^{\infty} (a_k + b_k)$$
$$- c * \sum_{k=1}^{\infty} a_k = \sum_{k=1}^{\infty} c * a_k$$
- $\exists N \in \mathbb{N} : (\sum_{k=N}^{\infty} a_k)_{\text{konv.}} \Rightarrow (\sum_{k=1}^{\infty} a_k)_{\text{konv.}}$ (Es reicht spätere Glieder zu betrachten)
- $(\sum_{k=1}^{\infty} a_k)_{\text{konv.}} \Rightarrow \forall N \in \mathbb{N} : (\sum_{k=N}^{\infty} a_k)_{\text{konv.}} \Rightarrow \lim_{N \rightarrow \infty} \sum_{k=N}^{\infty} a_k = 0$

Konvergenzkriterien

CAUCHY

(sum_{k=1}^inf a_k)_{konv.}

iff (sum_{k=1}^n a_k)_{n in N} is CAUCHY

iff for all epsilon > 0 there exists n_0 in N such that for all n > m > n_0:

|sum_{k=m+1}^n a_k| <= epsilon

Notwendige

(sum_{k=1}^inf a_k)_{konv.} implies limit_{n to inf} a_n = 0

limit_{n to inf} a_n != 0 implies (sum_{k=1}^inf a_k)_{div.}

Hinreichende

Lemma a_k >= 0 (implies mnt.) for all k in N

(sum_{k=1}^inf a_k)_{konv.} iff (sum_{k=1}^inf a_k)_{beschr.}

Majorante 0 <= a_k <= b_k for all k in N (Min. <= Major.)

(sum_{k=1}^inf b_k)_{konv.} iff (sum_{k=1}^inf a_k)_{konv.}

Algorithmus Handlungsvorschrift aus endlich vielen Einzelschritten zur Problemlösung.

- Korrektheit (Test-based dev.)
- Terminierung (TOURING)
- Effizienz (Komplexität)

Formen (High to low) Menschl. Sprache, Pseudocode, Mathematische Ausdrücke, Quellcode, Binärcode

Divide & Conquer

Divide Zerlegen in kleinere Teilprobleme

Conquer Lösen der Teilprobleme mit gleicher Methode (rekursiv)

Merge Zusammenführen der Teillösungen

Effizienz

Raum/Zeit-Tradeoff: schnell + großvs. klein + langsam.

Programmlaufzeit/-allokationen	Komplexität
Einfluss äußerer Faktoren	Unabh.
Konkrete Größe	Asymptotische Schätzung

Inputgröße n Jeweils

- Best-case C_B
- Average-case C_A
- Worst-case C_W

Asymptotische /Speicherkomplexität

Groß-O-Notation Kosten C_f(n) mit g: N -> R exists c > 0 exists n_0 > 0 for all n >= n_0

Untere Schranke Omega(f) C_f(n) >= c * g(n)

Obere Schranke O(f) C_f(n) <= c * g(n)

Exakte Schranke Theta(f) C_f(n) in Omega(f) intersect O(f) Polynomial k-th Grades in Theta(n^k)

(Beweis: g und c finden)

Groß-O	Wachstum	Klasse	
O(1)	Konstant		lösbar
O(log n)	Logarithmisch		
O(n)	Linear		
O(n log n)	Nlogn		
O(n^2)	Quadratisch	Polynomiell O(n^k)	
O(n^3)	Kubisch		hart
O(2^n)	Exponentiell	Exponentiell O(alpha^n)	
O(n!)	Fakultät		
O(n^n)			

Rechenregeln

Elementare Operationen, Kontrollstr. in O(1)

Schleifen in i Wiederholungen * O(f) teuerste Operation

Abfolge O(g) nach O(f) in O(max(f; g))

Rekursion in k Aufrufe * O(f) teuerste Operation

Mastertheorem a >= 1, b > 1, Theta > 0

T(n) = a * T(n/b) + Theta(n^k)

implies { Theta(n^k) if a < b^k, Theta(n^k * log n) if a = b^k, Theta(n^{log_b a}) if a > b^k

Floor/Ceiling Runden

Floor floor(x) nach unten

Ceiling ceil(x) nach oben

Suchverfahren

Lineare Liste endlich, geordnete (nicht sortierte) Folge n Elemente L := [a_0, ..., a_n] gleichen Typs.

Array Sequenzielle Abfolge im Speicher, statisch, Index O(1), schnelle Suchverfahren

Sequenziell C_A(n) = 1/n * sum^n i = (n+1)/2 in O(n)

Algorithm: Sequential Search

Input: Liste L, Predikat x

Output: Index i von x

for i <- 0 to L.len - 1 do if x == L[i] then return i end end return -1

Auswahlproblem Finde i-kleinstes Element in unsortierter Liste in Theta(n)

Algorithm: i-Smallest Element

Input: Unsortierte Liste L, Level i

Output: Kleinstes Element x

p <- L[L.len - 1]

for k = 0 to L.len - 1 do if L[k] < p then if L[k] < L[p] then Push(L <, L[k]) end if L[k] > p then Push(L >, L[k]) end end if L <.len == i - 1 then return p if L <.len > i - 1 then return i-Smallest Element L < if L <.len < i - 1 then return i-Smallest Element (L >, i - 1 - L <.len) end

Sortierte Listen

Binär C_W(n) = floor(log_2 n) + 1, C_A(n) approx log_2 n in O(log n)

Algorithm: Binary Search

Input: Sortierte Liste L, Predikat x

Output: Index i von x

if L.len == 0 then return -1 else m <- floor((L_2.len)/2) if x == L[m] then return m if x < L[m] then return Binary Search [L[0], ..., L[m - 1]] if x > L[m] then return m + 1 + Binary Search [L[m + 1], ..., L[L.len - 1]] end

Sprung Kosten Vergleich a, Sprung b mit optimaler Sprungweite:

m = floor(sqrt((a/b) * n))

C_A(n) = 1/2 * (ceil(n/m) * a + m * b) in O(sqrt(n))

Algorithm: Jump Search

Input: Sortierte Liste L, Predikat x

Output: Index i von x

m <- floor(sqrt(n))

while i < L.len do i <- i + m if x < L[i] then return Search [L[i - m], ..., L[i - 1]] end end return -1

- Rekursive in O(sqrt[3]{n}) Sprungsuche
- Partitionierung in Blöcke m möglich

Exponentiell in O(log x)

Algorithm: Exponential Search

Input: Sortierte Liste L, Predikat x

Output: Index i von x

while x >= L[i] do i <- 2 * i end return Search [L[i/2], ..., L[i - 1]]

- Unbekanntes n möglich

Interpolation C_A(n) = 1 + log_2 log_2 n, C_W(n) in O(n)

Algorithm: Searchposition

Input: Listengrenzen [u, v]

Output: Suchposition p

return floor(u + ((x - L[u]) / (L[v] - L[u])) * (v - u))

Algorithm: Interpolation Search

Input: Sortierte Liste [L[u], ..., L[v]], Predikat x

Output: Index i von x

if x < L[u] or x > L[v] then return -1 p <- Searchposition(u, v) if x == L[p] then return p if x > L[p] then return Interpolation Search(p + 1, v, x) else return Interpolation Search(u, p - 1, x) end

Häufigkeitsordnungen mit Zugriffswahrscheinlichkeit p_i: C_A(n) = sum_{i=0}^n i * p_i

Frequency-count Zugriffszähler pro Element

Transpose Tausch mit Vorgänger

Move-to-front

Verkettete Listen

Container Jedes Element p ist in der Form p -> (key | value | next). Index in O(n)

Löschen $\in O(1)$

```
Algorithm: Delete
Input: Zeiger p auf Vorgänger des lösches Elementes
if p ≠ ∅ ∧ p → next ≠ ∅ then
  | p → next ← (p → next) → next
end
```

- desh. sehr dynamisch

Suchen $C_A(n) = \frac{n+1}{2} \in O(n)$

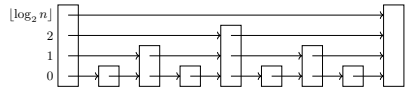
```
Algorithm: Search Linked List
Input: Verkettete Liste L, Predikat x
Output: Zeiger p auf x
p ← L.head while p → value ≠ x do
  | p ← p → next
end
return p
```

Doppelt Verkettet Zeiger auf Vorgänger

(key) value prev next

- Bestimmung des Vorgängers $\in O(1)$ statt $O(n)$
- Höherer Speicheraufwand

Skip



- Zeiger auf Ebene i zeigt zu nächstem 2^i Element
- Suchen $\in O(\log n)$

Perfekt Einfügen, Löschen $\in O(n)$ (Vollst. Reorga.)

Randomisiert Höhe zufällig (keine vollst. Reorga.)
 $P(h) = \frac{1}{2^{h+1}}$: Einfügen, Löschen $\in O(\log n)$

Spezielle Listen

ADT „Abstrakte Datentypen“

Stack $S = | „TOP“, \dots$ Operationen nur auf letztem Element $\in O(1)$

Queue $Q = | „HEAD“, \dots, „TAIL“$
Vorne Löschen, hinten einfügen $\in O(1)$

Priority Queue $P = \begin{bmatrix} p_0 & p_1 & \dots & p_n \\ a_0 & a_1 & \dots & a_n \end{bmatrix}$
Jedes Element hat Priorität; Entfernen von Element mit höchster („MIN“) Priorität

Sortierverfahren

Sortierproblem

Gegeben (endliche) Folge von Schlüssel (von Daten) $(K_i)_{i \in I}$

Gesucht Bijektive Abbildung $\pi : I \rightarrow I$ (Permutation), sodass $K_{\pi(i)} \leq K_{\pi(i+1)}$

mit Optimierung nach geringen

- Schlüsselvergleichen C
- Satzbewegungen M

Eigenschaften

Ordnung *Allgemein* vs. *speziell*: Ordnung wird nur über Schlüsselvergleiche hergestellt

Relation *Stabil* vs. *instabil*: Vorherig relative Reihenfolge bleibt erhalten

Speicher *In situ* vs. *ex situ*: Zusätzlicher Speicher notwendig

Lokal *Intern* vs. *extern*: Hauptspeicher oder Mischung vorsortierter externer Teilfolgen

Ordnung $\forall x, y \in X$

Reflexiv $x \leq x$

Antisym. $x \leq y \wedge y \leq x \Rightarrow x = y$

Transitiv $x \leq y \wedge y \leq z \Rightarrow x \leq z$

Total (Vollständig) $x \leq y \vee y \leq x$

(ohne Total: „*Halbordnung*“)

Grad der Sortierung

Anzahl der Inversionen Anzahl kleinerer Nachfolger für jedes Element:

$$\text{inv}(L) := |\{(i, j) \mid 0 \leq i < j \leq n - 1, L[i] \geq L[j]\}|$$

Anzahl der Runs Ein *Run* ist eine sortierte Teilliste, die nicht nach links oder rechts verlängert werden kann. Die Anzahl der Runs ist:

$$\text{runs}(L) := |\{i \mid 0 \leq i < n - 1, L[i + 1] < L[i]\}| + 1$$

Längster Run Anzahl der Elemente der längsten sortierten Teilliste:

$$\text{las}(L) := \max\{r.\text{len} \mid r \text{ ist Run in } L\}$$
$$\text{rem}(L) := L.\text{len} - \text{las}(L)$$

Einfache Sortierverfahren $O(n^2)$

Selection Entferne kleinstes Element in unsortierter Liste und füge es sortierter Liste an.

```
Algorithm: Selectionsort
Input: Liste L
Output: Sortierte Liste L
for i ← 0 to L.len - 2 do
  min ← i
  for j ← i + 1 to L.len - 1 do
    if L[i] < L[j] then
      | min ← j
  end
  if min ≠ i then
    | Swap L[min], L[i]
  end
end
if L.len = 0 then
  return -1
```

Insertion Verschiebe erstes Element aus unsortierter Liste von hinten durch sortierte Liste, bis das vorgehende Element kleiner ist.

```
Algorithm: Insertionsort
Input: Liste L
Output: Sortierte Liste L
for i ← 1 to L.len - 1 do
  if L[i] < L[i - 1] then
    temp ← L[i]
    j ← i
    while temp < L[j - 1] ∧ j > 0 do
      | L[j] ← L[j - 1]
      j ← j - 1
    end
    L[j] ← temp
  end
end
```

Bubble Vertausche benachbarte Elemente die nicht in Sortierordnung sind, durchlaufe bis nichts vertauscht wird. *Achtung*: Die hinteren Elemente können im Durchlauf ignoriert werden!

```
Algorithm: Bubblesort
Input: Liste L
Output: Sortierte Liste L
i ← L.len
swapped ← 1
while swapped do
  swapped ← 0
  for j ← 0 to i - 2 do
    if L[j] > L[j + 1] then
      | Swap L[j], L[j + 1]
      | swapped ← 1
    end
  end
  i ← i - 1
end
```

Verbesserte Sortierverfahren $O(n \log n)$

Shell Insertionsort, nur werden Elemente nicht mit Nachbarn getauscht, sondern in t Sprüngen h_i , die kleiner werden. Im letzten Schritt dann Insertionsort ($h_t = 1$); somit Sortierung von grob bis fein, also Reduzierung der Tauschvorgänge.

```
Algorithm: Shellsort
Input: Liste L, Absteigende Liste von Sprunggrößen H
Output: Sortierte Liste L
foreach h in H do
  for i ← h to L.len - 1 do
    temp ← L[i]
    for j ← i; temp < L[j - h] ∧ j ≥ h; j ← j - h do
      | L[j] ← L[j - h]
    end
    L[j] ← temp
  end
end
```

Quick Rekursiv: Pivot-Element in der Mitte, Teillisten $L_{<}$, $L_{>}$, sodass $\forall l_{<} \in L_{<} \forall l_{>} \in L_{>} : l_{<} < x < l_{>}$. Zerlegung: Durchlauf von Links bis $L[i] \geq x$ und von Rechts bis $L[j] \leq x$, dann tauschen.

```
Algorithm: Quicksort
Input: Liste L, Indices l, r
Output: L, sortiert zwischen l und r
if l ≥ r then
  | return
i ← l
j ← r
piv ← L[(l+r)/2]
do
  while L[i] < piv do
    | i ← i + 1
  end
  while L[j] > piv do
    | j ← j - 1
  end
  if i ≤ j then
    | Swap L[i], L[j]
    | i ← i + 1
    | j ← j - 1
  end
while i ≤ j;
Quicksort(L, l, j)
Quicksort(L, i, r)
```

Turnier Liste also Binärbaum, bestimme min L durch Austragen des

Turniers, entferne Sieger und wiederhole von Siegerpfad aus.

Heap Stelle Max-Heap (größtes Element in der Wurzel) her, gib Wurzel aus und ersetze mit Element ganz rechts in unterster Ebene.

```
Algorithm: Max-Heapify
Input: Liste L, Index i der MHE widerspricht und
      ∀ j > i erfüllen MHE
Output: Liste L mit MHE ∀ j ≥ i
l ← 2i + 1
r ← 2i + 2
if l < L.len ∧ L[l] > L[i] then
  | largest ← l
else
  | largest ← i
end
if r < L.len ∧ L[r] > L[largest] then
  | largest ← r
if largest ≠ i then
  | Swap L[i], L[largest]
  | Max-Heapify L, largest
```

```
Algorithm: Build-Max-Heap
Input: Liste L
Output: Sortierte Liste L
Build-Max-Heap L
for i ← ⌊ L.len/2 ⌋ - 1 to 0 do
  | Max-Heapify L, i
end
```

Algorithm: Heapsort

```
Input: Liste L
Output: Sortierte Liste L
Build-Max-Heap L
for i ← L.len - 1 to 1 do
  | Swap L[0], L[i]
  | L.len ← L.len - 1
  | Max-Heapify L, 0
end
```

Merge Zerlege Liste in k Teile, sortiere diese (mit Mergesort) und verschmelze die sortierten Teillisten (merge).

```
Algorithm: 2-Merge
Input: Liste L mit L[l ... m - 1] und L[m ... r]
      sortiert, Indices l, m, r
Output: Liste L mit L[l ... r] sortiert
j ← l
k ← m
for i ← 0 to r - l do
  if k > r ∨ (j < m ∧ L[j] ≤ L[k]) then
    | B[i] ← L[j]
    | j ← j + 1
  else
    | B[i] ← L[k]
    | k ← k + 1
  end
end
for i ← 0 to r - l do
  | L[l + i] ← B[i]
end
```

Algorithm: Rekursives 2-Mergesort

```
Input: Liste L, Indices l, r
Output: Liste L mit L[l ... r] sortiert
if l ≥ r then
  | return
else
  | m ← ⌊ (l+r+1)/2 ⌋
  | Mergesort L, l, m - 1
  | Mergesort L, m, r
  | Merge L, l, m, r
end
```


Iteratives 2-Mergesort

Algorithm: Iteratives 2-Mergesort

Input: Liste L

Output: Sortierte Liste L

for $k \leftarrow 2; k < n; k \leftarrow k * 2$ do

for $i \leftarrow 0; i + k \leq n; i \leftarrow i + k$ do

Merge $L, i, \min(i + k - 1, n - 1),$

$i + \frac{k}{2}$

end

end

Merge $L, 0, n - 1, \frac{k}{2}$

Natürliches Mergesort Verschmelzen von benachbarten Runs (Ausnutzen der Vorsortierung)

Untere Schranke allgemeiner Sortierverfahren

Jedes allgemeine Sortierverfahren benötigt im Worst- und Average-case Schlüsselvergleiche von mindestens:

$$\Omega(n \log n)$$

(Siehe Pfadlänge auf Entscheidungsbaum)

Spezielle Sortierverfahren $O(n)$

Distribution Abspeichern der Frequenz jedes Elementes k auf $F[k]$; Ausgeben jedes Index $F[k]$ mal.

Lexikographische Ordnung Sei $A = \{a_1, \dots, a_n\}$ ein Alphabet, dass sich mit gegebener Ordnung $a_1 < \dots < a_n$ wie folgt auf dem Lexikon $A^* = \bigcup_{n \in \mathbb{N}_0} A^n$ fortsetzt:

$$v = (v_1, \dots, v_p) \leq w = (w_1, \dots, w_q)$$

$$\Leftrightarrow \forall 1 \leq i \leq p : v_i = w_i \quad p \leq q$$
$$\vee \forall 1 \leq j \leq i : v_j = w_j \quad v_i < w_i$$

(Die antisymmetrische Relation \leq heißt Lexikographische Ordnung)

Fachverteilen Sortieren von n k -Tupeln in k Schritten: Sortieren nach letztem Element, vorletztem usw.

Große Datensätze sortieren

Indirekt Liste von Zeigern $Z[i] = i$ auf die eigentlichen Listenelemente.

Schlüsselvergleiche mit $L[Z[i]]$, Satzbewegungen nur als Zeigertausch in Z . Anschließend linear kopieren.

Extern Zerlegen in m Blöcke, sortieren im Hauptspeicher (Run) der mind. $m + 1$ Blöcke groß ist, verschmelzen der Runs (m -Wege-Merge).

Ausgeglichenes 2-Wege-Mergesort Daten auf Band n , sortieren von Block $r_1 < n$ auf zweites Band und r_2 auf drittes Band, löschen des ersten Bandes und Merge $2r$ abwechselnd auf erstes (neues $2r_1$) und viertes Band (neues $2r_2$) und wiederholen.

Replacement Selectionsort Lese $r < n$ Elemente auf Priority-Queue Q . Falls $x = \min(Q) \geq$ letztem Element auf zweiten Band, schreibe x aus, sonst schreibe Q auf Band. Wiederhole auf dritten Band und dann merge.

$$\frac{1}{2}n \log \frac{1}{2}n + \frac{1}{2}n \log \frac{1}{2}n$$

Algo.	Stabil	Mem.	Schlüsselvergleiche			Satzbewegungen		
			C_S	C_M	M_S	M_M	M_W	
Selection	\neq	1	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$	$3 \times (n-1)$	$3 \times (n-1)$	$3 \times (n-1)$	
Insertion	\surd	1	$\frac{n(n-1)}{2} + n - 1$	$\frac{n(n-1)}{2}$	$2 \times (n-1)$	$\frac{n(n-1)}{2} + n - 1$	$\frac{n(n-1)}{2}$	
Bubble	\surd	1	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$	0	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$	
			Best-case		Average case	Worst-case		
Shell	\neq	1						
Quick	\neq	$\log n$	$n \log n$	$n \log n$	$n \log n$	n^2	$O(n \log^2 n)$	
Timmer	\neq	$2n - 1$	$n \log n$	$n \log n$	$n \log n$	$n \log n$		
Heap	\neq	1	$n \log n$	$n \log n$	$n \log n$	$n \log n$		
Merge	\surd	n	$n \log n$	$n \log n$	$n \log n$	$n \log n$		
Untere Schranke $(1.5 \times \log n)$ für allgemeine Sortierverfahren								
Distribution	\surd	n	n	n	$n \log n, n^2$		$O(n)$	