



Informatik I Komplexität von Algorithmen

G. Zachmann
Clausthal University, Germany
zach@in.tu-clausthal.de



Leistungsverhalten von Algorithmen

- **Speicherplatzkomplexität:** Wird primärer & sekundärer Speicherplatz effizient genutzt?
- **Laufzeitkomplexität:** Steht die Laufzeit im akzeptablen / vernünftigen / optimalen Verhältnis zur Aufgabe?
- Theorie: liefert **untere Schranke**, die für jeden Algorithmus gilt, der das Problem löst.
- Spezieller Algorithmus liefert **obere Schranke** für die Lösung des Problems.
- Erforschung von oberen und unteren Schranken:
Effiziente Algorithmen und Komplexitätstheorie (Zweige der Theoretischen Informatik)



Laufzeit



- Definition:
Die **Laufzeit $T(x)$** eines Algorithmus A bei Eingabe x ist definiert als die **Anzahl von Basisoperationen**, die Algorithmus A zur Berechnung der Lösung bei Eingabe x benötigt.
- Laufzeit = Funktion der Größe der Eingabe
- Definition für **Eingabegröße** ist abhängig vom Problem



Laufzeitanalyse



- Sei P ein gegebenes Programm und x Eingabe für P , $|x|$ Länge von x , und $T_P(x)$ die Laufzeit von P auf x .
- Beschreibe Aufwand eines Algorithmus als Funktion **der Größe des Inputs** (kann verschieden gemessen werden):
 $T_P(n)$ = Laufzeit des Programms P für Eingaben der Länge n
- **Der beste Fall (best case)**: Oft leicht zu bestimmen, kommt in der Praxis jedoch selten vor:

$$T_P(n) = \inf\{T_P(x) \mid |x| = n, x \text{ Eingabe für } P\}$$

- **Der schlechteste Fall (worse case)**: Liefert garantierte Schranken, meist *relativ* leicht zu bestimmen. Oft zu pessimistisch:

$$T_P(n) = \sup\{T_P(x) \mid |x| = n, x \text{ Eingabe für } P\}$$



Kostenmaße



- **Einheitskostenmaß:** Annahme, jedes Datenelement belegt unabhängig von seiner Größe denselben Speicherplatz (in Einheitsgröße).
 - Damit: Größe der Eingabe bestimmt durch Anzahl der Datenelemente
 - Beispiel: Sortierproblem
- **Logarithmisches Kostenmaß (Bit-Komplexität):** Annahme, jedes Datenelement belegt einen von seiner Größe (logarithmisch) abhängigen Platz
 - Größe der Eingabe bestimmt durch die Summe der Größen der Elemente
 - Erinnerung: für $n > 0$ ist die # Bits zur Darstellung von $n = \lceil \log_2(n + 1) \rceil$
 - Beispiel: Zerlegung einer gegebenen großen Zahl in Primfaktoren
- Ab jetzt immer Einheitskostenmaß





Beispiel Minimum-Suche



- Eingabe : Folge von n Zahlen (a_1, a_2, \dots, a_n) .
- Ausgabe : Index i , so daß $a_i \leq a_j$ für alle Indizes $1 \leq j \leq n$.
- Beispiel:
 - Eingabe: 31, 41, 59, 26, 51, 48
 - Ausgabe: 4

```
def min(A):  
    min = 1  
    for j in range(2, len(A)):  
        if A[j] < A[min]:  
            min = j
```



	Kosten	Anzahl
<code>def min(A):</code>	c_1	1
<code> min = 0</code>	c_2	$n - 1$
<code> for j in range(1, len(A)):</code>	c_3	$n - 1$
<code> if A[j] < A[min]:</code>	c_4	$n - 1$
<code> min = j</code>		

- Zusammen: Zeit

$$T(n) = c_1 + (n - 1) \cdot (c_2 + c_3 + c_4) \leq \bar{c} \cdot n$$

- Eingabegröße = Größe des Arrays

G. Zachmann Informatik 1 - WS 05/06
Komplexität 7

Weiteres Beispiel für Aufwandsberechnung



- Wir betrachten folgende Funktion f1, die $1! \cdot 2! \cdots (n-2)! \cdot (n-1)!$ berechnet

```
def f1(n):
    r = 1
    while n > 0:
        i = 1
        while i < n:
            r *= i
            i += 1
        n -= 1
    return r
```

n	Z (n)	V (n)	M (n)	I (n)
1	2	2	0	1
2	3	5	1	1
3	5	3	1	3
4	8	6	3	6
5	12	10	6	10
6	17	15	10	15
7	23	21	15	21
8	30	28	21	28
9	38	36	28	36
10	47	45	36	45

- Exakte Bestimmung des Aufwandes:
 - M = Anzahl Mult, I = Anzahl Inkr.,
 - V = Anzahl Vergleiche, Z = Anzahl Zuweisungen

G. Zachmann Informatik 1 - WS 05/06
Komplexität 8

- Anzahl Mult $M(n)$

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

$$= \sum_{k=1}^{n-1} k = \frac{(n-1)(n-2)}{2}$$
- Anzahl der Inkrementierungen: $I(n) = n + M(n+1)$,
 woraus folgt:

$$I(n) = \frac{n(n+1)}{2}$$
- Die Anzahl der Vergleiche

$$V(n) = I(n+1) = \frac{(n+1)(n+2)}{2}$$
- Die Anzahl benötigter Zuweisungen $Z(n)$ ist gleich



$$Z(n) = 1 + n + I(n) = 1 + \frac{n(n+3)}{2}$$

```

r = 1
while n > 0 :
    i = 1
    while i < n:
        r *= i
        i += 1
    n -= 1
return r

```

G. Zachmann Informatik 1 - WS 05/06
Komplexität 9

Rechenmodell / Algorithmisches Modell

- Für eine präzise mathematische Laufzeitanalyse benötigen wir ein Rechenmodell, das definiert
 - Welche Operationen zulässig sind.
 - Welche Datentypen es gibt.
 - Wie Daten gespeichert werden.
 - Wie viel Zeit Operationen auf bestimmten Daten benötigen.
- Formal ist ein solches Rechenmodell gegeben durch die Random Access Maschine (RAM).
 - RAMs sind Idealisierung von 1-Prozessorrechner mit einfachem aber unbegrenzt großem Speicher.

G. Zachmann Informatik 1 - WS 05/06
Komplexität 10



Basisoperationen und deren Kosten



- *Definition* : Als **Basisoperationen** bezeichnen wir
 - **Arithmetische Operationen** – Addition, Multiplikation, Division, Ab-, Aufrunden. Auf Zahlen fester Längen (z.B 64 Bit = Double)
 - **Datenverwaltung** – Laden, Speichern, Kopieren von Datensätzen fester Größe
 - **Kontrolloperationen** – Verzweigungen, Sprünge, Wertübergaben.
- **Kosten**: Zur Vereinfachung nehmen wir an, daß jede dieser Operationen bei allen Operanden gleich viel Zeit benötigt (im Einheitskostenmaß)
 - Überwiegend unabhängig von der verwendeten Programmiersprache
 - Ablesbar aus Pseudocode oder Programmstück
 - Exakte Definition ist nicht bedeutend



Beispiele



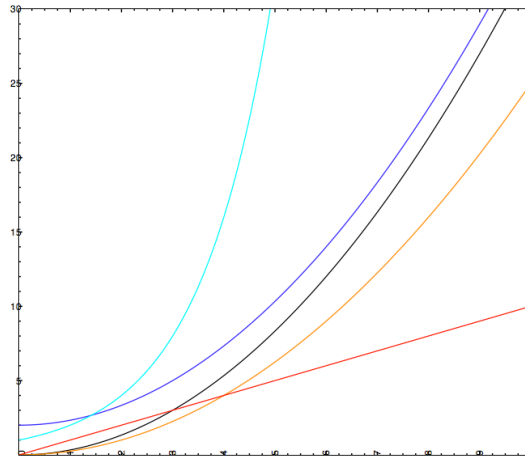
- einen Ausdruck auswerten
- einer Variablen einen Wert zuweisen
- Indizierung in einem Array
- Aufrufen einer Methode / Funktion
- Verlassen einer Methode / Funktion



Beispiel für Wachstum von Funktionen



$$\begin{aligned}f_1(n) &= \frac{1}{3}n^2, \\f_2(n) &= \frac{1}{4}n^2, \\f_3(n) &= n, \\f_4(n) &= \frac{1}{3}n^2 + 2, \\f_5(n) &= 2^n.\end{aligned}$$



Funktionsklassen



- Ziel:
Konstante Summanden und Faktoren dürfen bei der Aufwandsbestimmung vernachlässigt werden.
- Gründe:
 - Man ist an **asymptotischem Verhalten** für große Eingaben interessiert
 - Genaue Analyse kann technisch oft sehr aufwendig oder unmöglich sein
 - Lineare Beschleunigungen sind ohnehin immer möglich (schnellere Hardware)
- Idee:
 - Komplexitätsmessungen mit Hilfe von Funktionsklassen. Etwa $O(f)$ sind die Funktionen, die (höchstens) in der Größenordnung von f sind.
- **Groß-O-Notation:**
Mit O -, Ω - und Θ -Notation sollen obere, untere bzw. genaue Schranken für das Wachstum von Funktionen beschrieben werden.



"Groß-O"



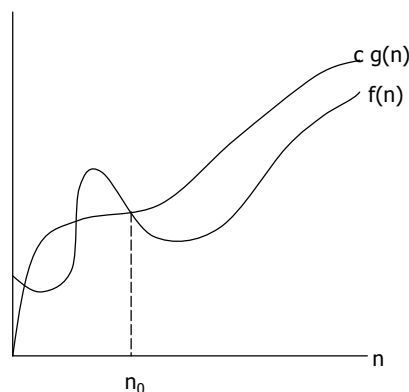
- Sei $f : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$
- Definition **Groß-O**: Die **Ordnung von f** (*the order of f*) ist die Menge
$$O(f(n)) = \{t : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0} \mid \exists c \in \mathbb{R}_{\geq 0} \exists n_0 \in \mathbb{N} \forall n \geq n_0 : t(n) \leq c \cdot f(n)\}$$
- Definition **Groß-Omega**: die Menge Ω ist wie folgt definiert:
$$\Omega(f(n)) = \{t : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0} \mid \exists c \in \mathbb{R}_{\geq 0} \exists n_0 \in \mathbb{N} \forall n \geq n_0 : t(n) \geq c \cdot f(n)\}$$
- Definition **Groß-Theta**: Die **exakte Ordnung Θ von $f(n)$** ist definiert als:
$$\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$$
- Terminologie: O , Ω , Θ , heißen manchmal auch **Landau'sche Symbole**



Veranschaulichung der O-Notation



- Die Funktion f gehört zur Menge $O(g)$, wenn es positive Konstante c , n_0 gibt, so daß $f(n)$ ab n_0 unterhalb $cg(n)$ liegt

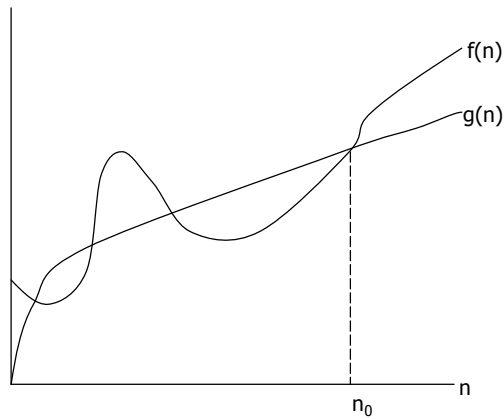




Veranschaulichung der Ω -Notation



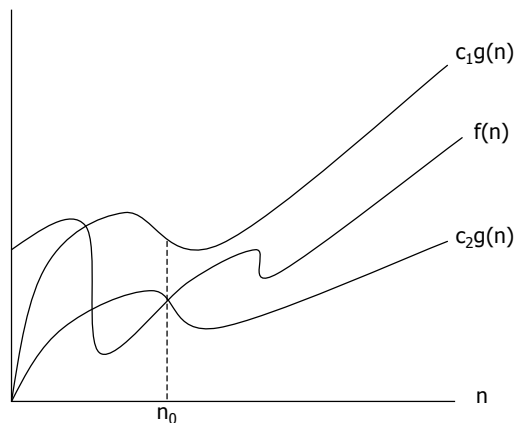
- Die Funktion f gehört zur Menge $\mathcal{O}(g)$, wenn es positive Konstante c , n_0 gibt, so daß $f(n)$ ab n_0 unterhalb $cg(n)$ liegt



Veranschaulichung der Θ -Notation



- Die Funktion f gehört zur Menge $\Theta(g)$, wenn es positive Konstante c_1 , c_2 , und n_0 gibt, so daß $f(n)$ ab n_0 zwischen $c_1g(n)$ und $c_2g(n)$ "eingepackt" werden kann





Bemerkungen zu den O-Notationen



- In manchen Quellen findet man leicht abweichende Definitionen, etwa
 $O(g) = \{f : \mathbb{N} \rightarrow \mathbb{R}^+ \mid \text{es gibt positive Konstanten } a \text{ und } b \text{ mit } f(n) \leq ag(n) + b \text{ für alle } n\}$
- Für die relevantesten Funktionen f (etwa die monoton steigenden f nicht kongruent 0) sind diese Definitionen äquivalent.
- Schreibweise (leider) oft : $f = O(g)$ statt $f \in O(g)$
- **Minimalität:** Die angegebene Größenordnung muß **nicht** minimal gewählt sein
- **Asymptotik:** Wie groß n_0 ist bleibt unklar (kann sehr groß sein)
- **„Verborgene Konstanten“:** Die Konstanten c und n_0 haben für kleine n großen Einfluß.



Beispiel Min-Search



- Behauptung: unser Minimum-Search-Algo besitzt Laufzeit $\Theta(n)$.

- Erinnerung:

$$T(n) \approx c_1 \cdot n + c_2$$

```
def min(A):  
    min = 1  
    for j in range(2, len(A)):  
        if A[j] < A[min]:  
            min = j
```

- Zum Beweis ist zu zeigen:
 1. Es gibt ein c_2 und n_2 , so daß die Laufzeit von Min-Search bei allen Eingaben der Größe $n \geq n_2$ immer höchstens $c_2 n$ ist. (Groß-O)
 2. Es gibt ein c_1 und n_1 , so daß für alle $n \geq n_1$ eine Eingabe der Größe n existiert, bei der Min-Search mindestens Laufzeit $c_1 n$ besitzt. (Omega)



Beispiele zu Funktionsklassen



- Ist $n^2 \in O(n^3)$?
 - Gesucht: $c \in \mathbb{R}^+$ und $n_0 \in \mathbb{N}$, so daß Bed. erfüllt,
also $\forall n > n_0 : n^2 \leq cn^3$
$$\Leftrightarrow 1 \leq c \cdot n \Leftrightarrow n \geq \frac{1}{c}$$

Wähle $c = 1, n_0 = 1$
- Ist $n^3 \in O(n^2)$?
 - Gesucht: $c \in \mathbb{R}^+, n_0 \in \mathbb{N}$, so daß $\forall n > n_0 : n^3 \leq cn^2$
$$\Leftrightarrow \exists c \in \mathbb{R}^+, n_0 \in \mathbb{N} \forall n > n_0 : n \leq c$$

Widerspruch!
- $f(n) = n \log n \notin O(n)$