Grundbegriffe der Informatik

Einheit 17: Quantitative Aspekte von Algorithmen

Thomas Worsch

KIT, Institut für Theoretische Informatik

Wintersemester 2015/2016

Überblick

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

- Ignorieren konstanter Faktoren
- Notation für obere und untere Schranken des Wachstums
- Eine grauenhafte Schreibweise
- Rechnen im O-Kalkül

Matrixmultiplikation

- Rückblick auf die Schulmethode
- Algorithmus von Strassen

Asymptotisches Verhalten implizit definierter Funktionen

- Laufzeit von Teile-und-Herrsche-Algorithmen
- Ineinander geschachtelte Schleifen

Wo sind wir?

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

Matrixmultiplikation

Asymptotisches Verhalten implizit definierter Funktionen

Zählen arithmetischer Operationen — in Abhängigkeit von der Größe der Objekte

im vorangegangenen Kapitel

- Addition von $n \times n$-Matrizen: n^2
- Multiplikation von $n \times n$-Matrizen $2n^3 - n^2$
- Berechnung der Wegematrix $n^5 - \frac{3}{2}n^4 + \frac{3}{2}n^3 + n^2$
  - oder weniger …

Idee: Anzahl durchgeführter Operationen

$\longleftrightarrow$ «Laufzeit» der Algorithmen

Ressourcen für Rechnungen

Laufzeit/Rechenzeit

Speicherplatzbedarf

- insbesondere für Zwischenergebnisse

Komplexitätsmaße

- im Sinne von *computational complexity*

Effizienz interessiert

- *aber:* «premature optimization is the root of all evil» (Knuth, 1974)

O, Θ, Ω — zur Notation asymptotischen Wachstums

wichtiges Handwerkszeug zum

- Reden über und
- Ausrechnen von

z. B. Laufzeiten

insbesondere «kontrollierte Ungenauigkeiten»

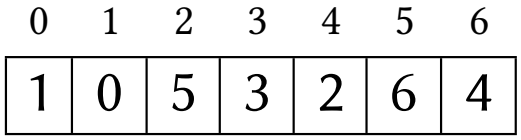
- O: «Groß-O»
  - eingeführt von Bachmann (oder früher)
  - von Landau bekannt gemacht
- Ω, Θ
  - von Knuth zumindest verbreitet

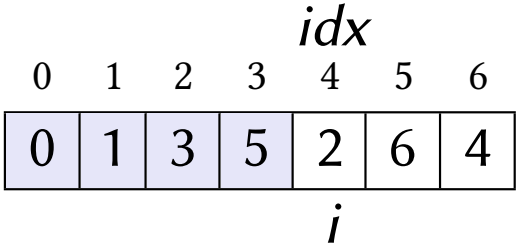
Insertionsort — Wieviele Vertauschungen sind nötig?

0	1	2	3	4	5	6
1	0	5	3	2	6	4

```
void sort(long[ ] a) {
    for (int i ← 1;   i < a.length;   i++) {
        insert(a, i);
    }
}
void insert(long[ ] a, int idx) {
    int i ← idx;
    // Tausche a[idx] nach links bis einsortiert
    while (i > 0  ∧  a[i − 1] > a[i]) {
        // Elemente a[i − 1] und a[i] vertauschen
        a[i − 1] ↔ a[i];
        i ← i − 1;
    }
}
```

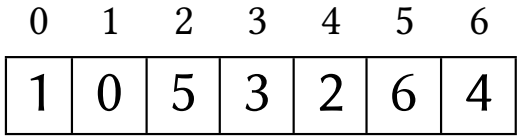
Insertionsort — Wieviele Vertauschungen sind nötig?

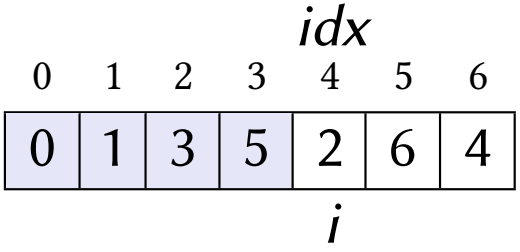


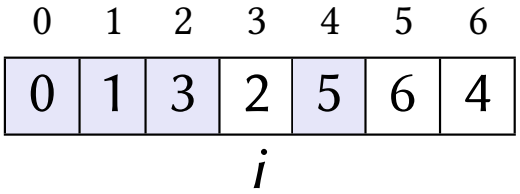



```
void sort(long[ ] a) {
    for (int i ← 1;   i < a.length;   i++) {
        insert(a, i);
    }
}
void insert(long[ ] a, int idx) {
    int i ← idx;
    // Tausche a[idx] nach links bis einsortiert
    while (i > 0  ∧  a[i − 1] > a[i]) {
        // Elemente a[i − 1] und a[i] vertauschen
        a[i − 1] ↔ a[i];
        i ← i − 1;
    }
}
```

Insertionsort — Wieviele Vertauschungen sind nötig?

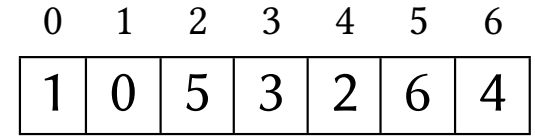


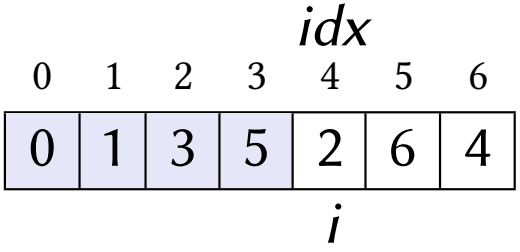


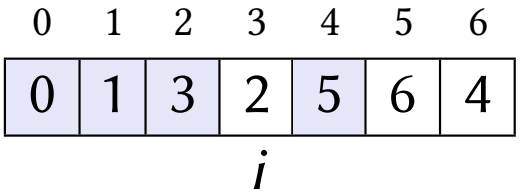


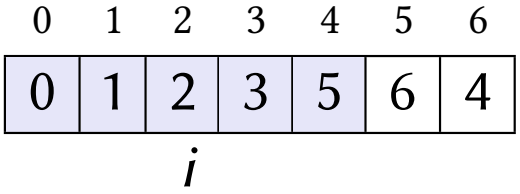

```
void sort(long[ ] a) {
    for (int i ← 1;   i < a.length;   i++) {
        insert(a, i);
    }
}
void insert(long[ ] a, int idx) {
    int i ← idx;
    // Tausche a[idx] nach links bis einsortiert
    while (i > 0  ∧  a[i − 1] > a[i]) {
        // Elemente a[i − 1] und a[i] vertauschen
        a[i − 1] ↔ a[i];
        i ← i − 1;
    }
}
```

Insertionsort — Wieviele Vertauschungen sind nötig?










```
void sort(long[ ] a) {
    for (int i ← 1;   i < a.length;   i++) {
        insert(a, i);
    }
}
void insert(long[ ] a, int idx) {
    int i ← idx;
    // Tausche a[idx] nach links bis einsortiert
    while (i > 0  ∧  a[i − 1] > a[i]) {
        // Elemente a[i − 1] und a[i] vertauschen
        a[i − 1] ↔ a[i];
        i ← i − 1;
    }
}
```

Insertionsort — Laufzeitabschätzung?

Laufzeit

≈

Anzahl Vergleiche

≈

Anzahl Vertauschungen (im wesentlichen …)

≈

Anzahl Schleifendurchläufe

- abhängig von Problemgröße $n = a.length$
- abhängig von konkreter Probleminstanz
  - wenn a anfangs an sortiert:
    while-Schleife wird überhaupt nicht ausgeführt.
  - wenn a anfangs entgegengesetzt geordnet:
    Schleifenrumpf $\sum_{i=1}^{n-1} i = n(n-1)/2$ mal ausgeführt

Ressourcenverbrauch — wie detailliert?

für jede Probleminstanz einzeln

- präzise aber oft unpraktikabel

gröber: nur in Abhängigkeit von der «Problemgröße»

wenn Ressourcenverbrauch
für verschiedene Instanzen gleicher Größe
unterschiedlich ist:

- bester Fall? (best case)
- Durchschnitt? (average case)
- schlechtester Fall? (worst case)

Ressourcenverbrauch — wie detailliert?

für jede Probleminstanz einzeln

- präzise aber oft unpraktikabel

gröber: nur in Abhängigkeit von der «Problemgröße»

wenn Ressourcenverbrauch
für verschiedene Instanzen gleicher Größe
unterschiedlich ist:

- bester Fall? (best case)
  - oft total uninteressant
- Durchschnitt? (average case)

- schlechtester Fall? (worst case)

Ressourcenverbrauch — wie detailliert?

für jede Probleminstanz einzeln

- präzise aber oft unpraktikabel

gröber: nur in Abhängigkeit von der «Problemgröße»

wenn Ressourcenverbrauch
für verschiedene Instanzen gleicher Größe
unterschiedlich ist:

- bester Fall? (best case)
  - oft total uninteressant
- Durchschnitt? (average case)
  - oft sehr schwer
- schlechtester Fall? (worst case)

Ressourcenverbrauch — wie detailliert?

für jede Probleminstanz einzeln

- präzise aber oft unpraktikabel

gröber: nur in Abhängigkeit von der «Problemgröße»

wenn Ressourcenverbrauch
für verschiedene Instanzen gleicher Größe
unterschiedlich ist:

- bester Fall? (best case)
  - oft total uninteressant
- Durchschnitt? (average case)
  - oft sehr schwer
- schlechtester Fall? (worst case)
  - oft angegeben
  - mit dem Hinweis, dass es auch besser sein kann …

Was ist wichtig

Das sollten Sie mitnehmen:

- Bedarf an
  - Rechenzeit und
  - Speicherplatzbedarf

  wichtige Komplexitätsmaße
- oft Quantifizierung in Abhängigkeit von Problemgröße
  - oft schlimmster Fall (worst case)
  - gelegentlich ein «mittlerer» Fall (average case)

Das sollten Sie üben:

- Abschätzen/ausrechnen wie oft ein Programmstück, z. B. ein Schleifenrumpf, durchlaufen wird.

Wo sind wir?

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

- Ignorieren konstanter Faktoren
- Notation für obere und untere Schranken des Wachstums
- Eine grauenhafte Schreibweise
- Rechnen im O-Kalkül

Matrixmultiplikation

Asymptotisches Verhalten implizit definierter Funktionen

Warum keine exakten Angaben?

Man will nicht.

Man kann nicht.

Man «soll» nicht.

Warum keine exakten Angaben?

Man will nicht.

- Faulheit
- Vergänglichkeit der genauen Werte
  - bald neuer Prozessor
- mangelndes Interesse an genauen Werten

Man kann nicht.

Man «soll» nicht.

Warum keine exakten Angaben?

Man will nicht.

- Faulheit
- Vergänglichkeit der genauen Werte
  - bald neuer Prozessor
- mangelndes Interesse an genauen Werten

Man kann nicht.

- Unfähigkeit
- Unkenntnis von Randbedingungen
- Ungenauigkeiten im „Algorithmus“ (??)

Man «soll» nicht.

Warum keine exakten Angaben?

Man will nicht.

- Faulheit
- Vergänglichkeit der genauen Werte
  - bald neuer Prozessor
- mangelndes Interesse an genauen Werten

Man kann nicht.

- Unfähigkeit
- Unkenntnis von Randbedingungen
- Ungenauigkeiten im „Algorithmus“ (??)

Man «soll» nicht.

- Unabhängigkeit von konkreten Probleminstanzen
- Unabhängigkeit von Programmiersprache
- Unabhängigkeit von Prozessor

Wie ungenau wollen wir über Funktionen reden?

Ignorieren konstanter Faktoren

- Motivation: Geschwindigkeitssteigerungen bei Prozessoren irrelevant

nur obere (bzw. untere) Schranken

- Motivation: können nur schlechtesten Fall analysieren

Wo sind wir?

Zu Notation und Redeweise

Notation:

- $\mathbb{R}_+$: Menge der positiven reellen Zahlen (*ohne* 0)
- $\mathbb{R}_0^+$: Menge der nichtnegativen rellen Zahlen, $\mathbb{R}_0^+ = \mathbb{R}_+ \cup \{0\}$.
- betrachten Funktionen $f : \mathbb{N}_0 \to \mathbb{R}_0^+$.

Zu Notation und Redeweise

Notation:

- $\mathbb{R}_+$: Menge der positiven reellen Zahlen (*ohne* 0)
- $\mathbb{R}_0^+$: Menge der nichtnegativen rellen Zahlen, $\mathbb{R}_0^+ = \mathbb{R}_+ \cup \{0\}$.
- betrachten Funktionen $f : \mathbb{N}_0 \to \mathbb{R}_0^+$.

Redeweisen:

- *asymptotisches Wachstum* oder
- *größenordnungsmäßiges Wachstum* von Funktionen

Zu Notation und Redeweise

Notation:

- $\mathbb{R}_+$: Menge der positiven reellen Zahlen (*ohne* 0)
- $\mathbb{R}_0^+$: Menge der nichtnegativen rellen Zahlen, $\mathbb{R}_0^+ = \mathbb{R}_+ \cup \{0\}$.
- betrachten Funktionen $f : \mathbb{N}_0 \to \mathbb{R}_0^+$.

Redeweisen:

- *asymptotisches Wachstum* oder
- *größenordnungsmäßiges Wachstum* von Funktionen

Definition:

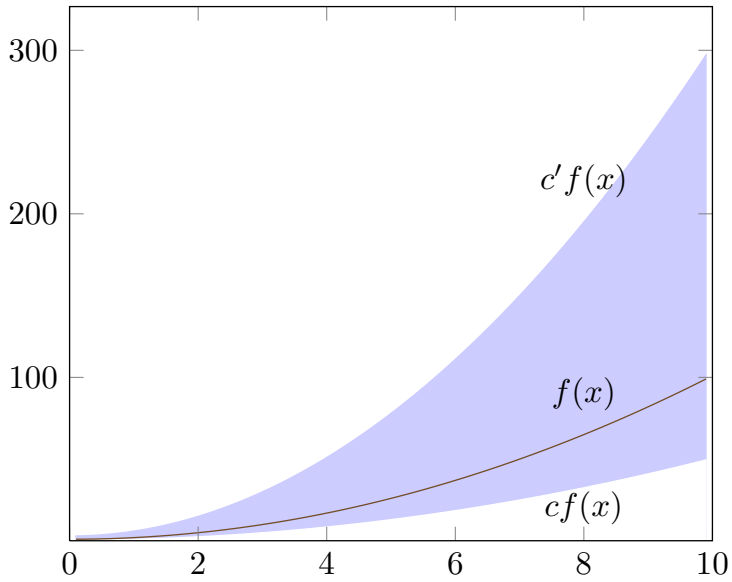
- Funktion $g : \mathbb{N}_0 \to \mathbb{R}_0^+$ wächst asymptotisch genauso schnell wie Funktion $f : \mathbb{N}_0 \to \mathbb{R}_0^+$, wenn gilt:

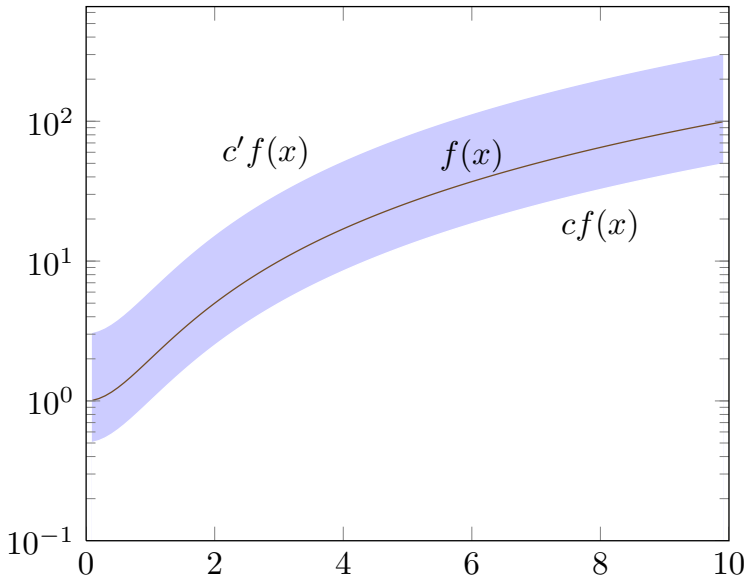
$$\exists c, c' \in \mathbb{R}_+ : \exists n_0 \in \mathbb{N}_0 : \forall n \geq n_0 : c f(n) \leq g(n) \leq c' f(n)$$

- schreibe $f \asymp g$ oder $f(n) \asymp g(n)$

Erläuterungen zur Definition von $f \asymp g$ (1)

$$\exists c, c' \in \mathbb{R}_+ : \exists n_0 \in \mathbb{N}_0 : \forall n \geq n_0 : cf(n) \leq g(n) \leq c'f(n)$$

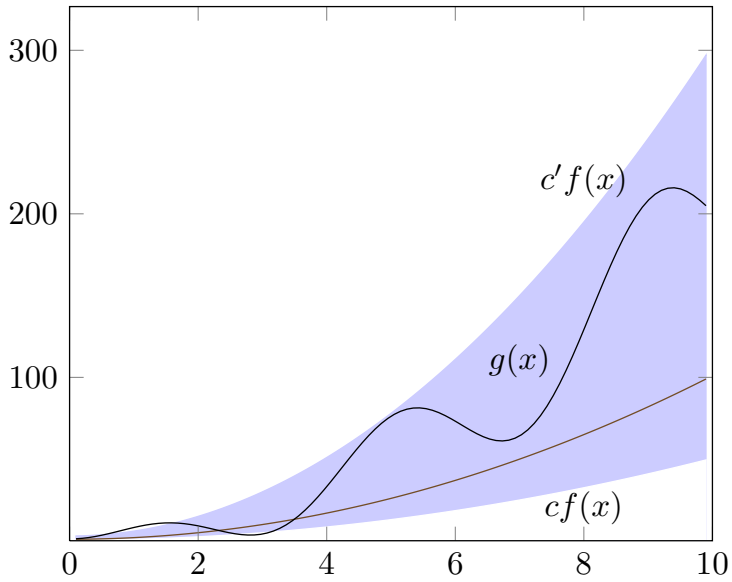


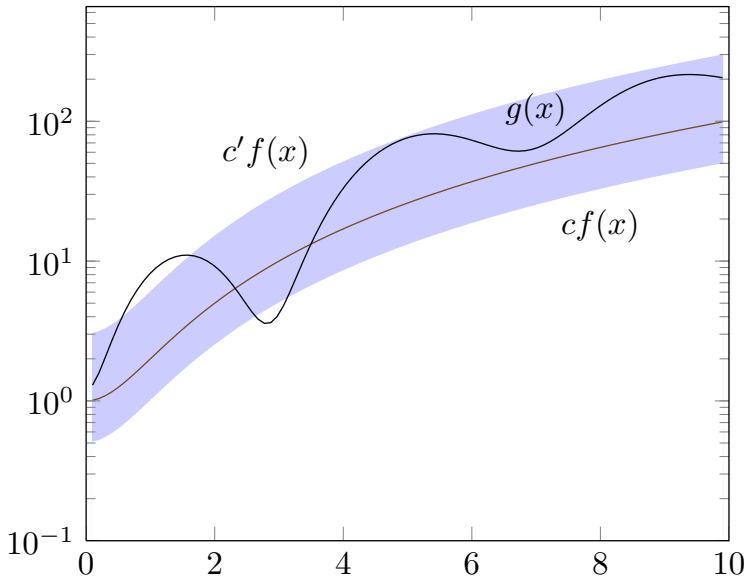


Achtung: kontinuierliche Funktionen gezeichnet,
aber Definition für diskrete Funktionen

Erläuterungen zur Definition von $f \asymp g$ (2)

$$\exists c, c' \in \mathbb{R}_+ : \exists n_0 \in \mathbb{N}_0 : \forall n \geq n_0 : cf(n) \leq g(n) \leq c'f(n)$$





Achtung: kontinuierliche Funktionen gezeichnet,
aber Definition für diskrete Funktionen

Beispiel

$f : n \mapsto 3n^2$ und $g : n \mapsto 10^{-2}n^2$.

Behauptung: $f \asymp g$

Beispiel

$f : n \mapsto 3n^2$ und $g : n \mapsto 10^{-2}n^2$.

Behauptung: $f \asymp g$

- $cf(n) \leq g(n)$: für $c = 10^{-3}$ und $n_0 = 0$ gilt

$$\forall n \geq n_0 : cf(n) = 10^{-3} \cdot 3n^2 \leq 10^{-2}n^2 = g(n)$$

- $g(n) \leq c'f(n)$ gilt z. B. für $c' = 1$ und $n_0 = 0$:

$$\forall n \geq n_0 : g(n) = 10^{-2}n^2 \leq 3n^2 = c'f(n)$$

Beispiel

$f : n \mapsto 3n^2$ und $g : n \mapsto 10^{-2}n^2$.

Behauptung: $f \asymp g$

- $cf(n) \leq g(n)$: für $c = 10^{-3}$ und $n_0 = 0$ gilt

$$\forall n \geq n_0 : cf(n) = 10^{-3} \cdot 3n^2 \leq 10^{-2}n^2 = g(n)$$

- $g(n) \leq c'f(n)$ gilt z. B. für $c' = 1$ und $n_0 = 0$:

$$\forall n \geq n_0 : g(n) = 10^{-2}n^2 \leq 3n^2 = c'f(n)$$

Rechenregel

Für jedes $f : \mathbb{N}_0 \to \mathbb{R}_0^+$ gilt:

$$\forall a, b \in \mathbb{R}_+ : af \asymp bf$$

Beispiel (2)

$f : x \mapsto n^3 + 5n^2$ und $g : n \mapsto 3n^3 - n$

Behauptung $f \asymp g$

Beispiel (2)

$f : x \mapsto n^3 + 5n^2$ und $g : n \mapsto 3n^3 - n$

Behauptung $f \asymp g$

- für $n \geq 0$ ist
$$\begin{aligned} f(n) &= n^3 + 5n^2 \\ &\leq n^3 + 5n^3 = 6n^3 \\ &= 9n^3 - 3n^3 \\ &\leq 9n^3 - 3n \\ &= 3(3n^3 - n) = 3g(n) \end{aligned}$$
also $\frac{1}{3} f(n) \leq g(n)$
- Andererseits ist
$$g(n) = 3n^3 - n \leq 3n^3 \leq 3(n^3 + 5n^2) = 3f(n)$$

Nichtbeispiele für $\asymp$ (1)

betrachte $f : n \mapsto n^2$ und $g : n \mapsto n^3$

Behauptung: $f \not\asymp g$

Begründung:

- sonst $g(n) \leq c' f(n)$ (für …)

Nichtbeispiele für $\asymp$ (1)

betrachte $f : n \mapsto n^2$ und $g : n \mapsto n^3$

Behauptung: $f \not\asymp g$

Begründung:

- sonst $g(n) \leq c' f(n)$ (für ...)
- für $f(n) \neq 0$ äquivalent zu $g(n)/f(n) \leq c' \in \mathbb{R}_+$

Nichtbeispiele für $\asymp$ (1)

betrachte $f : n \mapsto n^2$ und $g : n \mapsto n^3$

Behauptung: $f \not\asymp g$

Begründung:

- sonst $g(n) \leq c' f(n)$ (für ...)
- für $f(n) \neq 0$ äquivalent zu $g(n)/f(n) \leq c' \in \mathbb{R}_+$
- ab einem n_0 für jedes n

Nichtbeispiele für $\asymp$ (1)

betrachte $f : n \mapsto n^2$ und $g : n \mapsto n^3$

Behauptung: $f \not\asymp g$

Begründung:

- sonst $g(n) \leq c' f(n)$ (für …)
- für $f(n) \neq 0$ äquivalent zu $g(n)/f(n) \leq c' \in \mathbb{R}_+$
- ab einem n_0 für jedes n
- aber $g(n)/f(n) = n$ nicht beschränkbar

Nichtbeispiele für $\asymp$ (2)

betrachte $f : n \mapsto n^2$ und $g : n \mapsto 2^n$

Behauptung: $f \not\asymp g$

Begründung:

- für $f \asymp g$ müsste insbesondere $g(n)/f(n) \leq c'$ gelten (für …)
- Aber $2^n/n^2$ kann durch keine Konstante beschränkt werden:
  - einfache Grenzwertbetrachtung, oder
  - betrachte die $n_i = 2^{i+2}$ und zeige durch Induktion:

$$\forall i \in \mathbb{N}_0 : 2^{n_i} \geq 4^i n_i^2$$

Äquivalenzrelation $\asymp$

Zeichen $\asymp$ erinnert an das Gleichheitszeichen.

Lemma

Die Relation $\asymp$ ist eine Äquivalenzrelation.

Erinnerung: Eine Äquivalenzrelation ist per definitionem

- reflexiv,
- symmetrisch,
- transitiv.

Relation $\asymp$ ist reflexiv und symmetrisch

Reflexitvität: $f \asymp f$
für $c = c' = 1$ und $n_0 = 0$ gilt:
für jedes $n \geq n_0$ ist $cf(n) \leq f(n) \leq c'f(n)$

Symmetrie: Wenn $f \asymp g$, dann auch $g \asymp f$
Wenn für Konstanten $c, c' \in \mathbb{R}_+$, $n_0 \in \mathbb{N}_0$ und jedes $n \geq n_0$

$$cf(n) \leq g(n) \leq c'f(n) \ ,$$

dann gilt für die gleichen $n \geq n_0$ und
die Konstanten $d = 1/c$ und $d' = 1/c'$:

$$d'g(n) \leq f(n) \leq dg(n) \ .$$

Relation $\asymp$ ist reflexiv und symmetrisch

Reflexitvität: $f \asymp f$
für $c = c' = 1$ und $n_0 = 0$ gilt:
für jedes $n \geq n_0$ ist $cf(n) \leq f(n) \leq c'f(n)$

Symmetrie: Wenn $f \asymp g$, dann auch $g \asymp f$
Wenn für Konstanten $c, c' \in \mathbb{R}_+$, $n_0 \in \mathbb{N}_0$ und jedes $n \geq n_0$

$$cf(n) \leq g(n) \leq c'f(n) \ ,$$

dann gilt für die gleichen $n \geq n_0$ und
die Konstanten $d = 1/c$ und $d' = 1/c'$:

$$d'g(n) \leq f(n) \leq dg(n) \ .$$

Relation $\asymp$ ist reflexiv und symmetrisch

Reflexitvität: $f \asymp f$
für $c = c' = 1$ und $n_0 = 0$ gilt:
für jedes $n \geq n_0$ ist $cf(n) \leq f(n) \leq c'f(n)$

Symmetrie: Wenn $f \asymp g$, dann auch $g \asymp f$
Wenn für Konstanten $c, c' \in \mathbb{R}_+$, $n_0 \in \mathbb{N}_0$ und jedes $n \geq n_0$

$$cf(n) \leq g(n) \leq c'f(n) \ ,$$

dann gilt für die gleichen $n \geq n_0$ und
die Konstanten $d = 1/c$ und $d' = 1/c'$:

$$d'g(n) \leq f(n) \leq dg(n) \ .$$

Relation $\asymp$ ist transitiv

Transitivität: wenn $f \asymp g$ und $g \asymp h$, dann $f \asymp h$.
gelte für Konstanten $c, c' \in \mathbb{R}_+$ und jedes $n \geq n_0$

$$cf(n) \leq g(n) \leq c'f(n)$$

und für Konstanten $d, d' \in \mathbb{R}_+$ und jedes $n \geq n_1$

$$dg(n) \leq h(n) \leq d'g(n) \ .$$

Dann gilt für jedes $n \geq \max(n_0, n_1)$

$$dcf(n) \leq dg(n) \leq h(n) \leq d'g(n) \leq d'c'f(n) \ ,$$

wobei auch die Konstanten dc und $d'c'$ wieder positiv sind.

Groß-Θ

$\Theta(f)$: Menge aller Funktionen, die zu gegebenem f im Sinne von $\asymp$ äquivalent sind

Also:

$$\begin{aligned}\Theta(f) &= \left\{ g \;\middle|\; f \asymp g \right\} \\ &= \Big\{ g \;\Big|\; \exists c, c' \in \mathbb{R}_+ : \exists n_0 \in \mathbb{N}_0 : \forall n \geq n_0 : \\ &\qquad\qquad c f(n) \leq g(n) \leq c' f(n) \Big\}\end{aligned}$$

einfache Rechenregel für Θ

Rechenregel

Für alle $f : \mathbb{N}_0 \to \mathbb{R}_0^+$ und jedes Konstanten $a, b \in \mathbb{R}_+$ gilt:

$$\Theta(af) = \Theta(bf)$$

Wo sind wir?

Obere und untere Schranken

manchmal für Funktion nur obere (oder/und untere) Schranke bekannt

- Erinnerung: Anzahl Schleifendurchläufe bei Insertionsort

Definition:

- $O(f) = \{g \mid \exists c \in \mathbb{R}_+ : \exists n_0 \in \mathbb{N}_0 : \forall n \geq n_0 : g(n) \leq c f(n)\}$
- $\Omega(f) = \{g \mid \exists c \in \mathbb{R}_+ : \exists n_0 \in \mathbb{N}_0 : \forall n \geq n_0 : g(n) \geq c f(n)\}$

gelegentlich auch

- $g \preceq f$ falls $g \in O(f)$
- $g \succeq f$ falls $g \in \Omega(f)$

Redeweise

- g wächst asymptotisch höchstens so schnell wie f (für $g \preceq f$)
- g wächst asymptotisch mindestens so schnell wie f (für $g \succeq f$)

Beispiel (1)

Es sei $g\colon n \mapsto 10^{90} n^7$ und $f\colon n \mapsto 10^{-90} n^8$

Behauptung: $g \in O(f)$

Begründung:

- wähle $c = 10^{180}$ und $n_0 = 0$
- dann für jedes $n \geq 0$: $10^{90} n^7 \leq c \cdot 10^{-90} n^8$.

in $O(\cdot)$ usw. können *große* Konstanten stecken

Beispiel (2)

Was ist $O(1)$?

Beispiel (2)

Was ist $O(1)$?

1 steht für die konstante Funktion $\mathbb{1} : n \mapsto 1$

Definition: alle Funktionen g, für die es $c \in \mathbb{R}_+$ und $n_0 \in \mathbb{N}_0$ gibt, so dass für jedes $n \geq n_0$ gilt:

$$g(n) \leq c \cdot 1 = c$$

alle Funktionen, die durch Konstante beschränkbar sind

- alle konstanten Funktionen
- aber auch Funktionen wie $3 + \sin(n)$

Beispiel (3)

n^2/n nicht für große n durch Konstante beschränkbar

also gilt *nicht* $n^2 \preceq n$

aber $n \preceq n^2$

Relation $\preceq$ ist also *nicht* symmetrisch

für alle positive reelle Konstanten $0 < a < b$ gilt

$$
\begin{array}{lllll}
 & n^a \preceq n^b & \text{aber} & n^b \not\preceq n^a \\
\text{also} & n^a \in O\left(n^b\right) & \text{aber} & n^b \notin O\left(n^a\right) \\
\text{also} & n^b \in \Omega\left(n^a\right) & \text{aber} & n^a \notin \Omega\left(n^b\right)
\end{array}
$$

Beispiel (4)

$2^n/n^2$ nicht für große n durch Konstante beschränkbar

also gilt *nicht* $2^n \preceq n^2$.

aber $n^2 \preceq 2^n$.

für alle reellen Konstanten a und b, beide echt größer 1, gilt

	$n^a \preceq b^n$	aber	$b^n \not\preceq n^a$
also	$n^a \in O(b^n)$	aber	$b^n \notin O(n^a)$
also	$b^n \in \Omega(n^a)$	aber	$n^a \notin \Omega(b^n)$

Einfache Beobachtungen

in Ungleichung $g(n) \leq cf(n)$ Konstante
auf die andere Seite bringen liefert

Rechenregel

Für alle Funktionen $f : \mathbb{N}_0 \to \mathbb{R}_0^+$ und $g : \mathbb{N}_0 \to \mathbb{R}_0^+$ gilt:
$g \in O(f) \longleftrightarrow f \in \Omega(g)$ also $g \preceq f \longleftrightarrow f \succeq g$

Man kann auch zeigen:

$$\Theta(f) = O(f) \cap \Omega(f)$$

also $$g \asymp f \longleftrightarrow g \preceq f \wedge g \succeq f$$

Wo sind wir?

Für die Lektüre leider unverzichtbar

sehr unschöne Variante der O-Notation, aber weit verbreitet

Man schreibt

$$g(n) = O(f(n)) \quad \text{statt} \quad g(n) \in O(f(n)) \ ,$$
$$g(n) = \Theta(f(n)) \quad \text{statt} \quad g(n) \in \Theta(f(n)) \ ,$$
$$g(n) = \Omega(f(n)) \quad \text{statt} \quad g(n) \in \Omega(f(n)) \ .$$

Ausdrücke auf der linken Seite sind keine Gleichungen!

Lassen Sie daher bitte immer große Vorsicht walten:

Für die Lektüre leider unverzichtbar

sehr unschöne Variante der O-Notation, aber weit verbreitet

Man schreibt

$$g(n) = O(f(n)) \quad \text{statt} \quad g(n) \in O(f(n)) \ ,$$
$$g(n) = \Theta(f(n)) \quad \text{statt} \quad g(n) \in \Theta(f(n)) \ ,$$
$$g(n) = \Omega(f(n)) \quad \text{statt} \quad g(n) \in \Omega(f(n)) \ .$$

Ausdrücke auf der linken Seite sind keine Gleichungen!

Lassen Sie daher bitte immer große Vorsicht walten:

- Es ist falsch, aus $g(n) = O(f_1(n))$ und $g(n) = O(f_2(n))$ zu folgern, dass $O(f_1(n)) = O(f_2(n))$ ist.
- Es ist falsch, aus $g_1(n) = O(f(n))$ und $g_2(n) = O(f(n))$ zu folgern, dass $g_1(n) = g_2(n)$ ist.

Wo sind wir?

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

- Ignorieren konstanter Faktoren
- Notation für obere und untere Schranken des Wachstums
- Eine grauenhafte Schreibweise
- Rechnen im O-Kalkül

Matrixmultiplikation

Asymptotisches Verhalten implizit definierter Funktionen

Eine nützliche Rechenregel

Ist $g_1 \preceq f_1$ und $g_2 \preceq f_2$, dann ist auch $g_1 + g_2 \preceq f_1 + f_2$.

Ist umgekehrt $g \preceq f_1 + f_2$, dann kann man g in der Form $g = g_1 + g_2$ schreiben mit $g_1 \preceq f_1$ und $g_2 \preceq f_2$.

Das schreiben wir auch so:

Lemma

Für alle Funktionen $f_1, f_2 : \mathbb{N}_0 \to \mathbb{R}_0^+$ gilt:

$$O(f_1) + O(f_2) = O(f_1 + f_2)$$

Pluszeichen auf der linken Seite bedarf der Erläuterung ...

Komplexoperationen

Sind M_1 und M_2 Mengen, deren Elemente man addieren bzw. multiplizieren kann, dann sei

$$M_1 + M_2 = \{g_1 + g_2 \mid g_1 \in M_1 \wedge g_2 \in M_2\}$$
$$M_1 \cdot M_2 = \{g_1 \cdot g_2 \mid g_1 \in M_1 \wedge g_2 \in M_2\}$$

nichts Neues: analoge Definition des Produkts formaler Sprachen

Komplexoperationen (2)

Wenn eine der Mengen M_i einelementig ist, lässt man manchmal die Mengenklammern darum weg.

Beispiele

- mit Zahlenmengen

  statt $\{3\} \cdot \mathbb{N}_0 + \{1\}$ kürzer $3\mathbb{N}_0 + 1$

- mit Funktionenmengen

  statt $\left\{n^3\right\} + O\left(n^2\right)$ kürzer $n^3 + O\left(n^2\right)$

Beweis von $O(f_1) + O(f_2) = O(f_1 + f_2)$

„$\subseteq$“: wenn für $n \geq n_{01}$ gilt: $g_1(n) \leq c_1 f_1(n)$ und
wenn für $n \geq n_{02}$ gilt: $g_2(n) \leq c_2 f_2(n)$, dann
für $n \geq n_0 = \max(n_{01}, n_{02})$ und $c = \max(c_1, c_2)$:

$$\begin{aligned} g_1(n) + g_2(n) &\leq c_1 f_1(n) + c_2 f_2(n) \\ &\leq c f_1(n) + c f_2(n) \\ &= c(f_1(n) + f_2(n)) \end{aligned}$$

„$\supseteq$“: zu $g \in O(f_1 + f_2)$ finde g_1 und g_2 …:

Beweis von $O(f_1) + O(f_2) = O(f_1 + f_2)$

„$\subseteq$“: wenn für $n \geq n_{01}$ gilt: $g_1(n) \leq c_1 f_1(n)$ und
wenn für $n \geq n_{02}$ gilt: $g_2(n) \leq c_2 f_2(n)$, dann
für $n \geq n_0 = \max(n_{01}, n_{02})$ und $c = \max(c_1, c_2)$:

$$\begin{aligned} g_1(n) + g_2(n) &\leq c_1 f_1(n) + c_2 f_2(n) \\ &\leq c f_1(n) + c f_2(n) \\ &= c(f_1(n) + f_2(n)) \end{aligned}$$

„$\supseteq$“: zu $g \in O(f_1 + f_2)$ finde g_1 und g_2 …:

$$\text{definiere} \qquad g_1(n) = \begin{cases} g(n) & \text{falls } g(n) \leq c f_1(n) \\ c f_1(n) & \text{falls } g(n) > c f_1(n) \end{cases}$$

$$\text{und} \qquad g_2(n) = g(n) - g_1(n)$$

Der Rest ist einfache Rechnung.

Weitere Regeln

Rechenregel

Wenn $g_1 \preceq f_1$ ist, und wenn $g_1 \asymp g_2$ und $f_1 \asymp f_2$,
dann gilt auch $g_2 \preceq f_2$.

Rechenregel

Wenn $g \preceq f$ ist, also $g \in O(f)$,
dann ist auch $O(g) \subseteq O(f)$ und $O(g + f) = O(f)$.

Was ist wichtig

Das sollten Sie mitnehmen:

- Definitionen von $O(f)$, $\Theta(f)$, $\Omega(f)$
- Definitionen von $\asymp$, $\preceq$, $\succeq$

Das sollten Sie üben:

- Anschauung für $O(f)$, $\Theta(f)$, $\Omega(f)$
- rechnen mit $O(f)$, $\Theta(f)$, $\Omega(f)$

Wo sind wir?

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

Matrixmultiplikation

- Rückblick auf die Schulmethode
- Algorithmus von Strassen

Asymptotisches Verhalten implizit definierter Funktionen

Wo sind wir?

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

Matrixmultiplikation

- Rückblick auf die Schulmethode
- Algorithmus von Strassen

Asymptotisches Verhalten implizit definierter Funktionen

Multiplikation von 2×2-Matrizen

		b_{11}	b_{12}
		b_{21}	b_{22}
a_{11}	a_{12}	$a_{11}b_{11} + a_{12}b_{21}$	$a_{11}b_{12} + a_{12}b_{22}$
a_{21}	a_{22}	$a_{21}b_{11} + a_{22}b_{21}$	$a_{21}b_{12} + a_{22}b_{22}$

Anzahl Multiplikationen: $N_{\mathsf{mult}}(2) = 2^2 \cdot 2 = 8$

Anzahl Additionen: $N_{\mathsf{add}}(2) = 2^2 \cdot (2 - 1) = 4.$

Multiplikation von $n \times n$-Matrizen mit Blockaufteilung

n sei gerade

Verwendung von Blockmatrizen der Größe $n/2 \times n/2$

		B_{11}	B_{12}
		B_{21}	B_{22}
A_{11}	A_{12}	$A_{11}B_{11} + A_{12}B_{21}$	$A_{11}B_{12} + A_{12}B_{22}$
A_{21}	A_{22}	$A_{21}B_{11} + A_{22}B_{21}$	$A_{21}B_{12} + A_{22}B_{22}$

Mitteilung: Das ist auch die Produktmatrix!

8 Multiplikationen von Blockmatrizen und
4 Additionen von Blockmatrizen.

Anzahl elementarer Operationen

- $N_{\text{mult}}(n) = 8 \cdot N_{\text{mult}}(n/2)$ und
- $N_{\text{add}}(n) = 8 \cdot N_{\text{add}}(n/2) + 4 \cdot (n/2)^2 = 8 \cdot N_{\text{add}}(n/2) + n^2$.

Multiplikation von $2^k \times 2^k$-Matrizen ($n \neq 2^k$ «ähnlich»)

aus $N_{\text{mult}}(n) = 8 \cdot N_{\text{mult}}(n/2)$ folgt (Induktion)

$$\begin{aligned} N_{\text{mult}}(2^k) &= 8 \cdot N_{\text{mult}}(2^{k-1}) = 8 \cdot 8 \cdot N_{\text{mult}}(2^{k-2}) = \cdots \\ &= 8^k \cdot N_{\text{mult}}(1) \\ &= 8^k = (2^3)^k = (2^k)^3 = n^3 \end{aligned}$$

Aus $N_{\text{add}}(n) = 8 \cdot N_{\text{add}}(n/2) + n^2$ folgt

$$\begin{aligned} N_{\text{add}}(2^k) &= 8 \cdot N_{\text{add}}(2^{k-1}) + 4^k \\ &= 8 \cdot 8 \cdot N_{\text{add}}(2^{k-2}) + 8 \cdot 4^{k-1} + 4^k = \cdots \\ &= 8 \cdot 8 \cdot N_{\text{add}}(2^{k-2}) + 2 \cdot 4^k + 4^k = \cdots \\ &= 8^k N_{\text{add}}(2^0) + (2^{k-1} + \cdots + 1) \cdot 4^k = \\ &= 2^k \cdot 4^k - 4^k = n^3 - n^2 \end{aligned}$$

Multiplikation von $2^k \times 2^k$-Matrizen — nichts neues

Das wussten wir doch schon:

- $N_{\text{mult}}(n) = n^3$
- $N_{\text{add}}(n) = n^3 - n^2$

und?

Wo sind wir?

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

Matrixmultiplikation

- Rückblick auf die Schulmethode
- Algorithmus von Strassen

Asymptotisches Verhalten implizit definierter Funktionen

Die Idee von Volker Strassen (1969)

		B_{11}	B_{12}
		B_{21}	B_{22}
A_{11}	A_{12}	C_{11}	C_{12}
A_{21}	A_{22}	C_{21}	C_{22}

Einträge C_{ij} des Produktes ergeben sich auch so:

$$C_{11} = M_1 + M_4 - M_5 + M_7 \qquad C_{12} = M_3 + M_5$$
$$C_{21} = M_2 + M_4 \qquad C_{22} = M_1 - M_2 + M_3 + M_6$$

wobei
$$M_1 = (A_{11} + A_{22})(B_{11} + B_{22}) \qquad M_2 = (A_{21} + A_{22})B_{11}$$
$$M_3 = A_{11}(B_{12} - B_{22}) \qquad M_4 = A_{22}(B_{21} - B_{11})$$
$$M_5 = (A_{11} + A_{12})B_{22} \qquad M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

Die Idee von Strassen (2)

$$C_{11} = M_1 + M_4 - M_5 + M_7 \qquad C_{12} = M_3 + M_5$$
$$C_{21} = M_2 + M_4 \qquad C_{22} = M_1 - M_2 + M_3 + M_6$$

wobei
$$M_1 = (A_{11} + A_{22})(B_{11} + B_{22}) \qquad M_2 = (A_{21} + A_{22})B_{11}$$
$$M_3 = A_{11}(B_{12} - B_{22}) \qquad M_4 = A_{22}(B_{21} - B_{11})$$
$$M_5 = (A_{11} + A_{12})B_{22} \qquad M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

18 Additionen statt 4

7 Multiplikationen statt 8

Die Idee von Strassen (3)

18 Additionen statt 4

7 Multiplikationen statt 8

ja und?

Die Idee von Strassen (3)

18 Additionen statt 4

7 Multiplikationen statt 8

ja und?

ganze Block*matrizen*:
Additionen sind *viel* «billiger» als Multiplikationen

Aufwandsabschätzung für den Algorithmus von Strassen

Anzahl elementarer Operationen:

- $N_{\text{mult}}(n) = 7 \cdot N_{\text{mult}}(n/2)$
- $N_{\text{add}}(n) = 7 \cdot N_{\text{add}}(n/2) + 18 \cdot (n/2)^2 = 7 \cdot N_{\text{add}}(n/2) + 4.5 \cdot n^2$

Für den Fall $n = 2^k$ ergibt sich:

$$N_{\text{mult}}(2^k) = 7 \cdot N_{\text{mult}}(2^{k-1}) = 7 \cdot 7 \cdot N_{\text{mult}}(2^{k-2}) = \cdots$$
$$= 7^k = (2^{\log_2 7})^k = n^{\log_2 7} \approx n^{2.807\ldots}$$

analog $N_{\text{add}}(n) \in \Theta(n^{\log_2 7})$

Aufwandsabschätzung für den Algorithmus von Strassen

Anzahl elementarer Operationen:

- $N_{\text{mult}}(n) = 7 \cdot N_{\text{mult}}(n/2)$
- $N_{\text{add}}(n) = 7 \cdot N_{\text{add}}(n/2) + 18 \cdot (n/2)^2 = 7 \cdot N_{\text{add}}(n/2) + 4.5 \cdot n^2$

Für den Fall $n = 2^k$ ergibt sich:

$$N_{\text{mult}}(2^k) = 7 \cdot N_{\text{mult}}(2^{k-1}) = 7 \cdot 7 \cdot N_{\text{mult}}(2^{k-2}) = \cdots$$
$$= 7^k = (2^{\log_2 7})^k = n^{\log_2 7} \approx n^{2.807\ldots}$$

analog $N_{\text{add}}(n) \in \Theta(n^{\log_2 7})$

Aufwandsabschätzung für den Algorithmus von Strassen

Anzahl elementarer Operationen:

- $N_{\text{mult}}(n) = 7 \cdot N_{\text{mult}}(n/2)$
- $N_{\text{add}}(n) = 7 \cdot N_{\text{add}}(n/2) + 18 \cdot (n/2)^2 = 7 \cdot N_{\text{add}}(n/2) + 4.5 \cdot n^2$

Für den Fall $n = 2^k$ ergibt sich:

$$N_{\text{mult}}(2^k) = 7 \cdot N_{\text{mult}}(2^{k-1}) = 7 \cdot 7 \cdot N_{\text{mult}}(2^{k-2}) = \cdots$$
$$= 7^k = (2^{\log_2 7})^k = n^{\log_2 7} \approx n^{2.807\ldots}$$

analog $N_{\text{add}}(n) \in \Theta(n^{\log_2 7})$

Gesamtzahl elementarer Operationen ist also in

$$\Theta(n^{\log_2 7}) + \Theta(n^{\log_2 7}) = \Theta(n^{\log_2 7}) \approx \Theta(n^{2.807\ldots})$$

besser als n^3 für große n !

Matrizenmultiplikation — geht es noch schneller?

Coppersmith und Winograd (1990): $O\left(n^{2.376\ldots}\right)$

- der konstante Faktor ist *groß*

Vassilevska Williams (2012): noch ein bisschen schneller

unklar: Genügen $O\left(n^2\right)$ Operationen?

Teile und herrsche — engl. divide and conquer

«zerhacke» Probleminstanz in kleinere Teile

- mehr oder weniger viele

bearbeite Teile rekursiv nach gleichen Verfahren

konstruiere aus Teilergebnissen das
Resultat für die ursprüngliche Eingabe

Was ist wichtig

Das sollten Sie mitnehmen:

- bei algorithmischen Problemen kann man überraschende Dinge tun
- teile und herrsche
- Rekursionsformel für Abschätzung von (z. B.) Laufzeiten

Das sollten Sie üben:

- in dieser Vorlesung: rechnen mit $O(\cdot)$, $\Theta(\cdot)$, $\Omega(\cdot)$
  - spätestens in kommenden Semestern: rekursive Algorithmen

Wo sind wir?

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

Matrixmultiplikation

Asymptotisches Verhalten implizit definierter Funktionen

- Laufzeit von Teile-und-Herrsche-Algorithmen
- Ineinander geschachtelte Schleifen

Wo sind wir?

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

Matrixmultiplikation

Asymptotisches Verhalten implizit definierter Funktionen

- Laufzeit von Teile-und-Herrsche-Algorithmen
- Ineinander geschachtelte Schleifen

Laufzeit von Teile-und-Herrsche-Algorithmen

manchmal

- Problem der Größe n zerhackt
  - in konstante Anzahl a von
  - Teilproblemen gleicher Größe n/b
  - sinnvollerweise $a \geq 1$ und $b > 1$
- aus Teilergebnissen Gesamtergebnis zusammengesetzt
- Zerhacken vorher und Zusammensetzen hinterher kosten $f(n)$

Laufzeit von Teile-und-Herrsche-Algorithmen

manchmal

- Problem der Größe n zerhackt
  - in konstante Anzahl a von
  - Teilproblemen gleicher Größe n/b
  - sinnvollerweise $a \geq 1$ und $b > 1$
- aus Teilergebnissen Gesamtergebnis zusammengesetzt
- Zerhacken vorher und Zusammensetzen hinterher kosten $f(n)$

Abschätzung (z. B.) der Laufzeit $T(n)$ liefert
Rekursionsformel, die «grob gesagt» die Form hat

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Laufzeit von Teile-und-Herrsche-Algorithmen

manchmal

- Problem der Größe n zerhackt
  - in konstante Anzahl a von
  - Teilproblemen gleicher Größe n/b
  - sinnvollerweise $a \geq 1$ und $b > 1$
- aus Teilergebnissen Gesamtergebnis zusammengesetzt
- Zerhacken vorher und Zusammensetzen hinterher kosten $f(n)$

Abschätzung (z. B.) der Laufzeit $T(n)$ liefert
Rekursionsformel, die «grob gesagt» die Form hat

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

genau genommen $\lfloor n/b \rfloor$ oder $\lceil n/b + c \rceil$ oder …

Laufzeit von Teile-und-Herrsche-Algorithmen

manchmal

- Problem der Größe n zerhackt
  - in konstante Anzahl a von
  - Teilproblemen gleicher Größe n/b
  - sinnvollerweise $a \geq 1$ und $b > 1$
- aus Teilergebnissen Gesamtergebnis zusammengesetzt
- Zerhacken vorher und Zusammensetzen hinterher kosten $f(n)$

Abschätzung (z. B.) der Laufzeit $T(n)$ liefert
Rekursionsformel, die «grob gesagt» die Form hat

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

genau genommen $\lfloor n/b \rfloor$ oder $\lceil n/b + c \rceil$ oder …

gesucht: explizite Formel für $T(n)$

Mastertheorem — bescheidener hätte auch gereicht

drei Kochrezepte, in denen $f(n)$ und $\log_b a$ vorkommen:

Fall 1: Wenn $f(n) \in O\left(n^{(\log_b a)-\varepsilon}\right)$ für ein $\varepsilon > 0$ ist, dann ist $T(n) \in \Theta\left(n^{\log_b a}\right)$.

Fall 2: Wenn $f(n) \in \Theta\left(n^{\log_b a}\right)$ ist, dann ist $T(n) \in \Theta\left(n^{\log_b a} \log n\right)$.

Fall 3: Wenn $f(n) \in \Omega\left(n^{(\log_b a)+\varepsilon}\right)$ für ein $\varepsilon > 0$, und wenn es eine Konstante d gibt mit $0 < d < 1$, so dass für jedes hinreichend große n gilt $af(n/b) \leq df(n)$, dann ist $T(n) \in \Theta(f(n))$.

Achtung: Diese Fallunterscheidung ist *nicht vollständig!*

Mastertheorem — bei Multiplikation von Matrizen

Schulmethode

- $a = 8$ Multiplikationen von $n/2 \times n/2$Matrizen: also $b = 2$
- $\log_b a = 3$
- zusätzlicher Aufwand: 4 kleine Matrixadditionen, also $f(n) = 4 \cdot n^2/4 = n^2$
- $f(n) \in O\left(n^{3-\varepsilon}\right)$ (z. B. für $\varepsilon = 1$)
- Mastertheorem, Fall 1: $T(n) \in \Theta\left(n^3\right)$

Mastertheorem — bei Multiplikation von Matrizen

Schulmethode

- $a = 8$ Multiplikationen von $n/2 \times n/2$Matrizen: also $b = 2$
- $\log_b a = 3$
- zusätzlicher Aufwand: 4 kleine Matrixadditionen, also $f(n) = 4 \cdot n^2/4 = n^2$
- $f(n) \in O\left(n^{3-\varepsilon}\right)$ (z. B. für $\varepsilon = 1$)
- Mastertheorem, Fall 1: $T(n) \in \Theta\left(n^3\right)$

Algorithmus von Strassen

- $a = 7$ Multiplikationen von $n/2 \times n/2$Matrizen: also $b = 2$
- $\log_b a \approx 2.807\ldots$
- zusätzlicher Aufwand: 18 kleine Matrixadditionen, also $f(n) = 18 \cdot n^2/4 \in \Theta\left(n^2\right)$
- $f(n) \in O\left(n^{\log_b a-\varepsilon}\right) = O\left(n^{\log_2 7-\varepsilon}\right)$ (z. B. für $\varepsilon = 0.1$)
- Mastertheorem, Fall 1: $T(n) \in \Theta\left(n^{\log_2 7}\right) = \Theta\left(n^{2.807\ldots}\right)$

Wo sind wir?

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

Matrixmultiplikation

Asymptotisches Verhalten implizit definierter Funktionen

- Laufzeit von Teile-und-Herrsche-Algorithmen
- Ineinander geschachtelte Schleifen

Weitere Anwendung des Mastertheorems

$\langle$*Eingabewert* $n \in \mathbb{N}_0\rangle$
$x \leftarrow n$;
$i \leftarrow 0$;
while $x \geq 2$ **do**
    $i++$;
    $x \leftarrow x$ **div** 2;
od
$\langle$*Ausgabewert* $i\rangle$

Weitere Anwendung des Mastertheorems

$\langle$*Eingabewert* $n \in \mathbb{N}_0\rangle$
$x \leftarrow n$;
$i \leftarrow 0$;
while $x \geq 2$ **do**
  $i++$;
  $x \leftarrow x$ **div** 2;
od
$\langle$*Ausgabewert* $i\rangle$

Wert von i am Ende?

- Anzahl $D(n)$ Schleifendurchläufe

Weitere Anwendung des Mastertheorems

$\langle$*Eingabewert* $n \in \mathbb{N}_0\rangle$
$x \leftarrow n$;
$i \leftarrow 0$;
while $x \geq 2$ **do**
  $i{+}{+}$;
  $x \leftarrow x$ **div** 2;
od
$\langle$*Ausgabewert* $i\rangle$

Wert von i am Ende?

- Anzahl $D(n)$ Schleifendurchläufe

man sieht:

- $D(n) = \begin{cases} 0 & \text{falls } n \leq 1 \\ 1 + D(n/2) & \text{sonst} \end{cases}$

Weitere Anwendung des Mastertheorems

⟨*Eingabewert* $n \in \mathbb{N}_0$⟩
$x \leftarrow n$;
$i \leftarrow 0$;
while $x \geq 2$ **do**
  $i{+}{+}$;
  $x \leftarrow x$ **div** 2;
od
⟨*Ausgabewert* i⟩

Wert von i am Ende?

- Anzahl $D(n)$ Schleifendurchläufe

man sieht:

- $D(n) = \begin{cases} 0 & \text{falls } n \leq 1 \\ 1 + D(n/2) & \text{sonst} \end{cases}$

Mastertheorem:

- $a = 1$, $b = 2$, $f(n) = 1$
- $\log_b a = 0$, also Fall 2: $f(n) \in \Theta\left(n^0\right)$
- also $D(n) \in \Theta(\log n)$

Hier ist das Mastertheorem *nicht* anwendbar

$\langle$*Eingabewert* $n \in \mathbb{N}_0\rangle$
$x \leftarrow n;$
$i \leftarrow 0;$
while $x \geq 2$ **do**
  $i{+}{+};$
  $x \leftarrow x - 2;$
od
$\langle$*Ausgabewert* $i\rangle$

Hier ist das Mastertheorem *nicht* anwendbar

⟨*Eingabewert* $n \in \mathbb{N}_0$⟩
$x \leftarrow n$;
$i \leftarrow 0$;
while $x \geq 2$ **do**
  $i{+}{+}$;
  $x \leftarrow x - 2$;
od
⟨*Ausgabewert* i⟩

Anzahl Schleifendurchläufe

$$D(n) = \begin{cases} 0 & \text{falls } n \leq 1 \\ 1 + D(n-2) & \text{sonst} \end{cases}$$

Kochrezept nicht anwendbar

Einfache **for**-Schleifen

$D \leftarrow 0;$
for $i \leftarrow 0$ **to** $n-1$ **do**
  $S[i] \leftarrow V_1[i] + V_2[i]$
  $D++;$
od

Anzahl Schleifendurchläufe
offensichtlich: $D(n) = n$

Geschachtelte **for**-Schleifen

$D \leftarrow 0;$
for $i \leftarrow 0$ **to** $n-1$ **do**
  $E \leftarrow 0;$
  for $j \leftarrow 0$ **to** $n-1$ **do**
    $C[i,j] \leftarrow A[i,j] + B[i,j]$
    $E++;$
  od
  $D \leftarrow D + E;$
od

Geschachtelte **for**-Schleifen

```
D ← 0;
for i ← 0 to n − 1 do
    E ← 0;
    for j ← 0 to n − 1 do
        C[i, j] ← A[i, j] + B[i, j]
        E + +;
    od
    D ← D + E;
od
```

Anzahl Durchläufe

- innerer Schleifenrumpf

$$E(i, n) = n$$

- insgesamt

$$D(n) = \sum_{i=0}^{n-1} E(i, n) = \sum_{i=0}^{n-1} n = n^2$$

Geschachtelte **for**-Schleifen

$D \leftarrow 0;$
for $i \leftarrow 0$ **to** $n-1$ **do**
  $E \leftarrow 0;$
  for $j \leftarrow 0$ **to** i **do**
    $C[i,j] \leftarrow A[i,j] + B[i,j]$
    $E++;$
  od
  $D \leftarrow D + E;$
od

Geschachtelte **for**-Schleifen

```
D ← 0;
for i ← 0 to n − 1 do
    E ← 0;
    for j ← 0 to i do
        C[i, j] ← A[i, j] + B[i, j]
        E + +;
    od
    D ← D + E;
od
```

Anzahl Durchläufe

- innerer Schleifenrumpf

$$E(i,n) = i + 1$$

- insgesamt

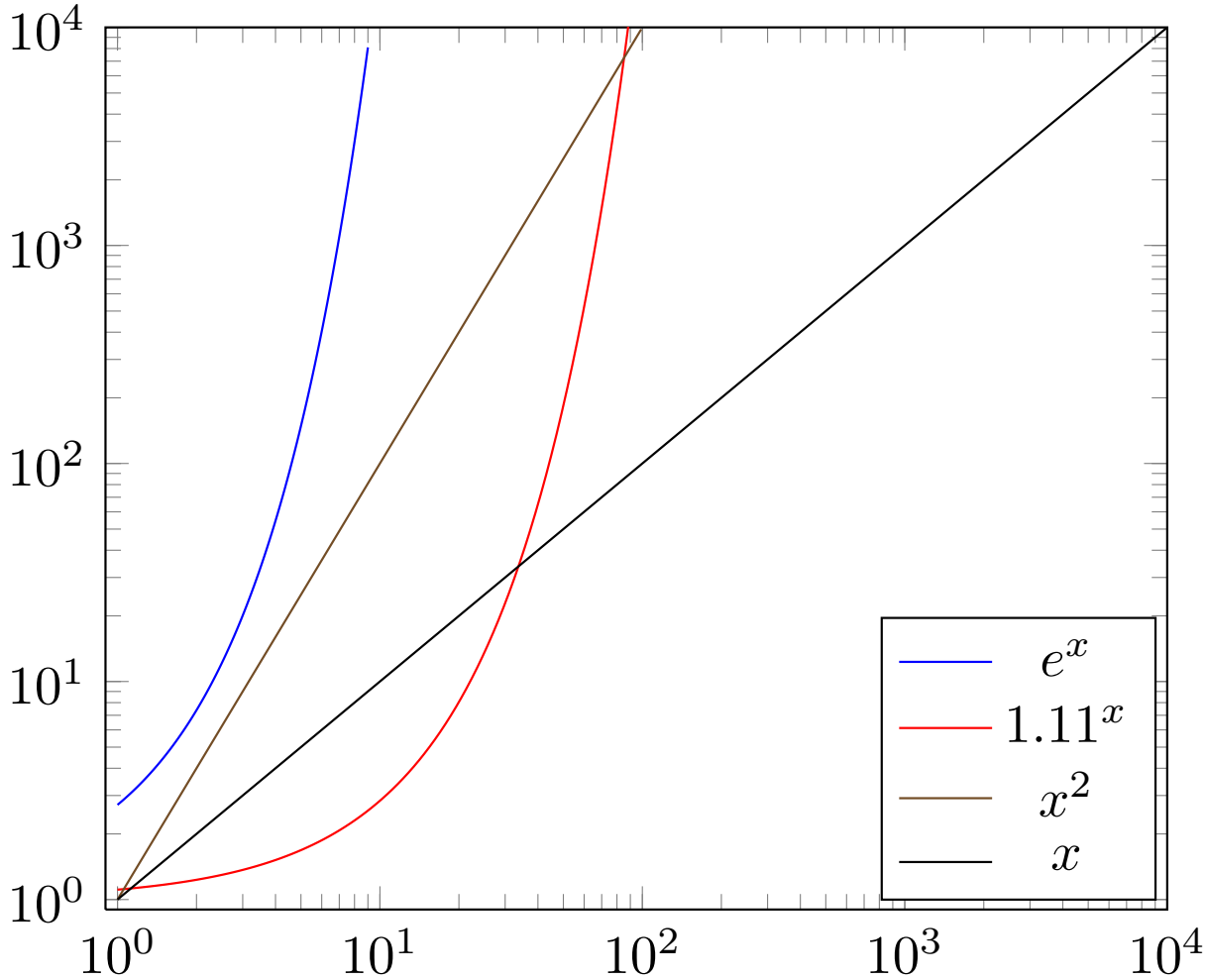
$$D(n) = \sum_{i=0}^{n-1} E(i,n) = \sum_{i=0}^{n-1} i + 1$$
$$= \frac{n(n+1)}{2} \in \Theta\left(n^2\right)$$

Rechenzeiten

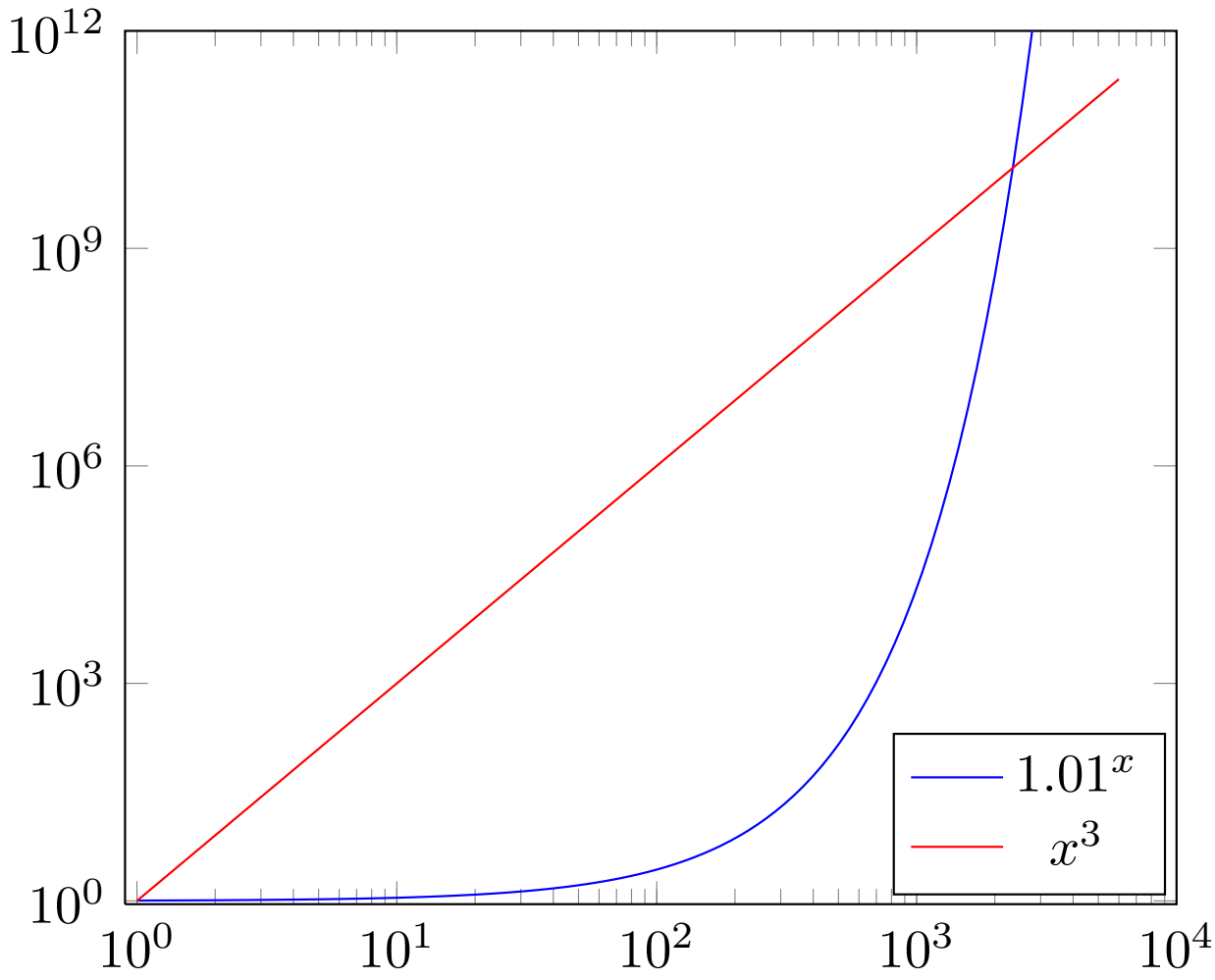
reale Rechenzeiten für $T(n)$ Schritte, wenn jeder Schritt 1 ns benötigt

$T(n)$	n					
	10	10^2	10^3	10^4	10^5	10^6
$\log_2(n)$	3.32 ns	6.64 ns	9.97 ns	13.29 ns	16.61 ns	19.93 ns
$\sqrt{n}$	3.16 ns	10.00 ns	31.62 ns	100.00 ns	316.23 ns	1.00 µs
n	10.00 ns	100.00 ns	1.00 µs	10.00 µs	100.00 µs	1.00 ms
$n \cdot \log_2(n)$	33.22 ns	664.39 ns	9.97 µs	132.88 µs	1.66 ms	19.93 ms
n^2	100.00 ns	10.00 µs	1.00 ms	100.00 ms	10.00 s	0.27 h
n^3	1.00 µs	1.00 ms	1.00 s	0.27 h	11.57 d	31.71 a
1.01^n	1.10 ns	2.70 ns	20.96 µs			
1.1^n	2.59 ns	13.78 µs				
2^n	1.02 µs					
n^n	10 s					

Rechenzeiten



Rechenzeiten (2)



Was ist wichtig

Das sollten Sie mitnehmen:

- Mastertheorem: Kochrezepte zum Nachschlagen des asymptotischen Wachstums nach einem einfachen Rezept rekursiv definierter Funktionen
- ineinandergeschachtelte Schleifen

Das sollten Sie üben:

- Mastertheorem anwenden
- Schleifen «auseinander nehmen»

Zusammenfassung

Komplexitätsmaße

- Laufzeitbedarf
- Speicherplatzbedarf

Abschätzung asymptotischen Wachstums

- «bis auf konstante Faktoren»
- nach oben mit $O(\cdot)$
- nach oben und unten mit $\Theta(\cdot)$
- nach unten mit $\Omega(\cdot)$

algorithmisches Prinzip

- Teile und Herrsche (divide and conquer)
- am Beispiel Matrixmultiplikation

Mastertheorem