

Grundbegriffe der Informatik

Einheit 13: Quantitative Aspekte von Algorithmen

Prof. Dr. Tanja Schultz

Karlsruher Institut für Technologie, Fakultät für Informatik

Wintersemester 2012/2013

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

- Ignorieren konstanter Faktoren

- Notation für obere und untere Schranken des Wachstums

- Eine furchtbare 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

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

Ignorieren konstanter Faktoren

Notation für obere und untere Schranken des Wachstums

Eine furchtbare 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

Einheit über Graphalgorithmen:

- ▶ Zählen elementarer arithmetischer Operationen
 - ▶ für Addition von $n \times n$ -Matrizen: n^2
 - ▶ für Multiplikation von $n \times n$ -Matrizen $2n^3 - n^2$
 - ▶ für Berechnung der Wegematrix $n^5 - \frac{3}{2}n^4 + \frac{3}{2}n^3 + n^2$
 - ▶ oder weniger ...
- ▶ Idee: Das hat was mit der Laufzeit der Algorithmen zu tun.

- ▶ Laufzeit/Rechenzeit
- ▶ Speicherplatzbedarf
 - ▶ insbesondere für „Zwischenergebnisse“
- ▶ das sind sogenannte *Komplexitätsmaße*
 - ▶ im Sinne von *computational complexity*
 - ▶ tauchen an vielen Stellen wieder auf
 - ▶ es gibt auch noch andere ...

- ▶ wichtiges Handwerkszeug zum
 - ▶ Reden über und
 - ▶ Ausrechnen vonz. B. Laufzeiten
- ▶ insbesondere: „kontrollierte Ungenauigkeiten“
 - ▶ „Groß-O“: stammt von Bachmann (oder früher), von Landau bekannt gemacht
 - ▶ Ω , Θ : von Knuth zumindest verbreitet
- ▶ nicht genau,
 - ▶ weil man nicht will
 - ▶ weil man nicht kannein Beispiel kommt gleich . . .

```
public class InsertionSort {  
    public static void sort(long[] a) {  
        for (int i ← 1; i < a.length; i++) {  
            insert(a, i);  
        }  
    }  
    private static void insert(long[] a, int idx) {  
        int i ← idx;  
        // Tausche a[idx] nach links bis es einsortiert ist  
        while (i > 0 ∧ a[i - 1] > a[i]) {  
            // Feldelemente a[i - 1] und a[i] vertauschen  
            a[i - 1] ↔ a[i];  
            i --;  
        }  
    }  
}
```

Wie oft wird die **while**-Schleife in der Methode *insert* ausgeführt?

- ▶ hängt von der Problemgröße $n = a.length$ ab
- ▶ aber nicht nur davon, sondern
- ▶ auch von der konkreten Probleminstance
 - ▶ Wenn Array a anfangs an sortiert:
while-Schleife wird überhaupt nicht ausgeführt.
 - ▶ Wenn Array a anfangs in entgegengesetzter Richtung sortiert:
Schleifenrumpf wird $\sum_{i=1}^{n-1} i = n(n-1)/2$ mal ausgeführt.

Wie beschreibt man den Ressourcenverbrauch?

- ▶ für jede Problemistanz einzeln:
 - ▶ präzise aber oft unpraktikabel
- ▶ gröber: nur in Abhängigkeit von der „Problemgröße“
- ▶ Frage: Was angeben, wenn Ressourcenverbrauch für verschiedene Instanzen gleicher Größe unterschiedlich?
 - ▶ 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 ...

Wie beschreibt man den Ressourcenverbrauch?

- ▶ für jede Problemistanz einzeln:
 - ▶ präzise aber oft unpraktikabel
- ▶ gröber: nur in Abhängigkeit von der „Problemgröße“
- ▶ Frage: Was angeben, wenn Ressourcenverbrauch für verschiedene Instanzen gleicher Größe unterschiedlich?
 - ▶ 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 ...

Wie beschreibt man den Ressourcenverbrauch?

- ▶ für jede Problemistanz einzeln:
 - ▶ präzise aber oft unpraktikabel
- ▶ gröber: nur in Abhängigkeit von der „Problemgröße“
- ▶ Frage: Was angeben, wenn Ressourcenverbrauch für verschiedene Instanzen gleicher Größe unterschiedlich?
 - ▶ 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 ...

Wie beschreibt man den Ressourcenverbrauch?

- ▶ für jede Problemistanz einzeln:
 - ▶ präzise aber oft unpraktikabel
- ▶ gröber: nur in Abhängigkeit von der „Problemgröße“
- ▶ Frage: Was angeben, wenn Ressourcenverbrauch für verschiedene Instanzen gleicher Größe unterschiedlich?
 - ▶ 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 ...

Das sollten Sie mitnehmen:

- ▶ Bedarf an
 - ▶ Rechenzeit und
 - ▶ Speicherplatzbedarf

wichtige **Komplexitätsmaße**

- ▶ Meist will/kann man nur die Abhängigkeit von der Problemgröße quantifizieren
 - ▶ üblicherweise den schlimmsten Fall (**worst case**)
 - ▶ gelegentlich einen mittleren Fall (average case)

Das sollten Sie üben:

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

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

- Ignorieren konstanter Faktoren

- Notation für obere und untere Schranken des Wachstums

- Eine furchtbare 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

Warum keine exakten Angaben?

- ▶ Man will nicht.
 - ▶ Faulheit
 - ▶ Vergänglichkeit der genauen Werte
 - ▶ Prozessor bald höher getaktet
 - ▶ Prozessor bald mit schnellerer Architektur
 - ▶ mangelndes Interesse an genauen Werten
 - ▶ will nur prozessorunabhängige Aussagen
- ▶ Man kann nicht.
 - ▶ Dummheit
 - ▶ Unwissenheit der genauen Randbedingungen
 - ▶ welcher Prozessor?
 - ▶ Ungenauigkeiten bei der Formulierung des Algorithmus
 - ▶ unabhängig von Programmiersprache
- ▶ Man „soll“ nicht.
 - ▶ nur vergrößernde Angaben in Abhängigkeit von Problemgröße

Warum keine exakten Angaben?

- ▶ Man will nicht.
 - ▶ Faulheit
 - ▶ Vergänglichkeit der genauen Werte
 - ▶ Prozessor bald höher getaktet
 - ▶ Prozessor bald mit schnellerer Architektur
 - ▶ mangelndes Interesse an genauen Werten
 - ▶ will nur prozessorunabhängige Aussagen
- ▶ Man kann nicht.
 - ▶ Dummheit
 - ▶ Unwissenheit der genauen Randbedingungen
 - ▶ welcher Prozessor?
 - ▶ Ungenauigkeiten bei der Formulierung des Algorithmus
 - ▶ unabhängig von Programmiersprache
- ▶ Man „soll“ nicht.
 - ▶ nur vergrößernde Angaben in Abhängigkeit von Problemgröße

Warum keine exakten Angaben?

- ▶ Man will nicht.
 - ▶ Faulheit
 - ▶ Vergänglichkeit der genauen Werte
 - ▶ Prozessor bald höher getaktet
 - ▶ Prozessor bald mit schnellerer Architektur
 - ▶ mangelndes Interesse an genauen Werten
 - ▶ will nur prozessorunabhängige Aussagen
- ▶ Man kann nicht.
 - ▶ Dummheit
 - ▶ Unwissenheit der genauen Randbedingungen
 - ▶ welcher Prozessor?
 - ▶ Ungenauigkeiten bei der Formulierung des Algorithmus
 - ▶ unabhängig von Programmiersprache
- ▶ Man „soll“ nicht.
 - ▶ nur vergrößernde Angaben in Abhängigkeit von Problemgröße

Warum keine exakten Angaben?

- ▶ Man will nicht.
 - ▶ Faulheit
 - ▶ Vergänglichkeit der genauen Werte
 - ▶ Prozessor bald höher getaktet
 - ▶ Prozessor bald mit schnellerer Architektur
 - ▶ mangelndes Interesse an genauen Werten
 - ▶ will nur prozessorunabhängige Aussagen
- ▶ Man kann nicht.
 - ▶ Dummheit
 - ▶ Unwissenheit der genauen Randbedingungen
 - ▶ welcher Prozessor?
 - ▶ Ungenauigkeiten bei der Formulierung des Algorithmus
 - ▶ unabhängig von Programmiersprache
- ▶ Man „soll“ nicht.
 - ▶ nur vergrößernde Angaben in Abhängigkeit von Problemgröße

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

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

- Ignorieren konstanter Faktoren

- Notation für obere und untere Schranken des Wachstums

- Eine furchtbare 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

► Notation:

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

► Redeweisen:

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

► Definition:

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

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

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

► Notation:

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

► Redeweisen:

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

► Definition:

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

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

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

► Notation:

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

► Redeweisen:

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

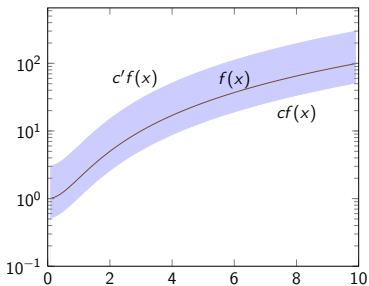
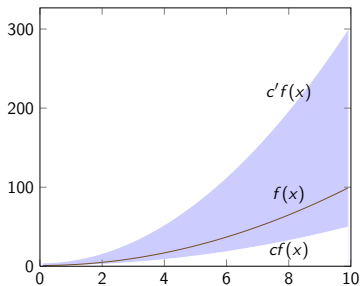
► Definition:

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

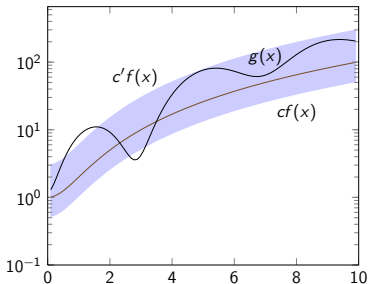
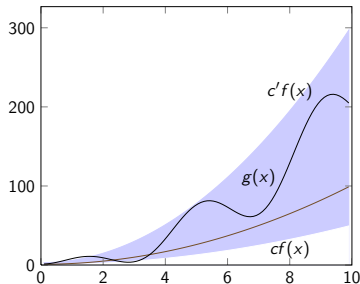
$$\exists c, c' \in \mathbb{R}_+ : \exists n_0 \in \mathbb{N}_0 : \forall n \geq n_0 : cf(n) \leq g(n) \leq c'f(n) .$$

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

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



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



► $f(n) = 3n^2$ und $g(n) = 10^{-2}n^2$.

► Behauptung: $f(n) \asymp g(n)$

► $cf(n) \leq g(n)$: 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 $n_0 = 0$:

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

Das lässt sich leicht etwas allgemeiner rechnen. Dann sieht man:

Rechenregel

Für alle $f : \mathbb{N}_0 \rightarrow \mathbb{R}_0^+$ gilt:

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

► $f(n) = 3n^2$ und $g(n) = 10^{-2}n^2$.

► Behauptung: $f(n) \asymp g(n)$

► $cf(n) \leq g(n)$: 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 $n_0 = 0$:

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

Das lässt sich leicht etwas allgemeiner rechnen. Dann sieht man:

Rechenregel

Für alle $f : \mathbb{N}_0 \rightarrow \mathbb{R}_0^+$ gilt:

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

► $f(n) = 3n^2$ und $g(n) = 10^{-2}n^2$.

► Behauptung: $f(n) \asymp g(n)$

► $cf(n) \leq g(n)$: 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 $n_0 = 0$:

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

Das lässt sich leicht etwas allgemeiner rechnen. Dann sieht man:

Rechenregel

Für alle $f : \mathbb{N}_0 \rightarrow \mathbb{R}_0^+$ gilt:

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

Beispiel (2)

- ▶ $f(n) = n^3 + 5n^2$ und $g(n) = 3n^3 - n$
- ▶ Behauptung $f(n) \asymp g(n)$
 - ▶ einerseits ist für $n \geq 0$ offensichtlich

$$\begin{aligned}f(n) &= n^3 + 5n^2 \\&\leq n^3 + 5n^3 \\&= 6n^3 \\&= 9n^3 - 3n^3 \\&\leq 9n^3 - 3n \\&= 3(3n^3 - n) = 3g(n)\end{aligned}$$

also $\frac{1}{3}f(n) \leq g(n)$

- ▶ Andererseits ist

$$g(n) = 3n^3 - n \leq 3n^3 \leq 3(n^3 + 5n^2) = 3f(n)$$

Beispiel (2)

- ▶ $f(n) = n^3 + 5n^2$ und $g(n) = 3n^3 - n$
- ▶ Behauptung $f(n) \asymp g(n)$
 - ▶ einerseits ist für $n \geq 0$ offensichtlich

$$\begin{aligned} f(n) &= n^3 + 5n^2 \\ &\leq n^3 + 5n^3 \\ &= 6n^3 \\ &= 9n^3 - 3n^3 \\ &\leq 9n^3 - 3n \\ &= 3(3n^3 - n) = 3g(n) \end{aligned}$$

also $\frac{1}{3}f(n) \leq g(n)$

- ▶ Andererseits ist

$$g(n) = 3n^3 - n \leq 3n^3 \leq 3(n^3 + 5n^2) = 3f(n)$$

- ▶ betrachte $f(n) = n^2$ und $g(n) = n^3$
- ▶ Behauptung: $f \not\asymp g$
- ▶ Begründung:
 - ▶ insbesondere müsste sonst $g(n) \leq c'f(n)$ gelten (für ...)
 - ▶ für $f(n) \neq 0$ äquivalent zu $g(n)/f(n) \leq c'$
 - ▶ Das müsste also für ein $c' \in \mathbb{R}_+$ ab einem n_0 für alle n gelten.
 - ▶ Aber $g(n)/f(n) = n$ kann durch keine Konstante beschränkt werden.

- ▶ betrachte $f(n) = n^2$ und $g(n) = n^3$
- ▶ Behauptung: $f \not\asymp g$
- ▶ Begründung:
 - ▶ insbesondere müsste sonst $g(n) \leq c' f(n)$ gelten (für ...)
 - ▶ für $f(n) \neq 0$ äquivalent zu $g(n)/f(n) \leq c'$
 - ▶ Das müsste also für ein $c' \in \mathbb{R}_+$ ab einem n_0 für alle n gelten.
 - ▶ Aber $g(n)/f(n) = n$ kann durch keine Konstante beschränkt werden.

- ▶ betrachte $f(n) = n^2$ und $g(n) = n^3$
- ▶ Behauptung: $f \not\asymp g$
- ▶ Begründung:
 - ▶ insbesondere müsste sonst $g(n) \leq c' f(n)$ gelten (für ...)
 - ▶ für $f(n) \neq 0$ äquivalent zu $g(n)/f(n) \leq c'$
 - ▶ Das müsste also für ein $c' \in \mathbb{R}_+$ ab einem n_0 für alle n gelten.
 - ▶ Aber $g(n)/f(n) = n$ kann durch keine Konstante beschränkt werden.

- ▶ betrachte $f(n) = n^2$ und $g(n) = n^3$
- ▶ Behauptung: $f \not\asymp g$
- ▶ Begründung:
 - ▶ insbesondere müsste sonst $g(n) \leq c'f(n)$ gelten (für ...)
 - ▶ für $f(n) \neq 0$ äquivalent zu $g(n)/f(n) \leq c'$
 - ▶ Das müsste also für ein $c' \in \mathbb{R}_+$ ab einem n_0 für alle n gelten.
 - ▶ Aber $g(n)/f(n) = n$ kann durch keine Konstante beschränkt werden.

- ▶ betrachte $f(n) = n^2$ und $g(n) = 2^n$
- ▶ 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:

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

- ▶ Zeichen \asymp erinnert an das Gleichheitszeichen.
- ▶ Das ist Absicht: Relation \asymp hat wichtige Eigenschaften:

Lemma

Die Relation \asymp ist eine Äquivalenzrelation.

Zur Erinnerung: Eine Äquivalenzrelation ist per definitionem

- ▶ reflexiv,
- ▶ symmetrisch,
- ▶ transitiv.

- ▶ *Reflexivität*: $f \asymp f$

denn für $c = c' = 1$ und $n_0 = 0$ gilt:

für $n \geq n_0$ 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 alle $n \geq n_0$

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

dann gilt für die gleichen $n \geq n_0$ und

die Konstanten $d = 1/c$ und $d' = 1/c'$:

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

- ▶ *Reflexivität*: $f \asymp f$

denn für $c = c' = 1$ und $n_0 = 0$ gilt:

für $n \geq n_0$ 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 alle $n \geq n_0$

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

dann gilt für die gleichen $n \geq n_0$ und

die Konstanten $d = 1/c$ und $d' = 1/c'$:

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

- ▶ *Reflexivität*: $f \asymp f$

denn für $c = c' = 1$ und $n_0 = 0$ gilt:

für $n \geq n_0$ 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 alle $n \geq n_0$

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

dann gilt für die gleichen $n \geq n_0$ und

die Konstanten $d = 1/c$ und $d' = 1/c'$:

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

- ▶ *Reflexivität*: $f \asymp f$

denn für $c = c' = 1$ und $n_0 = 0$ gilt:

für $n \geq n_0$ 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 alle $n \geq n_0$

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

dann gilt für die gleichen $n \geq n_0$ und

die Konstanten $d = 1/c$ und $d' = 1/c'$:

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

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

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

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

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

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

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

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

- ▶ $\Theta(f)$: Menge aller Funktionen, die zu einer gegebenen Funktion $f(n)$ im Sinne von \asymp äquivalent sind
- ▶ Also:

$$\begin{aligned}\Theta(f(n)) &= \{g(n) \mid f(n) \asymp g(n)\} \\ &= \{g(n) \mid \exists c, c' \in \mathbb{R}_+ : \exists n_0 \in \mathbb{N}_0 : \forall n \geq n_0 : \\ &\quad cf(n) \leq g(n) \leq c'f(n)\}\end{aligned}$$

Rechenregel

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

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

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

Ignorieren konstanter Faktoren

Notation für obere und untere Schranken des Wachstums

Eine furchtbare 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

- ▶ Manchmal kennt man eine Funktion nicht mal bis auf einen konstanten Faktor, sondern nur obere (oder/und untere) Schranken.
 - ▶ Erinnerung: Anzahl Schleifendurchläufe bei Insertionsort
- ▶ Definition:
 - ▶ $O(f) = \{g \mid \exists c \in \mathbb{R}_+ : \exists n_0 \in \mathbb{N}_0 : \forall n \geq n_0 : g(n) \leq cf(n)\}$
 - ▶ $\Omega(f) = \{g \mid \exists c \in \mathbb{R}_+ : \exists n_0 \in \mathbb{N}_0 : \forall n \geq n_0 : g(n) \geq cf(n)\}$
- ▶ gelegentlich auch

$$\begin{array}{lll} g \preceq f & \text{falls} & g \in O(f) \\ g \succeq f & \text{falls} & g \in \Omega(f) \end{array}$$

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

Beispiel (1)

- ▶ Es sei $g(n) = 10^{90}n^7$ und $f(n) = 10^{-90}n^8$
- ▶ Behauptung: $g(n) \in O(f(n))$
- ▶ Begründung:
 - ▶ wähle $c = 10^{180}$ und $n_0 = 0$
 - ▶ dann für alle $n \geq 0$: $10^{90}n^7 \leq c \cdot 10^{-90}n^8$.
- ▶ Man sieht: In $O(\cdot)$ usw. können *große* Konstanten stecken.
- ▶ Deswegen: Ob z. B. Algorithmus mit Laufzeit in $O(n^8)$ in der Praxis wirklich tauglich ist, hängt durchaus von der Konstante c bei der oberen Schranke cn^8 ab.
 - ▶ $c = 10^{-90}$ dürfte okay sein,
 - ▶ $c = 10^{90}$ vermutlich nicht.

Beispiel (1)

- ▶ Es sei $g(n) = 10^{90}n^7$ und $f(n) = 10^{-90}n^8$
- ▶ Behauptung: $g(n) \in O(f(n))$
- ▶ Begründung:
 - ▶ wähle $c = 10^{180}$ und $n_0 = 0$
 - ▶ dann für alle $n \geq 0$: $10^{90}n^7 \leq c \cdot 10^{-90}n^8$.
- ▶ Man sieht: In $O(\cdot)$ usw. können *große* Konstanten stecken.
- ▶ Deswegen: Ob z. B. Algorithmus mit Laufzeit in $O(n^8)$ in der Praxis wirklich tauglich ist, hängt durchaus von der Konstante c bei der oberen Schranke cn^8 ab.
 - ▶ $c = 10^{-90}$ dürfte okay sein,
 - ▶ $c = 10^{90}$ vermutlich nicht.

- ▶ Was ist $O(1)$?
- ▶ Definition: alle Funktionen $g(n)$, für die es $c \in \mathbb{R}_+$ und $n_0 \in \mathbb{N}_0$ gibt, so dass für alle $n \geq n_0$ gilt:

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

- ▶ alle Funktionen, die durch eine Konstante beschränkbar sind
 - ▶ Dazu gehören etwa alle konstanten Funktionen,
 - ▶ aber auch Funktionen wie $3 + \sin(n)$
(So etwas habe ich aber noch nie eine Rolle spielen sehen.)

Beispiel (3)

- ▶ vorne: Quotient n^2/n nicht für alle hinreichend großen n durch eine Konstante beschränkbar
- ▶ Also gilt *nicht* $n^2 \preceq n$.
- ▶ Andererseits gilt $n \preceq n^2$.
- ▶ Die Relation \preceq ist also *nicht* symmetrisch.
- ▶ Allgemein für positive reelle Konstanten $0 < a < b$:

$$n^a \preceq n^b \quad \text{aber} \quad n^b \not\preceq n^a$$

also

$$n^a \in O(n^b) \quad \text{aber} \quad n^b \notin O(n^a)$$

also

$$n^b \in \Omega(n^a) \quad \text{aber} \quad n^a \notin \Omega(n^b)$$

Beispiel (4)

- ▶ vorne: Quotient $2^n/n^2$ nicht für alle hinreichend großen n durch eine Konstante beschränkbar
- ▶ Also gilt *nicht* $2^n \preceq n^2$.
- ▶ Andererseits gilt $n^2 \preceq 2^n$.
- ▶ Allgemein für reelle Konstanten a und b , beide echt größer 1:

$$n^a \preceq b^n \quad \text{aber} \quad b^n \not\preceq n^a$$

also

$$n^a \in O(b^n) \quad \text{aber} \quad b^n \notin O(n^a)$$

also

$$b^n \in \Omega(n^a) \quad \text{aber} \quad n^a \notin \Omega(b^n)$$

- ▶ in der Ungleichung $g(n) \leq cf(n)$ die Konstante auf die andere Seite bringen (hatten wir schon) liefert

Rechenregel

Für alle Funktionen $f : \mathbb{N}_0 \rightarrow \mathbb{R}_0^+$ und $g : \mathbb{N}_0 \rightarrow \mathbb{R}_0^+$ gilt:

$$g(n) \in O(f(n)) \iff f(n) \in \Omega(g(n)), \quad \text{also} \quad g \preceq f \iff f \succeq g$$

- ▶ Man kann auch zeigen:

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

$$\text{also} \quad g \asymp f \iff g \preceq f \wedge g \succeq f$$

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

Ignorieren konstanter Faktoren

Notation für obere und untere Schranken des Wachstums

Eine furchtbare 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

- ▶ 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.

- ▶ 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.

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

Ignorieren konstanter Faktoren

Notation für obere und untere Schranken des Wachstums

Eine furchtbare 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

- ▶ 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 $f_1, f_2 : \mathbb{N}_0 \rightarrow \mathbb{R}_0^+$ gilt:

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

Das Pluszeichen auf der linken Seite bedarf der Erläuterung ...

- Sind M_1 und M_2 Mengen von Elementen, die man addieren bzw. multiplizieren kann, dann sei

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

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

- Das ist nichts Neues: Definition des Produkts formaler Sprachen passt genau in dieses Schema.

- ▶ 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 $\{n^3\} + O(n^2)$ kürzer $n^3 + O(n^2)$

beide Inklusionen getrennt beweisen:

„ \subseteq “: das ist einfach

Wenn für alle $n \geq n_{01}$ gilt: $g_1(n) \leq c_1 f_1(n)$ und
wenn für alle $n \geq n_{02}$ gilt: $g_2(n) \leq c_2 f_2(n)$, dann
gilt 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 “: „schwieriger“, weil man g_1 und g_2 finden muss.
Definiere

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

und
$$g_2(n) = g(n) - g_1(n)$$

Der Rest ist einfache Rechnung.

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

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 $O(f)$, $\Theta(f)$, $\Omega(f)$

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

Ignorieren konstanter Faktoren

Notation für obere und untere Schranken des Wachstums

Eine furchtbare 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

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

- Ignorieren konstanter Faktoren

- Notation für obere und untere Schranken des Wachstums

- Eine furchtbare 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

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: $N_{mult}(2) = 2^2 \cdot 2 = 8$
- ▶ Anzahl Additionen: $N_{add}(2) = 2^2 \cdot (2 - 1) = 4$.

Multiplikation von $n \times n$ -Matrizen

- ▶ n sei gerade
- ▶ Verwendung von Blockmatrizen:

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

- ▶ alle Blöcke haben Größe $n/2 \times n/2$
- ▶ 8 Multiplikationen von Blockmatrizen und 4 Additionen von Blockmatrizen.
- ▶ Anzahl elementarer Operationen
 - ▶ $N_{mult}(n) = 8 \cdot N_{mult}(n/2)$ und
 - ▶ $N_{add}(n) = 8 \cdot N_{add}(n/2) + 4 \cdot (n/2)^2 = 8 \cdot N_{add}(n/2) + n^2$.

- ▶ Fälle $n \neq 2^k$ kann man mit mehr Aufwand analog behandeln.
- ▶ aus $N_{mult}(n) = 8 \cdot N_{mult}(n/2)$ folgt (Induktion)

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

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

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

- ▶ Das wussten wir doch schon:
 - ▶ $N_{mult}(n) = n^3$
 - ▶ $N_{add}(n) = 2n^3 - n^2$
- ▶ und?

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

- Ignorieren konstanter Faktoren

- Notation für obere und untere Schranken des Wachstums

- Eine furchtbare 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

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

Die Einträge C_{ij} des Matrixproduktes ergeben sich auch wie folgt:

$$C_{11} = M_1 + M_4 - M_5 + M_7$$

$$C_{12} = M_3 + M_5$$

$$C_{21} = M_2 + M_4$$

$$C_{22} = M_1 - M_2 + M_3 + M_6$$

wobei $M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$ $M_2 = (A_{21} + A_{22})B_{11}$

$$M_3 = A_{11}(B_{12} - B_{22})$$

$$M_4 = A_{22}(B_{21} - B_{11})$$

$$M_5 = (A_{11} + A_{12})B_{22}$$

$$M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$

$$M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

$$C_{11} = M_1 + M_4 - M_5 + M_7$$

$$C_{12} = M_3 + M_5$$

$$C_{21} = M_2 + M_4$$

$$C_{22} = M_1 - M_2 + M_3 + M_6$$

wobei $M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$

$$M_2 = (A_{21} + A_{22})B_{11}$$

$$M_3 = A_{11}(B_{12} - B_{22})$$

$$M_4 = A_{22}(B_{21} - B_{11})$$

$$M_5 = (A_{11} + A_{12})B_{22}$$

$$M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$

$$M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

- ▶ 18 Additionen statt 4
- ▶ 7 Multiplikationen statt 8

Die Idee von Strassen (3)

- ▶ 18 Additionen statt 4
- ▶ 7 Multiplikationen statt 8
- ▶ ja und?

- ▶ 18 Additionen statt 4
- ▶ 7 Multiplikationen statt 8
- ▶ ja und?
- ▶ ganze Blockmatrizen:
Additionen sind *viel* „billiger“ als Multiplikationen

- ▶ Anzahl elementarer Operationen:

- ▶ $N_{mult}(n) = 7 \cdot N_{mult}(n/2)$

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

- ▶ Für den Fall $n = 2^k$ ergibt sich:

$$\begin{aligned} N_{mult}(2^k) &= 7 \cdot N_{mult}(2^{k-1}) = 7 \cdot 7 \cdot N_{mult}(2^{k-2}) = \dots \\ &= 7^k \cdot N_{mult}(1) \\ &= 7^k = (2^{\log_2 7})^k = (2^k)^{\log_2 7} \approx n^{2.807\dots} \end{aligned}$$

- ▶ Analog: $N_{add}(n) \in \Theta(n^{\log_2 7})$.

- ▶ Gesamtzahl elementarer Operationen ist also in

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

das ist besser als n^3

- ▶ Ja: Algorithmus von Coppersmith und Winograd: $O(n^{2.376\dots})$
 - ▶ 2011 von Virginia Williams optimiert zu $O(n^{2.3727})$
 - ▶ der konstante Faktor ist *groß*
- ▶ unklar: Reichen $O(n^2)$ Operationen?

- ▶ Man teilt die Problemistanz in kleinere Teile auf
 - ▶ mehr oder weniger viele
- ▶ Man bearbeitet die Teile rekursiv nach dem gleichen Verfahren.
- ▶ Man benutzt die Teilergebnisse, um das Resultat für die ursprüngliche Eingabe zu berechnen.
- ▶ engl. *divide and conquer*

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

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

- Ignorieren konstanter Faktoren

- Notation für obere und untere Schranken des Wachstums

- Eine furchtbare 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

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

- Ignorieren konstanter Faktoren

- Notation für obere und untere Schranken des Wachstums

- Eine furchtbare 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

- ▶ in manchen Fällen:
 - ▶ Problem der Größe n wird in konstante Anzahl a von Teilprobleme gleicher Größe n/b zerhackt
 - ▶ sinnvollerweise $a \geq 1$ und $b > 1$
 - ▶ 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 $\lceil n/b \rceil$
 - ▶ oder gar $\lfloor n/b + c \rfloor$ oder $\lceil n/b + c \rceil$
 - ▶ Mitteilung: Das ändert nichts.
- ▶ **Gesucht:** explizite Formel für $T(n)$.

- ▶ in manchen Fällen:
 - ▶ Problem der Größe n wird in konstante Anzahl a von Teilprobleme gleicher Größe n/b zerhackt
 - ▶ sinnvollerweise $a \geq 1$ und $b > 1$
 - ▶ 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 $\lceil n/b \rceil$
 - ▶ oder gar $\lfloor n/b + c \rfloor$ oder $\lceil n/b + c \rceil$
 - ▶ Mitteilung: Das ändert nichts.
- ▶ **Gesucht:** explizite Formel für $T(n)$.

- ▶ in manchen Fällen:
 - ▶ Problem der Größe n wird in konstante Anzahl a von Teilprobleme gleicher Größe n/b zerhackt
 - ▶ sinnvollerweise $a \geq 1$ und $b > 1$
 - ▶ 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 $\lceil n/b \rceil$
 - ▶ oder gar $\lfloor n/b + c \rfloor$ oder $\lceil n/b + c \rceil$
 - ▶ Mitteilung: Das ändert nichts.
- ▶ **Gesucht:** explizite Formel für $T(n)$.

- ▶ in manchen Fällen:
 - ▶ Problem der Größe n wird in konstante Anzahl a von Teilprobleme gleicher Größe n/b zerhackt
 - ▶ sinnvollerweise $a \geq 1$ und $b > 1$
 - ▶ 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 $\lceil n/b \rceil$
 - ▶ oder gar $\lfloor n/b + c \rfloor$ oder $\lceil n/b + c \rceil$
 - ▶ Mitteilung: Das ändert nichts.
- ▶ **Gesucht:** explizite Formel für $T(n)$.

Drei Kochrezepte, in denen $f(n)$ und $\log_b a$ eine Rolle spielen:

- Fall 1: Wenn $f(n) \in O(n^{(\log_b a) - \varepsilon})$ für ein $\varepsilon > 0$ ist, dann ist $T(n) \in \Theta(n^{\log_b a})$.
- Fall 2: Wenn $f(n) \in \Theta(n^{\log_b a})$ ist, dann ist $T(n) \in \Theta(n^{\log_b a} \log n)$.
- Fall 3: Wenn $f(n) \in \Omega(n^{(\log_b a) + \varepsilon})$ für ein $\varepsilon > 0$ ist, und wenn es eine Konstante d gibt mit $0 < d < 1$, so dass für alle hinreichend großen n gilt $af(n/b) \leq df(n)$, dann ist $T(n) \in \Theta(f(n))$.

Achtung: Diese Fallunterscheidung ist *nicht vollständig!*

- ▶ „Problemgröße“ n : die Zeilen- bzw. Spaltenzahl
- ▶ Schulmethode
 - ▶ $a = 8$ Multiplikationen
 - ▶ von Matrizen der Größe $n/2$: also $b = 2$
 - ▶ $\log_b a = \log_2 8 = 3$
 - ▶ zusätzlicher Aufwand: 4 kleine Matrixadditionen, also $f(n) = 4 \cdot n^2/4 = n^2$
 - ▶ $f(n) \in O(n^{3-\varepsilon})$ (z. B. für $\varepsilon = 1$)
 - ▶ Mastertheorem, Fall 1: $T(n) \in \Theta(n^3)$
- ▶ Algorithmus von Strassen
 - ▶ $a = 7$ Multiplikationen
 - ▶ von Matrizen der Größe $n/2$: also $b = 2$
 - ▶ $\log_b a = \log_2 7 \approx 2.807 \dots$
 - ▶ zusätzlicher Aufwand: 18 kleine Matrixadditionen, also $f(n) = 18 \cdot n^2/4 \in \Theta(n^2)$
 - ▶ $f(n) \in O(n^{\log_b a - \varepsilon}) = O(n^{\log_2 7 - \varepsilon})$ (z. B. für $\varepsilon = 0.1$)
 - ▶ Mastertheorem, Fall 1: $T(n) \in \Theta(n^{\log_2 7}) = \Theta(n^{2.807\dots})$

- ▶ „Problemgröße“ n : die Zeilen- bzw. Spaltenzahl
- ▶ Schulmethode
 - ▶ $a = 8$ Multiplikationen
 - ▶ von Matrizen der Größe $n/2$: also $b = 2$
 - ▶ $\log_b a = \log_2 8 = 3$
 - ▶ zusätzlicher Aufwand: 4 kleine Matrixadditionen, also $f(n) = 4 \cdot n^2/4 = n^2$
 - ▶ $f(n) \in O(n^{3-\varepsilon})$ (z. B. für $\varepsilon = 1$)
 - ▶ Mastertheorem, Fall 1: $T(n) \in \Theta(n^3)$
- ▶ Algorithmus von Strassen
 - ▶ $a = 7$ Multiplikationen
 - ▶ von Matrizen der Größe $n/2$: also $b = 2$
 - ▶ $\log_b a = \log_2 7 \approx 2.807 \dots$
 - ▶ zusätzlicher Aufwand: 18 kleine Matrixadditionen, also $f(n) = 18 \cdot n^2/4 \in \Theta(n^2)$
 - ▶ $f(n) \in O(n^{\log_b a - \varepsilon}) = O(n^{\log_2 7 - \varepsilon})$ (z. B. für $\varepsilon = 0.1$)
 - ▶ Mastertheorem, Fall 1: $T(n) \in \Theta(n^{\log_2 7}) = \Theta(n^{2.807\dots})$

Ressourcenverbrauch bei Berechnungen

Groß-O-Notation

Ignorieren konstanter Faktoren

Notation für obere und untere Schranken des Wachstums

Eine furchtbare 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

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

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

- ▶ Welchen Wert hat am Ende i ?
- ▶ Wieviele Schleifendurchläufe $D(n)$ werden insgesamt gemacht?


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

- ▶ wie 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: $f(n) \in \Theta(n^0)$
 - ▶ also $D(n) \in \Theta(\log n)$

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

- ▶ Anzahl Schleifendurchläufe

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

- ▶ Kochrezept nicht anwendbar

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

- ▶ Anzahl Schleifendurchläufe

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

- ▶ Kochrezept nicht anwendbar

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

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

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

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

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

► Anzahl Durchläufe

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

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

► Anzahl Durchläufe

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

```
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

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


```
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
```

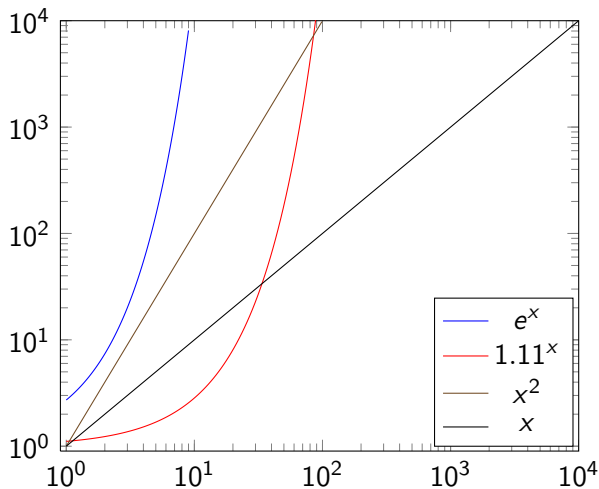
► Anzahl Durchläufe

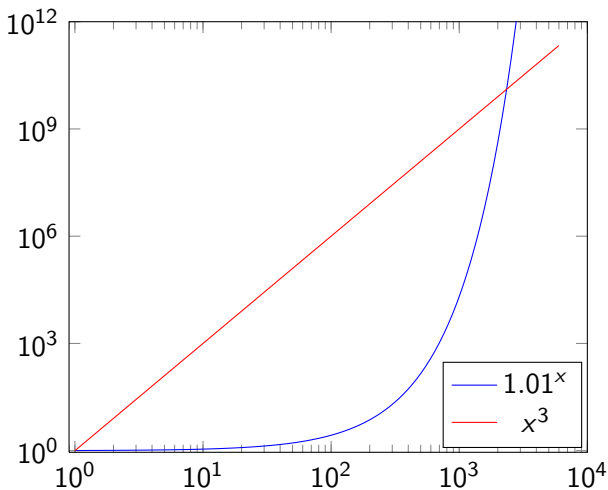
- innerer Schleifenrumpf: $E(i, n) = i + 1$
- insgesamt

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

„Wie großen Einfluss“ hat die Rechenzeitschranke?

$T(n)$	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 μ s
n	10.00 ns	100.00 ns	1.00 μ s	10.00 μ s	100.00 μ s	1.00 ms
$n \cdot \log_2(n)$	33.22 ns	664.39 ns	9.97 μ s	132.88 μ s	1.66 ms	19.93 ms
n^2	100.00 ns	10.00 μ s	1.00 ms	100.00 ms	10.00 s	0.27 h
n^3	1.00 μ s	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 μ s			
1.1^n	2.59 ns	13.78 μ s				
2^n	1.02 μ s					
n^n	10 s					





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“

- ▶ **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**