

Theoretische Informatik 3 WS 2006/07

Volker Strehl
Informatik 8

19. Oktober 2006



Organisatorisches

- ▶ Vorlesungstermine
Montags und Donnerstag, 16:00–17:30 Uhr im H9
- ▶ Übungen in 7 Gruppen, Termine und Eintragung → Webseite



Organisatorisches

- ▶ Vorlesungstermine
Montags und Donnerstag, 16:00–17:30 Uhr im H9



Organisatorisches

- ▶ Vorlesungstermine
Montags und Donnerstag, 16:00–17:30 Uhr im H9
- ▶ Übungen in 7 Gruppen, Termine und Eintragung → Webseite
- ▶ Webseite:
www8.informatik.uni-erlangen.de/IMMD8/Lectures/THINF/
enthält Materialien zur Vorlesung, Übungsaufgaben, aktuelle Informationen etc. (NB: nicht alles ist von aussen zugänglich)



Organisatorisches

- ▶ Vorlesungstermine
 - Montags und Donnerstag, 16:00–17:30 Uhr im H9
- ▶ Übungen in 7 Gruppen, Termine und Eintragung → Webseite
- ▶ Webseite:
 - www8.informatik.uni-erlangen.de/IMMD8/Lectures/THINF/
 - enthält Materialien zur Vorlesung, Übungsaufgaben, aktuelle Informationen etc. (NB: nicht alles ist von aussen zugänglich)
- ▶ Kontakt:
 - ▶ Adresse: Haberstrasse 2, 3. Stock
 - ▶ Telefon: 85-28712
 - ▶ email: strehl@cs.fau.de
 - ▶ Sprechstunde: Termin per email vereinbaren



Übersicht

- ▶ Einleitung – Begriffe und Beispiele



Empfohlene Literatur

- ▶ M. AIGNER, Diskrete Mathematik, Vieweg, 2005 (5. Aufl.).
- ▶ T. CORMEN, C. LEISERSON, R. RIVEST, Introduction to Algorithms, MIT Press, 2001 (2. A.).
- ▶ R. GRAHAM, D. KNUTH, O. PATASHNIK, Concrete Mathematics, Addison-Wesley, 1994 (2. A.).
- ▶ V. HEUN, Grundlegende Algorithmen, Vieweg, 2003 (2. A.).
- ▶ D. KNUTH, The Art of Computer Programming (1–3), Addison-Wesley, 1962–1997.
- ▶ U. SCHÖNING, Algorithmik, Spektrum-Verlag, 2001.
- ▶ R. SEDGEWICK, P. FLAJOLET, An Introduction to the Analysis of Algorithms, Addison-Wesley, 1996.
- ▶ H. WILF, Algorithms and Complexity, Prentice-Hall 1986.



Übersicht

- ▶ Einleitung – Begriffe und Beispiele
- ▶ Mathematische Hilfsmittel



Übersicht

- ▶ Einleitung – Begriffe und Beispiele
- ▶ Mathematische Hilfsmittel
- ▶ Exemplarische Analysen

Übersicht

- ▶ Einleitung – Begriffe und Beispiele
- ▶ Mathematische Hilfsmittel
- ▶ Exemplarische Analysen
- ▶ Rekursion und Komplexität
- ▶ Information und Komplexität

Übersicht

- ▶ Einleitung – Begriffe und Beispiele
- ▶ Mathematische Hilfsmittel
- ▶ Exemplarische Analysen
- ▶ Rekursion und Komplexität

Übersicht

- ▶ Einleitung – Begriffe und Beispiele
- ▶ Mathematische Hilfsmittel
- ▶ Exemplarische Analysen
- ▶ Rekursion und Komplexität
- ▶ Information und Komplexität
- ▶ Arithmetik und Komplexität

Teil I

Einleitung



Probleme

- ▶ Was ist ein Problem?



Probleme

- ▶ Was ist ein Problem?
 - ▶ D = domain = Definitionsbereich = "Instanzen"



Probleme

- ▶ Was ist ein Problem?
 - ▶ D = domain = Definitionsbereich = "Instanzen"
 - ▶ R = range = Wertebereich

Probleme

- ▶ Was ist ein Problem?
 - ▶ D = domain = Definitionsbereich = "Instanzen"
 - ▶ R = range = Wertebereich
 - ▶ $f : D \rightarrow R$ berechenbare Funktion

Probleme

- ▶ Was ist ein Problem?
 - ▶ D = domain = Definitionsbereich = "Instanzen"
 - ▶ R = range = Wertebereich
 - ▶ $f : D \rightarrow R$ berechenbare Funktion
 - ▶ falls $R = \{\text{true}, \text{false}\}$: Entscheidungsproblem

Probleme

- ▶ Was ist ein Problem?
 - ▶ D = domain = Definitionsbereich = "Instanzen"
 - ▶ R = range = Wertebereich
 - ▶ $f : D \rightarrow R$ berechenbare Funktion
 - ▶ falls $R = \{\text{true}, \text{false}\}$: Entscheidungsproblem
 - ▶ Grösse von Instanzen $\text{size} : D \rightarrow \mathbb{N}$

Probleme

- ▶ Was ist ein Problem?
 - ▶ D = domain = Definitionsbereich = "Instanzen"
 - ▶ R = range = Wertebereich
 - ▶ $f : D \rightarrow R$ berechenbare Funktion
 - ▶ falls $R = \{\text{true}, \text{false}\}$: Entscheidungsproblem
 - ▶ Grösse von Instanzen $\text{size} : D \rightarrow \mathbb{N}$
 - ▶ oft: $D_n = \{d \in D; \text{size}(d) = n\}$ ist endlich für $n \geq 0$, dann $d_n = \#D_n$

Kosten

- ▶ Algorithmen und Kosten

Kosten

- ▶ Algorithmen und Kosten
 - ▶ \mathcal{A} : Algorithmus zur Berechnung von $f : D \rightarrow R$ (auf TM oder RAM oder dergl.), d.h.

$$\forall d \in D : \mathcal{A}(d) = f(d)$$

Kosten

- ▶ Algorithmen und Kosten
 - ▶ \mathcal{A} : Algorithmus zur Berechnung von $f : D \rightarrow R$ (auf TM oder RAM oder dergl.), d.h.

$$\forall d \in D : \mathcal{A}(d) = f(d)$$

- ▶ Kostenfunktion (modellabhängig)

$$\text{cost}_{\mathcal{A}} : D \rightarrow \mathbb{N} : d \mapsto \text{cost}_{\mathcal{A}}(d)$$

soll Laufzeitverhalten (oder Speicherverbrauch) von \mathcal{A} auf einem Maschinenmodell (TM, RAM) anhand der bestimmenden Einflussgrößen (z.B. Anzahl der elementaren Rechenschritte einer TM, Anzahl der arithmetischen oder Vergleichsoperationen einer RAM) wiedergeben

- ▶ worst-case und average-case

- ▶ worst-case und average-case

- ▶ worst-case Komplexität

$$\text{cost}_{\mathcal{A}}(n) = \max_{d \in D_n} \text{cost}_{\mathcal{A}}(d)$$

- ▶ worst-case und average-case

- ▶ worst-case Komplexität

$$\text{cost}_{\mathcal{A}}(n) = \max_{d \in D_n} \text{cost}_{\mathcal{A}}(d)$$

- ▶ average-case Komplexität

$$\overline{\text{cost}_{\mathcal{A}}}(n) = \frac{1}{d_n} \sum_{d \in D_n} \text{cost}_{\mathcal{A}}(d)$$

- ▶ worst-case und average-case

- ▶ worst-case Komplexität

$$\text{cost}_{\mathcal{A}}(n) = \max_{d \in D_n} \text{cost}_{\mathcal{A}}(d)$$

- ▶ average-case Komplexität

$$\overline{\text{cost}_{\mathcal{A}}}(n) = \frac{1}{d_n} \sum_{d \in D_n} \text{cost}_{\mathcal{A}}(d)$$

- ▶ Hauptproblem:

Wie skaliert der Berechnungsaufwand mit der Problemgröße?

Wie verhalten sich die Folgen $(\text{cost}_{\mathcal{A}}(n))_{n \geq 0}$ bzw.

$(\overline{\text{cost}_{\mathcal{A}}}(n))_{n \geq 0}$ asymptotisch, d.h. für $n \rightarrow \infty$?

Zwei wichtige Bemerkungen

- ▶ Eine exakte Bestimmung (“Formel”) von $\text{cost}_{\mathcal{A}}(n)$ bzw. $\overline{\text{cost}_{\mathcal{A}}}(n)$ ist meist nicht möglich. Man muss sich mit “asymptotischen” Aussagen im Sinne der LANDAU-Notation begnügen, z.B. :

$$\text{cost}_{\mathcal{A}}(n) \in \mathcal{O}(n^2), \in \Omega(n^{3/2}), \in \Theta(n \log n), \sim 7n^{\log 3}, \dots$$

Zwei wichtige Bemerkungen

- ▶ Eine exakte Bestimmung (“Formel”) von $\text{cost}_{\mathcal{A}}(n)$ bzw. $\overline{\text{cost}_{\mathcal{A}}}(n)$ ist meist nicht möglich. Man muss sich mit “asymptotischen” Aussagen im Sinne der LANDAU-Notation begnügen, z.B. :

$$\text{cost}_{\mathcal{A}}(n) \in \mathcal{O}(n^2), \in \Omega(n^{3/2}), \in \Theta(n \log n), \sim 7n^{\log 3}, \dots$$

- ▶ Bei rekursiven Algorithmen genügen die Aufwandsfunktionen “divide-and-conquer” Rekursionsgleichungen, z.B.

$$T(2n) = 7T(n) + \Theta(n^2)$$

Wie kann man daraus Aussagen über das asymptotische Verhalten von $T(n)$ gewinnen?

Wichtige Unterscheidung

- ▶ Komplexität von Algorithmen

Wichtige Unterscheidung

- ▶ Komplexität von Algorithmen
 - ▶ Jeder konkrete Algorithmus \mathcal{A} für ein Problem $f : D \rightarrow R$ hat eine einem Berechnungsmodell bezüglich einer Grössenfunktion size und eines Komplexitätsmasses cost eine Komplexitätsfunktion.

Wichtige Unterscheidung

- ▶ Komplexität von Algorithmen
 - ▶ Jeder konkrete Algorithmus \mathcal{A} für ein Problem $f : D \rightarrow R$ hat eine einem Berechnungsmodell bezüglich einer Grössenfunktion $size$ und eines Komplexitätsmasses $cost$ eine Komplexitätsfunktion.
- ▶ Komplexität von Problemen

Wichtige Unterscheidung

- ▶ Komplexität von Algorithmen
 - ▶ Jeder konkrete Algorithmus \mathcal{A} für ein Problem $f : D \rightarrow R$ hat eine einem Berechnungsmodell bezüglich einer Grössenfunktion $size$ und eines Komplexitätsmasses $cost$ eine Komplexitätsfunktion.
- ▶ Komplexität von Problemen
 - ▶ Welchen Aufwand benötigt jeder Algorithmus zur Lösung eines Problems (in einem gegebenen Berechnungsmodell)? Das ist eine *inhärente* Eigenschaft des Problems!

- ▶ Konkrete Algorithmen liefern obere Schranken für die Problemkomplexität — “je kleiner, desto besser”.

- ▶ Konkrete Algorithmen liefern obere Schranken für die Problemkomplexität — “je kleiner, desto besser”.
- ▶ Untere Schranken treffen auf *alle* Algorithmen für ein Problem zu – “je grösser, desto besser” – sind aber meist sehr schwer zu gewinnen!

- ▶ Konkrete Algorithmen liefern obere Schranken für die Problemkomplexität — “je kleiner, desto besser”.
- ▶ Untere Schranken treffen auf *alle* Algorithmen für ein Problem zu – “je grösser, desto besser” – sind aber meist sehr schwer zu gewinnen!
- ▶ Für *optimale* Algorithmen stimmt die Algorithmen-Komplexität mit einer unteren Schranke für die Problemkomplexität überein.

- ▶ Konkrete Algorithmen liefern obere Schranken für die Problemkomplexität — “je kleiner, desto besser”.
- ▶ Untere Schranken treffen auf *alle* Algorithmen für ein Problem zu – “je grösser, desto besser” – sind aber meist sehr schwer zu gewinnen!
- ▶ Für *optimale* Algorithmen stimmt die Algorithmen-Komplexität mit einer unteren Schranke für die Problemkomplexität überein.
- ▶ Für die meisten Probleme sind gute untere Schranke nicht bekannt – und damit keine optimalen Algorithmen!

Begriffe

- ▶ *Komplexitätsanalyse* beschäftigt sich mit dem Entwurf der Analyse von möglichst effizienten Algorithmen für konkrete Probleme

Begriffe

- ▶ *Komplexitätsanalyse* beschäftigt sich mit dem Entwurf der Analyse von möglichst effizienten Algorithmen für konkrete Probleme
- ▶ *Komplexitätstheorie* untersucht die Komplexität von Problemen und die Komplexitäts-Beziehungen zwischen Problemen. *Komplexitätsklassen* (wie P, NP, EXPTIME, PSPACE) fassen Probleme “gleicher Schwierigkeit” zu Klassen zusammen und untersuchen Beziehungen zwischen diesen Klassen.

Zwei formale Sprachen

- ▶ Reguläre Sprache $F = (a + bb)^* \subseteq \{a, b\}^*$.
Wieviele Wörter der Länge n enthält F ?

Zwei formale Sprachen

- ▶ Reguläre Sprache $F = (a + bb)^* \subseteq \{a, b\}^*$.
Wieviele Wörter der Länge n enthält F ?
- ▶ Kontextfreie Sprache der korrekten Klammerungen
 $D \subseteq \{a, b\}^*$, erzeugt durch

$$D \rightarrow aDbD \mid \lambda.$$

Wieviele Wörter der Länge n enthält D ?

Zwei formale Sprachen

- ▶ Reguläre Sprache $F = (a + bb)^* \subseteq \{a, b\}^*$.
Wieviele Wörter der Länge n enthält F ?
- ▶ Kontextfreie Sprache der korrekten Klammerungen
 $D \subseteq \{a, b\}^*$, erzeugt durch

$$D \rightarrow aDbD \mid \lambda.$$

Wieviele Wörter der Länge n enthält D ?

- ▶ Allgemein:
 Σ endliches Alphabet, $L \subseteq \Sigma^*$ formale Sprache
Wie verhält sich $\ell_n(L) := \#(L \cap \Sigma^n)$ für $n \rightarrow \infty$?
Welche Rolle spielt dabei der Chomsky-Typ von L ?

Beispiel: reguläre formale Sprache

- ▶ Für $F = (a + bb)^* \subseteq \{a, b\}^*$ gilt

$$\ell_0(F) = \ell_1(F) = 1$$

$$\ell_{n+2}(F) = \ell_{n+1}(F) + \ell_n(F) \quad (n \geq 0)$$

Beispiel: reguläre formale Sprache

- Für $F = (a + bb)^* \subseteq \{a, b\}^*$ gilt

$$\begin{aligned} \ell_0(F) &= \ell_1(F) = 1 \\ \ell_{n+2}(F) &= \ell_{n+1}(F) + \ell_n(F) \quad (n \geq 0) \end{aligned}$$

und daher

$$\ell_n(F) = F_{n+1} \text{ Fibonacci-Zahl}$$

Beispiel: reguläre formale Sprache

- Für $F = (a + bb)^* \subseteq \{a, b\}^*$ gilt

$$\begin{aligned} \ell_0(F) &= \ell_1(F) = 1 \\ \ell_{n+2}(F) &= \ell_{n+1}(F) + \ell_n(F) \quad (n \geq 0) \end{aligned}$$

und daher

$$\ell_n(F) = F_{n+1} \text{ Fibonacci-Zahl}$$

Asymptotisch exponentielles Wachstum:

$$F_n = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}} \sim \frac{\phi^n}{\sqrt{5}}$$

mit $\phi = \frac{1+\sqrt{5}}{2} = 1.61803\dots$ (goldener Schnitt),
 $\hat{\phi} = \frac{1-\sqrt{5}}{2} = -0.61803\dots$

Beispiel: kontextfreie formale Sprache

- Für die durch

$$D \rightarrow aDbD \mid \lambda$$

erzeugte kontextfreie Sprache der korrekten Klammerungen
(äquivalent: Binärbäume) gilt

Beispiel: kontextfreie formale Sprache

- Für die durch

$$D \rightarrow aDbD \mid \lambda$$

erzeugte kontextfreie Sprache der korrekten Klammerungen
(äquivalent: Binärbäume) gilt $\ell_{2n+1}(D) = 0$ für $n \geq 0$ und für
 $d_n := \ell_{2n}(D)$

Beispiel: kontextfreie formale Sprache

- Für die durch

$$D \rightarrow aDbD \mid \lambda$$

erzeugte kontextfreie Sprache der korrekten Klammerungen (äquivalent: Binärbäume) gilt $\ell_{2n+1}(D) = 0$ für $n \geq 0$ und für $d_n := \ell_{2n}(D)$

$$d_{n+1} = d_0 \cdot d_n + d_1 \cdot d_{n-1} + d_2 \cdot d_{n-2} + \dots + d_n \cdot d_0$$



Beispiel: kontextfreie formale Sprache

- Für die durch

$$D \rightarrow aDbD \mid \lambda$$

erzeugte kontextfreie Sprache der korrekten Klammerungen (äquivalent: Binärbäume) gilt $\ell_{2n+1}(D) = 0$ für $n \geq 0$ und für $d_n := \ell_{2n}(D)$

$$d_{n+1} = d_0 \cdot d_n + d_1 \cdot d_{n-1} + d_2 \cdot d_{n-2} + \dots + d_n \cdot d_0$$

Wir werden später sehen:

$$d_n = \ell_{2n}(D) = \frac{1}{n+1} \binom{2n}{n} \sim \frac{4^n}{\sqrt{\pi n^3}}$$



Polynome

Konventionen:

- Polynom \simeq Folge der Koeffizienten

$$a(X) = a_0 + a_1X + a_2X^2 + \dots + a_mX^m \leftrightarrow (a_0, a_1, a_2, \dots, a_m)$$



Polynome

Konventionen:

- Polynom \simeq Folge der Koeffizienten

$$a(X) = a_0 + a_1X + a_2X^2 + \dots + a_mX^m \leftrightarrow (a_0, a_1, a_2, \dots, a_m)$$

- "Grösse" eines Polynoms = Anzahl der Koeffizienten (i.w. Grad)



Polynome

Konventionen:

- ▶ Polynom \simeq Folge der Koeffizienten

$$a(X) = a_0 + a_1X + a_2X^2 + \dots + a_mX^m \leftrightarrow (a_0, a_1, a_2, \dots, a_m)$$

- ▶ “Grösse” eines Polynoms = Anzahl der Koeffizienten (i.w. Grad)
- ▶ Aufwand für Polynomoperationen wird gemessen in Anzahl der Operationen im Koeffizientenbereich (oft auch nur: Multiplikationen im Koeffizientenbereich)

Multiplikation

$$a(X) = a_0 + a_1X + a_2X^2 + \dots + a_mX^m \leftrightarrow (a_0, a_1, a_2, \dots, a_m)$$

$$b(X) = b_0 + b_1X + b_2X^2 + \dots + b_nX^n \leftrightarrow (b_0, b_1, b_2, \dots, b_n)$$

$$\begin{aligned} c(X) &= a(X) \cdot b(X) \\ &= c_0 + c_1X + \dots + c_{m+n}X^{m+n} \leftrightarrow (c_0, c_1, c_2, \dots, c_{m+n}) \end{aligned}$$

mit

$$c_k = \sum_{i+j=k} a_i \cdot b_j \quad (0 \leq i \leq m, 0 \leq j \leq n, 0 \leq k \leq m+n)$$

Berechnung mittels dieser Formel erfordert $(m+1)(n+1)$ Multiplikationen im Koeffizientenbereich \Rightarrow Aufwand wächst (für $m = n$) *quadratisch* mit der Grösse der Instanzen

Komplexität der Multiplikation

- ▶ “klassisch”: $\mathcal{O}(n^2)$

Komplexität der Multiplikation

- ▶ “klassisch”: $\mathcal{O}(n^2)$
- ▶ das rekursive (divide-and-conquer) Verfahren von KARATSUBA leistet dies mit einem Aufwand $M_{kara}(n)$, für den gilt

$$M_{kara}(2n) = 3 \cdot M_{kara}(n) + \Theta(n)$$

und das ergibt $M_{kara}(n) \in \mathcal{O}(n^{\log_2 3})$, wobei $\log_2 3 \approx 1.585$.

Bemerkung: analoge Aussagen gelten für die Multiplikation von ganzen Zahlen

Komplexität der Multiplikation

- ▶ “klassisch”: $\mathcal{O}(n^2)$
- ▶ das rekursive (divide-and-conquer) Verfahren von KARATSUBA leistet dies mit einem Aufwand $M_{kara}(n)$, für den gilt

$$M_{kara}(2n) = 3 \cdot M_{kara}(n) + \Theta(n)$$

und das ergibt $M_{kara}(n) \in \mathcal{O}(n^{\log_2 3})$, wobei $\log_2 3 \approx 1.585$.

- ▶ Man kann Polynome auch ganz anders multiplizieren:
Evaluation und Interpolation

Bemerkung: analoge Aussagen gelten für die Multiplikation von ganzen Zahlen



one-way?

- ▶ *Multiplizieren* ist effizient machbar, aber gilt das auch für das *Faktorisieren*?



Komplexität der Multiplikation

- ▶ “klassisch”: $\mathcal{O}(n^2)$
- ▶ das rekursive (divide-and-conquer) Verfahren von KARATSUBA leistet dies mit einem Aufwand $M_{kara}(n)$, für den gilt

$$M_{kara}(2n) = 3 \cdot M_{kara}(n) + \Theta(n)$$

und das ergibt $M_{kara}(n) \in \mathcal{O}(n^{\log_2 3})$, wobei $\log_2 3 \approx 1.585$.

- ▶ Man kann Polynome auch ganz anders multiplizieren:
Evaluation und Interpolation
- ▶ mittels schneller FOURIER-Transformation (FFT=spezielles *rekursives* Evaluations-Interpolationsschema) kann man einen Aufwand $\mathcal{O}(n \log n)$ erreichen

Bemerkung: analoge Aussagen gelten für die Multiplikation von ganzen Zahlen



one-way?

- ▶ *Multiplizieren* ist effizient machbar, aber gilt das auch für das *Faktorisieren*?
- ▶ *Exponentiation* ist effizient machbar, aber gilt das auch für das *Logarithmieren*?



one-way?

- ▶ *Multiplizieren* ist effizient machbar, aber gilt das auch für das *Faktorisieren*?
- ▶ *Exponentiation* ist effizient machbar, aber gilt das auch für das *Logarithmieren*?
- ▶ Antworten auf diese Fragen wären essentiell für die Sicherheit gängiger kryptographischer Verfahren!

Sortieren

- ▶ Allgemein: (A, \leq) totalgeordnete Menge
Sortieren = Umordnen von Elementen einer A -Liste (oder eines A -Feldes) auf- oder absteigend bezgl. \leq

one-way?

- ▶ *Multiplizieren* ist effizient machbar, aber gilt das auch für das *Faktorisieren*?
- ▶ *Exponentiation* ist effizient machbar, aber gilt das auch für das *Logarithmieren*?
- ▶ Antworten auf diese Fragen wären essentiell für die Sicherheit gängiger kryptographischer Verfahren!
- ▶ Faktorisieren und Logarithmieren sind effizient durchführbar mit einem Computer, den es noch nicht gibt:
Quantencomputing.

Sortieren

- ▶ Allgemein: (A, \leq) totalgeordnete Menge
Sortieren = Umordnen von Elementen einer A -Liste (oder eines A -Feldes) auf- oder absteigend bezgl. \leq
- ▶ Instanzengröße = Listenlänge

Sortieren

- ▶ Allgemein: (A, \leq) totalgeordnete Menge
Sortieren = Umordnen von Elementen einer A -Liste (oder eines A -Feldes) auf- oder absteigend bezgl. \leq
- ▶ Instanzengrösse = Listenlänge
- ▶ Aufwand = Anzahl der paarweisen Vergleiche von Listenelementen

Sortieren

- ▶ Allgemein: (A, \leq) totalgeordnete Menge
Sortieren = Umordnen von Elementen einer A -Liste (oder eines A -Feldes) auf- oder absteigend bezgl. \leq
- ▶ Instanzengrösse = Listenlänge
- ▶ Aufwand = Anzahl der paarweisen Vergleiche von Listenelementen
- ▶ Vereinfachung (*Permutationsmodell*): Listenelemente sind paarweise verschieden, es kommt nur auf die relativen Ordnungsbeziehungen zwischen den Listenelementen an, nicht auf absolute Werte
→ Instanzen der Grösse $n =$ Permutationen von $\{1, 2, \dots, n\}$

Sortieren

- ▶ Allgemein: (A, \leq) totalgeordnete Menge
Sortieren = Umordnen von Elementen einer A -Liste (oder eines A -Feldes) auf- oder absteigend bezgl. \leq
- ▶ Instanzengrösse = Listenlänge
- ▶ Aufwand = Anzahl der paarweisen Vergleiche von Listenelementen
- ▶ Vereinfachung (*Permutationsmodell*): Listenelemente sind paarweise verschieden, es kommt nur auf die relativen Ordnungsbeziehungen zwischen den Listenelementen an, nicht auf absolute Werte
→ Instanzen der Grösse $n =$ Permutationen von $\{1, 2, \dots, n\}$
- ▶ NB: es gibt $n! \sim (n/e)^n \cdot \sqrt{2\pi n}$ Instanzen der Grösse n

Sortieralgorithmen

- ▶ Es gibt viele verschiedene (experimentell und theoretisch gut untersuchte) Sortieralgorithmen (→ KNUTH, TAOCP 3).

Sortieralgorithmen

- ▶ Es gibt viele verschiedene (experimentell und theoretisch gut untersuchte) Sortieralgorithmen (→ KNUTH, TAOCP 3).
- ▶ Bei einigen der “simplen” Algorithmen (INSERTIONSORT, SELECTIONSORT, BUBBLESORT) wächst der Aufwand im worst- und im average-case quadratisch in der Listenlänge $\mathcal{O}(n^2)$.

Sortieralgorithmen

- ▶ Es gibt viele verschiedene (experimentell und theoretisch gut untersuchte) Sortieralgorithmen (→ KNUTH, TAOCP 3).
- ▶ Bei einigen der “simplen” Algorithmen (INSERTIONSORT, SELECTIONSORT, BUBBLESORT) wächst der Aufwand im worst- und im average-case quadratisch in der Listenlänge $\mathcal{O}(n^2)$.
- ▶ Bei “guten” Algorithmen (HEAPSORT, MERGESORT) wächst der Aufwand im worst-case (und auch im average-case) wie $\mathcal{O}(n \log n)$.
- ▶ Bei QUICKSORT wächst der Aufwand im worst-case quadratisch, im average-case aber wie $\mathcal{O}(n \log n)$.

Sortieralgorithmen

- ▶ Es gibt viele verschiedene (experimentell und theoretisch gut untersuchte) Sortieralgorithmen (→ KNUTH, TAOCP 3).
- ▶ Bei einigen der “simplen” Algorithmen (INSERTIONSORT, SELECTIONSORT, BUBBLESORT) wächst der Aufwand im worst- und im average-case quadratisch in der Listenlänge $\mathcal{O}(n^2)$.
- ▶ Bei “guten” Algorithmen (HEAPSORT, MERGESORT) wächst der Aufwand im worst-case (und auch im average-case) wie $\mathcal{O}(n \log n)$.

Sortieralgorithmen

- ▶ Es gibt viele verschiedene (experimentell und theoretisch gut untersuchte) Sortieralgorithmen (→ KNUTH, TAOCP 3).
- ▶ Bei einigen der “simplen” Algorithmen (INSERTIONSORT, SELECTIONSORT, BUBBLESORT) wächst der Aufwand im worst- und im average-case quadratisch in der Listenlänge $\mathcal{O}(n^2)$.
- ▶ Bei “guten” Algorithmen (HEAPSORT, MERGESORT) wächst der Aufwand im worst-case (und auch im average-case) wie $\mathcal{O}(n \log n)$.
- ▶ Bei QUICKSORT wächst der Aufwand im worst-case quadratisch, im average-case aber wie $\mathcal{O}(n \log n)$.
- ▶ Bei SHELLSORT kann man einen Aufwand wie $\mathcal{O}(n^{1+\epsilon})$ (für beliebig kleines $\epsilon > 0$) erreichen.

Untere Schranke für die Sortierkomplexität

- ▶ Jeder auf paarweisen Vergleichen basierende Sortieralgorithmus benötigt auf Listen der Länge n im worst-case und im average-case mindestens

$$\log n! \sim n \log n - \Theta(n)$$

Vergleichsoperationen (*informationstheoretische Schranke*)

Untere Schranke für die Sortierkomplexität

- ▶ Jeder auf paarweisen Vergleichen basierende Sortieralgorithmus benötigt auf Listen der Länge n im worst-case und im average-case mindestens

$$\log n! \sim n \log n - \Theta(n)$$

Vergleichsoperationen (*informationstheoretische Schranke*)

- ▶ In die average-case-Aussage geht der *Entropiebegriff* der Informationstheorie entscheidend ein.

Das Problem k -SAT

- ▶ $X = \{x_1, x_2, \dots, x_n\}$ Menge von a.l. Variablen,
 $\bar{X} = \{\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n\}$ negierte Variablen
 $L = X \cup \bar{X}$: Menge der Literale

Das Problem k -SAT

- ▶ $X = \{x_1, x_2, \dots, x_n\}$ Menge von a.l. Variablen,
 $\bar{X} = \{\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n\}$ negierte Variablen
 $L = X \cup \bar{X}$: Menge der Literale
- ▶ Klausel: Disjunktion von Literalen

$$K = l_1 \vee l_2 \vee \dots \vee l_s$$

Das Problem k -SAT

- ▶ $X = \{x_1, x_2, \dots, x_n\}$ Menge von a.l. Variablen,
 $\bar{X} = \{\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n\}$ negierte Variablen
 $L = X \cup \bar{X}$: Menge der Literale
- ▶ Klausel: Disjunktion von Literalen

$$K = l_1 \vee l_2 \vee \dots \vee l_s$$

- ▶ AL-Formel in KNF: Konjunktion von Literalen

$$F = K_1 \wedge K_2 \wedge \dots \wedge K_t$$

Das Problem k -SAT

- ▶ $X = \{x_1, x_2, \dots, x_n\}$ Menge von a.l. Variablen,
 $\bar{X} = \{\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n\}$ negierte Variablen
 $L = X \cup \bar{X}$: Menge der Literale
- ▶ Klausel: Disjunktion von Literalen

$$K = l_1 \vee l_2 \vee \dots \vee l_s$$

- ▶ AL-Formel in KNF: Konjunktion von Literalen

$$F = K_1 \wedge K_2 \wedge \dots \wedge K_t$$

- ▶ k -SAT Formel: KNF-Formel mit $s \leq k$ Literalen pro Klausel

Das Problem k -SAT

- ▶ $X = \{x_1, x_2, \dots, x_n\}$ Menge von a.l. Variablen,
 $\bar{X} = \{\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n\}$ negierte Variablen
 $L = X \cup \bar{X}$: Menge der Literale
- ▶ Klausel: Disjunktion von Literalen

$$K = l_1 \vee l_2 \vee \dots \vee l_s$$

- ▶ AL-Formel in KNF: Konjunktion von Literalen

$$F = K_1 \wedge K_2 \wedge \dots \wedge K_t$$

- ▶ k -SAT Formel: KNF-Formel mit $s \leq k$ Literalen pro Klausel
- ▶ Entscheidungsproblem: Erfüllbarkeit von KNF-Formeln testen!

Algorithmen für k -SAT

- ▶ Brute-force Methode:
 alle 2^n Bewertungen der n Variablen testen
 → exponentielle Komplexität

Algorithmen für k -SAT

- ▶ Brute-force Methode:
alle 2^n Bewertungen der n Variablen testen
→ exponentielle Komplexität
- ▶ Bis heute kein effizientes Verfahren bekannt:
zwar ist 2-SAT effizient entscheidbar, aber bereits 3-SAT ist NP-vollständig!

Algorithmen für k -SAT

- ▶ Brute-force Methode:
alle 2^n Bewertungen der n Variablen testen
→ exponentielle Komplexität
- ▶ Bis heute kein effizientes Verfahren bekannt:
zwar ist 2-SAT effizient entscheidbar, aber bereits 3-SAT ist NP-vollständig!
- ▶ Es gibt wesentlich bessere (aber immer noch exponentielle) Verfahren als die brute-force Methode!