

# Komplexität von Algorithmen

## SS 2009

Volker Strehl  
Informatik 8

20. April 2009

## Organisatorisches

- ▶ Vorlesungstermine

Montags und Freitag, 10:15–11:45 Uhr im H4

## Organisatorisches

- ▶ Vorlesungstermine  
Montags und Freitag, 10:15–11:45 Uhr im H4
- ▶ Übungen in 5 Gruppen, Termine und Eintragung → Webseite

## Organisatorisches

- ▶ Vorlesungstermine

Montags und Freitag, 10:15–11:45 Uhr im H4

- ▶ Übungen in 5 Gruppen, Termine und Eintragung → Webseite

- ▶ Webseite:

`www8.informatik.uni-erlangen.de/IMMD8/Lectures/THINF/`  
enthält Materialien zur Vorlesung, Übungsaufgaben, aktuelle Informationen etc.

## Organisatorisches

- ▶ Vorlesungstermine

Montags und Freitag, 10:15–11:45 Uhr im H4

- ▶ Übungen in 5 Gruppen, Termine und Eintragung → Webseite

- ▶ Webseite:

`www8.informatik.uni-erlangen.de/IMMD8/Lectures/THINF/`  
enthält Materialien zur Vorlesung, Übungsaufgaben, aktuelle Informationen etc.

- ▶ Kontakt:

- ▶ Adresse: Haberstrasse 2, 3. Stock
- ▶ Telefon: 85-28712
- ▶ email: `strehl@informatik.uni-erlangen.de`
- ▶ Sprechstunde: Termin per email vereinbaren

## 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, Probleme, Beispiele

# Übersicht

- ▶ Einleitung – Begriffe, Probleme, Beispiele
- ▶ Mathematische Hilfsmittel

# Übersicht

- ▶ Einleitung – Begriffe, Probleme, Beispiele
- ▶ Mathematische Hilfsmittel
- ▶ Rekursion und Komplexität

# Übersicht

- ▶ Einleitung – Begriffe, Probleme, Beispiele
- ▶ Mathematische Hilfsmittel
- ▶ Rekursion und Komplexität
- ▶ Information und Komplexität

# Übersicht

- ▶ Einleitung – Begriffe, Probleme, Beispiele
- ▶ Mathematische Hilfsmittel
- ▶ Rekursion und Komplexität
- ▶ Information und Komplexität
- ▶ Arithmetik und Komplexität

# Probleme

- ▶ Was ist ein Problem?

# Probleme

- ▶ Was ist ein Problem?
  - ▶  $D = \text{domain} = \text{Definitionsbereich} = \text{“Instanzen”}$

# Probleme

- ▶ Was ist ein Problem?
  - ▶  $D = \text{domain} = \text{Definitionsbereich} = \text{“Instanzen”}$
  - ▶  $R = \text{range} = \text{Wertebereich}$

# Probleme

- ▶ Was ist ein Problem?
  - ▶  $D = \text{domain} = \text{Definitionsbereich} = \text{“Instanzen”}$
  - ▶  $R = \text{range} = \text{Wertebereich}$
  - ▶  $f : D \rightarrow R$  berechenbare Funktion

# Probleme

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

# Probleme

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

# Probleme

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

# Algorithmen und Kosten

## Algorithmen und Kosten

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

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

## 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)

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

z.B. Laufzeitverhalten, modelliert durch

- ▶ Anz. elementare Rechenoperationen (Bit-Komplexität)
- ▶ Anz. arithmetische Operationen (arithmetische Komplexität)
- ▶ Anz. Vergleichsoperationen

## 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:

Skalierung des Berechnungsaufwandes 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  $n \rightarrow \infty$ ?

## asymptotisches Verhalten

- ▶ Exakte Bestimmung (“Formel”) von  $\text{cost}_{\mathcal{A}}(n)$  bzw.  $\overline{\text{cost}_{\mathcal{A}}}(n)$  ist meist nicht möglich. “Asymptotischen” Aussagen im Sinne der LANDAU-Notation sind erwünscht, z.B. :

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

## asymptotisches Verhalten

- ▶ Exakte Bestimmung (“Formel”) von  $\text{cost}_{\mathcal{A}}(n)$  bzw.  $\overline{\text{cost}_{\mathcal{A}}}(n)$  ist meist nicht möglich. “Asymptotischen” Aussagen im Sinne der LANDAU-Notation sind erwünscht, 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.

$$S(2n) = 2 \cdot S(n) + \Theta(n)$$

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

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

## Grundsätzliche Unterscheidung

In einem gegebenen Berechnungsmodell,  
bezüglich einer Grössenfunktion  $size$   
und eines Komplexitätsmasses  $cost$

## Grundsätzliche Unterscheidung

In einem gegebenen Berechnungsmodell,  
bezüglich einer Grössenfunktion  $size$   
und eines Komplexitätsmasses  $cost$

- ▶ hat jeder konkrete Algorithmus  $\mathcal{A}$  für ein Problem  $f : D \rightarrow R$   
eine Komplexitätsfunktion  $n \mapsto cost_{\mathcal{A}}(n)$

⇒ Komplexität von Algorithmen

## Grundsätzliche Unterscheidung

In einem gegebenen Berechnungsmodell,  
bezüglich einer Grössenfunktion  $size$   
und eines Komplexitätsmasses  $cost$

- ▶ hat jeder konkrete Algorithmus  $\mathcal{A}$  für ein Problem  $f : D \rightarrow R$  eine Komplexitätsfunktion  $n \mapsto cost_{\mathcal{A}}(n)$

⇒ Komplexität von Algorithmen

- ▶ hat jeder mögliche Algorithmus  $\mathcal{A}$  zur Lösung eines Problems  $f : D \rightarrow R$  einen Mindestaufwand  $n \mapsto \min_{\mathcal{A}} cost_{\mathcal{A}}(n)$ .

Das ist eine *inhärente* Eigenschaft des Problems!

⇒ Komplexität von Problemen

- ▶ 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 (asymptotisch) 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 (asymptotisch) überein.
- ▶ Für die meisten Probleme sind gute untere Schranke nicht bekannt – und damit keine optimale 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.

# 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.

## Beispiel: Polynome

Konventionen:

- ▶ Polynom  $\simeq$  Folge der Koeffizienten

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

## Beispiel: Polynome

Konventionen:

- ▶ Polynom  $\simeq$  Folge der Koeffizienten

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

- ▶ “Grösse” eines Polynoms = Anzahl der Koeffizienten (i.w. Grad)

## Beispiel: Polynome

Konventionen:

- ▶ Polynom  $\simeq$  Folge der Koeffizienten

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

- ▶ “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

$$a(X) = a_0 + a_1X + a_2X^2 + \cdots + a_mX^m$$

$$b(X) = b_0 + b_1X + b_2X^2 + \cdots + b_nX^n$$

$$c(X) = a(X) \cdot b(X)$$

$$= c_0 + c_1X + \cdots + c_{m+n}X^{m+n}$$

mit

$$c_k = \sum_{i+j=k} a_i \cdot b_j$$

# Multiplikation

$$a(X) = a_0 + a_1X + a_2X^2 + \cdots + a_mX^m$$

$$b(X) = b_0 + b_1X + b_2X^2 + \cdots + b_nX^n$$

$$c(X) = a(X) \cdot b(X)$$

$$= c_0 + c_1X + \cdots + c_{m+n}X^{m+n}$$

mit

$$c_k = \sum_{i+j=k} a_i \cdot b_j$$

- ▶ Berechnung (mittels dieser Formel!) erfordert  $(m+1)(n+1)$  Multiplikationen im Koeffizientenbereich  
⇒ Aufwand wächst (für  $m=n$ ) *quadratisch* mit der Input-Grösse

## Komplexität der Multiplikation

- ▶ “klassisch”:  $\mathcal{O}(n^2)$

## Komplexität der Multiplikation

- ▶ “klassisch”:  $\mathcal{O}(n^2)$
- ▶ KARATSUBA (divide-and-conquer) Verfahren leistet dies mit 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  $\log_2 3 \approx 1.585$ .

## Komplexität der Multiplikation

- ▶ “klassisch”:  $\mathcal{O}(n^2)$
- ▶ KARATSUBA (divide-and-conquer) Verfahren leistet dies mit 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*

## Komplexität der Multiplikation

- ▶ “klassisch”:  $\mathcal{O}(n^2)$
- ▶ KARATSUBA (divide-and-conquer) Verfahren leistet dies mit 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

## Komplexität der Multiplikation

- ▶ “klassisch”:  $\mathcal{O}(n^2)$
- ▶ KARATSUBA (divide-and-conquer) Verfahren leistet dies mit 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
- ▶ Problemkomplexität ist unbekannt!

Analoge Aussagen gelten für die Multiplikation von ganzen Zahlen

## one-way?

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

## 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!

## 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.  $\leq$

# 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 pw verschieden, es kommt nur  
auf die relativen Ordnungsbeziehungen zwischen  
den Listenelementen an, nicht auf absolute Werte

# 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 pw 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 pw 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 ( $\rightarrow$  KNUTH, TAOCP 3).

# Sortieralgorithmen

- ▶ Es gibt viele verschiedene (experimentell und theoretisch gut untersuchte) Sortieralgorithmen ( $\rightarrow$  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 ( $\rightarrow$  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 ( $\rightarrow$  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 ( $\rightarrow$  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.