Algorithmen & Komplexität

Angelika Steger

Institut für Theoretische Informatik

Was ist ein Algorithmus ?

Ein Algorithmus ist eine eindeutige Handlungsvorschrift,

[bestehend] aus endlich vielen, wohldefinierten Einzelschritten.

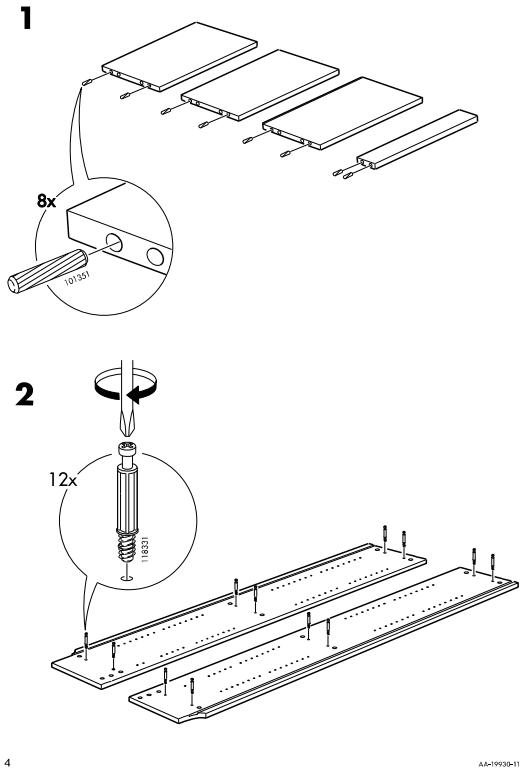
(Wikipedia)

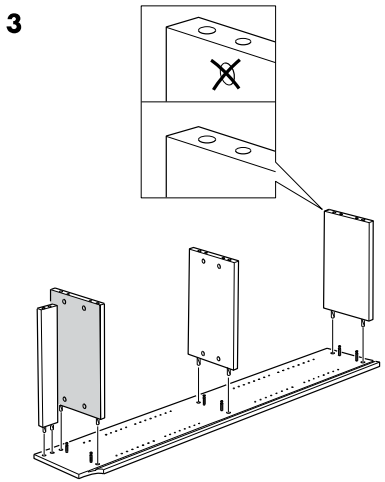
Algorithmen – Beispiele

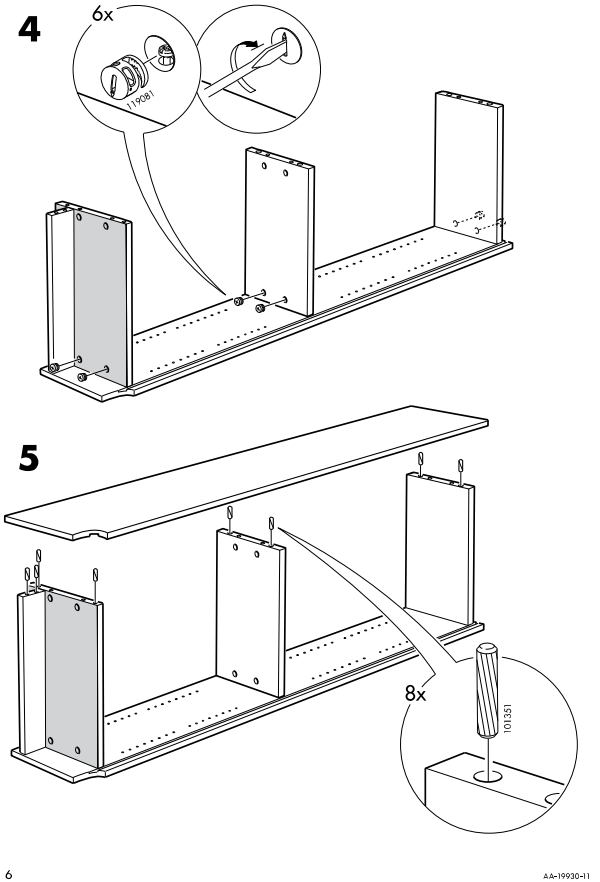
Kräuter-Omelette

Eier aufschlagen und mit Salz und Pfeffer würzen. 50 Gramm in Stückchen geschnittene kalte Butter und Kräuter zufügen. Solange verquirlen, bis sich Eiweiß und Eigelb verbunden haben. Restliche Butter in einer Pfanne erhitzen, sobald sie schäumt, Eimasse zugeben und stocken lassen. Omelette zusammenklappen und sofort servieren.

Algorithmen – Beispiele







Algorithmen der Vorlesung

Algorithmen zur Lösung von Problem auf Computern

Grösster gemeinsamer Teiler

ggt(n,m)

if n < m then swap(n,m);

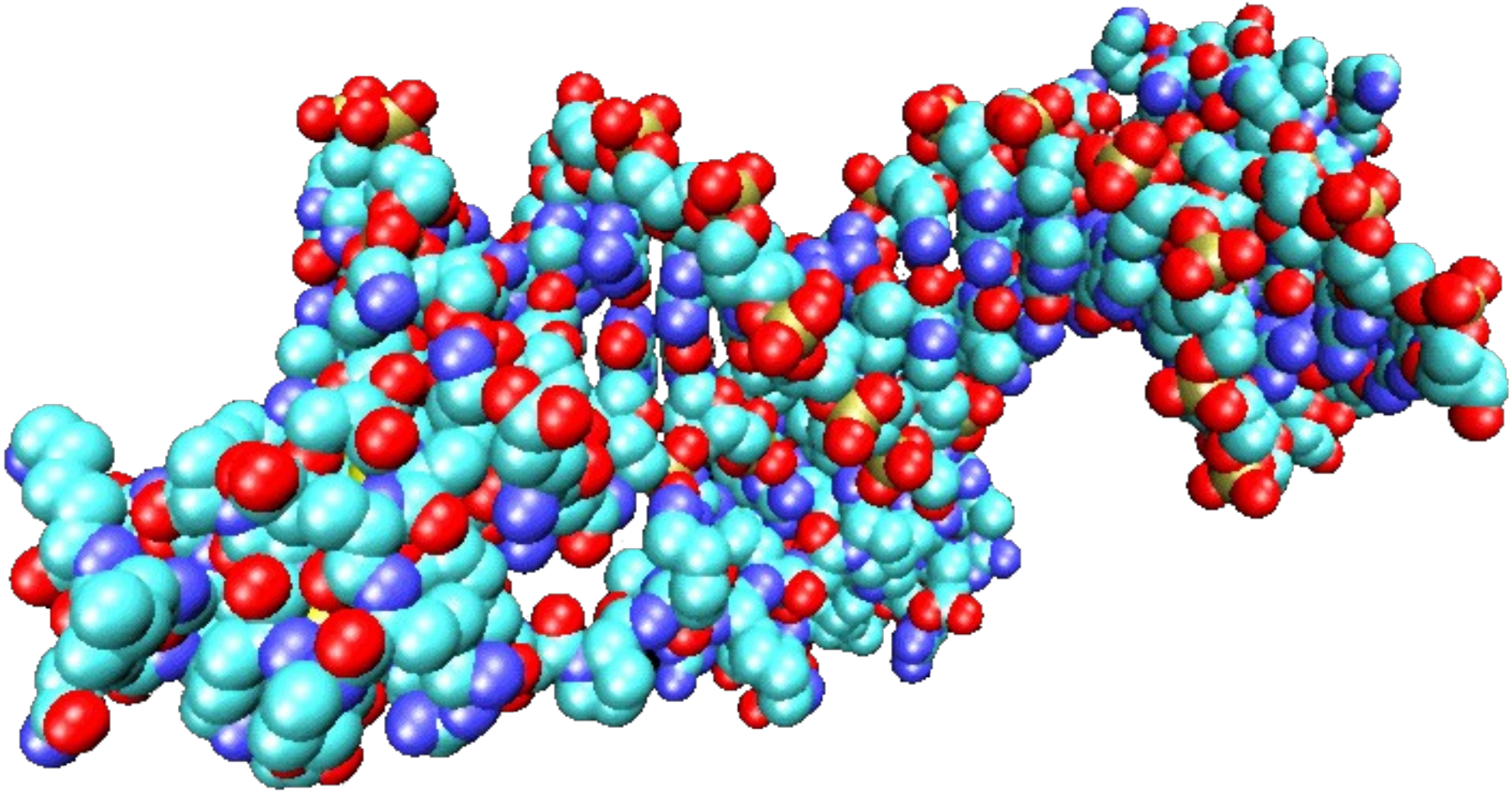
if m | n then return m

else return ggt(m, n mod m);

Beispiel: ggt(102,66) = ggt(66,36) = ggt(36, 30)
= ggt(30,6) = 6;

Probe: 102= 2*3*17; 66=2*3*11;

Was bedeutet Komplexität ?



Komplexitätsbegriff der Vorlesung

Berechenbarkeitstheorie:

Eine Funktion heisst berechenbar, wenn mittels einer abstrakten und/oder mechanischen Vorgehensweise zu gegebenen Eingaben ihre Ausgabe bestimmt werden kann.

Komplexitätstheorie:

Stellt zusätzlich die Frage nach der Effizienz der Berechnung.

Algorithmen & Komplexität

ca. 300 v. Chr: Euklids Algorithmus

ca. 800 n. Chr: Der persische Mathematiker alChoresmi veröffentlicht eine Aufgabensammlung für Kaufleute, die später als Liber Algorithmi erscheint

Algorithmus: Kunstwort aus alChoresmi und arithmos (griech. Zahl)

1936: Alan Turing: erste formale Definition von Berechenbarkeit („*Turingmaschine*“)

Church‘sche These: „*Jedes intuitiv berechenbare Problem kann durch eine Turingmaschine gelöst werden.*“

Inhalt der Vorlesung

1. Einführung: Was ist ein Algorithmus?
2. Beispiele effizienter Algorithmen
3. Algorithmische Grundprinzipien
4. Effiziente Datenstrukturen
5. Komplexitätstheorie
6. Ausblick

Wieso ist die Vorlesung wichtig?

- Algorithmen sind wichtig für die Praxis
- Algorithmische Ideen und Beweisstrategien nehmen vermehrt Einzug in die Mathematik
- Algorithmische Beweise erfordern eine andere / neue Herangehensweise
- Theoretische Informatik und Mathematik haben eine immer grössere Schnittmenge (Geometrie, Kombinatorik, Algebra, …)

Übungen, Testate, Klausur

Übungsleitung:

Ralph Keusch, Florian Meier

Webseiten der Vorlesung:

www.cadmo.ethz.ch/education/lectures/HS16/ac

Übungsblätter:

Ausgabe: dienstags über die Webseite der Vorlesung (ab heute!)

Abgabe: in der folgenden Woche in der Pause der Vorlesung

Testat:

an der ETH probeweise für drei Jahre abgeschafft

Empfehlung: - nutzen Sie die Möglichkeit für Feedback

- geben Sie jede Woche die Übungen ab

Klausur:

innerhalb der Prüfungsperiode, Hilfsmittel: 10 handbeschriebene Blätter

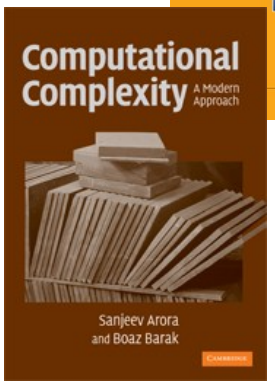
Literatur

Cormen, Leiserson, Rivest, Stein
(Introduction to Algorithms)

Kleinberg, Tardos
(Algorithm Design)

Ottmann, Widmayer
(Algorithmen und Datenstrukturen)

Arora, Barak
(Comput



Skript: Abgabe in der Pause für CHF 10,00.
(identisch zu letztem Jahr)

1. Was ist ein Algorithmus?

- Wie beschreibt man einen Algorithmus?
- Wie misst man die Laufzeit eines Algorithmus?
- Wie beweist man seine Korrektheit?

Beispiel

Szenario:

Gegeben: $n=10^9$ Zahlen $a_1,\ldots,a_n$;

Aufgabe: Sortiere sie!

Alternative Formulierungen:

- Schreibe ein Programm, das 10^9 Zahlen einliest und sie sortiert wieder ausgibt.

- Entwerfe einen Algorithmus Sort(a,n), der die Elemente in einem Feld a[ ] der Länge n sortiert.

1. Ansatz: BubbleSort

BubbleSort: C++ Programm

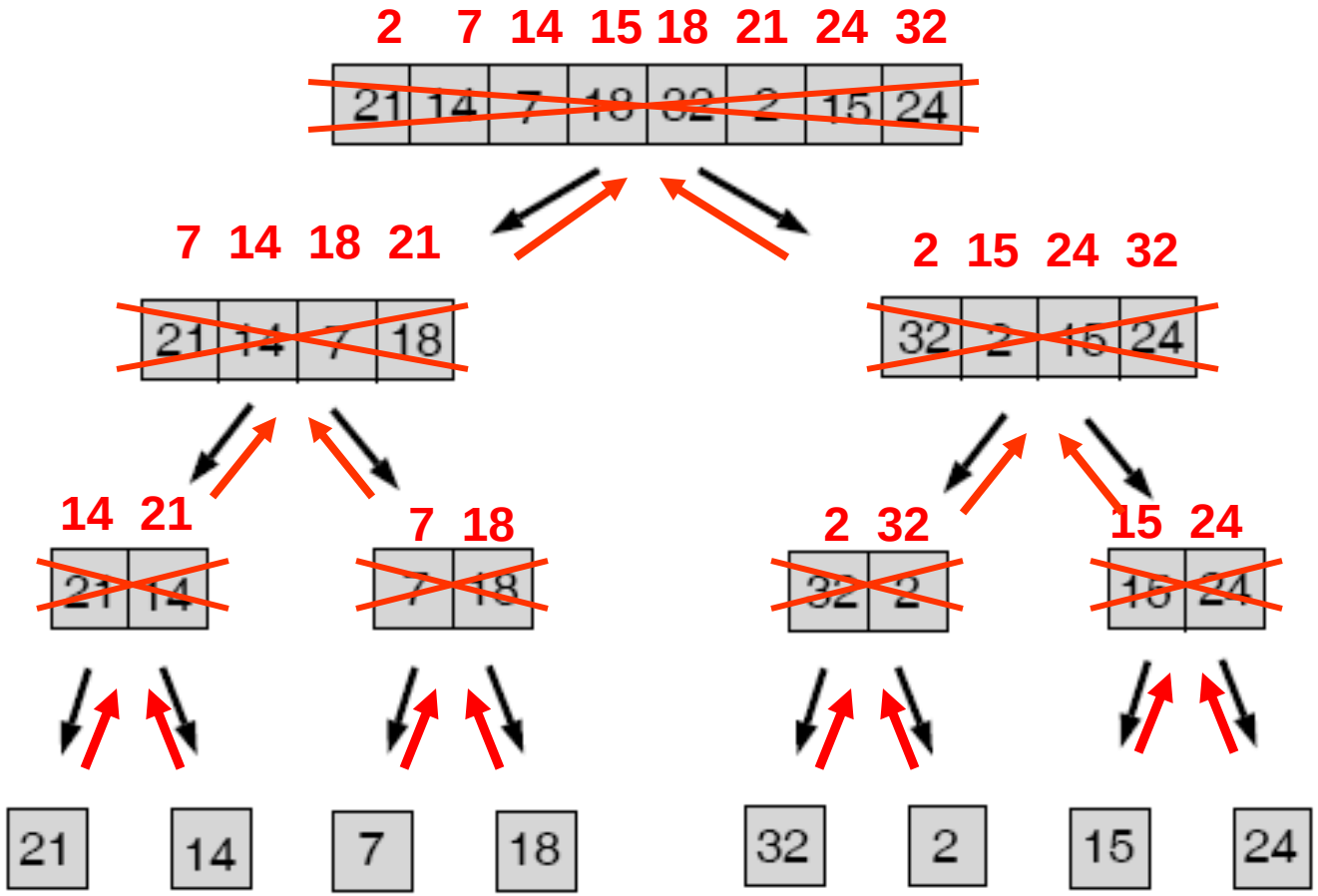
```
void bubble_sort (vector<int>& array) {
  int numSorted = 1;
  int n = int(array.size());

  while (numSorted < n) {
    for (int i = numSorted; i > 0; --i)
      if (array[i] < array[i-1])
        swap (array[i], array[i-1]);
    ++numSorted;
  }
}
```

BubbleSort: „intuitives“ Programm

BubbleSort:

```
for i=2 to n do
    for j = i downto 2 do
        if a[j] < a[j-1]  then swap(a[j],a[j-1]);
```

MergeSort - Beispiel



Welcher Algorithmus ist besser?

BubbleSort

oder

MergeSort

Welcher Algorithmus ist besser?

Naheliegende Antwort:

BubbleSort ist viel besser, denn er lässt sich viel einfacher implementieren!

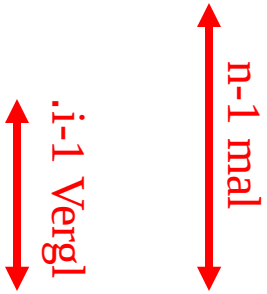
Aber:

Das Implementieren lohnt sich nicht, denn der Algorithmus ist für unsere Zwecke völlig unbrauchbar!!

BubbleSort: Aufwandsabschätzung

BubbleSort:

```
for i=2 to n do
    for j = i downto 2 do
        if a[j] < a[j-1]  then swap(a[j],a[j-1]);
```



Wie viele Vergleiche werden durchgeführt?

Anzahl Vergleiche:

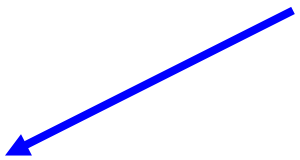
$$\sum_{i=2}^{n}(i-1) = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}$$

BubbleSort: Aufwandsabschätzung (2)

Anzahl Vergleiche für n zu sortierende Elemente:

$$\sum_{i=2}^{n}(i-1) = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}$$

Millionen Instruktionen pro Sekunde



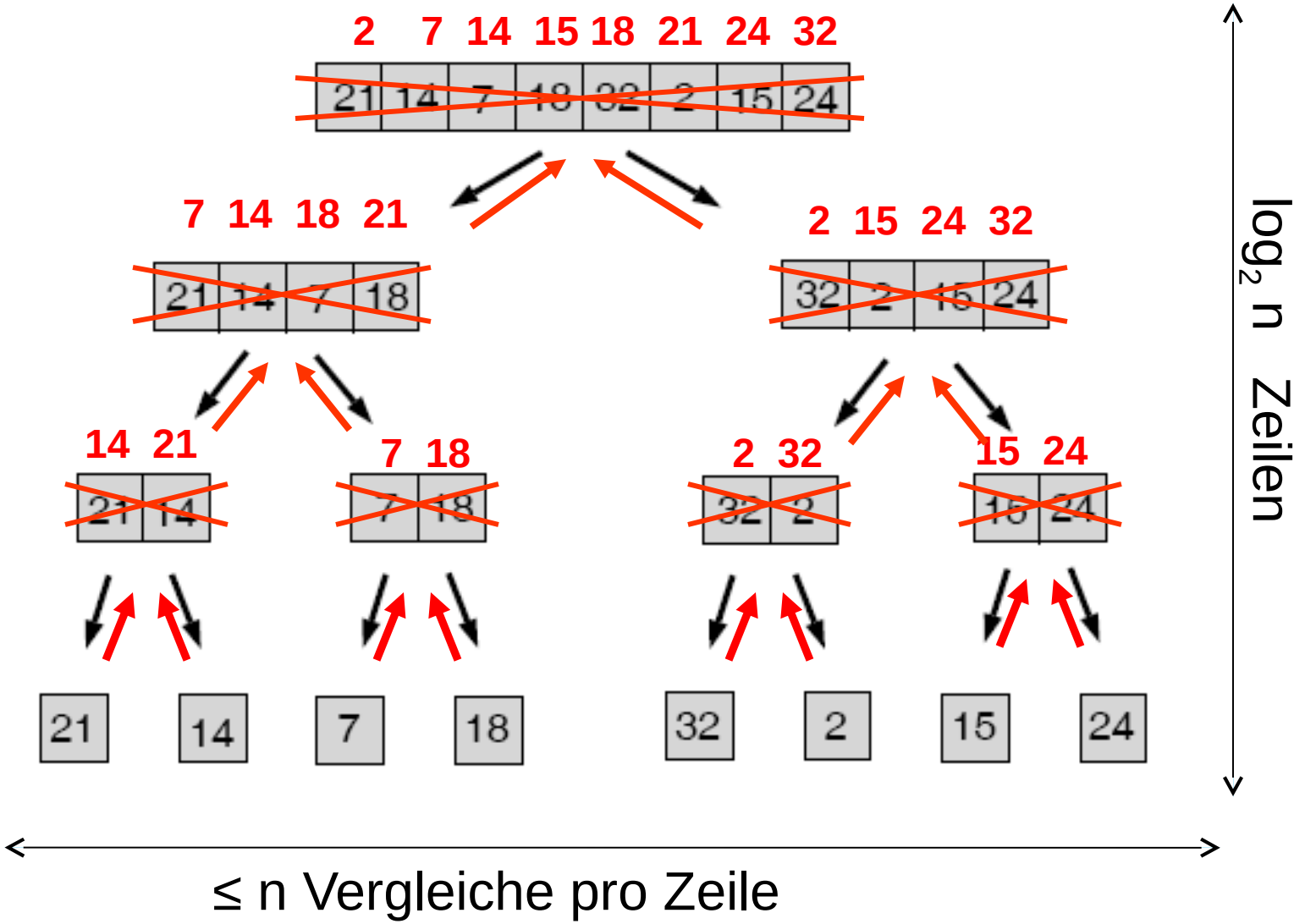
Unser Szenario: $n=10^9$

Annahme: 10^4 MIPS Rechner

Programm benötigt $\geq \frac{(10^9-1)\cdot 10^9}{2} \cdot \frac{1}{10^4 \cdot 10^6}$ Sek.

$\approx$ 1.5 Jahre

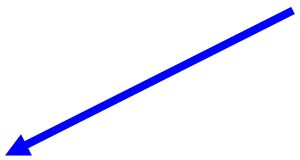
MergeSort - Beispiel



MergeSort: Aufwandsabschätzung (2)

Anzahl Vergleiche für n zu sortierende Elemente:

$$n \cdot \log_2 n$$



Millionen Instruktionen pro Sekunde

Unser Szenario: $n=10^9$

Annahme: 10^4 MIPS Rechner

Programm benötigt $\geq 10^9 \cdot \log_2 10^9 \cdot \frac{1}{10^4 \cdot 10^6}$ Sek.

≈ 3 Sek.

Algorithmen – Ziel der Vorlesung

Lerne,

gegeben ein Szenario/Aufgabenstellung,

dafür einen „passenden“ Algorithmus zu

- entwickeln und zu
- analysieren

Wie analysiert man einen Algorithmus?

Ziele:

- Zeige, dass der Algorithmus korrekt ist.

  D.h., zeige dass er für alle (erlaubten) Eingaben das richtige Ergebnis liefert.

- Bestimme, die Laufzeit des Algorithmus.

  Wie?? — Was für ein Rechner??

  „abstrakter“ Rechner: Random Access Machine

  Welche Eingaben??

  z.B. Worst Case Analyse:

  zeige dass Alg. für alle Eingaben einer gegebenen Länge höchstens soundso lange läuft.

Das Rechenmodell: Was ist wichtig?

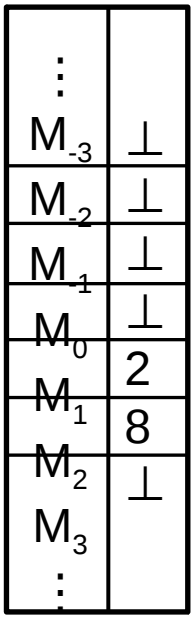
1. Wir haben einen Speicher auf den wir mit Operationen zugreifen können.

2. Eine gewisse, kleine Menge von Basis-Operationen.

3. Es ist möglich, das Rechenmodell formal zu definieren.

Speicher

$M: \mathbb{Z} \mapsto \mathbb{Z} \cup \{\perp\}$



$M_i = M(i)$

Random Access Machine

$M_i \leftarrow 1$	("*schreibe die Konstante 1 in Speicherzelle* M_i"),
$M_i \leftarrow M_j + M_k$	("*schreibe die Summe der Zahlen, die in den Zellen* M_j *und* M_k *stehen, in die Zelle* M_i"),
$M_i \leftarrow M_j - M_k$	("*schreibe die Differenz der Zahlen, die in den Zellen* M_j *und* M_k *stehen, in die Zelle* M_i"),
$M_i \leftarrow M_j * M_k$	("*schreibe das Produkt der Zahlen, die in den Zellen* M_j *und* M_k *stehen, in die Zelle* M_i"),
$M_i \leftarrow \lfloor M_j / M_k \rfloor$	("*teile die Zahl in* M_j *durch die Zahl, die in* M_k *steht, und schreibe den ganzzahligen Teil des Ergebnisses in Zelle* M_i"),
$M_i \leftarrow M_{M_j}$	("*schreibe in Zelle* M_i *die Zahl in der Zelle, deren Index in Zelle* M_j *steht*"),
$M_{M_i} \leftarrow M_j$	("*schreibe die Zahl, die in* M_j *steht, in die Zelle, deren Index in* M_i *steht*"),
GOTO m IF $M_i > 0$	("*wenn die Zahl, die in* M_i *steht, grösser als 0 ist, fahre mit dem* m*ten Befehl des Programms fort; ansonsten fahre mit dem nächsten fort*").

RAM vs. „heutiger“ Computer

- grösserer Befehlssatz
- Instruktionen werden geschickt verzahnt ausgeführt („pipelining“)
- moderne Programmiersprachen bieten „high level“ Zugang

Aber: Modelle im Prinzip sehr ähnlich,
Laufzeiten verhalten sich ähnlich

Was heisst „ähnlich“ ??

Vorlesung/Algorithmentheorie:

Wir analysieren Laufzeit von Algorithmen nur
bis auf einen konstanten Faktor
genau.

Beispiel: BubbleSort

BubbleSort:

```
for i=2 to n do
    for j = i downto 2 do
        if a[j] < a[j-1]  then swap(a[j],a[j-1]);
```

Beobachtung:

Laufzeit ist *proportional* zur Anzahl Vergleich.

Wir zählen *nur* die Anzahl Vergleiche.

Notation

Schreibweise

- für „bis auf konstanten Faktor genau“

bzw.

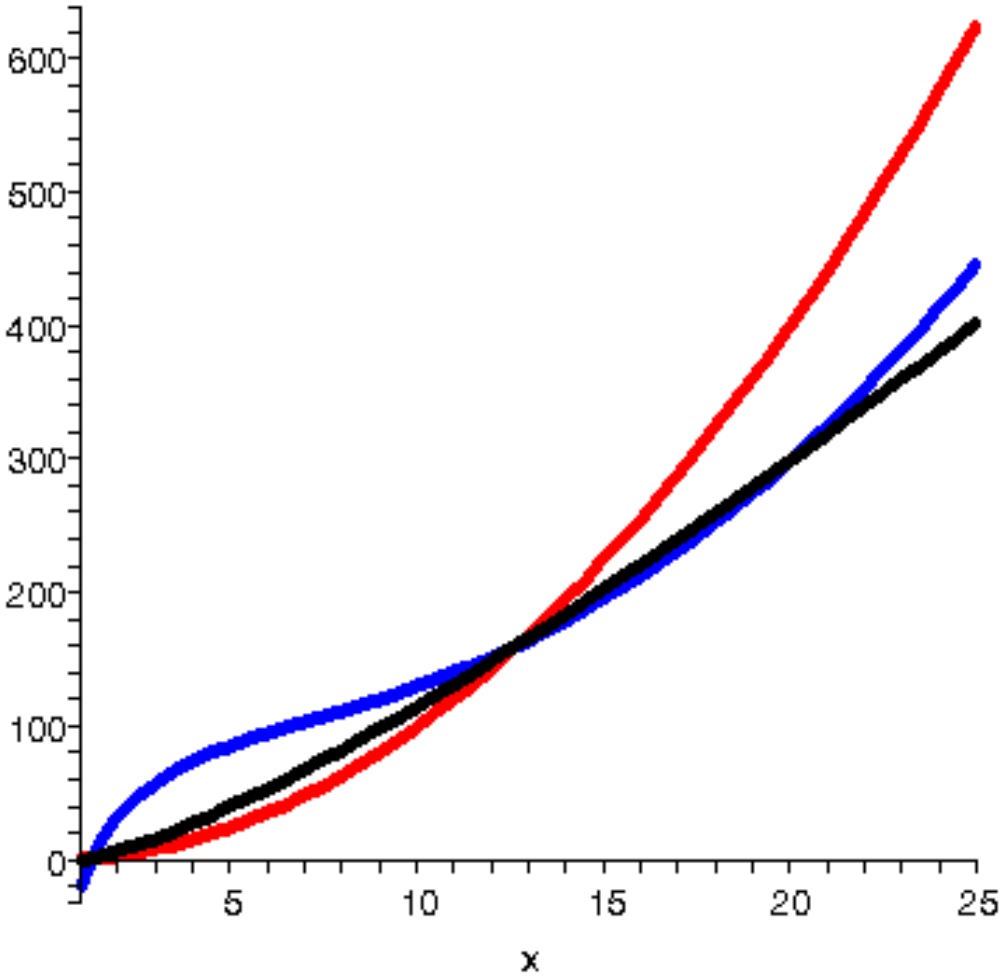
- für „proportional zu“

Gross-Oh-Notation

Landau-Symbole

Notation	*Definition*	*Bedeutung*
$f(n) = \mathcal{O}(g(n))$	$\limsup_{n\to\infty} \frac{\lvert f(n)\rvert}{\lvert g(n)\rvert} < \infty$	$f(n)$ grows at most as fast as $g(n)$
$f(n) = \Omega(g(n))$	$\liminf_{n\to\infty} \frac{\lvert f(n)\rvert}{\lvert g(n)\rvert} > 0$	$f(n)$ grows at least as fast as $g(n)$
$f(n) = \Theta(g(n))$	$f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$	$f(n)$ grows as fast as $g(n)$
$f(n) = o(g(n))$	$\lim_{n\to\infty} \frac{\lvert f(n)\rvert}{\lvert g(n)\rvert} = 0$	$f(n)$ grows slower than $g(n)$
$f(n) = \omega(g(n))$	$\frac{\lvert f(n)\rvert}{\lvert g(n)\rvert} \to \infty$	$f(n)$ grows faster than $g(n)$

Beispiel

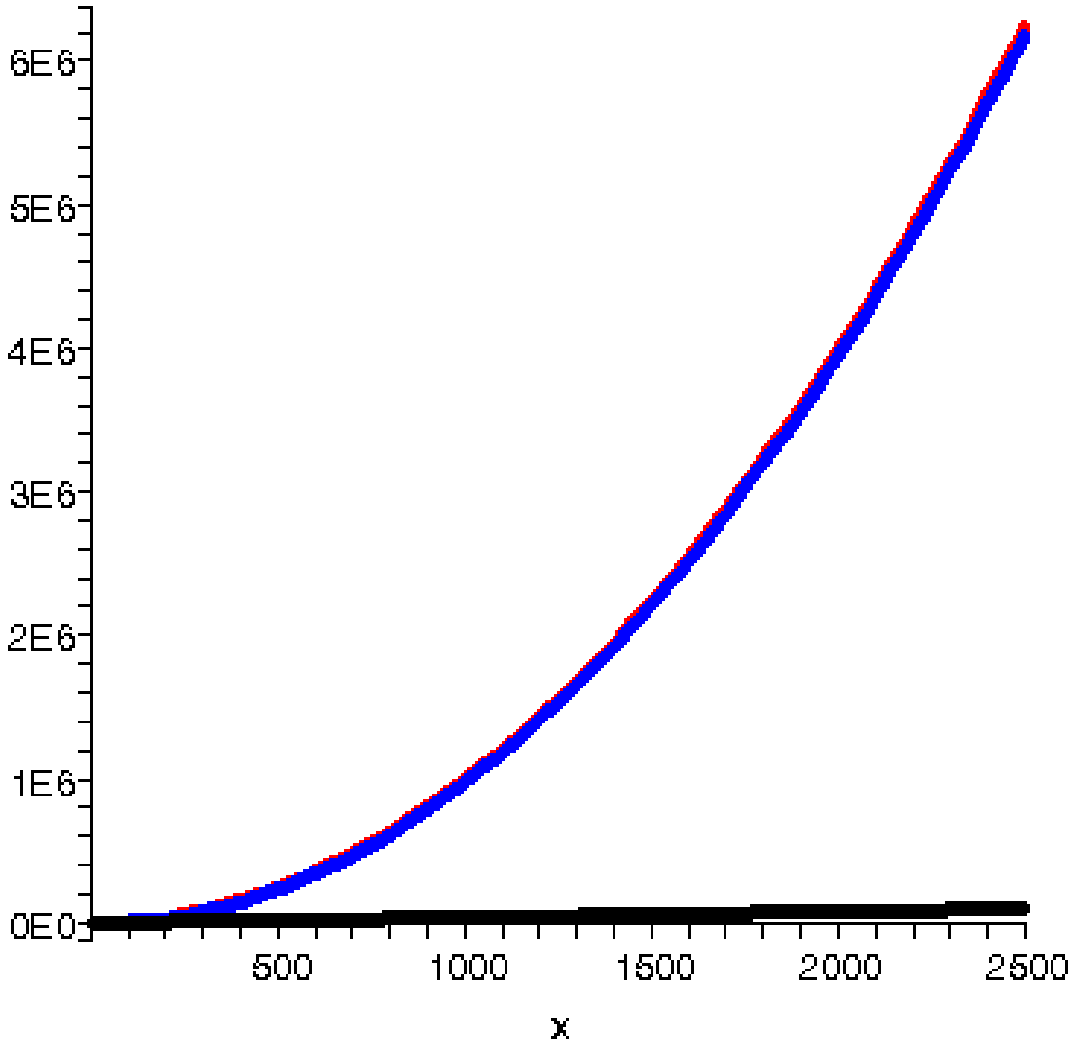


x^2

$x^2-20*x+100*\log(x)$

$5*x*\log(x)$

Beispiel



x^2

x^2-20*x+100*log(x)

5*x*log(x)

Wie misst man die Laufzeit??

Anzahl arithmetischer Operationen

bis auf einen konstanten Faktor genau

… aber für welche Eingabe??

Idee: „Worst Case Analyse“

$T_{\mathcal{A}lg}(n) := \max\{T(I) \mid I \text{ ist Eingabe der Länge } n\}$

Beispiel: BubbleSort hat Laufzeit $O(n^2)$

MergeSort hat Laufzeit $O(n \log n)$

Algorithmen-Klassifikation

linearer Algorithmus, $T_{\mathcal{A}lg}(n) = \mathcal{O}(n)$,

quasi-linearer Algorithmus, $T_{\mathcal{A}lg}(n) = \mathcal{O}(n \log n)$,

quadratischer Algorithmus, $T_{\mathcal{A}lg}(n) = \mathcal{O}(n^2)$,

polynomieller Algorithmus, $T_{\mathcal{A}lg}(n) = \mathcal{O}(n^k)$ für ein $k \in \mathbb{N}$,

exponentieller Algorithmus, $T_{\mathcal{A}lg}(n) = 2^{\mathcal{O}(n)}$.

Achtung: Man kann „schummeln“!!

Berechnung von n!

```
x := 1
for j = 1 to n do
    x = x * j;
return(x)
```

Anzahl Multiplikationen: n

Länge der Ausgabe: $\log_2(n!) \approx n \log_2(n)$

Es ist unrealistisch, dass man in n Schritten
$n \log_2(n)$ Bits schreiben kann!!

Erklärung

- die Länge der Eingabe ist $\log_2 n$

- die vom Algorithmus berechneten Zahlen sind aber (exponentiell) viel grösser

In diesem Fall muss man statt der Anzahl arithmetischer Operationen die Anzahl Bit-Operationen zählen.

Problem tritt in der Vorlesung nicht auf…

…wir zählen Anzahl arithmetischer Operationen.

Graphenalgorithmen

Szenario

Gegeben:

Flugplan einer Airline

Aufgabe:

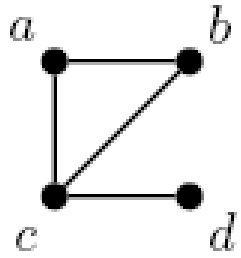
Schreibe ein Programm, das Anfragen der Form

„Kann man von A nach B mit höchstens einmal umsteigen gelangen?“

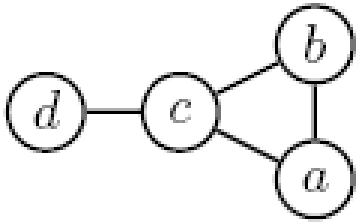
beantwortet.

Modellierung

Graph



oder auch so



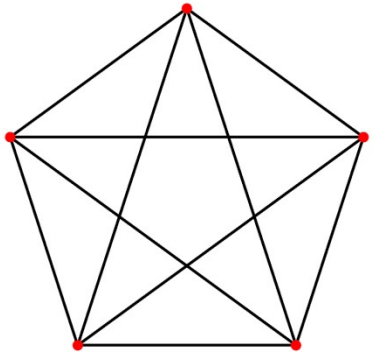
Ein Graph G ist ein Tupel (V,E), wobei V eine (endliche) nichtleere Menge von Knoten ist.

Die Menge E ist eine Teilmenge der zweielementigen Teilmengen von V, also $E \subseteq \{ \{x,y\} : x,y \in V, x \neq y\}$.

Die Elemente der Menge E bezeichnet man als Kanten.

Einige spezielle Graphenklassen (I)

Vollständiger Graph K_n

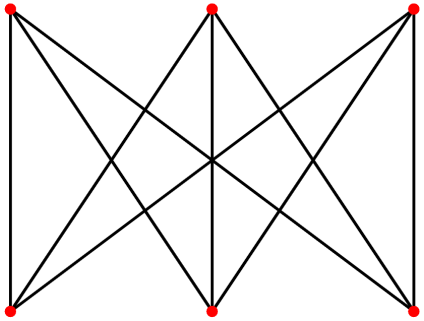


Kreis C_n

Pfad P_n

Einige spezielle Graphenklassen (II)

Vollständiger bipartiter Graph $K_{n,n}$



Hyperwürfel Q_d