

Datenstrukturen und Algorithmen

SS 2017

Prof. Dr. Christian Scheideler

Organisatorisches

Vorlesung:

- Mo 11:15 - 12:45 Hörsaal L1
- Fr 11:15 - 12:45 Hörsaal L1

Zentralübung:

- **Fr 13:00 – 13:45** Hörsaal L1
- Beginn: 12. Mai

Übungen:

- Beginn: übernächste Woche

Organisatorisches

Übungen:

- Jede Woche ein Übungsblatt (Freitags)
- 1. Blatt nächsten Freitag auf der Kurswebseite
- Abgabe: Montag übernächste Woche bis 11:15 Uhr
verspätete Abgaben werden nicht berücksichtigt!
- Gruppengröße: bis zu 3 Personen
- Vorstellung von Musterlösungen in der Zentralübung
- **Zulassung zur Klausur:**
bei 75% der Zettel mindestens 20% der Punkte
- **Bonuspunkt** (falls Klausur bestanden):
mindestens 60% der Punkte und einmal vorrechnen

Organisatorisches

Übungsgruppen:

- Anmeldung über PAUL
- Übungsgruppenzeiten: siehe Kurswebseite

<http://cs.uni-paderborn.de/ti/lehre/veranstaltungen/ss-2017/datenstrukturen-und-algorithmen/>

Organisatorisches

Klausur:

- Eine Korrelation mit den Übungsaufgaben ist zu erwarten
- Es gab in der Vergangenheit einen direkten Zusammenhang zwischen Übungsteilnahme/abgabe und gutem Abschneiden bei Klausuren
- Hilfsmittel: ein handbeschriebenes DIN A4-Blatt

Sprechzeiten:

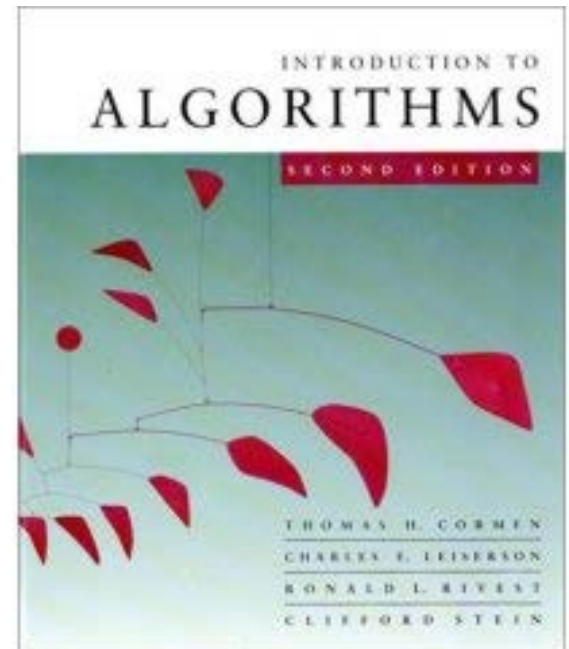
- Do 16:00 – 17:00 (Raum F2.326; Fürstenallee)
- Fragen/Anmerkungen zunächst an den **eigenen Tutor**

Organisatorisches

Literatur:

Cormen, Leiserson, Rivest, Stein:
Introduction to Algorithms, 3rd ed.
MIT Press/McGraw-Hill
ISBN 0-262-53305-8

Weitere Literatur siehe Webseite



Inhaltsangabe:

- Einleitung, Motivation
- Pseudocode, Invarianten, Laufzeitanalyse
- Groß-O-Notation

} Grundlagen

- Inkrementelle Algorithmen: Insertion-Sort
- Divide & Conquer Algorithmen: Merge-Sort
- Quick-Sort Analyse und Varianten
- Rekursionsgleichungen: Master-Theorem
- Algorithmen mit Datenstrukturen: Heaps, Heap-Sort
- Untere Schranke für Vergleichssortierer
- Counting-Sort

} Sortieralgorithmen

- ADTs und Datenstrukturen: Stacks, Queues, Listen, Bäume
- Hashtabellen
- Binäre Suchbäume

} Datenstrukturen

- Elementare Graphalgorithmen: Tiefen- und Breitensuche
- Zusammenhangskomponenten
- Kürzeste Wege: Algorithmus von Dijkstra
- Minimale Spannbäume: Algorithmus von Prim und Kruskal

} Graphalgorithmen

- Gierige Algorithmen: Scheduling
- Dynamische Programmierung: Rucksack-Problem
- Amortisierte Laufzeitanalyse
- Optimale Suchbäume

} Entwurfsmethoden

1. Einführung

- Was ist ein Algorithmus (bzw. eine Datenstruktur)?
- Welche Probleme kann man damit lösen?
- Warum betrachten wir (effiziente) Algorithmen?
- Wie beschreiben wir Algorithmen?
- Nach welchen Kriterien beurteilen wir Algorithmen?
- Welche Algorithmen betrachten wir?
- Wie passt *Datenstrukturen und Algorithmen* ins Informatikstudium?

Was ist ein Algorithmus?

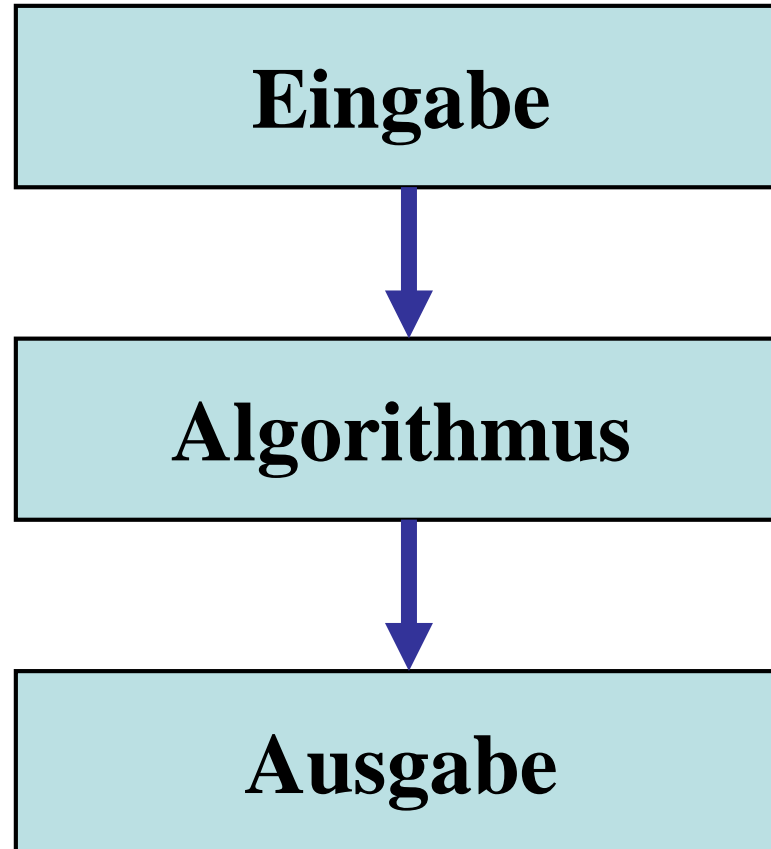
Definition 1.1: Ein *Algorithmus* ist eine eindeutige Beschreibung eines Verfahrens zur Lösung einer bestimmten Klasse von Problemen.

Hier: Ein Algorithmus ist eine Menge von Regeln für ein Verfahren, um aus gewissen Eingaben bestimmte Ausgaben zu berechnen. Dabei muss

1. Das Verfahren in einem endlichen Text beschreibbar sein.
2. Jeder Schritt des Verfahrens auch tatsächlich ausführbar sein.
3. Der Ablauf des Verfahrens zu jedem Zeitpunkt eindeutig definiert sein.

Was ist ein Algorithmus?

Anschaulich:



Beispiel Sortieren

Eingabe beim Sortieren: Folge von n Zahlen (a_1, a_2, \dots, a_n) .

Ausgabe beim Sortieren: Umordnung (b_1, b_2, \dots, b_n) der Eingabefolge, so dass $b_1 \leq b_2 \leq \dots \leq b_n$.

Sortieralgorithmus: Verfahren, das zu jeder Folge (a_1, a_2, \dots, a_n) sortierte Umordnung (b_1, b_2, \dots, b_n) berechnet.

Eingabe: (31,41,59,26,51,48)

Ausgabe: (26,31,41,48,51,59)

Was ist eine Datenstruktur?

Definition 1.2: Eine *Datenstruktur* ist eine bestimmte Art, Daten im Speicher eines Computers so anzuordnen, dass Operationen wie z.B. *Suchen*, *Einfügen*, *Löschen* einfach zu realisieren sind.

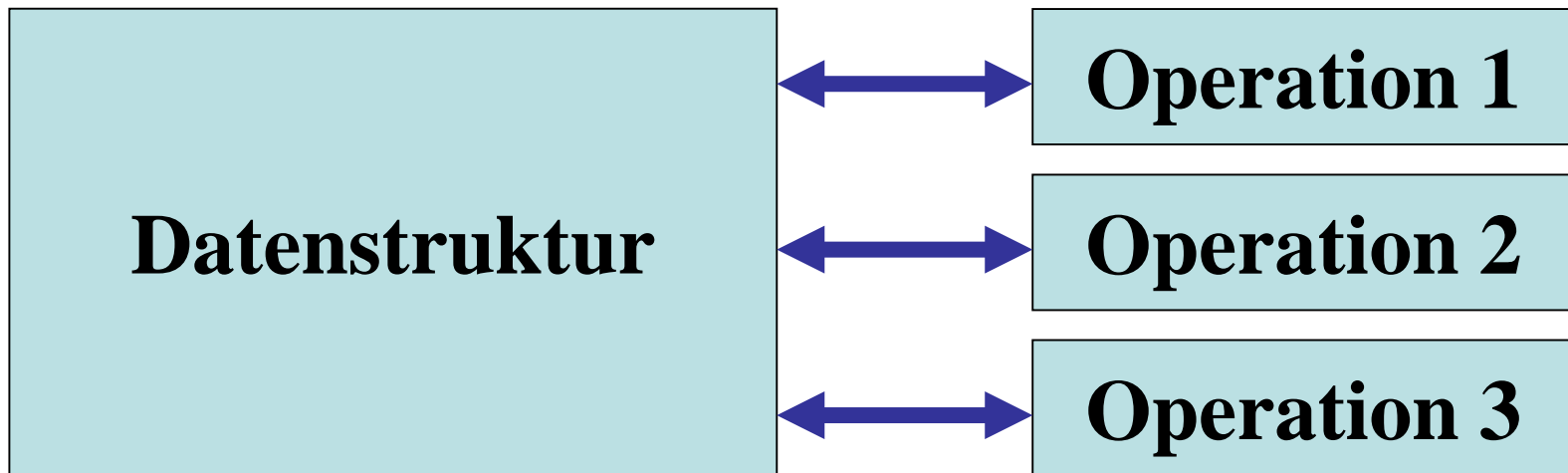
Einfache Beispiele:

- *Listen*
- *Arrays*

Datenstrukturen und Algorithmen sind eng verbunden: Gute Datenstrukturen sind häufig unerlässlich für gute Algorithmen.

Was ist eine Datenstruktur?

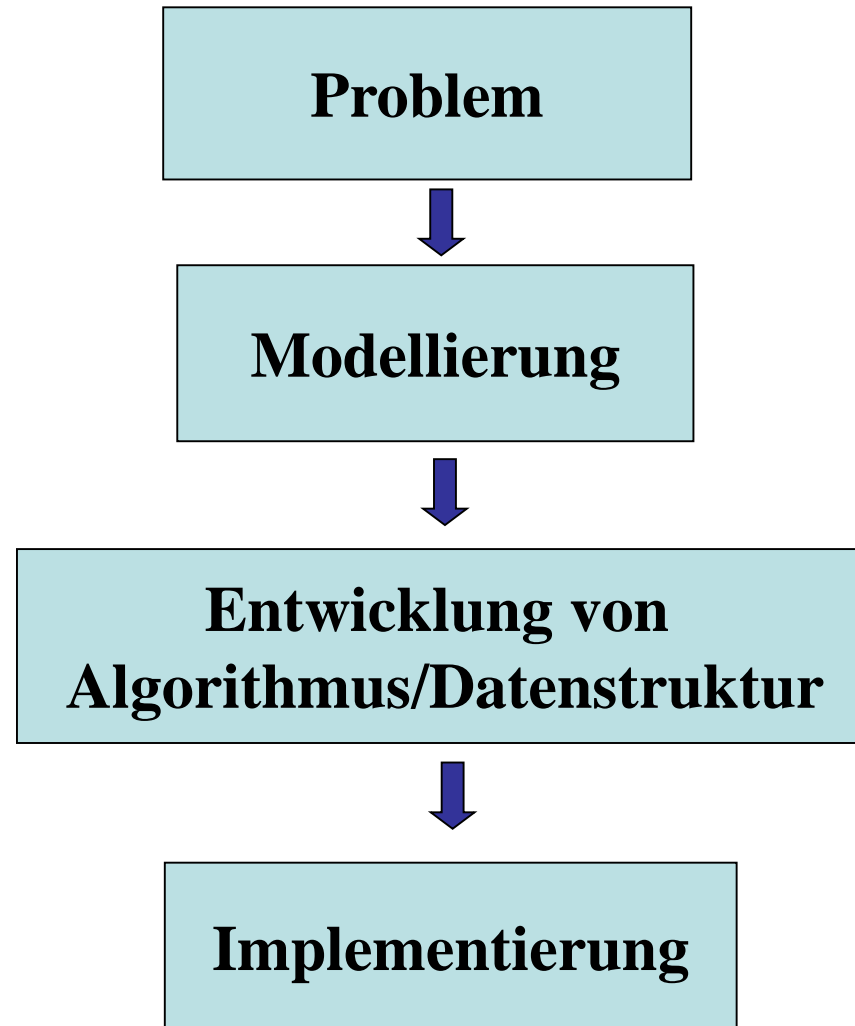
Anschaulich:



Beispielprobleme

- Wie findet ein Navigationssystem gute Verbindungen zwischen zwei Orten? Wie werden im Internet Informationen geroutet?
- Wie berechnet ein Unternehmen eine möglichst gute Aufteilung seiner Ressourcen, um seinen Gewinn zu maximieren?
- Wie werden etwa in Google Informationen schnell gefunden?
- Wie werden Gleichungssysteme der Form $Ax=b$ gelöst?

Softwareentwicklung



Kriterien für Algorithmen

- Algorithmen müssen korrekt sein.
 - ⇒ Benötigen *Korrektheitsbeweise*.
- Algorithmen sollen zeit- und speichereffizient sein.
 - ⇒ Benötigen *Analysemethoden für Zeit- und Speicherbedarf*.
- Analyse basiert *nicht auf* empirischen Untersuchungen, sondern auf *mathematischen Analysen*. Nutzen hierfür *Pseudocode* und *Basisoperationen*.

Algorithmenentwurf

Idealerweise gehören zum Entwurf eines Algorithmus

1. *Formale Beschreibung des Algorithmus*
2. *Korrektheitsbeweis*
3. *Zeit- bzw. Speicherbedarfsanalyse.*

Beschreibung von Algorithmen

- Zunächst informelles (bzw. abstraktes) Verfahren zur Lösung.
- Dann Präzisierung durch *Pseudocode*.
- *Keine* Beschreibung durch Programmcode in Java oder C, C++, aber wir planen Programmieraufgaben.

Warum mathematische Korrektheitsbeweise?

- Fehler können fatale Auswirkungen haben (Steuerungssoftware in Flugzeugen, Autos, AKWs)
- Fehler können selten auftreten („Austesten“ funktioniert nicht)

Besonders teurerer algorithmischer Fehler:

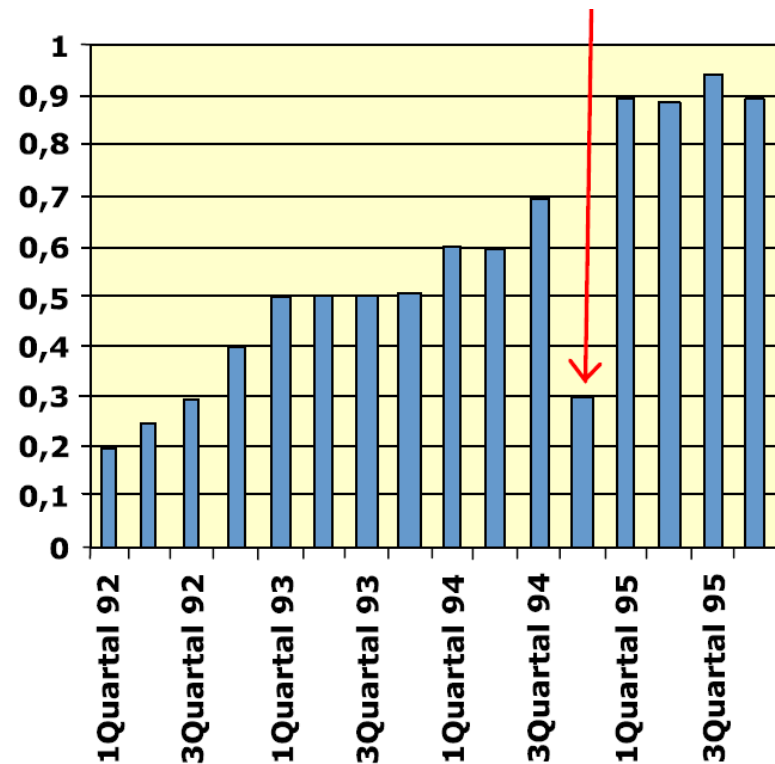
- Pentium bug (\approx \$500 Mio.)
- Enormer Imageschaden
- Trat relativ selten auf

Algorithmenentwurf

Die Berechnung von
 $z = x - (x/y)y$
sollte $z=0$ ergeben.

Der fehlerhafte
Pentium-Prozessor
berechnete statt dessen
mit den Werten
 $x=4195835$ und
 $y=3145727$
den Wert $z=256$

\$480 Mio. entgangener Gewinn



Quartalsergebnisse der Firma Intel

Wozu gute Algorithmen?

- Computer werden zwar immer schneller und Speicher immer größer und billiger, aber
- Geschwindigkeit und Speicher werden immer begrenzt sein.
- Daher muss mit den Ressourcen *Zeit* und *Speicher* geschickt umgegangen werden.
- Außerdem werden in Anwendungen immer größere Datenmengen verarbeitet. Diese wachsen schneller als Geschwindigkeit oder Speicher von Computern.

Effekt guter Algorithmen - Sortieren

Insertion - Sort :

Sortiert n Zahlen mit $c_1 n^2$ Vergleichen.

Computer A :

10^9 Vergleiche/Sek.

$c_1 = 2, n = 10^6$.

Dann benötigt A

$$\frac{2 \cdot (10^6)^2}{10^9} = 2000 \text{ Sek.}$$

Merge - Sort :

Sortiert n Zahlen mit $c_2 n \log(n)$ Vergleichen.

Computer B :

10^7 Vergleiche/Sek.

$c_2 = 50, n = 10^6$.

Dann benötigt B

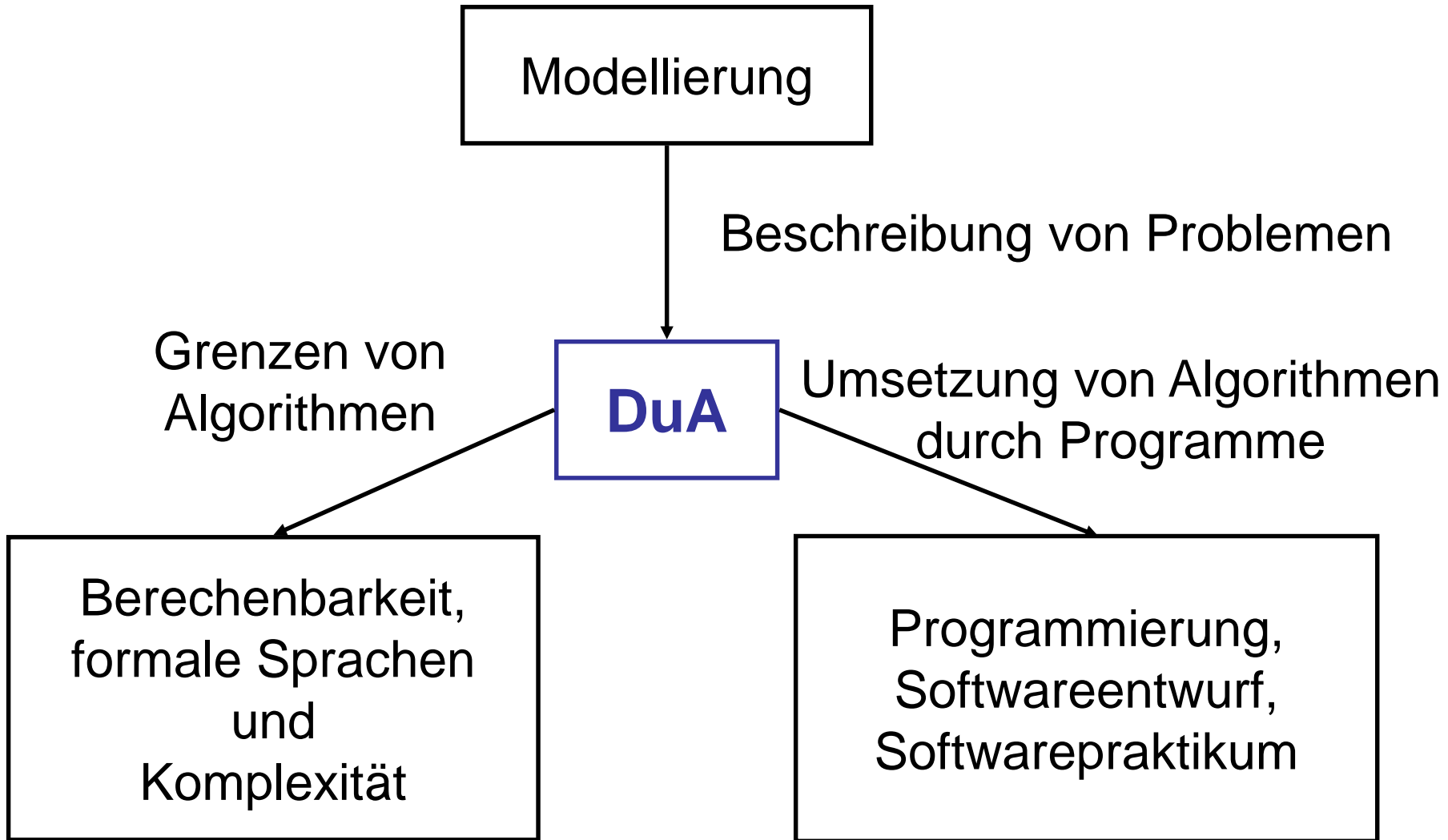
$$\frac{50 \cdot (10^6) \log(10^6)}{10^7} \approx 100 \text{ Sek.}$$

Probleme, Algorithmen, Ziele

Ziel der Vorlesung ist es

1. Wichtige Probleme, Algorithmen und Datenstrukturen kennen zu lernen.
2. Wichtige Algorithmentechniken und Entwurfsmethoden kennen und *anwenden* zu lernen.
3. Wichtige Analysemethoden kennen und *anwenden* zu lernen.
4. Das Zusammenspiel von Datenstrukturen und Algorithmen zu verstehen.

DuA + Info-Studium



Beispiele

Ziel: Einführung in effiziente Algorithmen am Beispiel ganzzahliger Arithmetik

- Notation
- Arithmetik auf großen Zahlen
 - Addition
 - Multiplikation
- Kreisberechnung

Notation

Ganzzahlige Arithmetik:

- Ganze Zahl: Ziffernfolge zur Basis B
- “Zur Basis B ”: Ziffern aus $[B] = \{0, \dots, B-1\}$
- Zahl $a = (a_{n-1} \dots a_0)_B$ zur Basis B hat den Wert

$$\sum_{i=0}^{n-1} a_i B^i$$

Wichtige Beispiele:

- $B=2$: Binärzahlen ($(101)_2 = 1 \cdot 2^2 + 1 \cdot 2^0 = 5$)
- $B=10$: Dezimalzahlen
- $B=16$: Hexadezimalzahlen ($(A)_{16} = 10$)
Ziffern: $\{0, \dots, 9\} \cup \{A, B, C, D, E, F\}$

Notation

Pseudocode:

- Schleifen (for, while, repeat)
- Bedingtes Verzweigen (if – then – else)
- (Unter-)Programmaufruf/Übergabe (return)
- Zuweisung durch \leftarrow
- Kommentar durch \triangleright
- Daten als Objekte mit einzelnen Feldern oder Eigenschaften (z.B. $length(A)$:= Länge des Arrays A)
- Blockstruktur durch Einrückung

Notation

Elementare arithmetische Operationen:

Seien $a, b, c, x, y \in [B]$.

- $(x \ y)_B \leftarrow a+b+c$
- $(x \ y)_B \leftarrow a \cdot b$

Beispiele für $B=10$:

- $(12)_{10} = 5+6+1$
- $(42)_{10} = 6 \cdot 7$

Basis B klar: lassen wir weg

Addition

Es gilt:

- $c_0 = 0$
- $(c_{i+1} \ s_i) = a_i + b_i + c_i$ für alle $i \geq 0$
- $s_n = c_n$

Algorithmus:

- $c \leftarrow 0$
- for $i \leftarrow 0$ to $n-1$ do
 $(c \ s_i) \leftarrow a_i + b_i + c$
- $s_n \leftarrow c$

for ... to ... do ... - Schleife

Elementaroperation

Addition

Satz 1.3: Die Addition zweier Zahlen der Länge n benötigt höchstens n Elementaradditionen

Beobachtung: Jede Addition n -stelliger Zahlen a und b benötigt mindestens n Elementaradditionen, d.h. unser Algo ist **optimal**.

Multiplikation

- Gegeben: zwei Zahlen $a=(a_{n-1}\dots a_0)$ und $b=(b_{n-1}\dots b_0)$
- Gesucht: $p=(p_{2n-1}\dots p_0)$ mit $p = a \cdot b$

Schulmethode:

- berechne $a \cdot b_j$ für alle $j \geq 0$
- addiere Ergebnisse versetzt zusammen

Multiplikation

$$\begin{array}{r} (a_{n-1} \dots a_0) \cdot b_j \xrightarrow{(c_i \ d_i) = a_i \cdot b_j} \begin{array}{r} c_{n-1} \ c_{n-2} \ \dots \ c_0 \ 0 \\ + \ d_{n-1} \ \dots \ d_1 \ d_0 \\ \text{Übertrag} \ e_n \ e_{n-1} \ \dots \ e_1 \ e_0 \\ \hline p_n \ p_{n-1} \ \dots \ p_1 \ p_0 \end{array} \end{array}$$

Algorithmus:

$e \leftarrow 0; c_{-1} \leftarrow 0$

for $i \leftarrow 0$ to $n-1$ do

$(c_i \ d_i) \leftarrow a_i \cdot b_j; (e \ p_i) \leftarrow c_{i-1} + d_i + e$

$p_n \leftarrow c_{n-1} + e$

Multiplikation

Lemma 1.4: Wir können eine Zahl der Länge n mit einer Ziffer in $2n+1$ elementaren arithmetischen Operationen multiplizieren.

Sei $p_j = a \cdot b_j$ mit $p_j = (p_{j,n} \cdots p_{j,0})$.

Dann müssen wir noch

$$p = \sum_{j=0}^{n-1} p_j \cdot B^j$$

berechnen.

Multiplikation

Algorithmus:

$p \leftarrow 0$

for $j \leftarrow 0$ to $n-1$ do

$p \leftarrow p + (a \cdot b_j) B^j$

Satz 1.3

Lemma 1.4

nur Positionierung

Anzahl Elementaroperationen maximal

- $\leq 2n$ wegen Satz 1.3, insgesamt $\leq 2n^2$
- $2n+1$ wegen Lemma 1.4, insgesamt $n(2n+1)$

Also insgesamt max. $4n^2+n$ Elementaroperationen

Multiplikation

Satz 1.5: Die Schulmethode multipliziert zwei Zahlen der Länge n mit höchstens $4n^2+n$ primitiven Operationen.

Können wir besser werden?

Multiplikation

Rekursive Version der Schulmethode:

Gegeben: zwei n -stellige Zahlen a, b zur Basis B , n gerade

Sei $a = a_1 \cdot B^k + a_0$ und $b = b_1 \cdot B^k + b_0$ mit $k = n/2$

Dann gilt:

$$\begin{aligned} a \cdot b &= (a_1 \cdot B^k + a_0) \cdot (b_1 \cdot B^k + b_0) \\ &= a_1 \cdot b_1 B^{2k} + (a_1 \cdot b_0 + a_0 \cdot b_1) B^k + a_0 \cdot b_0 \end{aligned}$$

4 rekursive Aufrufe zur Multiplikation $n/2$ -stelliger Zahlen

Multiplikation

Annahme: $|a|=|b|=n=2^c$ für ein $c \in \mathbb{N}$

Algorithmus Produkt(a,b): 

if $|a|=|b|=1$ then return $a \cdot b$

else

if ... then ... else ... - Anweisung

$k \leftarrow |a|/2$

return Produkt(a_1, b_1) $\cdot B^{2k}$ \oplus (Produkt(a_1, b_0)
 \oplus Produkt(a_0, b_1)) $\cdot B^k$ \oplus Produkt(a_0, b_0)

Multiplikation

Anzahl Elementaroperationen (\cdot , $+$):

$$T(n) = \begin{cases} 1 & n=1 \\ 4T(n/2) + 3 \cdot (2n) & n=2^c, c \in \mathbb{N} \end{cases}$$

Daraus ergibt sich $T(n) = 7n^2 - 6n$ für alle
 $n=2^c, c \in \mathbb{N}_0$

Beweis: durch Induktion

Multiplikation

Karatsubas Idee:

$$\begin{aligned} a \cdot b &= (a_1 \cdot B^k + a_0) \cdot (b_1 \cdot B^k + b_0) \\ &= a_1 b_1 \cdot B^{2k} + (a_1 b_0 + a_0 b_1) \cdot B^k + a_0 b_0 \\ &= a_1 b_1 B^{2k} + ((a_1 + a_0)(b_1 + b_0) - \\ &\quad (a_1 b_1 + a_0 b_0)) B^k + a_0 b_0 \end{aligned}$$

Nur noch **drei** rekursive Aufrufe für
 $a_1 b_1$, $a_0 b_0$, $(a_1 + a_0)(b_1 + b_0)$

Multiplikation

Algorithmus Karatsuba(a, b):

if $|a|=|b|=1$ then return $a \cdot b$

else

$k \leftarrow |a|/2$

$p_1 \leftarrow \text{Karatsuba}(a_1, b_1)$

$p_2 \leftarrow \text{Karatsuba}(a_1 + a_0, b_1 + b_0)$

$p_3 \leftarrow \text{Karatsuba}(a_0, b_0)$

return $p_1 \cdot B^{2k} + (p_2 - (p_1 + p_3)) \cdot B^k + p_3$

Multiplikation

Anzahl Elementaroperationen:

$$T(n) = \begin{cases} 1 & n=1 \\ 3T(n/2+1)+6(2n) & n>1 \end{cases}$$

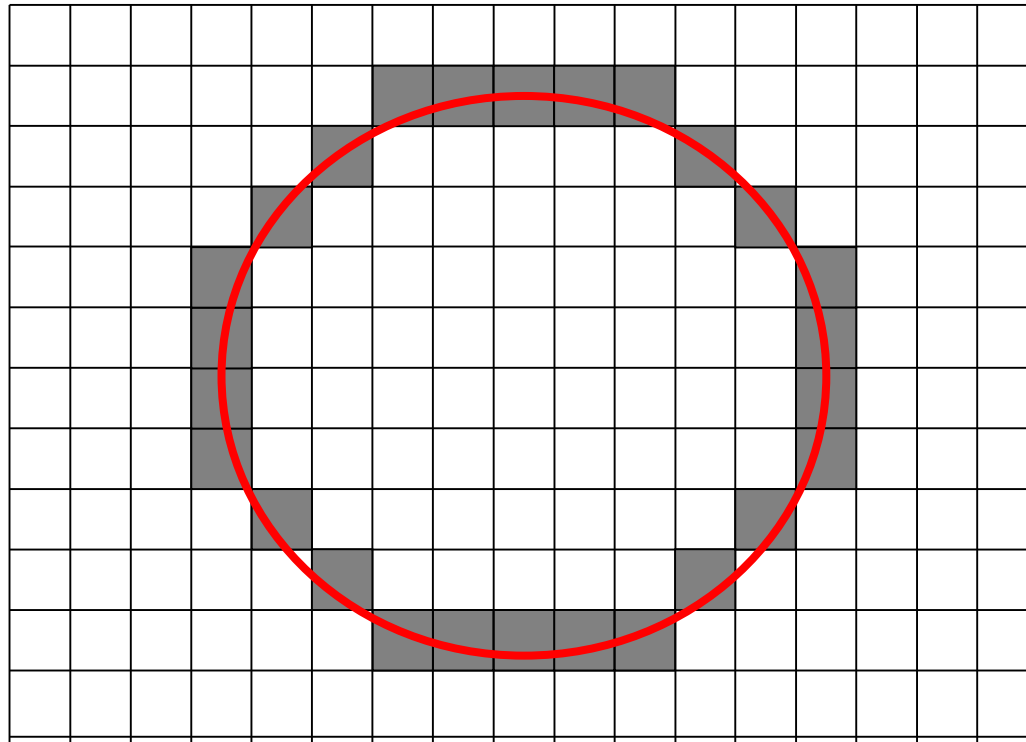
Daraus ergibt sich $T(n) \sim n^{\log 3} = n^{1,58\dots}$

Problem: Karatsuba erst für $n > 1.000.000$ besser als Schulmethode

Bester Algo für Multiplikation: $O(n \log n \log \log n)$

Kreiszeichnen

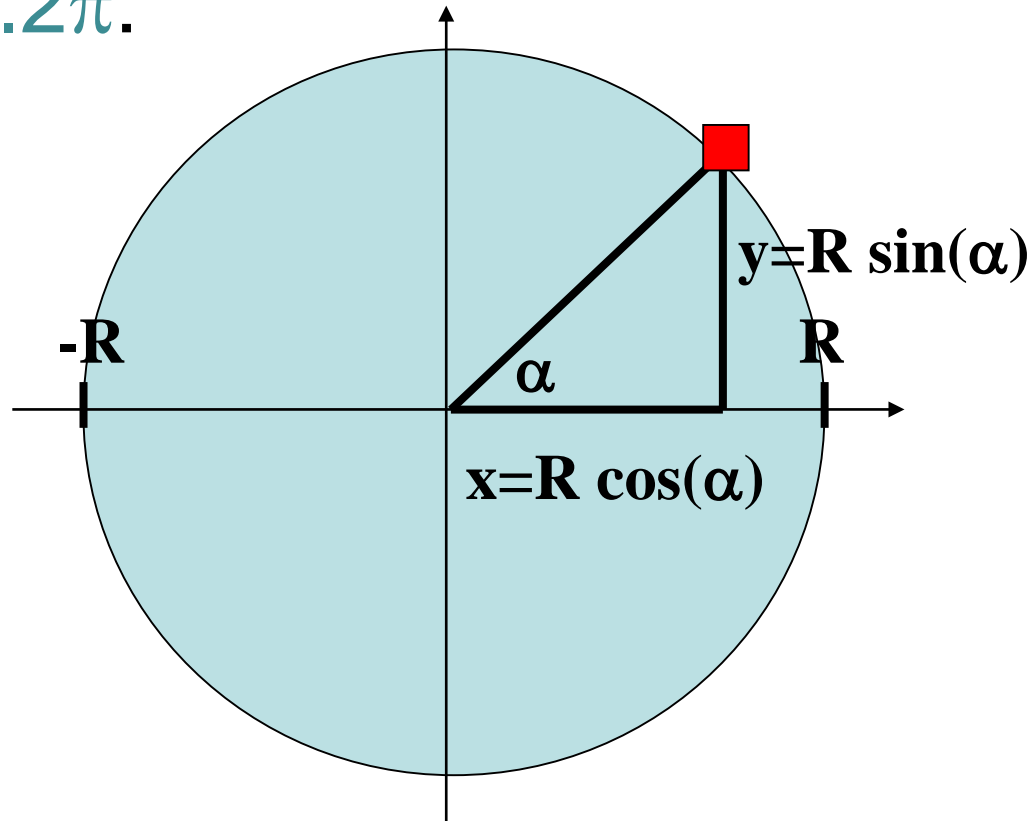
Wie zeichne ich schnell Kreise im Computer?



Kreiszeichnen

Naiver Ansatz: verwende **sin** und **cos**

Für $\alpha=0\dots 2\pi$:



Kreiszeichnen

plot(x,y): setze Punkt bei Position (x,y)

Algorithmus Kreis1:

```
for  $i \leftarrow 0$  to  $n-1$  do  
    plot( $R \cdot \cos(2 \cdot \pi \cdot i/n)$ ,  $R \cdot \sin(2 \cdot \pi \cdot i/n)$ )
```

Kreisumfang: $U=2\pi R$. Bei Pixelbreite von 1
Längeneinheit reicht dann $n=7R$.

Problem: sin, cos, ·, / teuer!

Kreiszeichnen

Besserer Ansatz:

$$x^2 + y^2 = R^2, \text{ also } y = \pm \sqrt{R^2 - x^2}$$

Algorithmus Kreis2:

$y_0 \leftarrow 0$

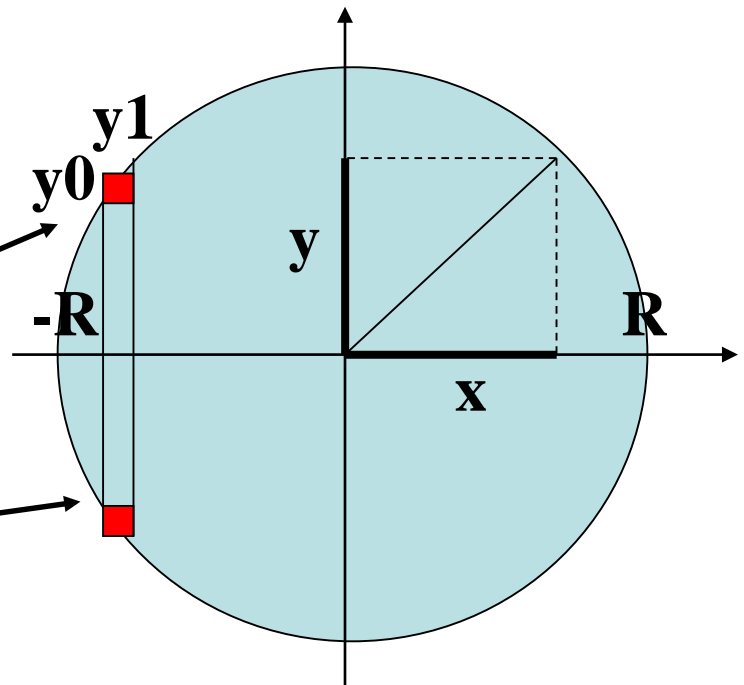
for $x \leftarrow -R$ to R do

$y_1 \leftarrow \text{sqrt}(R \cdot R - x \cdot x)$

for $y \leftarrow y_0$ to y_1 do

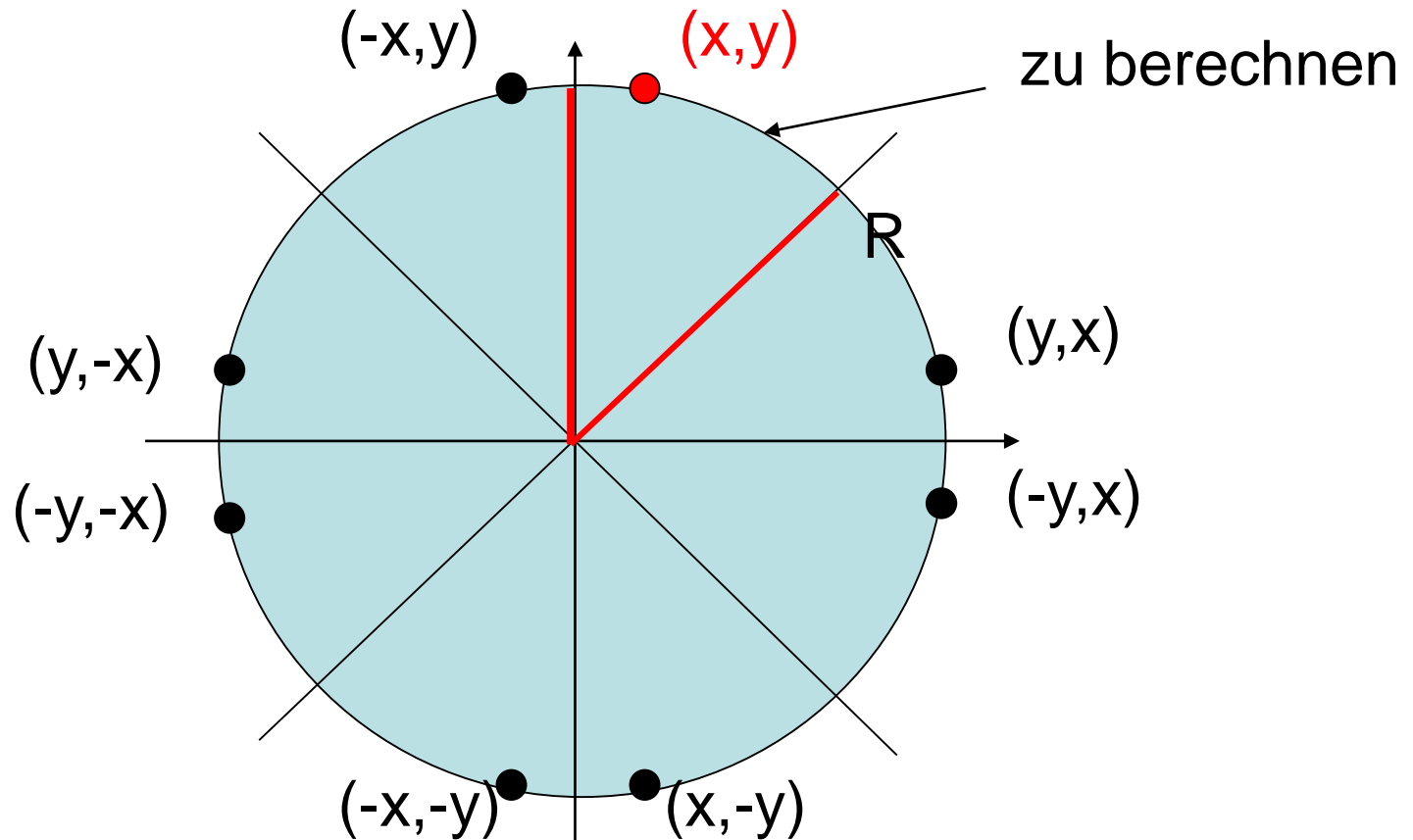
plot(x, y); plot($x, -y$)

$y_0 \leftarrow y_1$



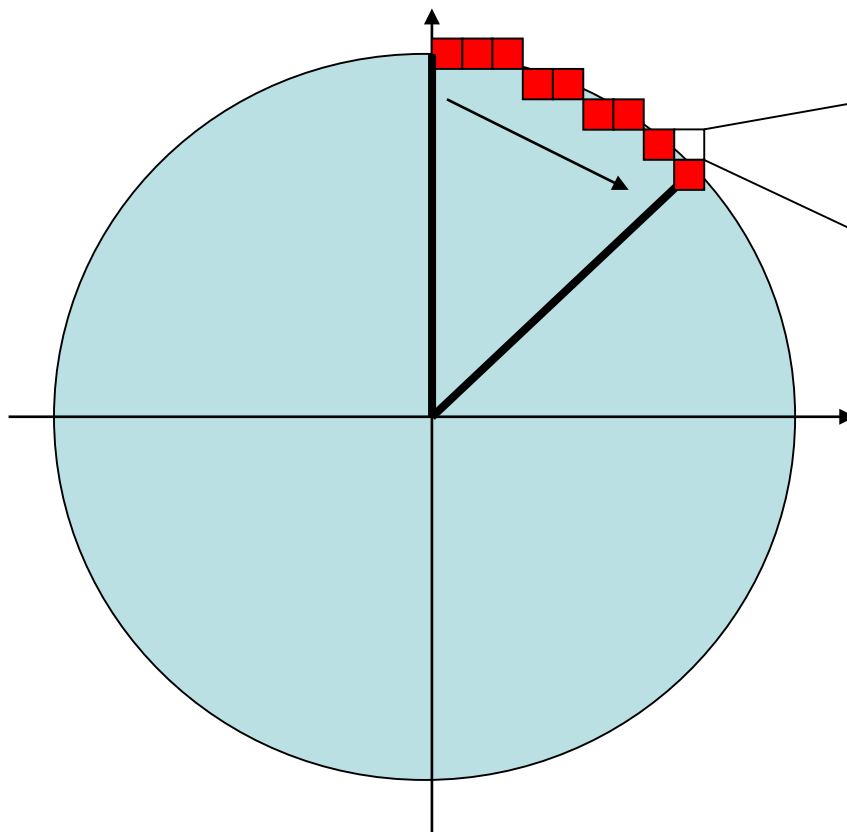
Kreiszeichnen

Noch besser: Ausnutzung von Spiegelungen



Kreiszeichnen



Aber: Algo verwendet immer noch Wurzel
Kommen wir mit $\cdot, +, -$ aus?



- innerhalb von Kreis?
ja: $x \leftarrow x+1$
nein: $x \leftarrow x+1; y \leftarrow y-1$

zeichne nur \blacksquare mit \bullet innerhalb

Kreiszeichnen

- Mittelpunkt des ersten  sei $(x,y)=(0,R)$
- Position seines \bullet : $(0,R-1/2)$ (1×1 -Quadrat)
- Test, ob (x,y) innerhalb Kreis:
teste, ob $F(x,y)=x^2+y^2-R^2 < 0$ ist
- Also Test, ob  mit Mittelpunkt $(1,R)$ noch innerhalb von Kreis ist: $F(1,R-1/2) < 0$?
- Es gilt:
$$F(x+1,y) = F(x,y)+2x+1$$
$$F(x+1,y-1) = F(x,y)+2x-2y+2$$

Bresenham Algorithmus

$(x,y) \leftarrow (0,R)$

$F \leftarrow 1-R$

{ eigentlich $F(1,R-1/2)=5/4-R$, aber $1-R$ gleiche Ausgabe }

$\text{plot}(0,R); \text{plot}(R,0); \text{plot}(0,-R); \text{plot}(-R,0)$

while $(x < y)$ do

 if $(F < 0)$ then

$x \leftarrow x+1$

$F \leftarrow F+2 \cdot x+1$

 { \square bei $(x+1,y)$ hat \bullet innerhalb Kreis? }

 { ja: zu Punkt $(x+1,y)$ von F }

 { von $F(x+1,y)$ zu $F(x+2,y)$ }

 else

$x \leftarrow x+1$

$F \leftarrow F+2 \cdot x-2 \cdot y+2$

$y \leftarrow y-1$

 { nein: erst zu Punkt $(x+1,y)$ von F }

 { von $F(x+1,y)$ zu $F(x+2,y-1)$ }

 { ...dann zu Punkt $(x+1,y-1)$ }

$\text{plot}(x,y); \text{plot}(y,x); \text{plot}(-x,y); \text{plot}(y,-x)$

$\text{plot}(x,-y); \text{plot}(-y,x); \text{plot}(-x,-y); \text{plot}(-y,-x)$

Bresenham Algorithmus

$(x,y) \leftarrow (0,R)$

$F \leftarrow 1-R$

{eigentlich $F(1, R-1/2) = 5/4 - R$. aber $1-R$ gleiche Ausgabe}

plot(0,R); plot(R,0) while ... do ... - Schleife

while ($x < y$) do

if ($F < 0$) then

$x \leftarrow x+1$

$F \leftarrow F+2 \cdot x+1$

else

$x \leftarrow x+1$

$F \leftarrow F+2 \cdot x-2 \cdot (y-1)$

$y \leftarrow y-1$

plot(x,y); plot(y,x); plot(-x,y); plot(y,-x)

plot(x,-y); plot(-y,x); plot(-x,-y); plot(-y,-x)

{ bei $(x+1,y)$ hat innerhalb Kreis? }

Besserer Bresenham Algorithmus

$(x,y) \leftarrow (0,R)$

$F \leftarrow 1-R$

plot(0,R); plot(R,0); plot(0,-R); plot(-R,0)

while ($x < y$) do

$x \leftarrow x+1$

 if ($F < 0$) then { \square bei $(x+1,y)$ hat \bullet innerhalb Kreis? }

$F \leftarrow F+2 \cdot x+1$

 else

$y \leftarrow y-1$

$F \leftarrow F+2 \cdot (x-y)$

 plot(x,y); plot(y,x); plot(-x,y); plot(y,-x)

 plot(x,-y); plot(-y,x); plot(-x,-y); plot(-y,-x)

2. : in Maschinencode sehr schnell (Linksshift der Bits)

Zusammenfassung

Beispiele:

- Arithmetik auf großen Zahlen
Beispiel für **asymptotische Verbesserung**
(klassische Algorithmik)
- Kreisberechnung
Beispiel für **Verbesserung der Konstanten**
(Algorithm Engineering)



DuA

Ideal: Anwendung beider Strategien