

## Algorithmen- und Programmier- Grundbegriffe

Autor: E. Engelhardt  
Stand: 09. Dezember 2007

### Inhalt



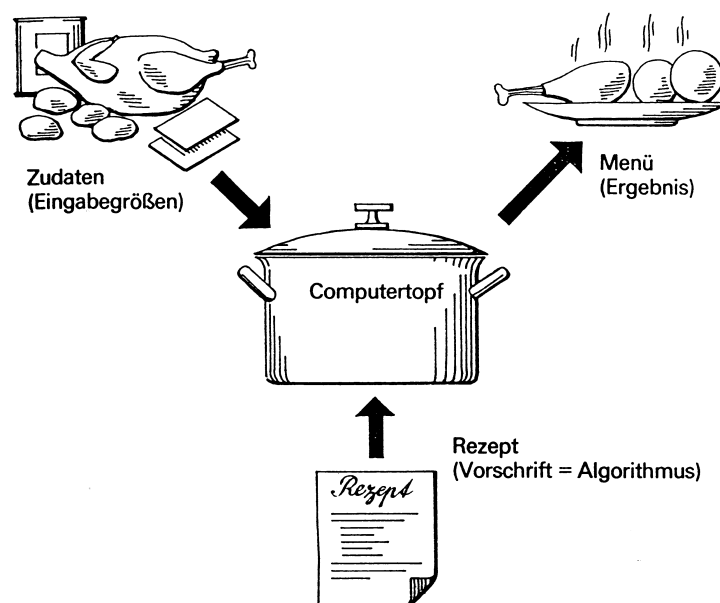
- ◆ Einleitung
- ◆ Algorithmus
- ◆ Klassische Algorithmen
- ◆ Darstellungsformen von Algorithmen
- ◆ Programmiersprachen
- ◆ Komplexität von Algorithmen
- ◆ Geschichte der Programmierung
- ◆ Problemlösungsstrategien

- ◆ Diese Präsentation erhebt nicht den Anspruch, eine theoretisch fundierte Einführung in die Informationstechnik zu geben.
- ◆ Es werden wichtige Begriffe und wesentliche Ergebnisse der theoretischen Informatik als Einführung für die Programmierung vorgestellt.
- ◆ Dabei wird auf ausführliche und mathematisch exakte Herleitungen verzichtet. Es soll unabhängig von anderen Lehrveranstaltungen Verständnis von für die Programmierung wichtigen Begriffen vermitteln.

- ◆ Einfache Definition:
  - Informatik ist die Wissenschaft von der systematischen Verarbeitung von Daten, insbesondere der automatischen Verarbeitung mit Hilfe von Digitalrechnern.
- ◆ Informatik ist weder Natur- noch Technikwissenschaft, sondern Strukturwissenschaft (wie Mathematik), d.h.:
  - Informatik behandelt und untersucht vom Menschen geschaffene formale Strukturen (Datenstrukturen, Sprachstrukturen, Systemstrukturen)
  - Informatik ist eine Disziplin der Informationswissenschaften
- ◆ Zentrales Thema der Informatik ist die Formulierung und Realisierung von Algorithmen

- ◆ Muhammed Ibn Musa al-Kwarizmi (ca. 813 - 846)
  - „*Al-kitab al-muhtasar fi hisab **al jabr** wa'l-muqabala*“ Rechenbuch in arabischer Sprache, das das indische Zahlensystem benutzt
  - Das Buch vom Rechnen durch Ergänzung und Gegenüberstellung
  - stammte aus dem Ort mit heutigem Namen Khiva (Usbekistan)
  - Arithmetik, Trigonometrie, Astronomie und Geographie
  - Übersetzung des Rechenbuches (Regeln der Wiedereinsetzung und Reduktion) zu Beginn des 12. Jahrhunderts in Latein: „Dixit **Algoritmi**: Laudes deo rectori nostro ...“
- ◆ Der Name Algorithmus wurde für jede Form von Rechen-vorschrift beibehalten
  - der Name „Algebra“ ist von „al jabr“ abgeleitet (siehe oben)

- ◆ Anleitung zur Lösung eines Problems wie Kochrezept, Montageanleitung oder Strickmuster



- ◆ Algorithmus muss von einer Maschine ausführbar sein, die Inhalt nicht versteht
  - Notation in einer Form mit Syntax (Einhaltung formaler Regeln) und Semantik (Bedeutung der Konstrukte)
  - man verwendet sogenannte Programmiersprachen
  
- ◆ Programm: Ein in einer Programmiersprache formulierter Algorithmus
  
- ◆ Programmierung: Alle Aktivitäten von der Problemstellung bis zum fertigen Programm

- ◆ Finden eines Lösungsweges
  - genaue Analyse des Problems
  - zur Verfügung stehende Daten
  - erwartete Ergebnisse
  - Anforderungen an Ressourcen
  - erfordert Kreativität
  - Entwurf von Algorithmen kann als „Kunst“ betrachtet werden
  - allgemeingültige Prinzipien und Techniken erleichtern das Finden von Algorithmen
  - Programmieren ist „nur“ die Umsetzung des Algorithmus in eine maschinenverständliche Sprache
  
- ◆ Es gibt keinen Algorithmus zum Entwerfen von Algorithmen!

- ◆ *Allgemeinheit / Universalität:*
  - Algorithmus löst im Allgemeinen Klasse von Problemen
  - Auswahl z.B. über Parameter
  - muss über die Formulierung geregelt werden
- ◆ *Endlichkeit (in der Beschreibung)*
  - Formulierung des Algorithmus erfolgt durch einen endlichen Text
- ◆ *Eindeutigkeit*
  - die einzelnen Schritte und ihre Aufeinanderfolge sind unmissverständlich beschrieben
- ◆ *Vollständigkeit*
  - es darf kein notwendiger Schritt ausgelassen werden

- ◆ *Determiniertheit:*
  - in der Regel determiniert, d.h. bei gleichen Eingabewerten und Startbedingungen folgt dasselbe Ergebnis (Ausgabewerte)
- ◆ *Determinismus:*
  - ein Algorithmus heißt deterministisch, wenn zu jedem Zeitpunkt seiner Bearbeitung höchstens eine Möglichkeit der Fortsetzung besteht
- ◆ *Korrektheit*
  - bei jeder gültigen Eingabe wird das richtige Ergebnis geliefert
- ◆ *Terminierung:*
  - Algorithmus terminiert nach endlich vielen Schritten (Ausnahme: z.B.: Betriebssystem-Funktionen)
- ◆ *Effizienz / Komplexität*
  - Maß für den Verbrauch von Ressourcen

- ◆ *Theorem von Böhm und Jacobini (sinngemäß):*
  - Gibt es zur Lösung eines Problems einen Algorithmus, so lässt sich dieser unter alleiniger Verwendung der drei Grundstrukturen Sequenz (Folge), Selektion (Verzweigung) und Zyklus (Iteration, Schleife) darstellen.
- ◆ Sequenz (Folge)
  - Folge von Aktionen, die genau einmal abgearbeitet werden
  - Algorithmen, die nur aus einer Sequenz bestehen heißen lineare Algorithmen (z. B.: Vertauschen der Inhalte zweier Variablen)
- ◆ Selektion (Auswahl, Verzweigung)
  - in Abhängigkeit einer Bedingung werden Aktionen ausgeführt oder nicht ausgeführt
- ◆ Zyklus / Iteration (Repetition, Wiederholung, Schleife)
  - Grundstrukturen werden wiederholt (mehrfach) abgearbeitet

- ◆ Hilbert:
  - Suche nach einem universellen Algorithmus, der für eine gegebene Aussage in einem formalen System entscheiden kann, ob die Aussage richtig oder falsch ist (Entscheidungsproblem 1928)
- ◆ Turing-Maschine (A.M. Turing 1936)
  - „On Computable Numbers with an Application to the Entscheidungsproblem“ (1936)
  - Algorithmus muss von einer Maschine ausführbar sein, die Inhalt nicht versteht
- ◆ Lambda-Kalkül (A. Church 1936)
- ◆ Algorithmenbegriff für Zeichenreihen; Markowsche Ketten (A.A. Markow 1951)

- ◆ Zu jedem, in irgendeiner Form definierten, Algorithmus kann eine Turing-Maschine gebaut werden

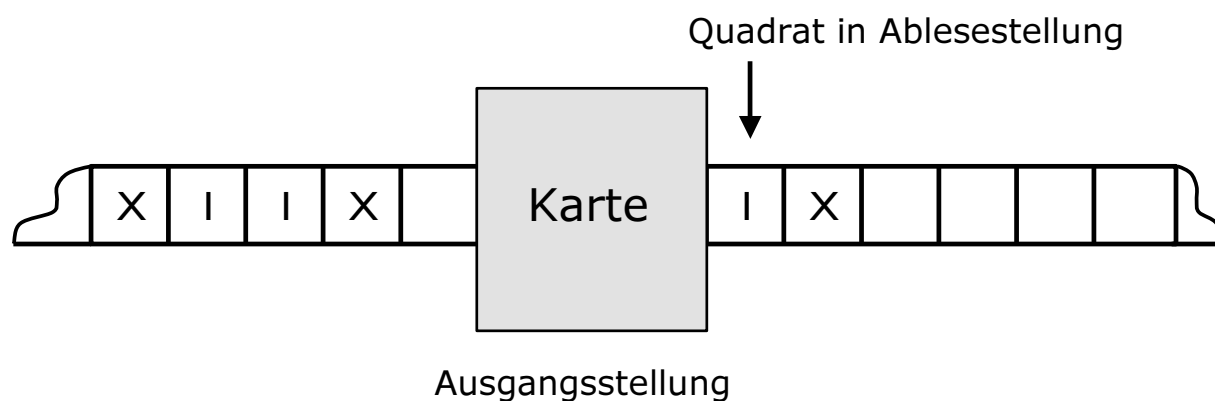
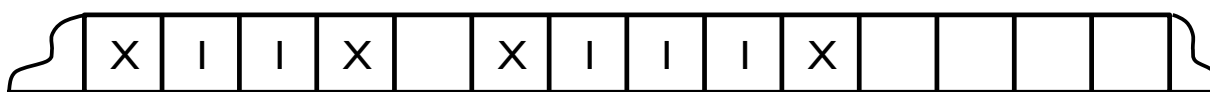


„Jede im intuitiven Sinne berechenbare Funktion ist auch Turing-berechenbar.“



- ◆ Die verschiedenen Algorithmenbegriffe sind äquivalent.
- ◆ These ist nicht bewiesen, aber allgemein anerkannt.

Eintragungen auf dem Band für die Rechenoperationen  $2+2$ ,  $2 \times 3$  und  $3^2$

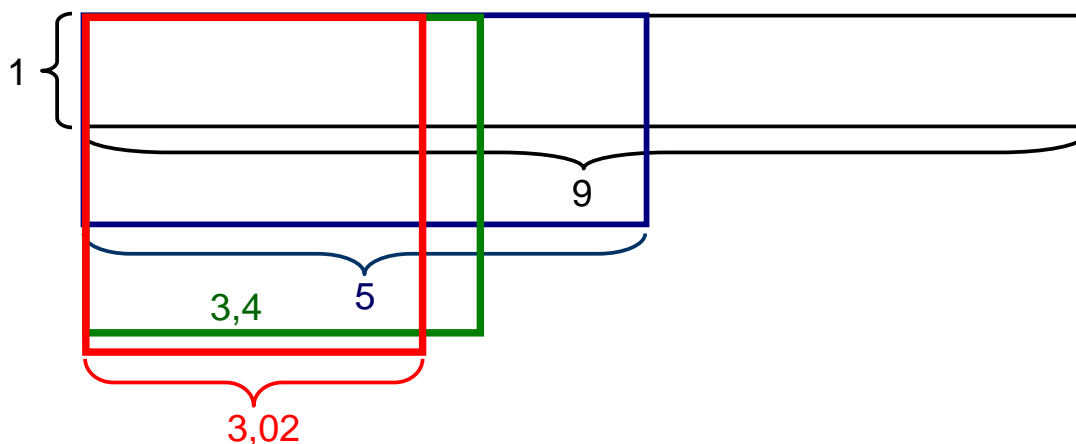


- ◆ Entstehung aus Problemen der Flächenmessung durch die Babylonier (etwa 1700 vor Christi)
- ◆ Aufgegriffen durch Heron von Alexandria (100 nach Christi)
  - Die Quadratwurzel soll berechnet werden
  - Dazu werden schrittweise immer quadratähnlichere Rechtecke konstruiert, ohne den Flächeninhalt zu verändern
  - Eine Seite wird als arithmetisches Mittel der beiden Rechteckseiten gewählt und die andere Seite wird berechnet, so dass der Flächeninhalt sich nicht verändert

$$x_1 = (x_0 + y_0) / 2 \quad \text{und} \quad y_1 = a/x_1$$

$$x_{n+1} = (x_n + y_n) / 2 \quad \text{und} \quad y_{n+1} = a/x_{n+1}$$

$$x_{n+1} = (x_n + a/x_n) / 2 \quad (\text{Pseudocode auf einer folgenden Folie})$$



- |   |           |                             |
|---|-----------|-----------------------------|
| 1. Schritt: $a = 9$                             | $x_0 = 9$ | $y_0 = 1$                   |
| 2. Schritt: $x_1 = (9 + 1) / 2 = 5$             |           | $y_1 = 9/5$                 |
| 3. Schritt: $x_2 = (5 + 9/5) / 2 = 34/10 = 3,4$ |           | $y_2 = 9 / (34/10) = 2,647$ |
| 4. Schritt: $x_3 = (3,4 + 2,647) / 2 = 3,02$    |           | $y_3 = 9 / (3,02) = 2,98$   |

- ◆ Problem des größten gemeinsamen Teilers (ggT)
  - wichtige Rolle in der Mathematik z. Bsp. Bruchrechnung
  
- ◆ Euklid von Alexandria (ca. 365 bis 300 v. Chr.)
 

„Nimmt man abwechselnd immer das Kleinere vom Größeren weg, dann muss der Rest schließlich die vorhergehende Größe messen ...“ („Die Elemente, Zehntes Buch §3“)

  - $a = q * b + r$       und       $0 \leq r < b$
  
  - Bsp.:       $18 \text{ und } 4 \mid 18 - 4 = 14 \rightarrow$   
                    $14 \text{ und } 4 \mid 14 - 4 = 10 \rightarrow$   
                            $10 \text{ und } 4 \mid 10 - 4 = 6 \rightarrow$   
                                    $6 \text{ und } 4 \mid 6 - 4 = 2 \rightarrow$   
    $4 \text{ und } 2 \mid 4 - 2 = 2 \quad -$

**> 2 ist ggT**

- ◆ Siebverfahren zur Ermittlung von Primzahlen
  - wichtige Rolle in der Zahlentheorie
  
- ◆ Eratosthenes von Kyrene (ca. 276 - 194 v. Chr.)
  1. Es werden alle Zahlen des zu untersuchenden Bereichs aufgeschrieben (nur Bitfelder notwendig)
  2. Es wird zur nächsten noch nicht gestrichenen Zahl vorgerückt.
  3. Von dieser Zahl ausgehend werden wieder alle Vielfachen gestrichen
  4. Gehe nach Schritt 2., wenn Zahl kleiner als Wurzel ... siehe Bem.

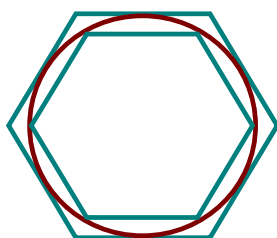
Alle nicht gestrichenen Zahlen des Feldes sind die Primzahlen

Bemerkung: Es müssen nur Vielfache einer Zahl gestrichen werden, höchstens bis zur Wurzel der größten zu ermittelnden Primzahl

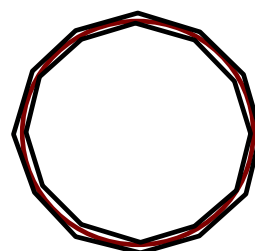
<del>1</del>	2	3	<del>4</del>	5	<del>6</del>	7	<del>8</del>	<del>9</del>	<del>10</del>
11	<del>12</del>	13	<del>14</del>	<del>15</del>	<del>16</del>	17	<del>18</del>	19	<del>20</del>
<del>21</del>	<del>22</del>	23	<del>24</del>	<del>25</del>	<del>26</del>	<del>27</del>	<del>28</del>	29	<del>30</del>
31	<del>32</del>	<del>33</del>	<del>34</del>	<del>35</del>	<del>36</del>	37	<del>38</del>	<del>39</del>	<del>40</del>
41	<del>42</del>	43	<del>44</del>	<del>45</del>	<del>46</del>	47	<del>48</del>	<del>49</del>	<del>50</del>
<del>51</del>	<del>52</del>	53	<del>54</del>	<del>55</del>	<del>56</del>	<del>57</del>	<del>58</del>	59	<del>60</del>
61	<del>62</del>	<del>63</del>	<del>64</del>	<del>65</del>	<del>66</del>	67	<del>68</del>	<del>69</del>	<del>70</del>
71	<del>72</del>	73	<del>74</del>	<del>75</del>	<del>76</del>	<del>77</del>	<del>78</del>	79	<del>80</del>
<del>81</del>	<del>82</del>	83	<del>84</del>	<del>85</del>	<del>86</del>	<del>87</del>	<del>88</del>	89	<del>90</del>
<del>91</del>	<del>92</del>	<del>93</del>	<del>94</del>	<del>95</del>	<del>96</del>	97	<del>98</del>	<del>99</del>	<del>100</del>

Klassische Algorithmen - Kreiszahl  $\pi$  (6/6)

- ◆ Approximation von  $\pi$ 
  - wichtige Rolle bei Flächen und Körperberechnungen
- ◆ Archimedes von Syrakus (ca. 287 - 212 v.Chr.)
  1. Die Annäherung des Kreises erfolgt durch einbeschriebene und umbeschriebene regelmäßige Vielecke.
  2. Die Umfänge der Vielecke werden berechnet unter Verwendung des Satzes von Pythagoras.
  3. Die Umfänge der einbeschriebenen und umbeschriebenen Vielecke nähern sich sehr stark an, so dass das 96-Eck schon einen auf drei Dezimalstellen genauen Wert für  $\pi$  liefert.



Sechseck

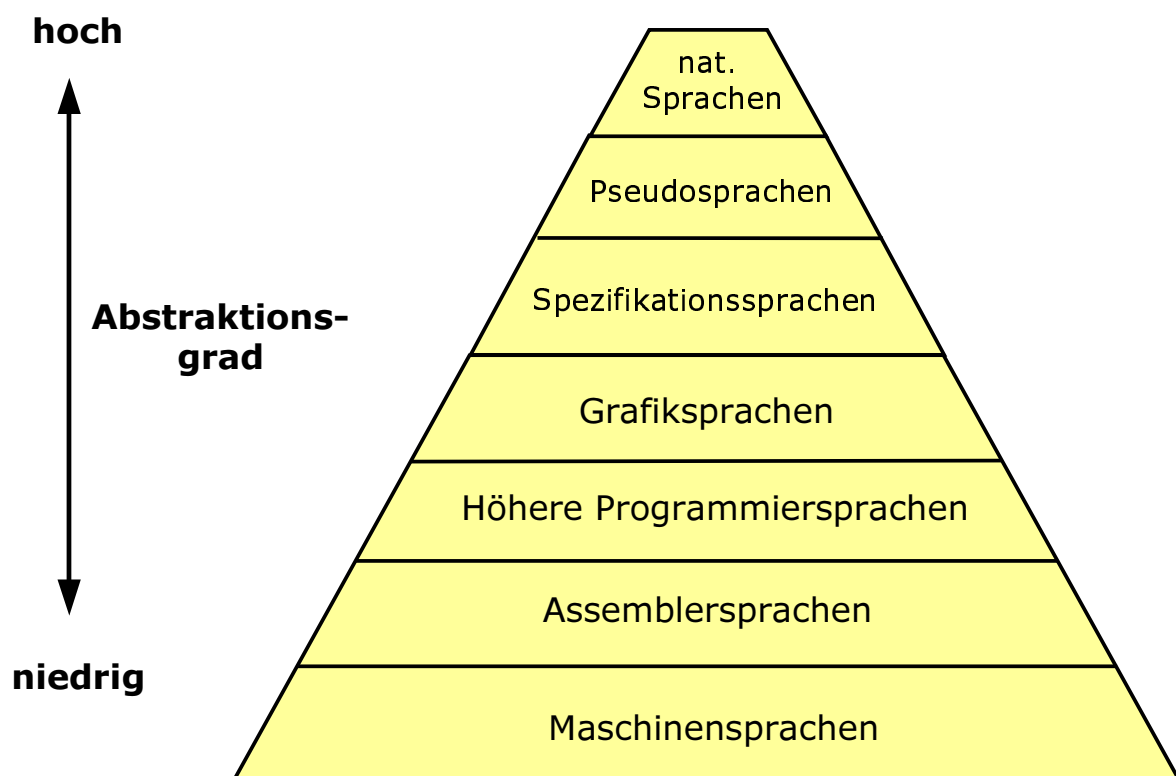


Zwölfeck

- ◆ Erste Generation: Maschinensprachen
  - Befehle und Adressen werden in Binärform eingegeben
  - Nur auf dem jeweiligen Prozessortyp lauffar
  - In der Anfangszeit der Computer-Programmierung war dies der einzige Weg der Programmierung
  
- ◆ Zweite Generation: Assemblersprachen
  - Für die Maschinen-Befehle werden „Mnemoniks“ (Namen, Bezeichnungen) eingeführt und für die Adressen werden symbolische Namen angegeben
  - Nur auf dem jeweiligen Prozessortyp lauffar
  - Für einen Prozessortyp kann es mehrere Assembler-Sprachen geben!
  - Die Übersetzungsprogramme heißen „Assembler“

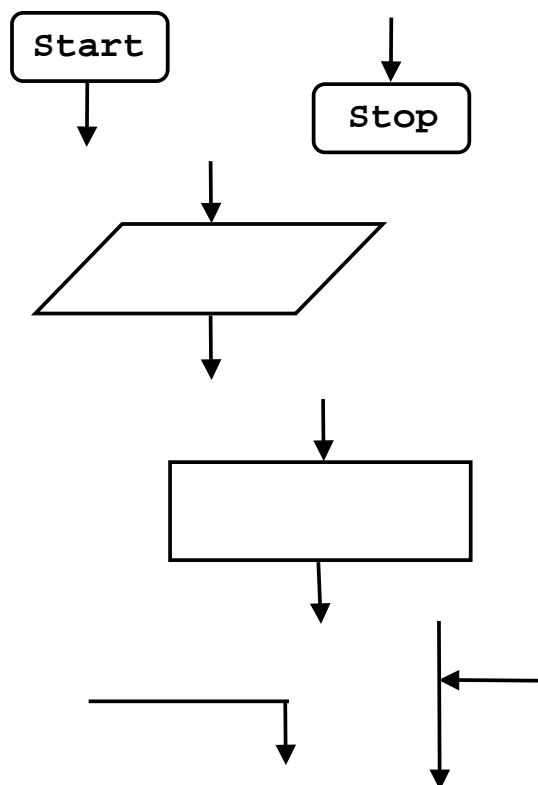
- ◆ Dritte Generation: Problemorientierte Sprachen
  - Ab dieser Generation beginnen die „höheren“ Programmiersprachen
  - werden auch als „prozedurale“ Sprachen bezeichnet (Beschreibung der Algorithmen und Abläufe)
  - Folge (Sequenz), Verzweigung (Selektion) und Schleife (Iteration) als Sprachelemente
  - Möglichkeiten für strukturierte Programmierung
  - Sehr viele Sprachen die oft auf bestimmte Aufgabengebiete zugeschnitten sind (FORTRAN, COBOL, ...)
  - Die Quellprogramm-Dateien werden durch Compiler in Zwischen-code-Dateien (Objekt-Dateien) oder direkt in Maschinencode übersetzt
  - Objekt-Dateien werden durch „Linker“ (Binder) in Maschinencode übertragen

- ◆ Vierte Generation: Deklarative Sprachen
  - Beschreiben, was das Programm leisten soll (nichtprozedurale Programmiersprachen)
  - Kein Einfluss auf interne Abläufe bei der Umsetzung
  - Meist auf ganz bestimmte Aufgabengebiete zugeschnitten (Datenbanken, Tabellenkalkulation, ...)
  - Beispiele: SQL, Open Access
- ◆ Fünfte Generation: Künstliche Intelligenz
  - Versuchen menschliche Intelligenz oder allgemein Intelligenz nachzuvollziehen
  - Spezielle Aufgabengebiete wie Spracherkennung, Wissensverarbeitung und Robotik
  - Beispiele: Lisp, Prolog, Smalltalk

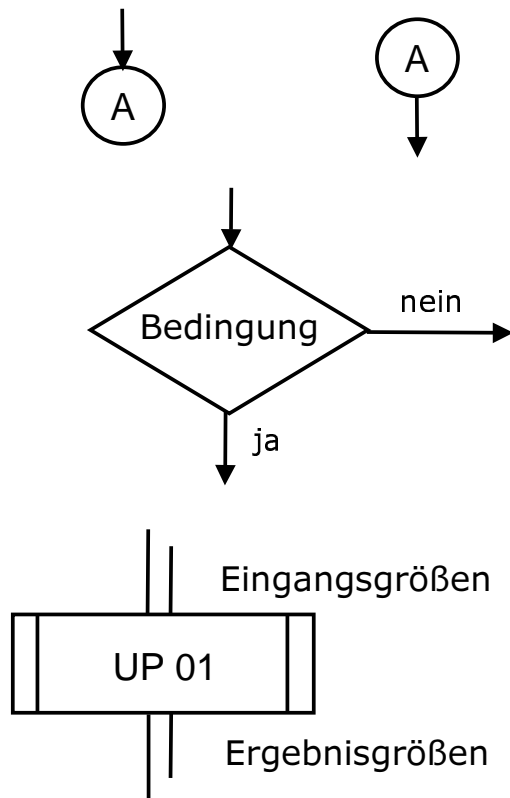


- ◆ Verbale Beschreibung der einzelnen Schritte
  - Oft schwer verständlich (Übersetzungsprobleme, Bedeutung von Wörtern, Zweideutigkeiten, ...)
  - Beschreibung muss sehr genau sein und kann deshalb lang und unübersichtlich werden
  - **Beispiel:**  
Größter gemeinsamer Teiler zweier natürlicher Zahlen; Euklid von Alexandria ca. 365 - 300 v. Chr.

„Nimmt man abwechselnd immer das Kleinere vom Größeren weg, dann muss der Rest schließlich die vorhergehende Größe messen ...“ („Die Elemente, Zehntes Buch §3“)

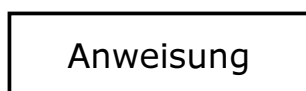


- ◆ Anfang / Ende
- ◆ Ein- / Ausgabe
- ◆ Anweisung (Aktion)
- ◆ Ablauf / Zusammenführung

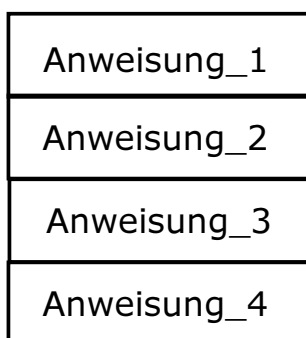


- ◆ Anschluss-Stellen
- ◆ Verzweigung
- ◆ Aufruf von Teilalgorithmen

- ◆ I. Nassi und B. Shneidermann schlugen 1973 diese Darstellungsart vor, deshalb auch Nassi-Shneidermann-Diagramm (DIN 66 261)
  - Grundform ist ein Strukturblock (Rechteck)
  - Abarbeitungsreihenfolge erfolgt von oben nach unten

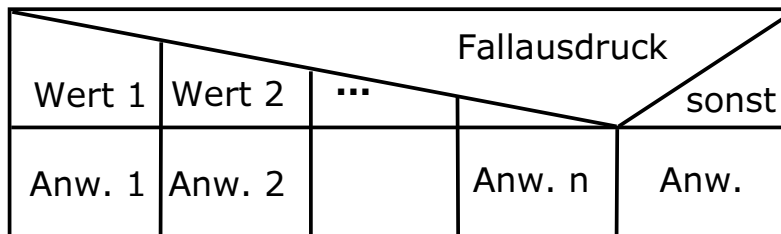
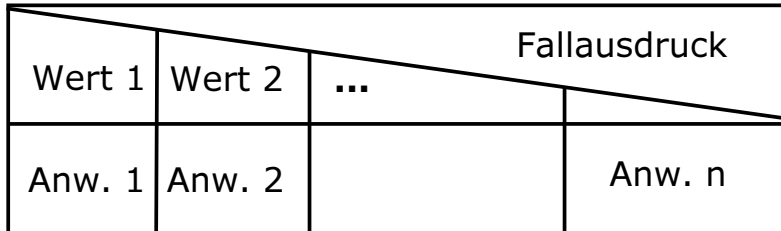
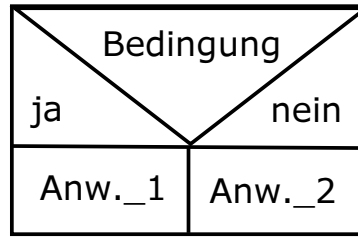
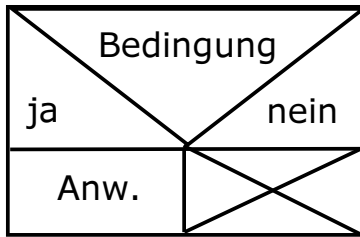


- ◆ Anweisung



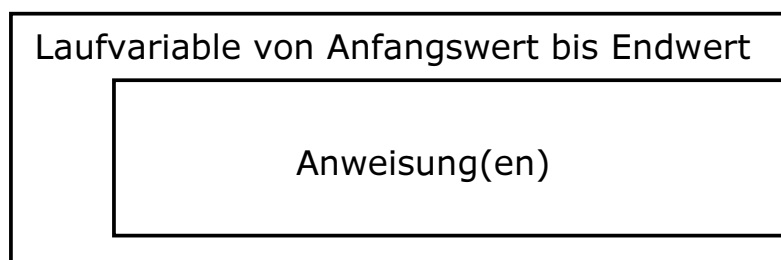
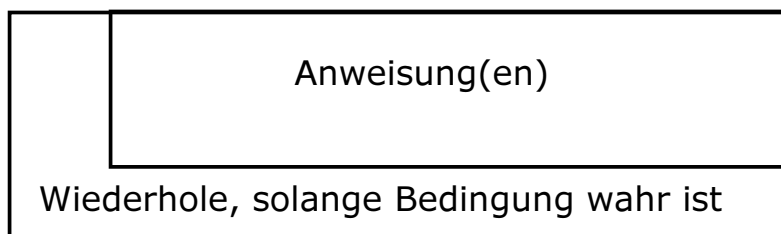
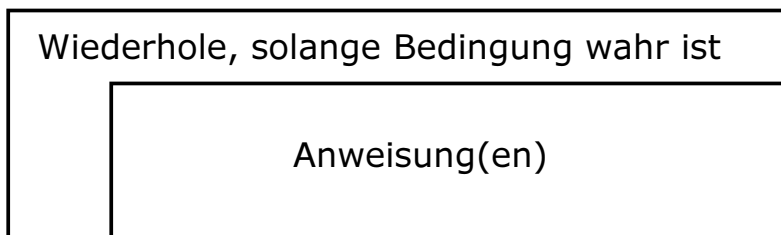
- ◆ Sequenz / Folge

## Darstellungsformen - Struktogramm (6/10)



- ◆ Selektion / Verzweigung mit und ohne Alternative
- ◆ Mehrfachverzweigung ohne „default“-Zweig
- ◆ Mehrfachverzweigung mit „default“-Zweig

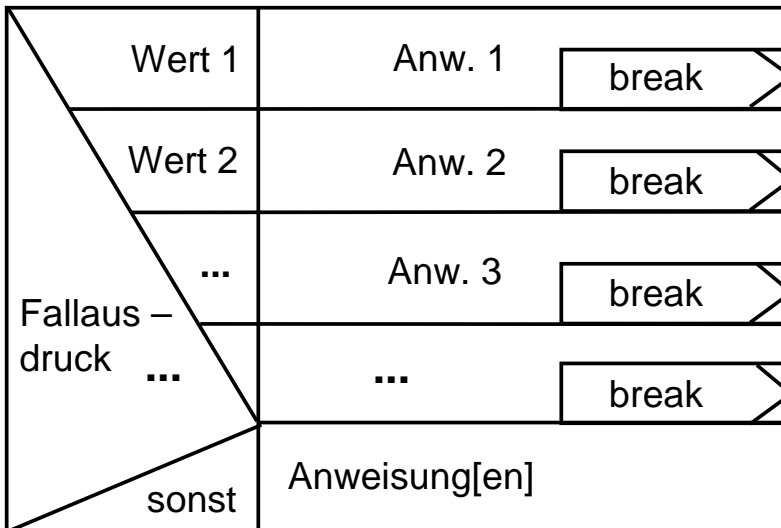
## Darstellungsformen - Struktogramm (7/10)



- ◆ Kopfgesteuerte Schleife (abweisender Zyklus)
- ◆ Fußgesteuerte Schleife (mindestens ein Durchlauf)
- ◆ Schleife von Anfangswert bis Endwert (Zählschleife)

## Darstellungsformen - Struktogramm (8/10)

- ◆ Vorschlag (E. Engelhardt Dez. 2002)
  - Bereitstellung von grafischen Elementen für Programmier-elemente aus C/C++, C#, Java, J++, JavaScript, ...
  - in oben genannten Sprachen funktioniert die Mehrfachver-zweigung anders als z.B. in ALGOL, Pascal, ...



- ◆ Mehrfachver-zweigung mit „default“-Zweig
- ◆ Anweisungen können „break“ enthalten oder auch nicht
- ◆ (ohne „default“-Zweig analog)

## Darstellungsformen - Pseudo-Sprachen (9/10)

- ◆ Werden oft zur Beschreibung von Algorithmen in Lehrbüchern und Veröffentlichungen verwendet
- ◆ Die Schlüsselwörter sind sehr stark an existierende Program-miersprachen angelehnt
- ◆ Beispiel: Heron-Verfahren

**Heron(a)**

```

Hilfsvariable: x, y, xneu, yneu;
(* lokale Hilfsvariablen *)
x:=a; y:=1;
Solange ((x**2 - a) > 0.000001) tue
(
  xneu:= (x+y)/2;
  yneu:= a/xneu;
  x:= xneu;
  y:= yneu;
)

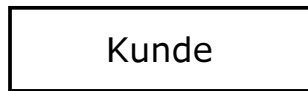
```

Rueckgabe x

**Ende**

## Darstellungsformen - ER-Modell (10/10)

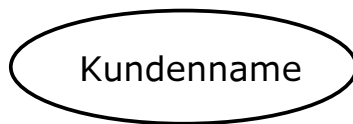
- ◆ Orientierung an Daten und deren Beziehungen
- ◆ Das Entity-Relationship-Modell ist ein Datenmodell
- ◆ Das ERM ist nicht zur Beschreibung von Algorithmen



- ◆ Entity (Datenobjekt)



- ◆ Relationship (Beziehung)



- ◆ Attribut (Eigenschaft eines Objekts)

## Programmiersprachen - Vorbetrachtungen (1/10)

- ◆ Theoretisch könnten alle Programme mit einer einzigen Programmiersprache programmiert werden
- ◆ Es sind nur die drei Grundelemente Sequenz, Selektion und Zyklus notwendig
- ◆ Für verschiedene Aufgabenstellungen (Wirtschaft, mathematische Verfahren, künstliche Intelligenz, usw.) wurden verschiedene an die Aufgabenstellung angepasste Programmiersprachen entwickelt
- ◆ Durch immer komplexer werdende Programme und Programmsysteme wurden immer wieder neue Betrachtungsweisen und entsprechende Programmiersprachen dafür entwickelt (Versuch, die immer komplexer werdenden Programme beherrschbar zu machen)

- ◆ Klassifizierung erfolgt nach Sprachtyp
- ◆ *Imperative Programmiersprachen*
  - anweisungsorientierte Programmiersprachen
  - abstrakte Datentypen und Modularisierung
  - Erstellen von Programmen für von-Neumann-Rechner
- ◆ *Prozedurale Programmiersprachen*
  - gehören zu den imperativen Programmiersprachen
  - Sprachen der ersten bis dritten Generation
  - Zergliederung der Aufgaben in Unterprogramme
  - Gebrauch von Variablen, Befehlen und Prozeduren (Funktionen)
  - Modul: abgeschlossene Programmeinheit, die für sich programmiert und getestet wird (Aufbau von Bibliotheken)
  - Prozedur: geben keinen Wert zurück
  - Funktion: können Wert zurück geben
  - C-Beispiel: `int sum (int a, int b) { return (a + b); }`

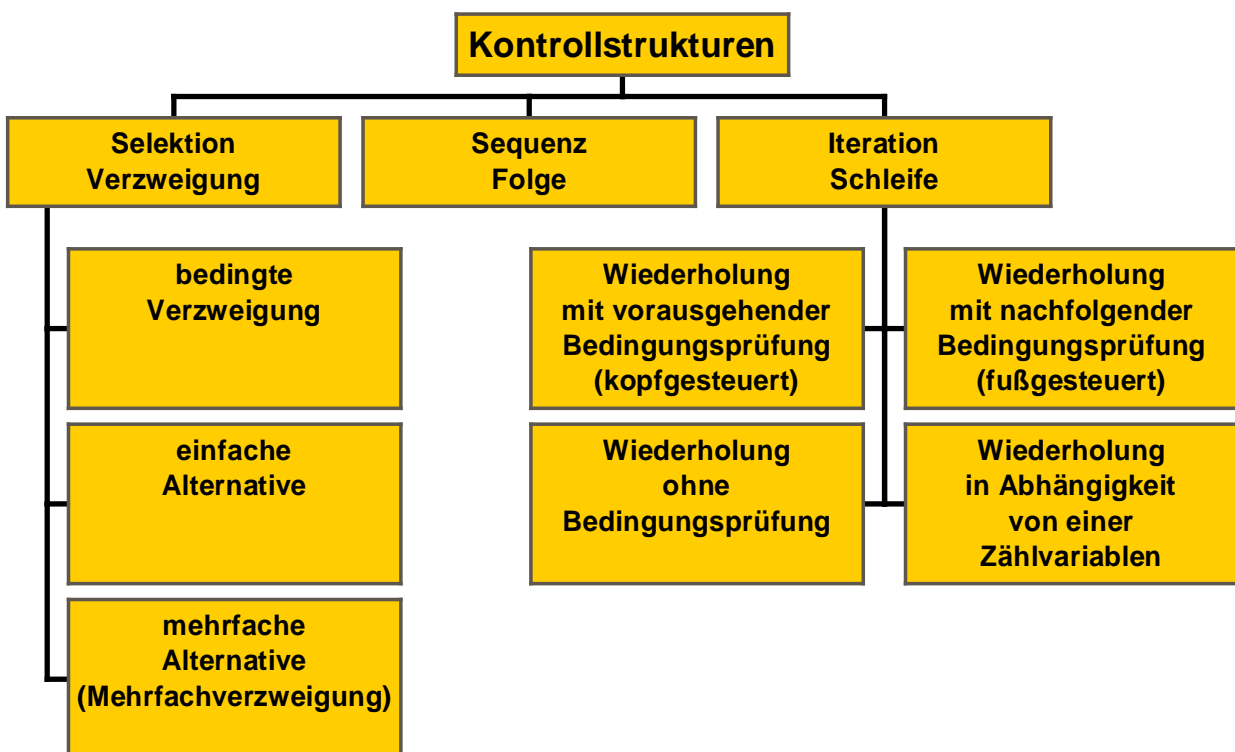
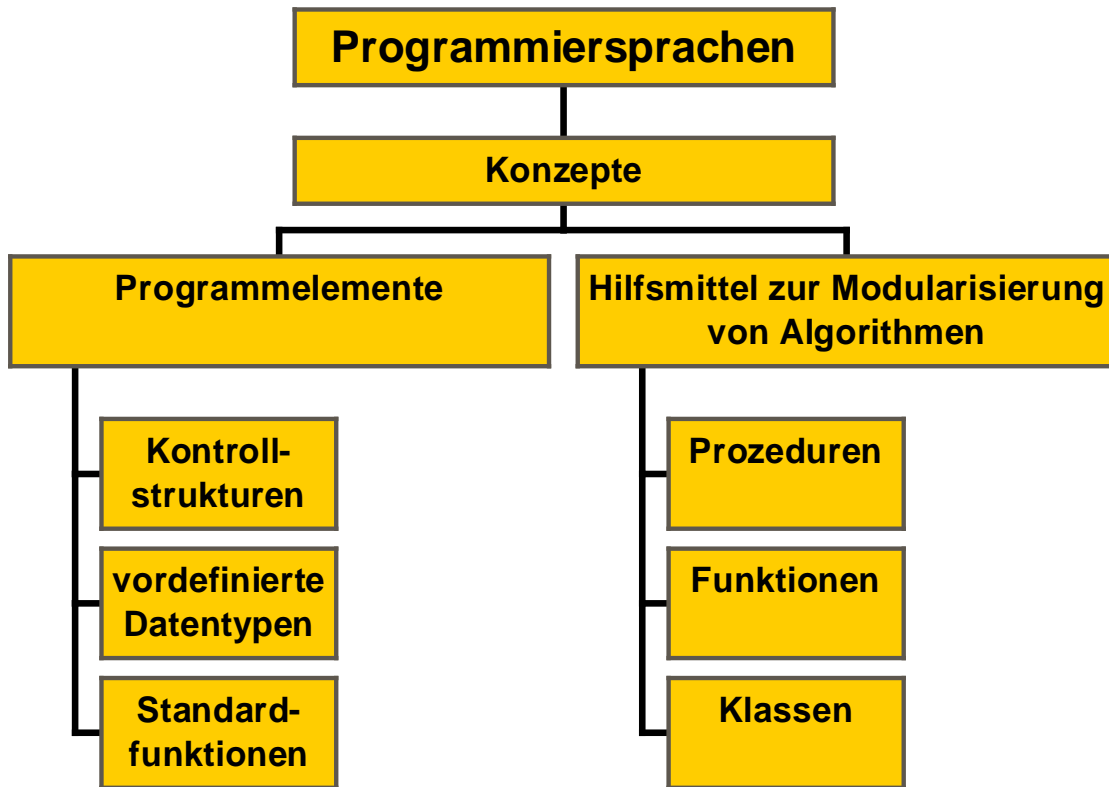
- ◆ *Deklarative (deskriptive) Programmiersprachen*
  - Beschreibung des Problems durch funktionale (Funktionen) oder logische Zusammenhänge (Programmklauseln)
- ◆ *logische Programmiersprachen*
  - Lösung von bestimmten Aufgaben (z.Bsp. Datenbanken)
  - es werden nur die Bedingungen für eine korrekte Lösung bestimmt (es wird kein Lösungsweg angegeben)
  - die Einzelheiten der Umsetzung werden nicht betrachtet
  - auch prädikative Programmiersprache (basierend auf dem Prädikatenkalkül); Rekursion und Backtracking
  - PROLOG-Beispiel: `sum(A, B, C) if C = A + B`

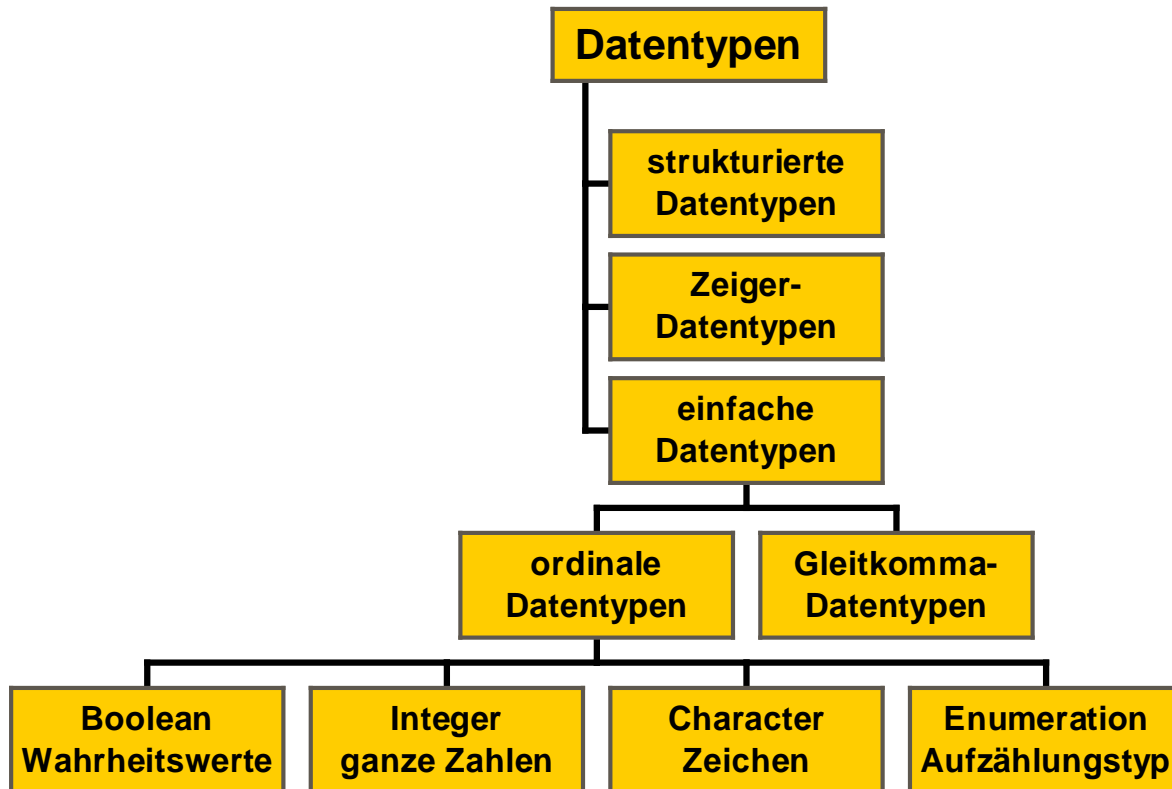
### ◆ *funktionale Programmiersprachen*

- Charakterisierung:
  1. große Anzahl eingebauter Funktionen
  2. kompakte Funktionsdefinition
  3. rekursive Aufrufbarkeit von Funktionen
  4. bedingte Ausdrücke
  5. Programme sind Terme
  6. Formeln werden als besondere Terme aufgefasst
  7. keine Einschränkung in Zahl und Typ der Argumente und Funktionswerte bei selbstdefinierten Funktionen
  8. Argumente und Wert von Funktionen dürfen Funktionen sein
- basierend auf dem Lambda-Kalkül (A. Church, 1936)
- idealerweise ohne Variablen
- LISP-Beispiel: `(DEFINE (SUM A B) (+ A B))`

### ◆ *Objektorientierte Programmiersprachen*

- Weiterentwicklung der modularen Programmierung
- Kapselung von Daten und dazugehörigen Methoden zu „Objekten“
- Möglichkeiten der Formulierung allgemeiner Beschreibungen in Form der „Klassen“
- Wiederverwendbarkeit durch Vererbung und Ableitung
- Ereignisgesteuerter Programmablauf





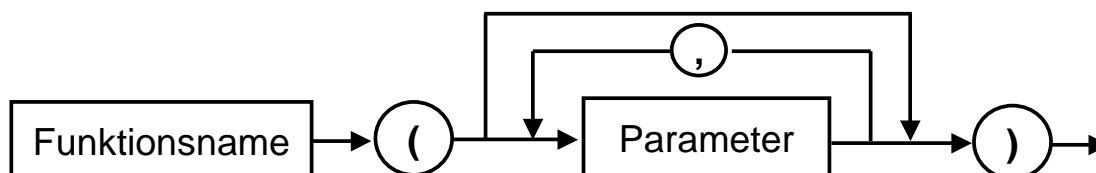
◆ *Backus-Naur-Form*

- entwickelt von Backus (USA) und Naur (Dänemark)
- Künstliche Sprache zur Beschreibung von Daten und Abläufen (ALGOL wurde in dieser Sprache beschrieben)
- Bezeichner stehen in spitzen Klammern (<Variable> ::= ... )
- Beschreibung beginnt mit einem Startwert
- Vereinfachtes Beispiel: Palindrome

<Palindrom> ::= A(Palindrom)A | B(Palindrom)B | ... | Z(Palindrom)Z |  
 (Buchstabe) | ε (ε ist leeres Wort)

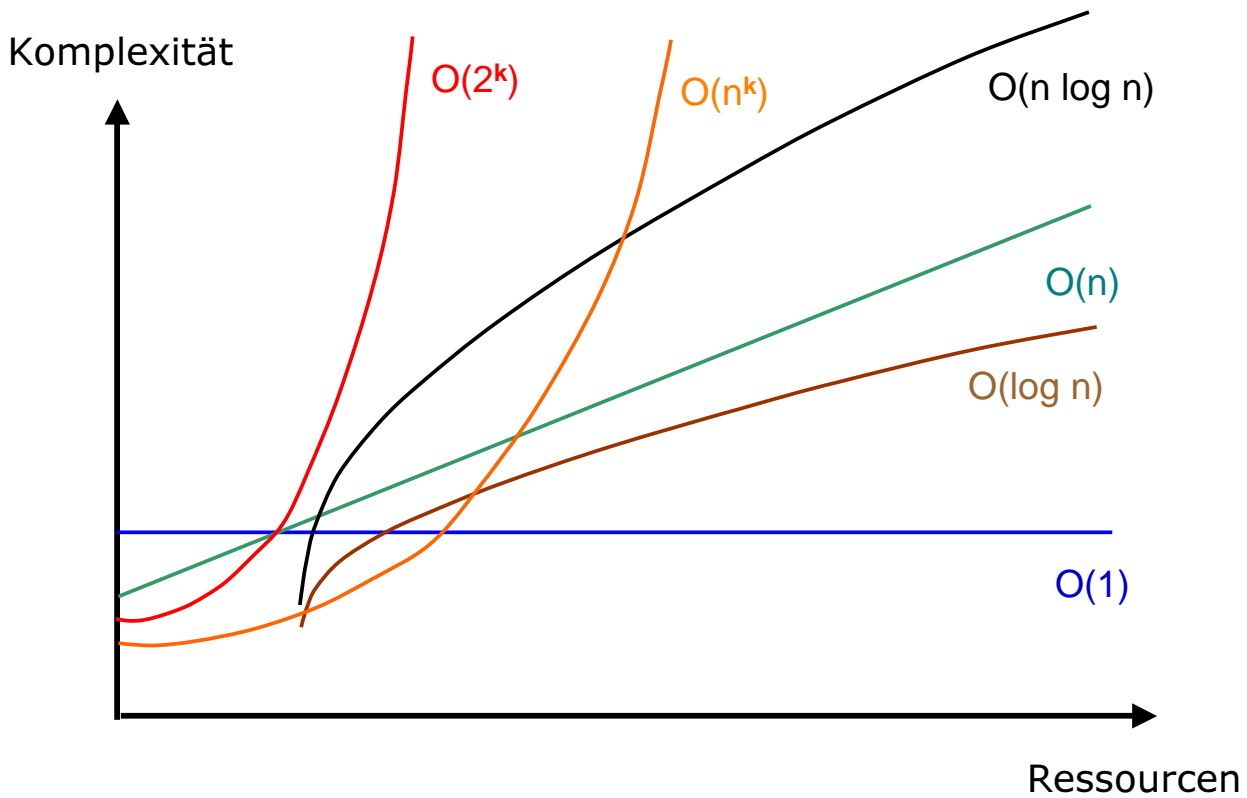
<Buchstabe> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

- ◆ Eine weitere Möglichkeit wäre eine Beschreibung mit Hilfe von *Syntaxdiagrammen* (Bsp. Funktionsaufruf)



- ◆  $[ \dots ]$  was zwischen den eckigen Klammern steht kann, muss aber nicht stehen.
- ◆  $\dots | \dots$  es gilt eine der Alternativen (die vor oder die nach dem Senkrechtstrich)
- ◆  $\langle \dots \rangle$  was zwischen den spitzen Klammern steht ist ein Platzhalter für einen Bezeichner
- ◆  $,$  Komma ist Trenner bei Aufzählungen
- ◆  $\{ \dots \}$  beliebige Anzahl des in Klammern stehenden Syntax-Elementes
- ◆  $\{ \dots \}_1$  wie oben, mindestens einmal

- ◆ Unter Komplexität von Algorithmen wird Verbrauch von Betriebsmitteln (Ressourcen) verstanden. Das kann der Speicher- oder Zeitverbrauch des Algorithmus sein
- ◆ Dieser Verbrauch wird mit der Formel  $O(\dots)$  ausgedrückt („groß O“ steht für Ordnung)
- ◆ Bei Algorithmen mit  $O(1)$  oder  $O(n^0)$  ist das Problem nicht abhängig von den Datenmengen oder der Rechenzeit
- ◆  $O(n)$  ist lineare Komplexität
- ◆ Mit  $O(n^2)$ ,  $O(n^3)$ , ... wird quadratische, kubische, ... Komplexität bezeichnet
- ◆  $O(n^k)$  mit  $k > 0$  heißt polynomiale Komplexität
- ◆  $O(\log n)$  oder  $O(n \log n)$  ist logarithmische Komplexität
- ◆ Exponentielle Komplexität ist z. Bsp.  $O(2^k)$



- ◆ Organisation einer optimalen Rundreise mit 70 Städten

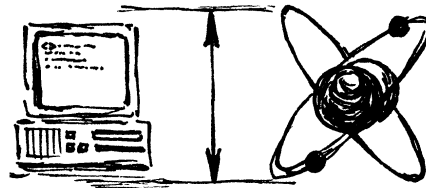
Abb.: Antje Elsner



Problem des Handelsreisenden

- ◆ Bewertung einer Rundreise mit einem Maschinenbefehl
- ◆ Befehl wird in der Zeit abgearbeitet, die Licht benötigt, um den Rechner zu durchqueren
- ◆ Rechner ist so groß wie Heliumatom

Abb.: Antje Elsner



- ◆ Es stehen so viele Rechner zur Verfügung, wie in eine Kugel mit dem Radius von 1 AE (astronomische Einheit) passen
- ◆ Die Rechner haben 10 Milliarden Jahre Zeit, um das Problem zu lösen

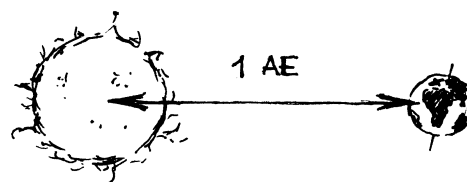


Abb.: Antje Elsner

- ◆ Die Rechner sind **nicht** in der Lage, das Problem in dieser Zeit zu lösen

- ◆ Alle Probleme, die mit polynomialen Aufwand lösbar sind, gehören zur Klasse  $\mathcal{P}$
- ◆ Probleme, die nicht mit polynomialen Aufwand lösbar sind, bilden die Klasse  $\mathcal{NP}$
- ◆ Viele Probleme der Klasse  $\mathcal{NP}$  sind eng miteinander verwandt, das heißt, wenn eines dieser Probleme mit polynomialen Aufwand lösbar wäre, dann sind es die anderen auch. Diese Probleme werden auch als  $\mathcal{NP}$ -vollständig bezeichnet.
- ◆ Somit wäre für diese Problemklassen  $\mathcal{P} = \mathcal{NP}$ . Das konnte bisher nicht bewiesen oder widerlegt werden!
- ◆ Es gibt aber Probleme, die nicht  $\mathcal{NP}$ -vollständig sind!  
(Problem, ob Zahl Primzahl ist oder nicht)

- ◆ Für die Lösung von Problemen muss zunächst die Art des Problems heraus gefunden werden
  - Nichtalgorithmische Probleme (Blei in Gold verwandeln)
  - Theoretisch nicht lösbare algorithmische Probleme (Halteproblem, Entscheidungsproblem)
  - Praktisch nicht lösbare bzw. sehr schwer lösbare Probleme (Problem des Handelsreisenden, Stundenplan-Problem)
  - Praktisch lösbare Probleme (Suchen, Sortieren)
- ◆ Die Einteilung kann unter verschiedenen Gesichtspunkten vorgenommen werden
  - Hier sollen zunächst einige heuristische Strategien genannt werden

- ◆ Der Begriff Heuristik kommt aus dem Griechischen
  - Der berühmteste Gelehrte des Altertums, Archimedes von Syrakus, soll nach der Entdeckung des Auftriebs während eines Bades „heureka“ rufend auf die Straße gelaufen sein
  - Heuristik ist die Lehre des Problemlösens

Abb.: Antje Elsner



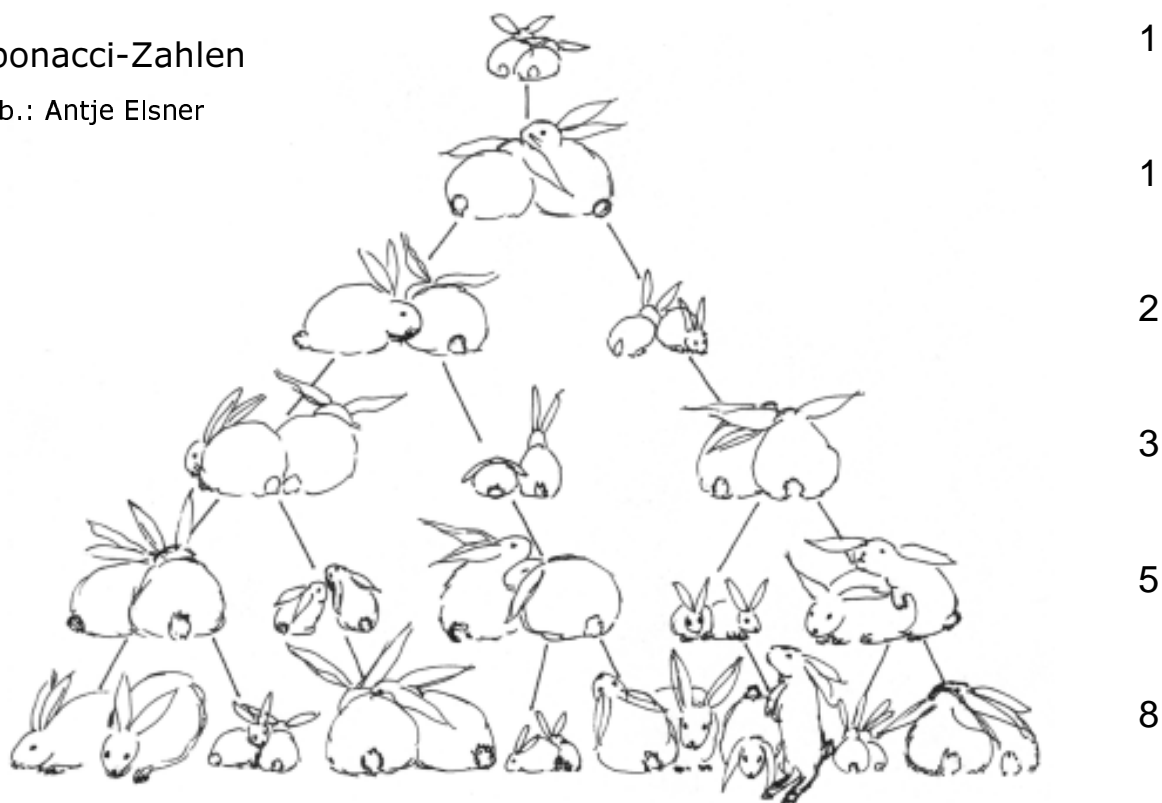
- ◆ Elementare (direkte, naive, straightforward) Methoden
  - Oft sehr allgemein. Sie sind für viele Fälle anwendbar
  - Raffiniertere Herangehensweisen sind oft nur bei bestimmten Fällen anwendbar
  
- ◆ Methode der rohen Gewalt „brute force method“
  - Durchprobieren aller Möglichkeiten
  - Diese Methode ist meist sehr aufwändig (hohe Komplexität)
  - Beispiel: Durchprobieren aller Züge bis zu einer bestimmten Tiefe beim Schachspiel
  
- ◆ Gierige Strategie „greedy strategy“
  - Es wird immer das nächste, für das Problem beste, Teilproblem erledigt
  - Beispiel TSP: Es wird immer zur nächstgelegenen Stadt gereist

- ◆ Modularität
  - Das Problem wird in Teilprobleme zerlegt, die einfacher zu lösen sind (Baukastenprinzip)
  - Lösung der einzelnen Teilprobleme
  - Einzellösungen werden zur Gesamtlösung zusammen gesetzt
  
- ◆ Rekursion
  - Fundamentales heuristische Prinzip
  - Viele Probleme sind „von Hause aus“ rekursiv
  - Selbstähnlichkeit (Ornamente, Fraktale)
  - Eng verbunden mit dem Prinzip der vollständigen Induktion
  - Kleinere Kopien desselben Problems bis zur kleinsten Stufe

- ◆ Wieviele Kaninchenpaare gibt es nach einem Jahr, wenn es anfangs ein junges Paar gibt, jedes Paar jeden Monat ein neues Paar zur Welt bringt und alle Jungen sich jeweils nach einem Monat fortpflanzen?
- ◆ Zahlenreihe:  
1 1 2 3 5 8 13 21 34 55 89 144 ...
- ◆ Mathematische Beschreibung:  
 $F_0 = 0, F_1 = 1, F_{n+2} = F_{n+1} + F_n; n \geq 0$
- ◆ Am Beispiel der Fibonacci-Zahlen lässt sich leicht zeigen, dass bei einfacher Übernahme der rekursiven Formel für die Programmierung bereits bei wenigen hundert Zahlen der Computer für Stunden oder Tage ausgelastet ist

Fibonacci-Zahlen

Abb.: Antje Elsner



- ◆ Prinzip „Teile und Herrsche“ („divide and conquer“)
  - Das Problem wird in etwa gleichgroße Teilprobleme zerlegt, die gelöst werden
  - Die Gesamtlösung wird aus den Teillösungen zusammen gesetzt
  - Algorithmen werden meist rekursiv formuliert
  - Beispiel: Quicksort
  
- ◆ Systematisches Durchlaufen von Baumstrukturen
  - Breitensuche („breadth first“)
  - Tiefensuche („depth first“)
  - Backtracking als Modifikation der Tiefensuche, das heißt, es wird nur so weit in die Tiefe gegangen, wie zum Erkennen einer Lösung nötig ist

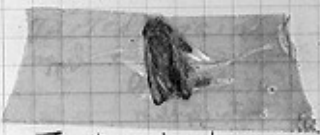
- ◆ Die gezielte mathematische Analyse
  - liefert oft sehr effiziente Lösungen
  - Beispiel: Lösung von quadratischen Gleichungen
  
- ◆ Probabilistische Verfahren
  - Verzicht auf hundertprozentige Sicherheit (Pseudo-Primzahlen)
  - Beispiel: Public Key Cryptography (RSA-Verfahren)
  
- ◆ Simulationsverfahren
  
- ◆ Neuronale Netze
  
- ◆ Genetische Algorithmen

9/2

9/9

0800 Antan started  
 1000 " stopped - antan ✓  
 1300 (033) MP-MC 1.95210000  
 (033) PRO 2 2.130476415 (033) 4.615925059(-2)  
 correct 2.130476415  
 Relays 6-2 in 033 failed special speed test  
 in Relay "1.000 test".  
 Relays changed

1100 Started Cosine Tape (Sine check)  
 1525 Started Multi-Adder Test.

1545  Relay #70 Panel F  
 (moth) in relay.

First actual case of bug being found.

~~1630~~ Antan started.  
 1700 closed down.

## Literaturangaben

- [1] Dietmar Herrmann: „Algorithmen Arbeitsbuch“, ADDISON-WESLEY Verlag, ISBN 3-89319-481-9
- [2] Siegfried Stein: „Software Engineering Methoden“, IBM Bildungszentrum NordWest, DOC SWE-ME SCRIPT 1992
- [3] Prof. Dr.-Ing. habil Hans-Ulrich Karl, Dr. rer. Nat. Barbara Speck Technische Universität Dresden, Fakultät Informatik „Grundlagen der Informatik“, VMS Verlag Modernes Studieren Hamburg - Dresden GmbH Bestellnummer: 1039 02 0
- [4] H. Schröder: „Grundlagen der Programmierung“, Herdt-Verlag für Bildungsmedien GmbH, PG99 30-0-07-02-01
- [5] Jochen Ziegenbalg: „Algorithmen von Hammurapi bis Gödel“, Spektrum Akademischer Verlag GmbH, ISBN 3-8274-0114-3
- [6] Andreas Solymosi / Ulrich Grude: „Grundkurs Algorithmen und Datenstrukturen“, Vieweg Verlagsgesellschaft mbH, ISBN 3-528-05743-2