Einführung in die Informatik, Algorithmen und Datenstrukturen

Teil 1 - Thema 2

Algorithmen

Inhalts- und Prozessbereiche der Bildungsstandards Informatik – Sek I

/Grundsätze und Standards für die Informatik in der Schule Empfehlungen der Gesellschaft für Informatik e. V., 2008 /

Entwurf der Bildungsstandards „Informatik für die Sek. I“ - Algorithmen

Algorithmen

Schülerinnen und Schüler aller Jahrgangsstufen

- kennen Algorithmen zum Lösen von Aufgaben und Problemen aus verschiedenen Anwendungsgebieten und lesen und interpretieren gegebene Algorithmen,
- entwerfen und realisieren Algorithmen mit den algorithmischen Grundbausteinen und stellen diese geeignet dar.

/Grundsätze und Standards für die Informatik in der Schule Empfehlungen der Gesellschaft für Informatik e. V., 2008 /

Bildungsstandards – Inhaltsbereich Algorithmen – Klasse 5 - 7

Beispielaufgabe 3.2.03:

Ein Roboter soll einen Garten mit einer Mauer umzäunen. Dies gelingt, wenn man ihm folgenden »Auftrag« erteilt:

```
Wiederhole 4-mal
  Wiederhole 8-mal
    Ziegel hinlegen
    Schritt nach vorn machen
  Ende Wiederholung
  nach links drehen
Ende Wiederholung
```

Versuche diese Handlungsvorschrift auf einem karierten Blatt Papier abzuarbeiten und beantworte dann folgende Fragen:

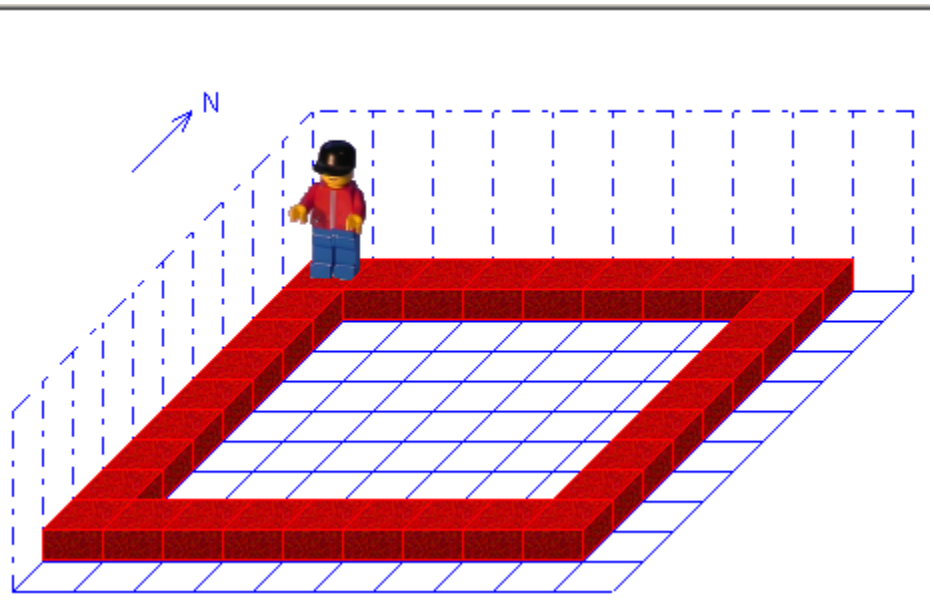
a) Wie groß muss die Welt mindestens sein, damit der Roboter alle Anweisungen ausführen kann?
b) Wo muss der Roboter am Anfang seines »Auftrags« stehen? Wo steht er am Ende?
c) Wie hoch wird die Mauer um den Garten?
d) Wie viele Ziegel liegen jeweils hintereinander?
e) Wie groß ist der Garten, den der Roboter umzäunt hat?

Stelle die oben benutzte Handlungsvorschrift grafisch dar (z. B. in einem Struktogramm).

/Grundsätze und Standards für die Informatik in der Schule Empfehlungen der Gesellschaft für Informatik e. V., 2008 /

Umsetzung mit Karol

```
Wiederhole 4 mal
  Wiederhole 8 mal
     Hinlegen
     Schritt
  *wiederhole
   LinksDrehen
*wiederhole
```



Algorithmenbegriff

Ein **Algorithmus** besitzt folgende Merkmale:

(1) Er besteht aus (endlich vielen) *Schritten*. Jeder dieser Schritte enthält eine unmißverständliche *Anweisung*.

(2) Es gibt einen eindeutig bestimmten Schritt, der *als erster* auszuführen ist. Nach jedem Schritt steht fest, ob der Algorithmus *beendet* ist bzw. welcher Schritt *als nächster* auszuführen ist.

(3) Er ist *für beliebige Werte* eines Grundbereiches ausführbar.

Man erkennt:

- Durch einen Algorithmus werden vorgegebenen *Eingabedaten* eindeutig bestimmte *Ausgabedaten* zugeordnet.
- Zu einer Aufgabenstellung können oft *voneinander verschiedene Algorithmen* gefunden werden.

  Es wurden z. B. zur Ermittlung von Werten des Terms $\frac{a \cdot b}{a + b}$ zwei Möglichkeiten angegeben; eine dritte wird in Aufg. 3 (↗ S. 21) genutzt.
- Algorithmen können auf *unterschiedliche Weise dargestellt* werden, z. B. durch eine Folge von Befehlssätzen, ein Diagramm, einen Ablaufplan.

/Lehrbuch Mathematik Klasse 9; Volk und Wissen Volkseigener Verlag Berlin 1987/

Algorithmenbegriff

Algorithmus ist eine Verarbeitungsvorschrift, die aus einer **endlichen Folge** von **eindeutig ausführbaren Anweisungen** besteht, mit der man eine **Vielzahl gleichartiger Aufgaben** lösen kann. Ein Algorithmus gibt an, wie Eingabegrößen schrittweise in Ausgabegrößen umgewandelt werden.

EVA-Prinzip

Algorithmusdefinitionen (intuitiv)

Algoṛithmus [mittellat.], Rechenvorgang, der nach einem bestimmten, sich wiederholenden Schema abläuft. Jede Aufgabe, deren Lösungsverfahren sich durch einen A. beschreiben lässt, kann auch mithilfe einer Rechenanlage gelöst werden.

(c) Meyers Lexikonverlag.

Algorithmus Al|go|ṛith|mus <*arab.-mlat.*> *der;* -, ...men:
1. (veraltet) Rechenart mit Dezimalzahlen.
2. Rechenvorgang, der nach einem bestimmten [sich wiederholenden] Schema abläuft (Arithmetik).
3. Verfahren zur schrittweisen Umformung von Zeichenreihen (math. Logik)

(c) Dudenverlag.

Algorithmusdefinitionen (intuitiv)

Algorithmus ::=
"eine **endliche Folge von Regeln**, nach denen sich nach **endlich vielen, eindeutig festgelegten Schritten** die Lösung einer Auf-gabe so ergibt, dass sie **eindeutig** den in ihr enthaltenen Bedingungen entspricht"
"ein System von Regeln, nach denen gegebene Größen (Eingabegrößen, Eingabeinformationen) in andere Größen (Ausgabegrößen, Ausgabeinformationen) umgeformt oder umgearbeitet werden können.„

Der Algorithmus ist eine **allgemeine Handlungsvorschrift**, die unabhängig von der Realisierung in einer bestimmten Programmiersprache ist. Den Vorgang der Abarbeitung eines Algorithmus bezeichnet man als **Prozess**, die ausführende Maschine (die ausführende Person) als **Prozessor**.

Algorithmusdefinitionen

Turing:

„Ein Algorithmus ist eine Menge von Anweisungen für eine einfache Maschine (Turingmaschine).“

Church-Turing-These

(1) Alle vernünftigen Definitionen von „Algorithmus“, soweit sie bekannt sind, sind gleichwertig und gleichbedeutend.

(2) Jede vernünftige Definition von „Algorithmus“, die jemals irgendwer aufgestellt hat, ist gleichwertig und gleichbedeutend zu denen, die wir kennen.

Eigenschaften von Algorithmen - Abstraktion

Abstraktion

Ein Algorithmus wird im allgemeinen **zur Lösung einer Klasse von Problemen** definiert. Die Wahl eines einzelnen, aktuell zu lösenden Problems aus dieser Klasse erfolgt über die Parametrisierung. Diese wird häufig über Eingabedaten realisiert.

Allgemeingültigkeit

*Ein Algorithmus muss auf **alle Aufgaben gleichen Typs** (Aufgabenklasse) anwendbar sein und (bei richtiger Anwendung) stets zum gesuchten Resultat (zur Lösung bzw. zur Einsicht, dass die Aufgabe nicht lösbar ist) führen.*

Eigenschaften von Algorithmen - Finitheit

Finitheit

Statische Finitheit bedeutet, dass der **Algorithmus von endlicher Länge** ist.

Als **dynamische Finitheit** wird die Eigenschaft bezeichnet, dass die für die Ausführung des **Algorithmus benötigten Ressourcen jederzeit endlich** sind.

Endlichkeit

Ein Algorithmus besteht aus endlich vielen Anweisungen (Verarbeitungsbefehlen bzw. Regeln) endlicher Länge.

Eigenschaften von Algorithmen - Terminiertheit

Terminiertheit
Ein Algorithmus heißt **terminierend**, wenn er für jede zulässige Eingabe nach **endlich vielen Schritten ein Ergebnis liefert**.

Endlichkeit *der Beschreibung bedeutet noch nicht, dass auch der beschriebene* ***Prozess endlich*** *sein muss. Man fordert, dass der beschriebene Prozess nach endlich vielen Schritten (deren Anzahl nicht bekannt sein muss) abbricht, d.h. nach einer endlichen Zeitspanne zum Ende kommt (terminiert). Dies wird für Iterationen z.B. dadurch erreicht, dass man die Genauigkeit bzw. eine maximale Durchlaufanzahl festlegt.*

Eigenschaften von Algorithmen – Determiniertheit

Determiniertheit

Eine weitere wesentliche Eigenschaft von Algorithmen ist die Determiniertheit. Existiert in einem Algorithmus **zu jeder Aktion höchstens eine Folgeaktion**, und ist der Berechnungsablauf dadurch eindeutig bestimmt, so heißt der Algorithmus **deterministisch**.

Eindeutigkeit

Mit jeder Anwendung ist auch die nächstfolgende festgelegt*, das heißt, die Reihenfolge der Abarbeitung der Anweisungen unterliegt nicht der Willkür des Ausführenden. (Man sagt auch: Algorithmen sind deterministisch.) Das heißt, dass bei gleichen Bedingungen gleiche Eingabegrößen bei wiederholter Abarbeitung eines Algorithmus auf dieselben Ausgabegrößen abgebildet werden.*

Nutzung von Algorithmeneigenschaften

Auflistung aller Primzahlen

❍ **Terminiertheit**
ⓧ **Determiniertheit**
ⓧ **Finitheit**
? **Abstraktion**

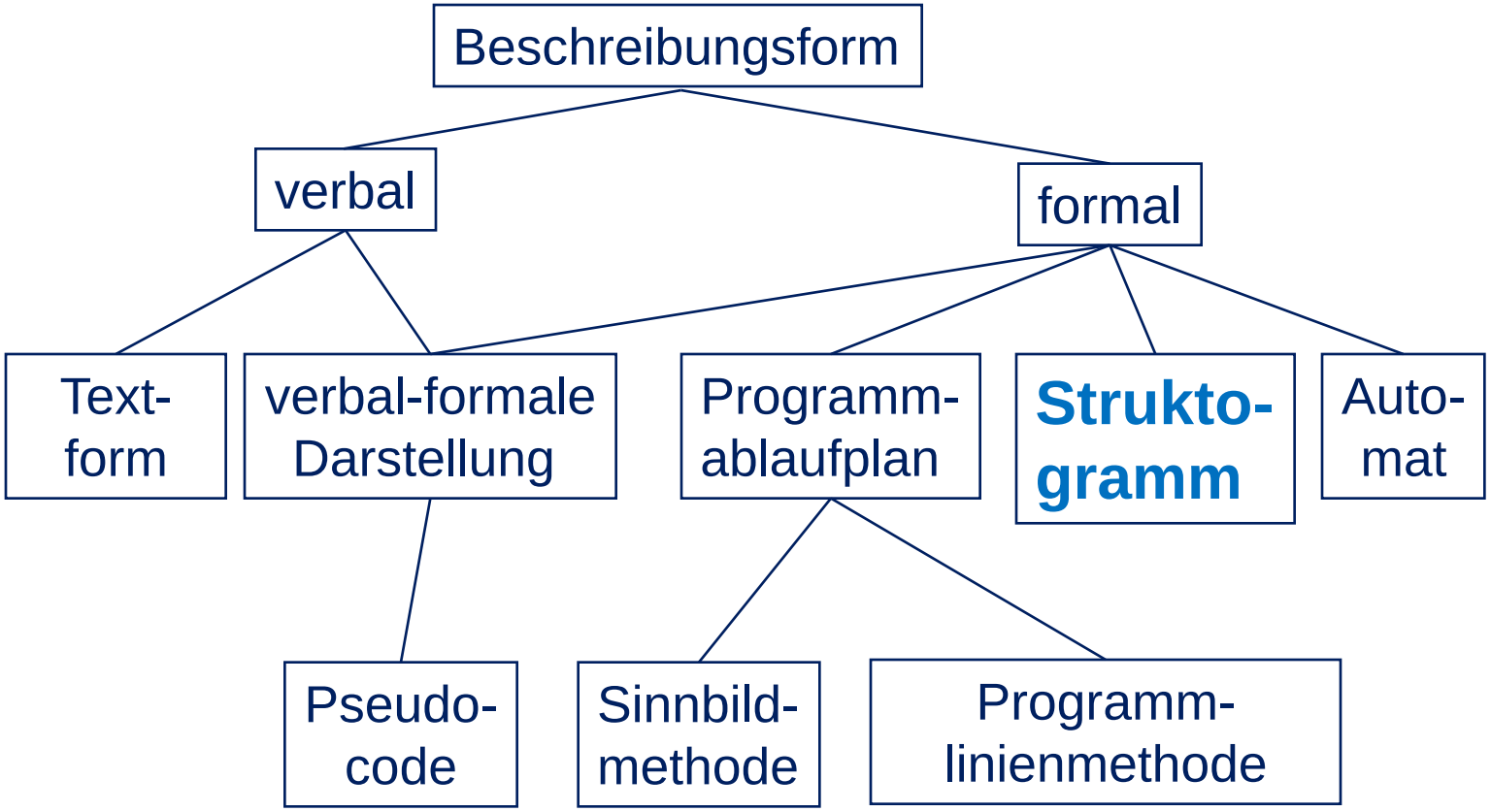
Auflistung aller Primzahlen im Intervall a bis b

Ⓧ **Terminiertheit**
Ⓧ **Determiniertheit**
Ⓧ **Finitheit**
Ⓧ **Abstraktion**

Benotung eines Aufsatzes

Ⓧ **Terminiertheit**
❍ **Determiniertheit**
ⓧ **Finitheit**
❍ **Abstraktion**

Beschreibungsmittel für Algorithmen



Darstellungsformen - verbale Notation

Grießflammeri:

- 1 Liter Milch,
- 200 g Weichweizengrieß,
- 80 g Zucker,
- 1 Prise (etwa 0,2 g) Salz,
- 20 g Butter,
- 3 Eier

Eier trennen und Eiweiß zu Schnee schlagen. Milch zum Kochen bringen. Grieß, Zucker und Salz unter Rühren hinzugeben und auf kleiner Flamme rühren, bis die Masse dick geworden ist. Der Klumpenbildung kann man entgegenwirken, indem man Grieß und Zucker vorher vermischt. Dann vom Feuer nehmen, Eigelb und Butter dazugeben und vermischen. Eischnee unterheben. Abkühlen lassen. Kann pur oder mit Kompott gegessen werden (Kirschkompott passt gut dazu).

/Quelle: http://de.wikipedia.org/wiki/Kochrezept/

Darstellungsformen – formale Notation

Die folgende Darstellungen beziehen sich auf **imperative Algorithmen**.

Diese sind dadurch gekennzeichnet, das die Problemlösung auf einer Folge elementarer Aktionen beruht, die Zustandsübergänge im ausführenden Rechner bewirken.

Die Zustandübergänge sind durch Werte von Variablen beschrieben.

(Elementare Aktionen: Eingabe, Wertzuweisung, Ausgabe)

Darstellungsformen – verbal formale Notation

Algorithmus:

Quicksort (F : folge);

Falls F die leere Folge ist oder F nur aus einem einzigen Element besteht, bleibt F unverändert; sonst:

Divide: Wähle ein Pivotelement k von F (z.B. das letzte) und teile F ohne k in die Teilfolgen F_1 und F_2 bzgl. k:

F_1 enthält nur Elemente von F ohne k, die <= k sind,

F_2 enthält nur Elemente von F ohne k, die >= k sind,

Conquer: Quicksort(F_1); Quicksort(F_2);

{ Nach Ausführung dieser beiden Aufrufe sind F_1 und F_2 sortiert }

Merge: Bilde die Ergebnisfolge F durch Hintereinanderhängen von F_1, k, F_2 in dieser Reihenfolge.

Darstellungsformen - Pseudocode

Aufgabe:

Es ist eine Anzahl von Messwerten einzulesen. Der Definitionsbereich liegt zwischen –255 und 255. Als Endekennzeichen ist ein Wert <= -999 einzugeben. Das arithmetische Mittel aller Messwerte ist zu berechnen und auszugeben.

```
BEGINN
lies (W);
S:=0;
I:=0;
SOLANGE W > -999
TUE
     BEGINN
     I := I + 1;
     S := S + W;
     lies (W);
     ENDE
D:= S/I;
schreib (D);
ENDE.
```

Darstellungsformen - Pseudocode

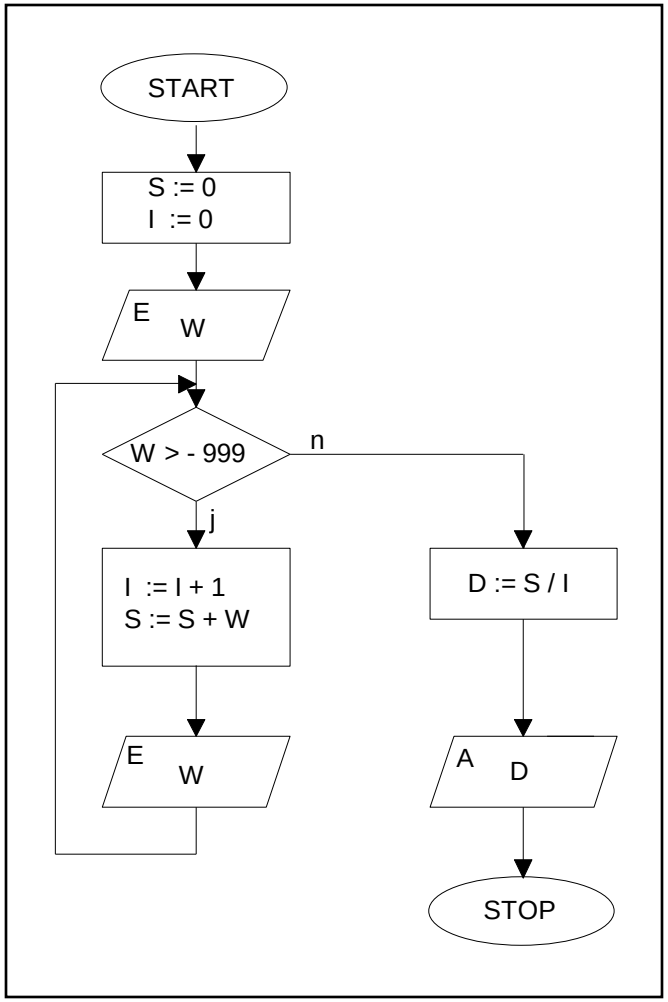
Es sind die reelen Nullstellen der quadratischen Gleichung
$y = a * x^2 + b * x + c$ mit Hilfe der Lösungsformel zu berechnen und auszugeben.

(1) lies A, B, C
(2) falls $A = 0$, dann (3), sonst (4)
(3) Lösung von $ax + b = 0$, wobei $a := B$ und $b := C$
(4) $p := \frac{B}{A}$ und $q := \frac{C}{A}$
(5) $D := \left(\frac{p}{2}\right)^2 - q$
(6) falls $D \geqq 0$, dann (7), sonst (9)
(7) $x_1 := -\frac{p}{2} + \sqrt{D}$ und $x_2 := -\frac{p}{2} - \sqrt{D}$
(8) drucke x_1 und x_2
(9) stopp

Darstellungsformen - PAP (Sinnbildmethode)

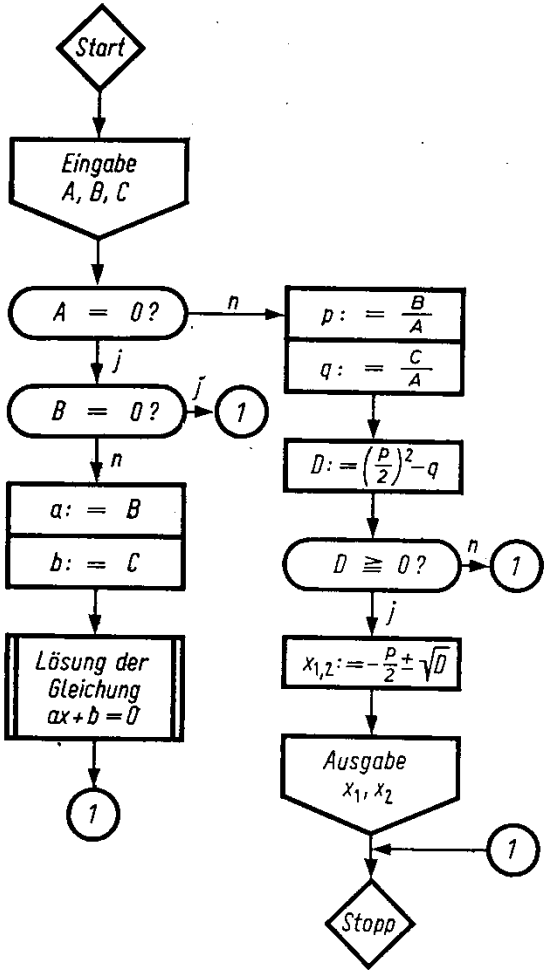
Aufgabe:

Es ist eine Anzahl von Messwerten einzulesen. Der Definitionsbereich liegt zwischen –255 und 255. Als Endekennzeichen ist ein Wert <= -999 einzugeben. Das arithmetische Mittel aller Messwerte ist zu berechnen und auszugeben.



Darstellungsformen - PAP (Sinnbildmethode)

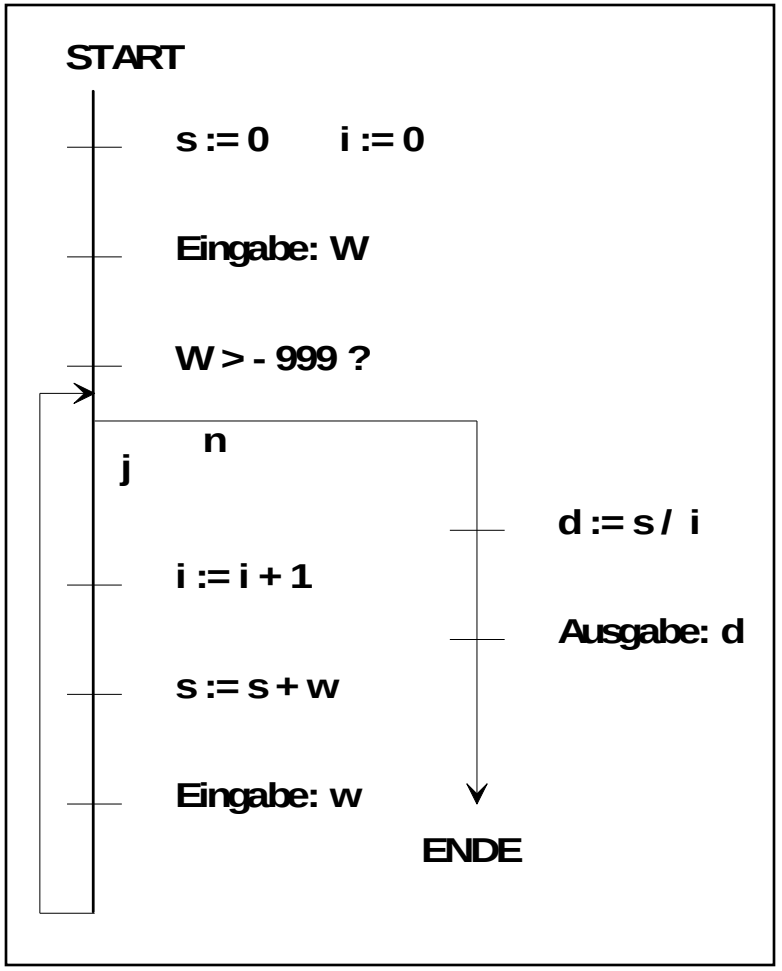
Es sind die reelen Nullstellen der quadratischen Gleichung $y = a * x^2 + b * x + c$ mit Hilfe der Lösungsformel zu berechnen und auszugeben.



Darstellungsformen - PAP (Programmlinienmethode)

Aufgabe:

Es ist eine Anzahl von Messwerten einzulesen. Der Definitionsbereich liegt zwischen –255 und 255. Als Endekennzeichen ist ein Wert <= -999 einzugeben. Das arithmetische Mittel aller Messwerte ist zu berechnen und auszugeben.

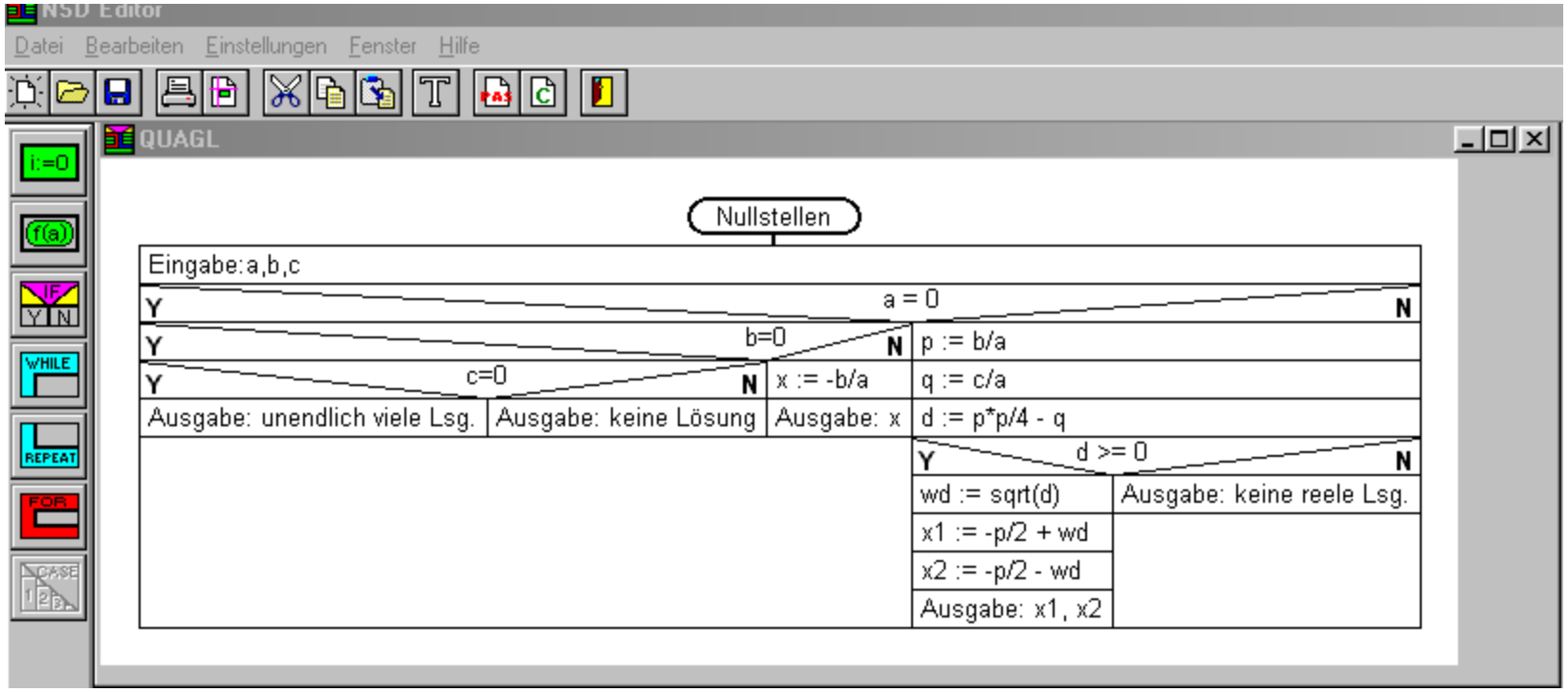


Darstellungsformen - Struktogramm

Struktorgramme werden auch als **Nassi-Shneiderman-Diagramme** (NSD) bezeichnet. Sie sind eine abstrakte Beschreibungsform für Algorithmen.

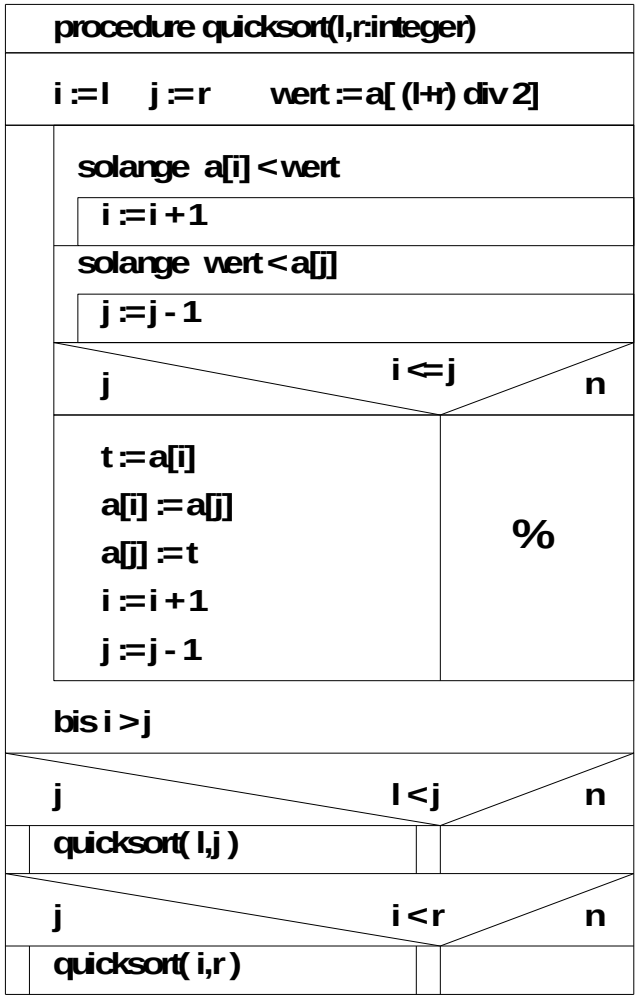
Sie erlauben es, prozedurale Programmabläufe zu beschreiben, zu entwickeln und zu dokumentieren. Dabei werden die erarbeiteten Prozeduren auf einer Code-unabhängigen Ebene beschrieben und unterstützen damit den «Top-Down-Entwurf».

Darstellungsformen - Struktogramm



/http://diuf.unifr.ch/softeng/student-projects/completed/kalt/ftp-nsd.html/

Darstellungsform Struktogramm - Programm




```
procedure quicksort3m(var fe:
feld;l,r:integer);
 var wert,t,i,j : integer;
 begin
   i := l;
   j := r;
   wert := fe[(l+r) div 2];
     repeat
       while fe[i] < wert do
         i := i + 1;
       while wert < fe[j] do
         j := j - 1;
       if i <= j then
         begin
         t := fe[i];
         fe[i] := fe[j];
         fe[j] := t;
         i := i + 1;
         j := j - 1;
         end; { of if }
     until i > j;
     if l < j then
       quicksort3m(fe,l,j );
     if i < r then
       quicksort3m(fe,i,r );
end; { of procedure quicksort3m }
```

Regeln für die Arbeit mit Struktogrammen

- Die Größe eines Struktogramms ist immer auf **eine Seite** (A4) beschränkt.
- In Struktogrammen gibt es immer nur e**inen Eingang** und **einen Ausgang**. Dies gilt auch für die einzelnen Grundsymbole.
- Der **Programmablauf** erfolgt grundsätzlich **von oben nach unten**.
- Die Grundsymbole können **ineinander geschachtelt** und **aneinander gereiht** werden.
- Alle Programmverzweigungen laufen an einer Stelle wieder zusammen.
- Bei der Entwicklung von Struktogrammen ist das **Prinzip der schrittweisen Verfeinerung** anzuwenden (TOP-DOWN).
- Bei größeren Programmen sollte man **zusätzlich einen Strukturbaum** (hierachisches Funktionsdiagramm) zur Beschreibung des Gesamtprogramms aufstellen.

Sinnbilder für Struktogramme nach Nassi-Shneiderman

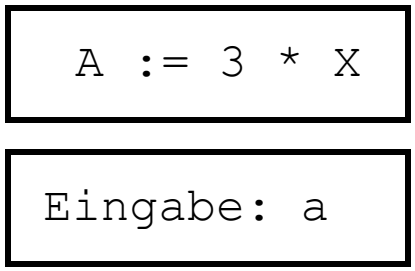
Grundlage: DIN 66261 / EN 28631

„Die Aussagen eines Programmablaufplanes erfolgen mit Hilfe von Sinnbildern und erläuternden Texten in den Sinnbildern. Der Programmablauf wird durch die Auswahl der Sinnbilder und deren Schachtelung dargestellt. Die Texte beschreiben inhaltlich die Bedingungen und die Verarbeitungen.“

(bei der Verwendung von Sinnbildern für Programmablaufpläne gilt DIN 66001)

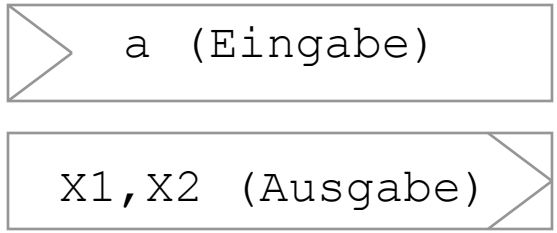
Strukturblöcke für Algorithmen - Verarbeitung

Dieses Programmkonstrukt besteht aus einen Verarbeitungsteil. Der Steuerungsteil ist implizit vorhanden. Der Verarbeitungsteil wird genau einmal ausgeführt, wenn das Konstrukt durchlaufen wird.



Folgende elementaren Anweisungen werden in Verarbeitungsblöcken dargestellt:

- **Wertzuweisung**
- **Eingabe- und Ausgabeoperation**



In der Schulinformatik verwendete Symbole, die nicht der DIN bzw. EN entsprechen.

Strukturblöcke für Algorithmen - Verarbeitung

Werden mehre Verarbeitungen unter einem Namen zusammengefasst, so kann das Symbol für Blöcke verwendet werden.

```
p := b / a
q := c / a
d := p*p/4-q
```

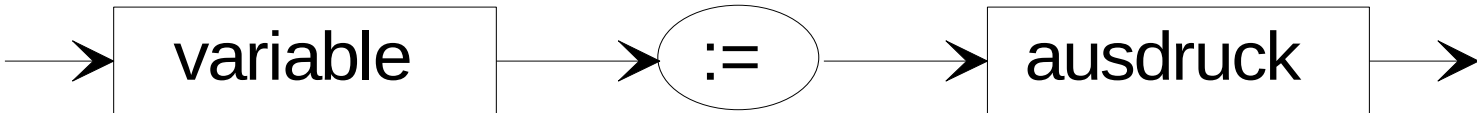
Der des Inhaltes der Anweisungen wird vom Anwender bestimmt.

```
Berechnen
   des
Mittelwertes
```

Object-Pascal Wertzuweisungen

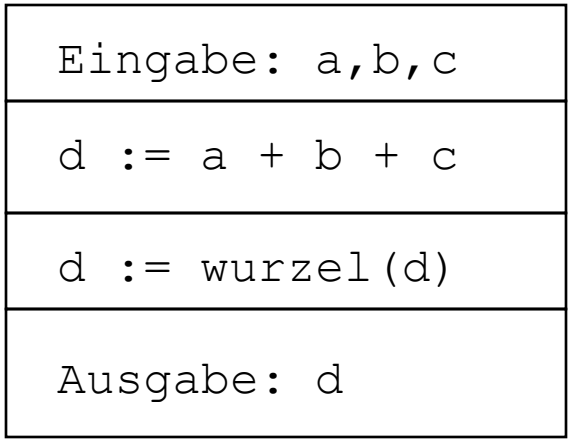
Die Ergibt-Anweisung überträgt den Wert, der bei der Auswertung des Ausdrucks in der Zuweisung ermittelt wird, auf die Variable. Der Wert des Ausdrucks muss zuweisungsverträglich zum Typ der Variablen sein. Variable, denen noch kein Wert übertragen wurde, heißen **undefiniert**.

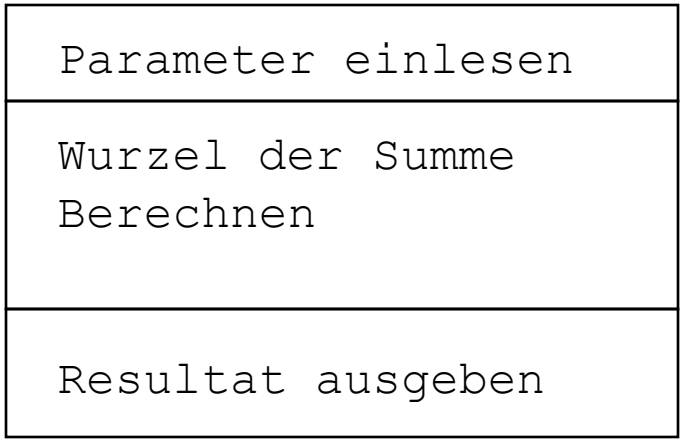
Syntax in Pascal/Delphi



Strukturblöcke für Algorithmen - Sequenz

Dieses Programmkonstrukt enthält zwei oder mehrere Verarbeitungsteile. Der Steuerungsteil ist implizit vorhanden, die Verarbeitungsteile werden genau einmal in der angegebenen Reihenfolge ausgeführt, wenn das Programmkonstrukt durchlaufen wird.

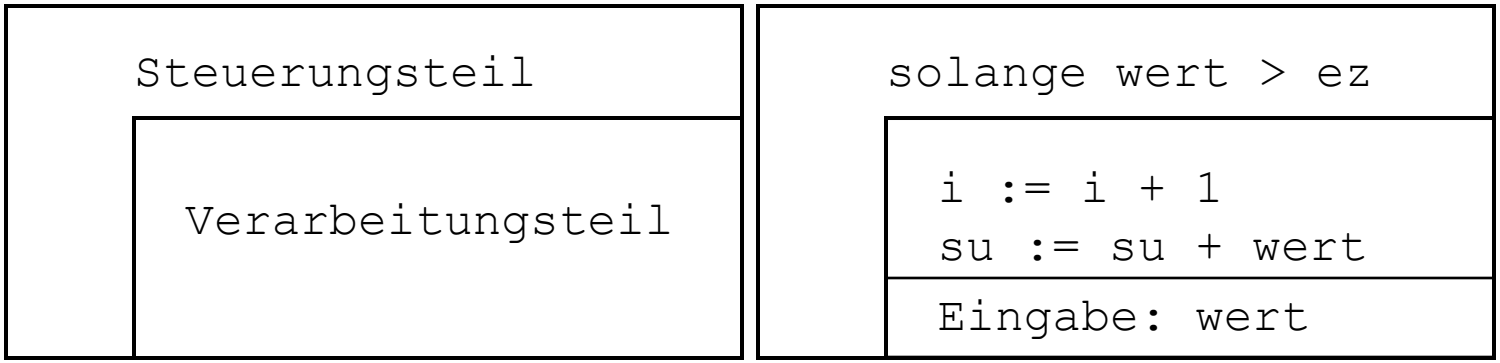




Strukturblöcke für Algoríthmen – Iterationsblöcke (Wiederholungen)

Wiederholungen mit vorausgehender Bedingungsprüfung – abweisende Zyklen (pre-tested iteration)

Dieses Programmkonstrukt besteht aus einem **Verarbeitungsteil** und einem **Steuerungsteil mit einer Bedingung**. Die Bedingung bestimmt, ob. bzw. wie häufig der Verarbeitungsteil ausgeführt wird, wenn das Programmkonstrukt durchlaufen wird.



Object-Pascal: WHILE-Schleife, FOR-Schleife

Terminiertheit von Schleifen

Es ist zu beachten, dass die Schleife terminiert (beendbar) ist. Dazu sind folgende Überprüfungen notwendig:

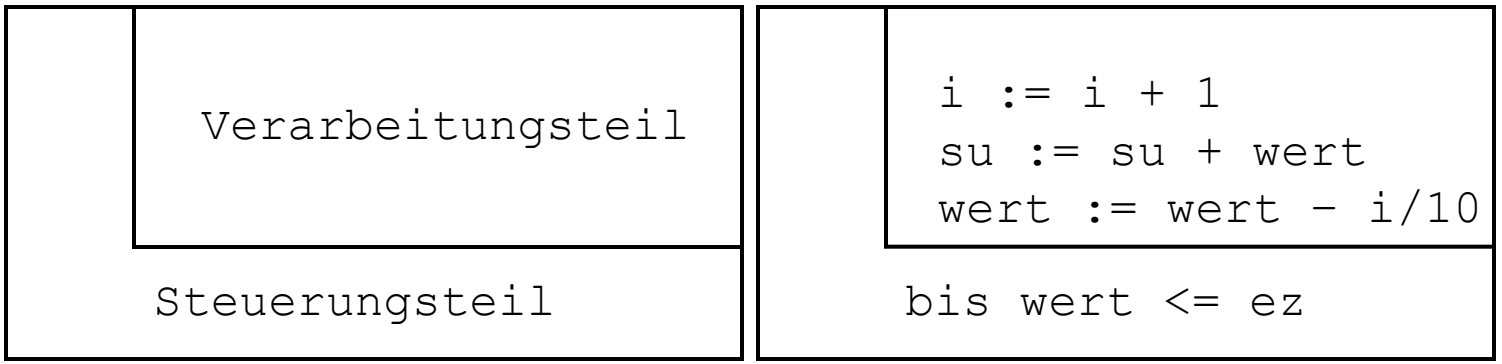
- Ist in der Anweisung überhaupt eine Variable vorhanden, die der Bedingung den für den Abbruch notwendigen Wert geben kann?

- Wird diese Bedingung jemals erreicht?

Strukturblöcke für Algorithmen – Iterationsblöcke (Wiederholungen)

Wiederholungen mit nachfolgender Bedingungsprüfung – nichtabweisende Zyklen (post-tested iteration)

Dieses Programmkonstrukt besteht aus einem **Verarbeitungsteil** und einem **Steuerungsteil mit einer Bedingung**. Die Bedingung bestimmt, ob. bzw. wie häufig der Verarbeitungsteil **nach der ersten Ausführung** wiederholt wird, wenn das Programmkonstrukt durchlaufen wird.



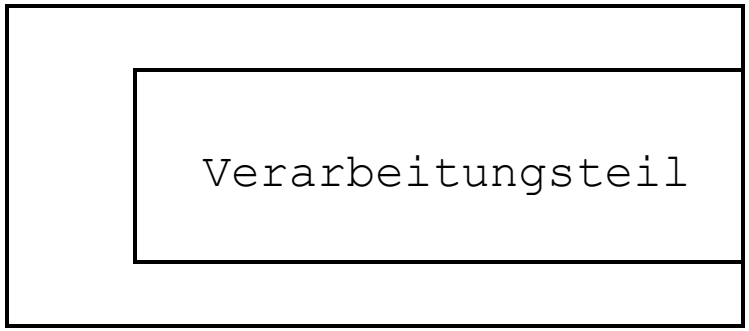
Object-Pascal: REPEAT-UNTIL-Schleife

Strukturblöcke für Algorithmen – Iterationsblöcke (Wiederholungen)

Wiederholungen ohne Bedingungsprüfung (continous iteration)

Dieses Programmkonstrukt enthält nur einem **Verarbeitungsteil.** Der Steuerungsteil ist implizit vorhanden. Der Verarbeitungsteil wird wiederholt ausgeführt, wenn das Programmkonstrukt durchlaufen wird.

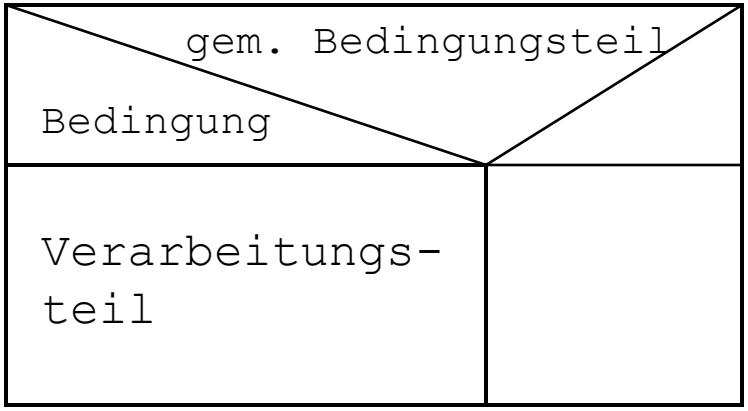
Diese Schleifenform entspricht einer Endlosschleife. Ein Abbruch kann nur durch einen Eingriff von außen erfolgen.

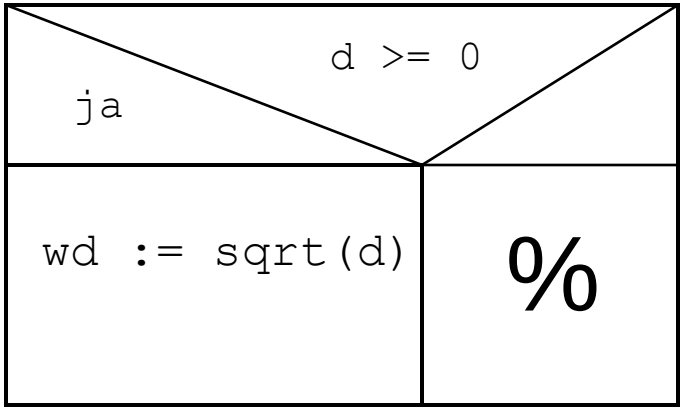


Strukturblöcke für Algorithmen – Alternative (Selektionsblöcke)

Bedingte Verarbeitung – unvollständige Alternative
(selected choice construct)

Dieses Programmkonstrukt besteht aus einem **Verarbeitungsteil** und einem **Steuerungsteil mit einer Bedingung**. Die Bedingung bestimmt, ob der Verarbeitungsteil ausgeführt wird, wenn das Programmkonstrukt durchlaufen wird.

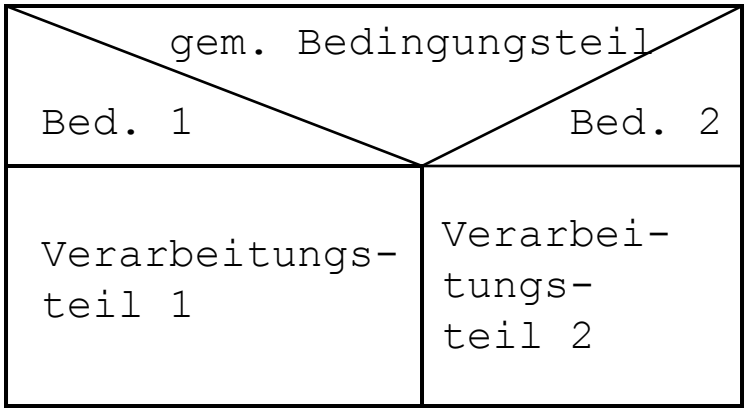


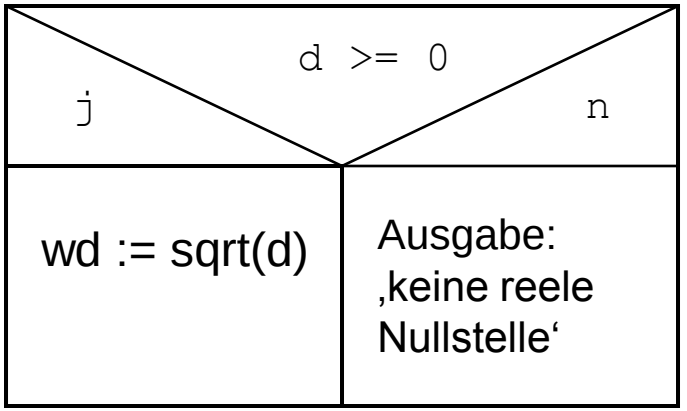


Strukturblöcke für Algorithmen – Alternative (Selektionsblöcke)

Einfache Alternative – vollständige Alternative
(dyadic selective)

Dieses Programmkonstrukt besteht aus zwei **Verarbeitungsteilen** und einem **Steuerungsteil mit einer Bedingung**. Die Bedingung bestimmt, welcher der beiden Verarbeitungsteile ausgeführt wird, wenn das Programmkonstrukt durchlaufen wird.



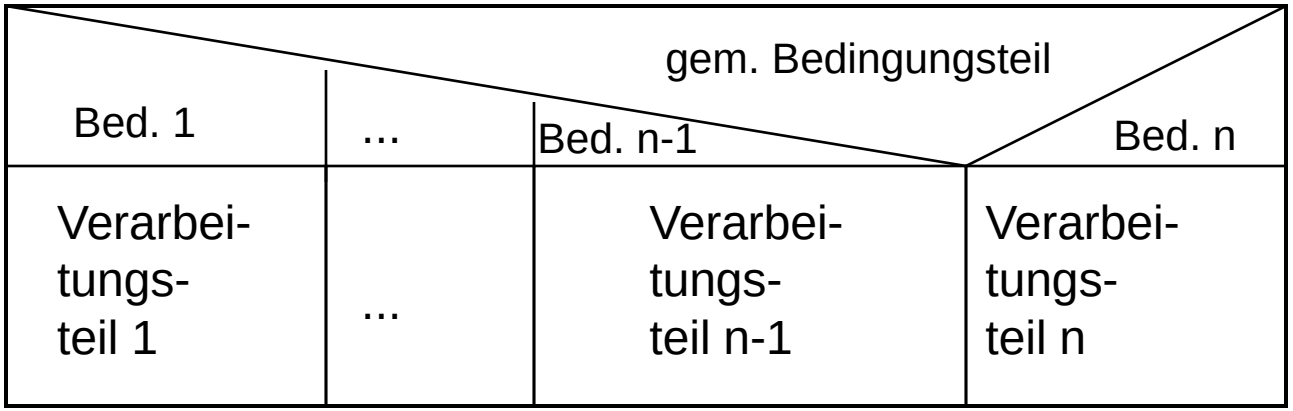


Object-Pascal: IF-Anweisung

Strukturblöcke für Algoríthmen – Alternative (Selektionsblöcke)

Mehrfache Alternative – Fallunterscheidung
(multiple exclusiv selective)

Dieses Programmkonstrukt besteht aus **mehreren Verarbeitungsteilen** und **einem Steuerungsteil mit der gleichen Anzahl einander ausschließender Bedingungen**. Der Steuerungsteil gibt mit diesen Bedingungen an, welcher der Verarbeitungsteile ausgeführt wird, wenn das Programmkonstrukt durchlaufen wird.



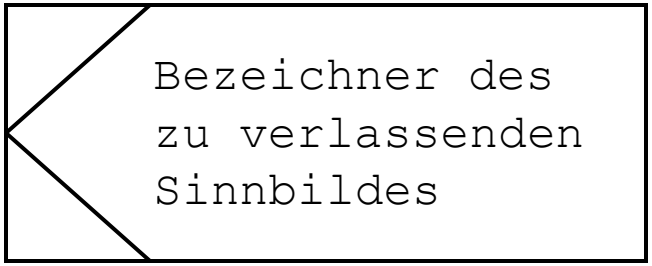
Object-Pascal: CASE-Anweisung

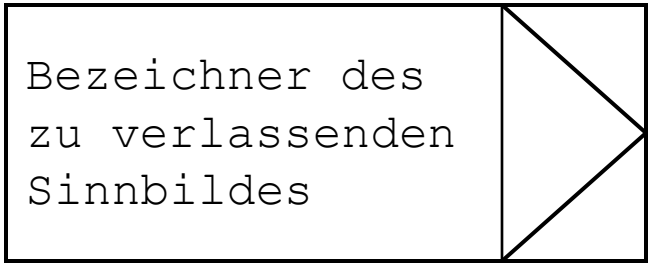
Strukturblöcke für Algorithmen - Abbruchanweisung

Die Ablaufsteuerung der Verarbeitungsteile in einem Programm wird durch die Steuerungsteile und die Schachtelung der Programmkonstrukte dargestellt. Darüber hinaus kann die Ausführung eines Programmkonstrukts durch eine Abbruchanweisung vorzeitig beendet werden. Die Abbruchanweisung ist eine den Programmablauf steuernde elementare Anweisung, die im Verarbeitungsteil jedes Programmkonstrukts enthalten sein kann.

Die Ausführung einer Abbruchanweisung bedeutet das unmittelbare beenden des bezeichneten Programmkonstrukts und damit auch aller darin enthaltenen eingeschachtelten Programmkonstrukte.

Darstellungsformen:





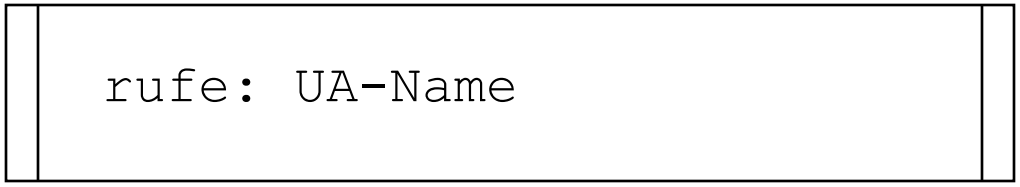
Strukturblöcke für Algorithmen – Regeln für abgeleitete Programmkonstrukte

Von der DIN-EN-Norm abgeleitete, geeignete Programmkonstrukte dürfen ebenfalls verwendet werden.
Bei der Kombination dieser Programmkonstrukte mit denen der Norm muss auf deren Verträglichkeit geachtet werden.

Bei der Auswahl zusätzlicher Programmkonstrukte besteht Normkonformität, wenn sie von gleicher funktionaler und konstruktiver Art sind und in der Weise angewendet werden, wie es diese Norm vorsieht.

Strukturblöcke für Algorithmen – Unteralgorithmen

Zur Strukturierung von Programmen und zur Mehrfachverwendung von Programmkonstrukten ist die Verwendung von Unteralgorithmen sinnvoll. Dazu stehen in den Programmiersprachen in der Regel die Sprachelemente Prozedur und Funktion zur Verfügung.
In der Schulinformatik wird für den Aufruf von Unteralgorithmen ein gesondertes Struktogrammsymbol verwendet.



Sprung-Anweisungen

Kontroverse Grundaussagen

1. Die GOTO-Anweisung ist **unverzichtbar** für Effizienz und eine gute Struktur.
2. Die Anwendung der GOTO-Anweisung in sorgfältig eingeschränkten Fällen **kann nützlich sein**.
3. Die GOTO-Anweisung ist ein Anachronismus, der in Zukunft aus allen Programmiersprachen **beseitigt werden sollte**.

Sprunganweisungen werden bei der Algorithmenbeschreibung durch Struktogramme nicht unterstützt.

Fragen eines nachdenklichen Programmierers

1. Was ist das gemeinsame, dass der großen Vielfalt von Computern zu Grunde liegt?
2. Wie könnte ein minimaler Computer aussehen, der ohne Rücksicht auf technische Grenzen wie Zeitverbrauch und Speicherkapazität alle Algorithmen im Prinzip auszuführen imstande ist?
3. Gibt es für jedes Problem, das sich mit Mitteln von Informatik und Mathematik beschreiben lässt, einen Lösungsalgorithmus, oder gibt es Probleme, für die sich nachweislich kein Algorithmus angeben lässt?
4. Wie kann man Algorithmen hinsichtlich ihrer Qualität, insbesondere ihrer Laufzeit miteinander vergleichen, ohne sich dabei auf eine bestimmte Maschine zu beziehen?
5. Kann man die Richtigkeit von Programmen nur durch Testen feststellen oder kann man sie, ähnlich einem mathematischen Satz, vielleicht beweisen?

Hilbertsches Entscheidungsproblem

Zitat Hilbert(1862-1943):

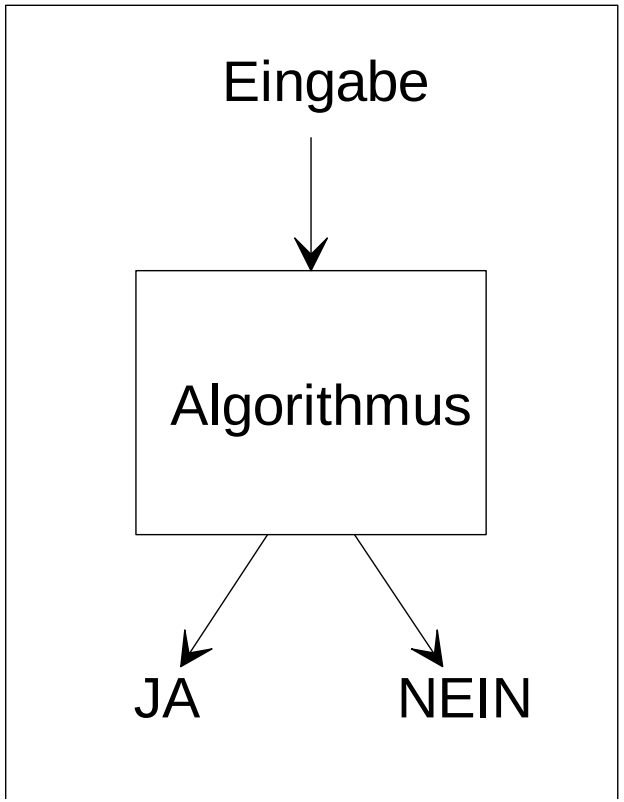
„... ich meine, die Überzeugung, daß ein jedes bestimmtes mathematisches Problem einer strengen Erledigung notwendigerweise fähig sein müsse, sei es, daß es gelingt, die Beantwortung einer gestellten Frage zu geben, sei es, daß die Unmöglichkeit einer Lösung und damit die Notwendigkeit des Mißlingens aller Versuche dargetan wird. ...Diese Überzeugung von der Lösbarkeit eines jeden Problems ist uns ein kräftiger Ansporn während der Arbeit; wir hören in uns den steten Zuruf: *Da ist das Problem, suche die Lösung. Du kannst sie durch reines Denken finden, denn in der Mathematik gibt es keinen Ignorabimus*.“

$\Rightarrow$ jedes wohldefinierte Problem ist einfach durch eine Algorithmenausführung lösbar

igno|ra|mus et igno|ra|bi|mus *[lat. "wir wissen (es) nicht u. werden (es auch) nicht wissen"]: Schlagwort für die Unlösbarkeit der Welträtsel*

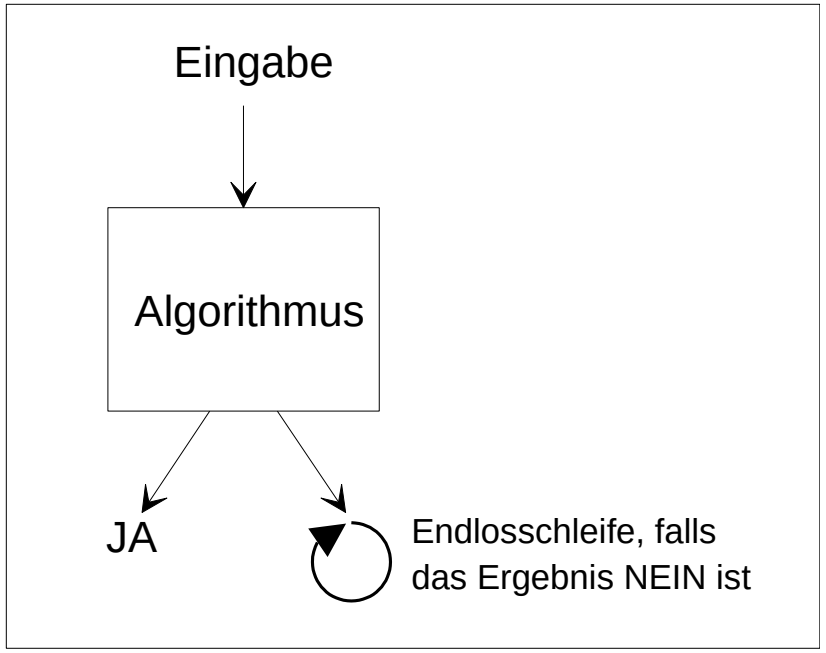
Berechenbarkeit

Mit Hilfe der Berechenbarkeit soll geklärt werden, ob ein Algorithmus mit Hilfe eines Computers berechnet werden kann. Probleme können berechenbar, partiell berechenbar oder nicht partiell berechenbar sein.
Ist ein **Problem berechenbar**, so gibt es einen Algorithmus, der für **jede** Eingabe korrekt **„JA“** oder **„NEIN“** antwortet.



Partielle Berechenbarkeit

Ist ein Problem nur **partiell berechenbar**, so gibt es einen Algorithmus, der mit „JA“ antwortet, falls die Antwort korrekt ist, jedoch eine **Endlosschleife ausführt, falls die Antwort „NEIN“** lautet.
Ist ein Problem **nicht berechenbar,** dann gibt es keinen Algorithmus, der immer korrekt mit „JA“ oder „NEIN“ antwortet. Es gibt keine Methode, die Korrektheit einer vorgeschlagenen Antwort zu beweisen.



Berechenbarkeitsmodelle

- **Allgemein-rekursive Funktionen (Gödel, Herbrand, Kleene, 1936)**
- **Lambda-Kalkül (Church, 1936)**
- **Minimum- und partiell-rekursive Funktionen (Gödel, Kleene, 1936)**
- **Turingmaschine (Turing, 1936)**
- **Post‘sche kanonische System (Post, 1943)**
- **Markov‘sche Algorithmen (Markov, 1951)**
- **Registermaschine (Sheperdson, Sturgis, 1963)**
- **Random-Access-Maschinen (1964)**

/HORN03/

Berechenbarkeitsmodelle

Turingmaschine (1936)

Bestandteile einer Turing-Maschine:

- Mehrere ($k \geq 1$) beidseitig unendliche, in Felder unterteilte Bänder. In jedem Feld steht ein Symbol aus einem unendlichen Bandalphabet. Es gibt ein Leerzeichen aus dem Alphabet, das das leere Element kennzeichnet.
- Ein Lese- und Schreibkopf für jedes Band, der sich von Feld zu Feld bewegen und bewegen und den Inhalt des jeweils betrachteten Feldes lesen und ändern kann.
- Eine Steuereinheit, die sich in einem der Zustände aus einer endlichen Zustandsmenge Z befindet, Informationen über die von den Köpfen gelesenen Symbole bekommt und deren Aktivitäten steuert. Es sind zwei Zustände besonders gekennzeichnet, der Startzustand und der Stoppzustand

Computermodelle der Berechenbarkeit

Random-Access-Maschinen (1964)

Die *Random-Access-Maschinen* sind ein mathematisches Modell für reale Rechner.

Eine RAM besteht aus einer Steuereinheit, aus einem Befehlsregister BR und und unendlich vielen durchnummerierten (Daten-)Registern R0, R1, R2.... . Die Nummer i des Registers Ri wird als seine Adresse bezeichnet.

Jedes Register enthält eine natürliche Zahl. Die Steuereinheit verfügt über ein Programm, das aus einer Liste von nummerierten Befehlen besteht.

Computermodelle der Berechenbarkeit

Programmiersprache PASCALLI

PASCALLI ist eine einfache Variante der Programmiersprache Pascal, die aber die volle Berechnungsstärke besitzt. Es sind alle Elemente enthalten, die die Fähigkeit ausmachen, Berechnungen auszuführen.

Die Beschränkungen der Elementgrößen und Felder wurden aufgehoben.

RAM $\subseteq$ PASCALLI

Komplexität und Effizienz

Als **Komplexität** bezeichnet man den für die Ausführung der Berechnung erforderlichen Aufwand an Rechenzeit und Speicherplatz. Die Abschätzung der Komplexität von Algorithmen ist Gegenstand der Komplexitätstheorie, einem Zweig der Theoretischen Informatik.
Laufzeit und Speicherplatzbedarf eines Algorithmus hängen in der Regel von der Größe der Eingabe ab.

„Ein Algorithmus heißt **effizient**, wenn er ein vorgegebenes Problem mit möglichst geringem Ressourcenaufwand (Zeit, Speicher) löst.“

/WAGENKN03/

O-Notation

In der Regel werden **Größenordnungen** der Laufzeit- und Speicherplatzfunktionen in Abhängigkeit von der Größe der Eingabedaten bestimmt. Um diese Größenordnungen (Wachstumsordnungen von Funktionen) auszudrücken und zu bestimmen, wird eine spezielle Notation verwendet. Diese wird als **Groß-Oh-** und **Groß-Omega-Notation** bezeichnet.

„T(N) ist von der Größenordnung N“ := (Wachstum der Funktion nach oben)

==> $T(N) = O(N)$ bzw. $T(N) \in O(N)$, $f \in O(g)$

$O(f) = \{ g | \exists c_1 > 0 : \exists c_2 > 0 : \forall N \in Z^+ : g(N) \leq c_1 * f(N) + c_2 \}$

Wachstum der Funktion nach unten: (untere Schranke für Laufzeit und Speicherplatzbedarf eines Algorithmus:

$f \in \Omega(g)$ bzw. $f = \Omega(g)$

Gilt: $f \in O(g)$ und $f \in \Omega(g)$ ==> $f = \theta(g)$

Ausführungszeiten von Algorithmen

Größe n der Eingabedaten	$\log_2 n$ Mikrosek.	n Mikrosek.	n^2 Mikrosek.	2^n Mikrosek.
10	0.000003 Sekunden	0.00001 Sekunden	0.0001 Sekunden	0.001 Sekunden
100	0.000007 Sekunden	0.0001 Sekunden	0.01 Sekunden	10^{14} Jahrhunderte
1000	0.00001 Sekunden	0.001 Sekunden	1 Sekunde	
10000	0.000013 Sekunden	0.01 Sekunden	1.7 Minuten	
100000	0.000017 Sekunden	0.1 Sekunden	2.8 Stunden	

Komplexität

$\log_2 n$ - Logarithmische Komplexität

Die Laufzeit wächst erheblich schwächer als die Zahl der Eingabedaten n.

***n* - Lineare Komplexität**

Dabei ist die Ausführungszeit proportional zur Anzahl der Eingabedaten.

***n* $\log_2 n$ - Leicht überlineare Komplexität**

Diese Komplexität ist nicht wesentlich schlechter als die lineare Komplexität, da der Logarithmus von n klein gegen n ist.

n^c - Polynomiale Komplexität

Algorithmen, deren asymptotisches Verhalten n, n^2, oder allgemein **n^c** für eine Konstante c ist, werden als **POLYNOMIALE ALGORITHMEN** bezeichnet.

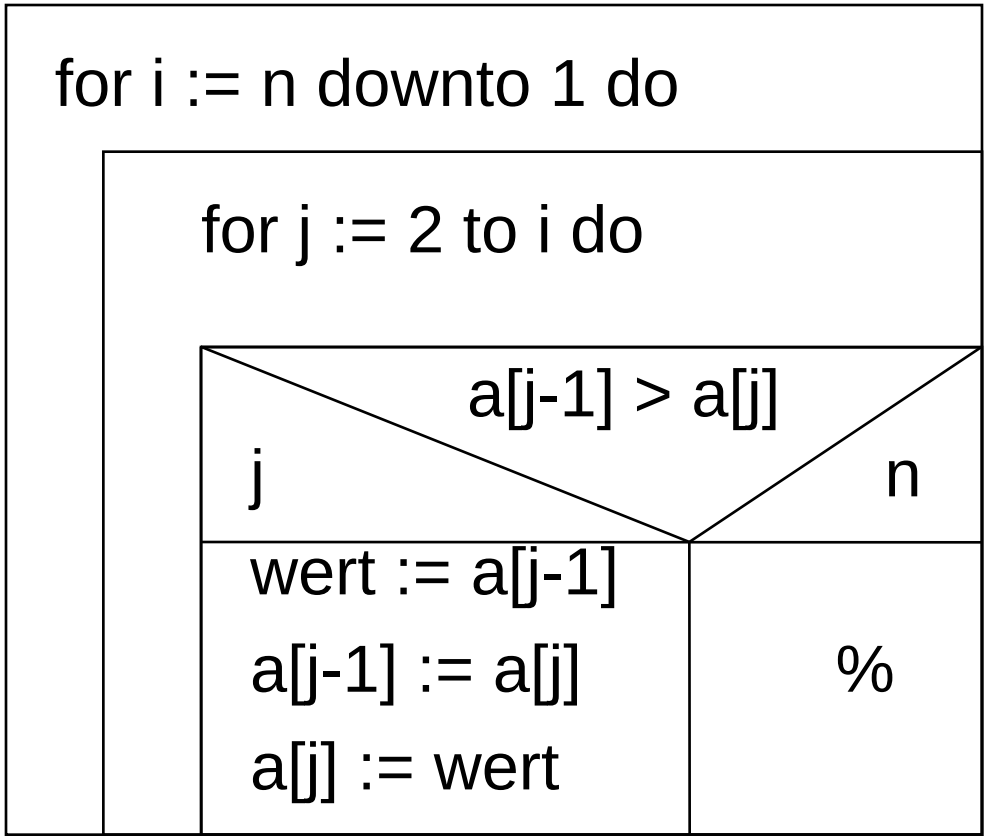
c^n - Exponentielle Komplexität

Algorithmen, deren asymptotisches Verhalten 2^n, oder allgemein **c^n** mit einer Konstanten c ist, werden als **EXPONENTIELLE ALGORITHMEN** bezeichnet.

Ausführungszeiten von Sortieralgorithmen

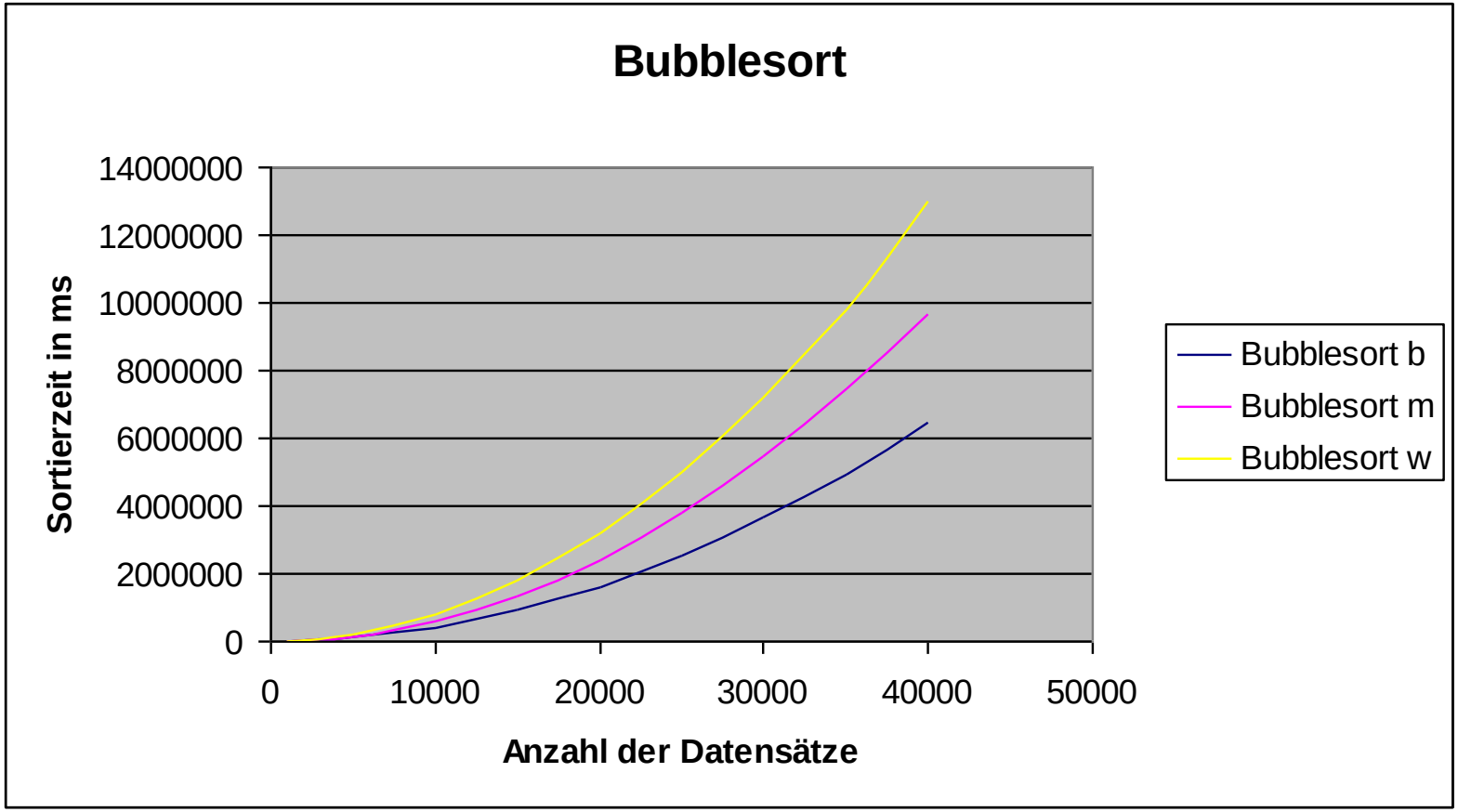
Datensätze	Selectionsort	Insertionsort	Bubblesort	Distribution-Counting	Quicksort
1000	0:00:05,062	0:00:02,031	0:00:06,000	0:00:00,031	0:00:00,109
5000	0:02:05,000	0:00:50,532	0:02:31,359	0:00:00,156	0:00:00,656
10000	0:08:27,688	0:03:25,547	0:10:09,953	0:00:00,328	0:00:01,328
15000	0:18:43,734	0:07:27,140	0:22:29,390	0:00:00,469	0:00:02,094
20000	0:32:06,453	0:13:22,734	0:39:54,704	0:00:00,641	0:00:02,906
25000	0:50:18,562	0:20:54,781	1:03:06,422	0:00:00,797	0:00:03,812
30000	1:12:34,843	0:30:26,828	1:31:00,782	0:00:00,953	0:00:04,563
35000	1:39:36,500	0:41:13,671	2:04:07,344	0:00:01,109	0:00:05,469
40000	2:09:00,422	0:53:57,234	2:40:47,453	0:00:01,297	0:00:06,407
45000				0:00:01,360	0:00:07,156
50000				0:00:01,594	0:00:08,016
55000				0:00:01,734	0:00:08,938
60000				0:00:01,890	0:00:10,344
65000				0:00:02,078	0:00:10,954
70000				0:00:02,219	0:00:11,625
75000				0:00:02,359	0:00:12,875
80000				0:00:02,532	0:00:13,891
85000				0:00:02,687	0:00:14,453

Sortierverfahren Bubble-Sort

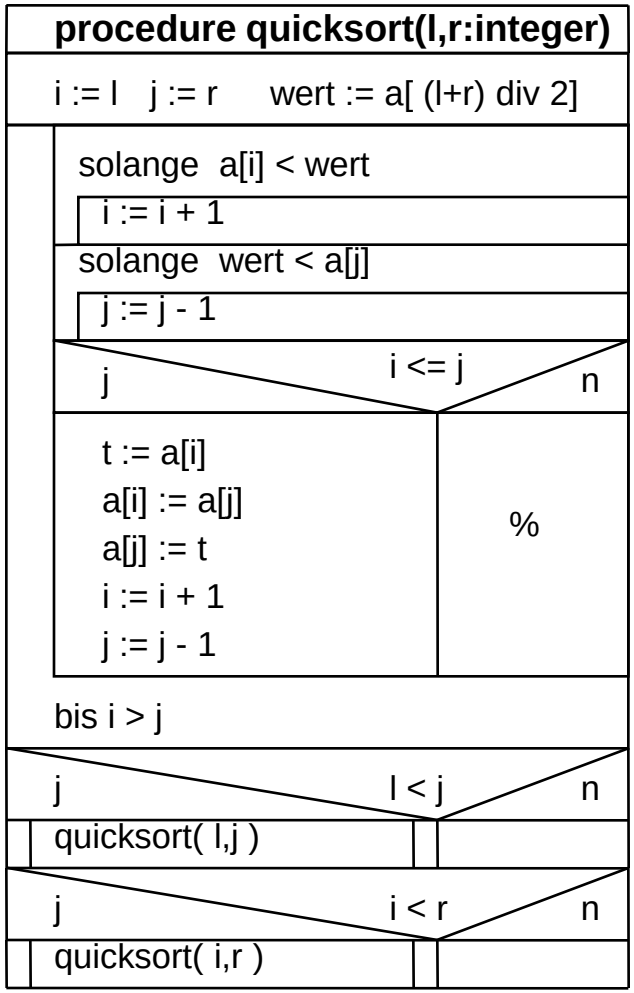


Bubble-Sort benötigt im Durchschnitt und im ungünstigsten Fall etwa
$N^2/2$ Vergleiche und
$N^2/2$ Austausch-operationen

Bubble Sort

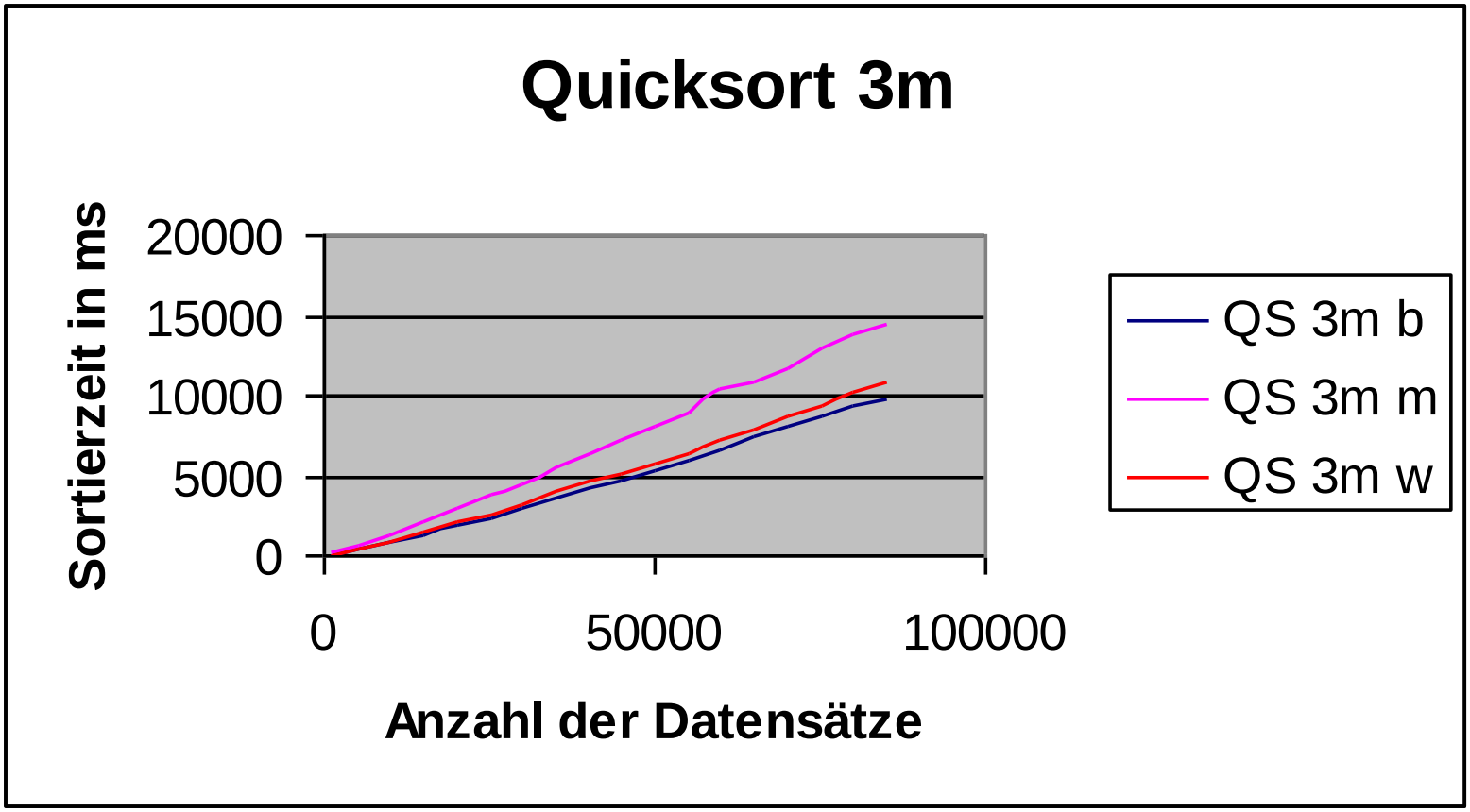

Bubblesort
Sortierzeit in ms
14000000
12000000
10000000
8000000
6000000
4000000
2000000
0
0
10000
20000
30000
40000
50000
Anzahl der Datensätze
Bubblesort b
Bubblesort m
Bubblesort w

Sortierverfahren Quick-Sort



Quich-Sort benötigt im Durchschnitt etwa
N log N Operationen und
im schlechtesten Fall
N² Austauschoperationen

Quicksort - 3 - Median - Strategie



Komplexitätsbetrachtungen

worst-case complexity

die Zeit, die maximal zur Ausführung eines Algorithmus benötigt wird.

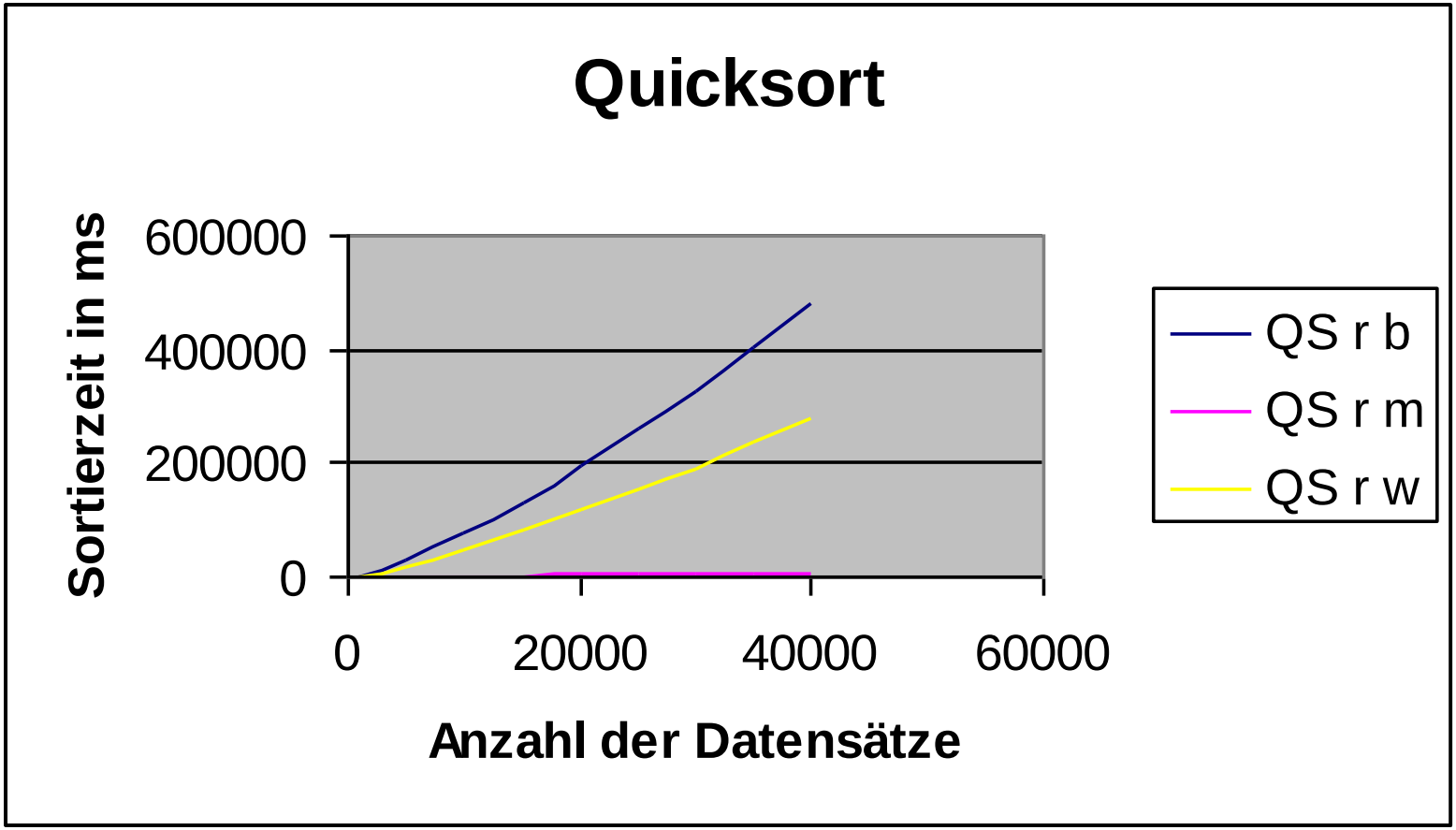
average-case complexity

durchschnittlicher Betriebsmittelbedarf für alle Eingaben. Dieser wird als Komplexität des Algorithmus im Durchschnittsfall bezeichnet.

best-case complexity

Betriebsmittelbedarf im günstigsten Fall.

Quicksort


Quicksort
Sortierzeit in ms
600000
400000
200000
0
0
20000
40000
60000
Anzahl der Datensätze
QS r b
QS r m
QS r w

Laufzeitermittlung von Programmen

Bewertungsprogramme - benchmarks

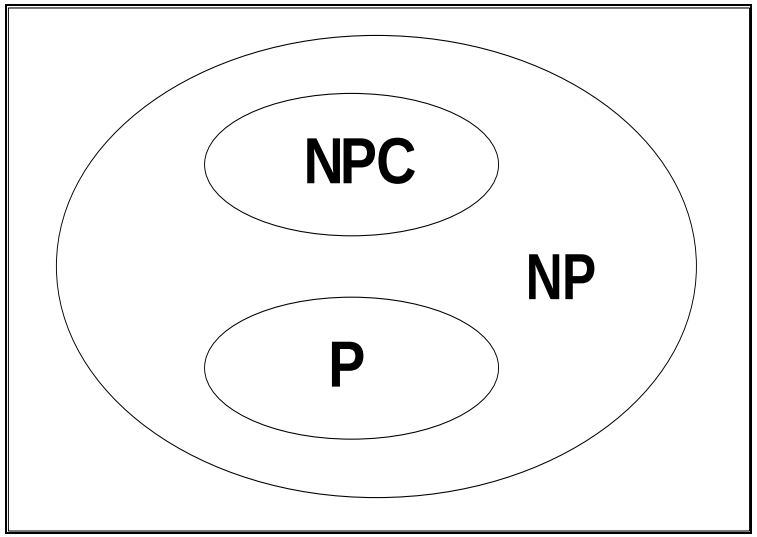
Benchmarks (Maßstab) sind Programme, die eine Sammlung typischer Eingaben bzw. Aufträge sind.

Analyse von Programmen

Programme werden analysiert um die Laufzeit allein aus dem Algorithmus und unabhängig von einem konkreten Computer abzuschätzen. Eine Möglichkeit der Beschreibung ist die O-Notation.

NP-Vollständigkeit von Problemen

- P entspricht der Familie der Probleme, die in polynomialer Zeit lösbar sind.
- NP eine Klasse von Problemen, für die bisher kein Polynomialzeitalgorithmus gefunden werden konnte. Die Überprüfung der Korrektheit der Lösung kann in Polynomialzeit erfolgen.
- NPC ist die Familie der NP-vollständigen (complete) Probleme, das sind die schwierigsten Probleme in NP.



Korrektheit von Programmen

Ein Programm gilt in der Regel als korrekt, wenn es die vorgegebene Aufgabenstellung erfüllt. Voraussetzung dafür ist, dass auch die Aufgabenstellung korrekt ist. Die Aufgabenstellung muss konsistent und vollständig in sich und im Bezug auf die Umwelt sein.

Weiterhin ist es notwendig, eine **Eingabespezifikation** und eine **Ausgabespezifikation** zu definieren. Die Eingabespezifikation ist eine vollständige Beschreibung der Situation beim Beginn des Programmablaufes.

Die Ausgabespezifikation ist die entsprechende Beschreibung am Ende des Programmablaufs. Damit wird festgelegt, welche Werte der Ausgangsparameter sich als Funktion der Eingangsparameter ergeben müssen.

Der Nachweis der Korrektheit eines Programms bedeutet, dass das Programm die Ausgangsparameter richtig berechnet und das es für alle zulässigen Eingangsparameter nach endlich vielen Schritten anhält.

Korrektheitskriterien

Der **Beweis** dient zur Bestätigung der Korrektheit eines Programms für alle zugelassenen Testdaten. Damit wird eine Aussage über das Gesamtverhalten angestrebt.

Eine Form des Beweises ist der **informelle Korrektheitsbeweis**, der besagt, dass das Programm verstanden wurde. Die Niederschrift dessen wird auch als Dokumentation bezeichnet.

Fehler

Die **Korrektheit eines Algorithmus** kann sinnvoll nur in Verbindung mit Annahmen über den Zweck des Algorithmus betrachtet werden. Das bedeutet, dass ein Algorithmus korrekt bezüglich seiner Spezifikation ist, wenn der Algorithmus festgelegte Ergebnisse über Eingabedaten liefert, wie sie in der Spezifikation vorgegeben sind.

Liefert ein Algorithmus, immer wenn er endet, das erwartete Ergebnis, so wird dieser Algorithmus als **partiell korrekt** bezeichnet. Es gibt jedoch keine Sicherheit dafür, dass dieser Algorithmus endet. Kann man zeigen, dass ein Algorithmus endet (terminiert ist), so wird dieser Algorithmus als **total korrekt** bezeichnet.

Fehler

Eine dritte wichtige Programmeigenschaft ist die **Ausführbarkeit**. Unter Zuhilfenahme der Komplexitätstheorie kann man Schranken des Betriebsmittelverbrauchs durch einen Algorithmus feststellen.

Nachweis: Bestätigung der Korrektheit des Programms für alle zugelassenen Eingabedaten (Gesamtverhalten des Programms).

Programmtest

Programmtest (Debugging)

Beim **Test** wird das Programm mit einer bestimmten Datenmenge, den Testdaten, ausgeführt. Die Ausführung erfolgt schrittweise. Es wird gezeigt, dass der Algorithmus für die gewählten Testdaten korrekt arbeitet.

Trockentest

Die Ausführung des Algorithmus kann auf einem Computer oder manuell erfolgen. Das Programm wird mit Testdaten Schritt für Schritt durchlaufen.

➔ Welche Ergebnisse werden mit welchen Testdaten erzeugt?

Nachweis: Das Programm arbeitet für die gewählten Testdaten korrekt.

Programmtest

"Durch Testen kann man stets nur die Anwesenheit, nie aber die Abwesenheit von Fehlern beweisen."

/Edsger Wybe Dijkstra ;The Humble Programmer, ACM Turing Lecture 1972/

Kontrollfragen I

1. Erklären Sie den Algorithmusbegriff mit Hilfe einer geeigneten Definition. Geben Sie die Eigenschaften von Algorithmen an und erläutern Sie diese kurz. Nennen Sie drei übliche Darstellungsformen für Algorithmen.
2. Erklären Sie für einen Algorithmus die Begriffe „berechenbar“ und „partiell berechenbar“. Nennen Sie mindestens zwei Modelle zur Berechenbarkeit von Algorithmen aus der theoretischen Informatik. Bewerten Sie die Aussage: „jedes wohldefinierte Problem ist einfach durch eine Algorithmenausführung lösbar“. Begründen Sie Ihre Entscheidung an einem Beispiel.
3. Erläutern Sie die Komplexität von Algorithmen. Gehen Sie dabei auf die unterschiedlichen Zeitkomplexitäten ein. Wie kann die Komplexität von Algorithmen ermittelt werden? Nennen Sie jeweils einen Algorithmus, dessen Bearbeitungszeit im mittleren Fall logarithmisch, linear, leicht überlinear bzw. quadratisch steigt.

Kontrollfragen II

4. Wann wird ein Algorithmus als effizient bezeichnet?
5. Erläutern Sie den Begriff Komplexität eines Algorithmus. Von welchen Größen ist die Komplexität eines Algorithmus abhängig?
6. Was versteht man unter Korrektheit von Programmen? Wie kann diese für einfache und komplexe Programme nachgewiesen werden?

Literatur:

/DUDEN03/ Basiswissen Schule – Informatik Abitur
PAETEC Verlag für Bildungsmedien Berlin, 2003
ISBN 3-89818-065-4

/HORN03/ Horn, Christian; Immo O. Kerner; Forbrig, Peter
Lehr- und Übungsbuch Informatik
Band 1: Grundlagen und Überblick
Fachbuchverlag Leipzig; 2003; ISBN 3-446-22543-9

/HORN00/ Horn, Christian; Immo O. Kerner
Lehr- und Übungsbuch Informatik
Band 3: Praktische Informatik
Fachbuchverlag Leipzig; 1997; ISBN 3-446-18696-4

/WAGNER94/ Wagner, Klaus W.
Theoretische Informatik - Grundlagen und Modelle
Springer-Verlag, Berlin, Heidelberg, New York, 1994
ISBN: 3-540-58139-1

/WAGENKN03/ Wagenknecht, Christian
Algorithmen und Komplexität
Fachbuchverlag Leipzig, 2003
ISBN: 3-446-22314-2