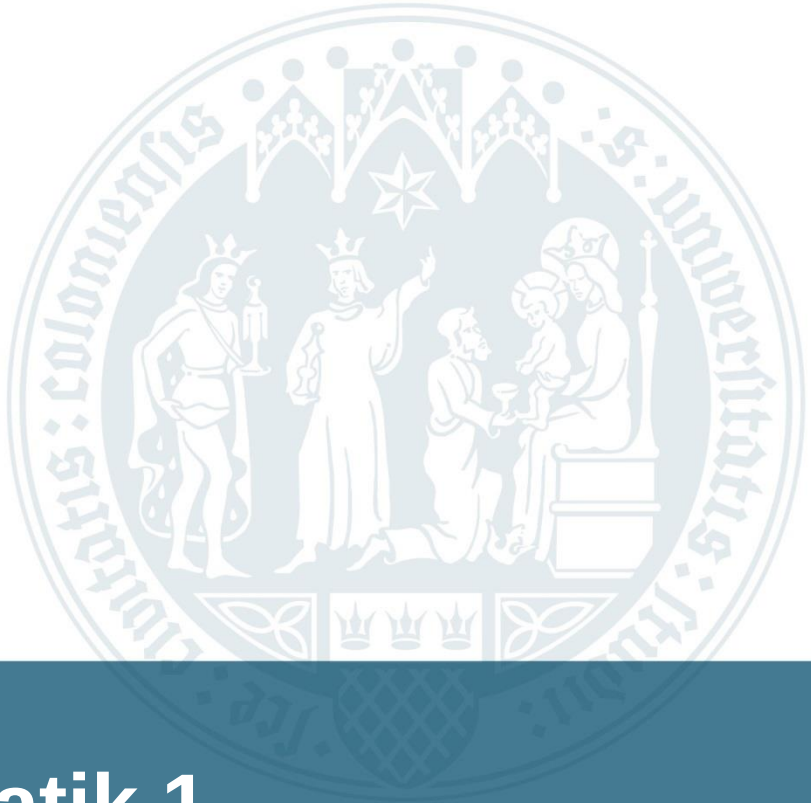
Universität zu Köln



Grundzüge der Informatik 1

Vorlesung 2

Überblick

Überblick

- Speicheraufbau und Datentypen
- Pseudocodebefehle und Laufzeit
- Laufzeitanalyse

Speicheraufbau und Datentypen

Elementare Datentypen

- ganze Zahlen
- reellwertige Zahlen
- Zeichen
- Zeiger

Speicher

- Beliebig viele Speicherzellen
- Speicherzellen sind mit 1 beginnend aufsteigend durchnummeriert
- Elementare Datentypen benötigen eine Speicherzelle

Speicheraufbau und Datentypen

```
Einfach()
1.  X=10
2.  Y= 5
3.  X= X*Y
4.  Y= X
```

Elementare Datentypen

- Wenn eine Variable zum ersten mal im Algorithmus auftaucht, wird für sie eine bisher unbelegte Speicherzelle reserviert
- Immer wenn sich die Variable ändert wird die zugehörige Speicherzelle entsprechend geändert
- Wir geben den Datentyp nicht explizit an

Speicheraufbau

Was ist ein Zeiger?

- Ein Zeiger ist eine ganze Zahl, die eine Speicherzelle bezeichnet
- Manchmal wird ein Zeiger auch als Referenz bezeichnet
- Zeiger können den Wert 0 bzw. NIL enthalten. Dies kennzeichnet den Fall, dass der Zeiger aktuell auf kein Objekt zeigt
- Zeiger dienen u.a. dazu, Felder oder Verbundobjekte zu referenzieren

Speicheraufbau und Datentypen

```
Einfach2()
1. A = new array[1..5]
2. A[4]=5
3. B=A
4. B[2] =3
5. output << A[2]
```

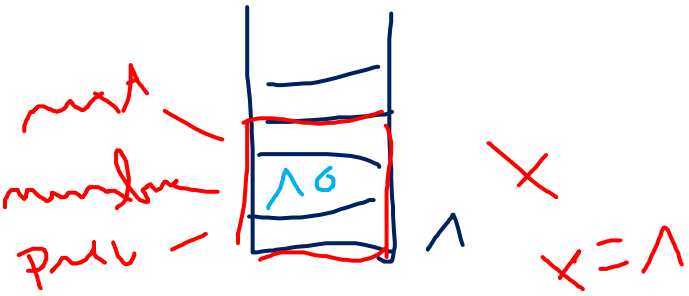
Felder

- Felder sind zusammenhängende Speicherbereiche, die denselben (elementaren) Datentyp enthalten
- Felder werden mit dem Befehl **new array** angelegt
- **new array**[1..n] reserviert einen zusammenhängenden Speicherblock der Größe n
- **new array** gibt einen Zeiger zurück, der die Nummer der ersten Zelle des Speicherblocks enthält

Speicheraufbau und Datentypen

```
Einfach3()
1.  x = new list_item
2.  number[x] = 10
```

```
Verbund list_item:
    previous
    number
    next
```



Verbunddaten

- Elementare Datentypen können als Verbund organisiert werden
- Der Verbund belegt dann wie ein Feld einen zusammenhängende Speicher-bereich
- Neue Verbundobjekte werden mit dem Befehl **new <Verbundtyp>** angelegt
- **new <Verbundtyp>** reserviert einen zusammenhängenden Speicherblock dessen Größe der Anzahl der Elemente des Verbunds entspricht
- **new <Verbundtyp>** gibt einen Zeiger zurück, der die Nummer der ersten Zelle des Speicherblocks enthält

Speicheraufbau und Datentypen

```
Einfach3()
1.  x = new list_item
2.  number[x] = 10
```

```
Verbund list_item:
    previous
    number
    next
```

Verbunddaten

- Auf die einzelnen Elemente eines Verbundes BEISPIEL wird mit <element_name> [BEISPIEL] zugegriffen

Speicheraufbau und Datentypen

```
Einfach4(n)
1.  A = new array[1..n]
2.  A[1] =10
3.  A[2] =20
4.  x = A[2]
5.  A[3] = x
```

Speicherbedarf

- Der Speicherbedarf eines Algorithmus ergibt sich nun aus der Summe der belegten Speicherzellen
- Im allgemeinen kann der Speicherbedarf von der Eingabe abhängen

Speicheraufbau und Datentypen

Zusammenfassung

- Elementare Datentypen belegen eine Speicherzelle
- Der Speicherbedarf eines Feldes oder Verbundes entspricht der Anzahl seiner Elemente
- Neue Felder oder Verbundobjekte werden mit **new** erzeugt
- Der Speicherbedarf eines Algorithmus ist die Summe der belegten Speicherzellen und kann von der Eingabe abhängen

Speicheraufbau und Datentypen

Zusammenfassung

- Elementare Datentypen belegen eine Speicherzelle
- Der Speicherbedarf eines Feldes oder Verbundes entspricht der Anzahl seiner Elemente
- Neue Felder oder Verbundobjekte werden mit **new** erzeugt
- Der Speicherbedarf eines Algorithmus ist die Summe der belegten Speicherzellen und kann von der Eingabe abhängen

Bemerkung

- Unser Rechenmodell abstrahiert von vielen Details moderner Hardware wie z.B. Speicherhierarchien

Pseudocodebefehle und Laufzeiten - Vorüberlegungen

Laufzeit hängt ab von

- Hardware (z.B. Prozessor, Cache, Pipelining, …)
- Software (z.B. Betriebssystem, Programmiersprache, Compiler)

Aber

- Analyse sollte unabhängig von der ausführenden Hard- und Software sein

Pseudocodebefehle und Laufzeiten - Vorüberlegungen

Rechenmodell

- Idee: Ignoriere rechnerabhängige Konstanten
- Eine Pseudocodeoperation benötigt einen Zeitschritt
- Wird eine Instruktion im Laufe des Algorithmus k-mal ausgeführt (z.B. in Schleifen), so benötigt sie insgesamt k Zeitschritte

Formales Modell

- Random Access Machines (RAM Modell)
- Details unterscheiden sich von unserem Modell

Pseudocode und Laufzeiten

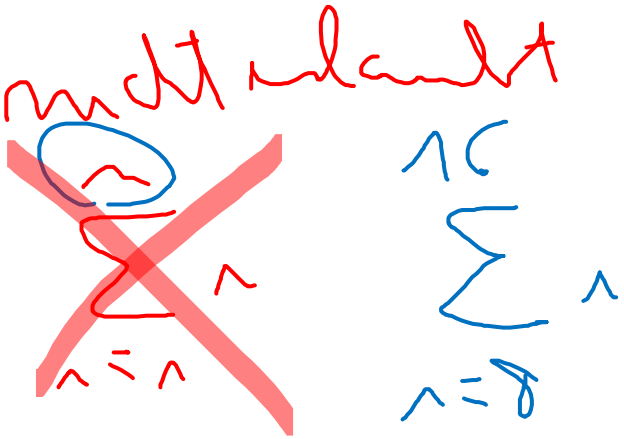
```
Beispiel1()
1.  X = 10
2.  Y = 20
3.  X = Y
4.  X = X * Y
```

Zuweisungen (Typ 1)

- Eine Zuweisung vom Typ 1 hat die Form
- <Variable1> = <Variable2>
- Variable1 und Variable2 haben dabei einen elementare Datentyp
- Es wird eine Kopie von Variable1 in Variable2 gespeichert
- Taucht Variable1 zum ersten mal auf, so wird eine Speicherzelle für Variable1 reserviert
- Variable2 muss vor der Zuweisung bereits verwendet worden sein

Pseudocode und Laufzeiten

```
Beispiel1()
1.  X = 10
2.  Y = 20
3.  X = Y
4.  X = X * Y
```



Zuweisungen (Typ 2)

- Eine Zuweisung vom Typ 2 hat die Form
- <Variable1> = <Mathematischer Ausdruck>
- Variable1 hat dabei einen elementaren Datentype
- Auf der rechten Seite steht ein konstant großer mathematischer Ausdruck
- Der mathematische Ausdruck kann Variablen verwenden, die bereits vorher verwendet wurden
- Taucht Variable1 zum ersten mal auf, so wird eine Speicherzelle für Variable1 reserviert
- Das Ergebnis der Auswertung des Ausdrucks wird in Variable1 gespeichert

Pseudocode und Laufzeiten

```
Beispiel1()
1. X = 10
2. Y = 20
3. X = Y
4. X = X * Y
```

Zuweisungen - Laufzeit

- Eine Zuweisung benötigt einen Rechenschritt

Pseudocode und Laufzeiten

```
Beispiel2()
1.  X = 10
2.  Y = 20
3.  if X>Y then output << X
4.  else output << Y
```

Bedingte Verzweigungen

- haben die Form
- **if** <logischer Ausdruck> **then** <Befehlsblock> **else** <Befehlsblock>
- Auswertung des logischen Ausdrucks erfolgt von links nach rechts
- Sobald die Auswertung klar ist, wird der restliche Ausdruck nicht weiter ausgewertet
- Der Befehlsblock nach **then** wird ausgeführt, wenn der logische Ausdruck wahr ist
- Ansonsten wird der Befehlsblock nach **else** ausgeführt

Pseudocode und Laufzeiten

```
Beispiel2()
1.  X = 10
2.  Y = 20
3.  if X>Y then output << X
4.  else output << Y
```

Bedingte Verzweigungen - Laufzeit

- Eine bedingt Verzweigung benötigt einen Rechenschritt

Pseudocode und Laufzeiten

```
Beispiel3(n)
1.  j=0
2.  for i=1 to n do
3.    j=j+i
4.  output << j
```

for-Schleifen

- haben die Form
- **for** <Zuweisung> **to** <Ende> **do**
- Beim ersten Erreichen der for-Schleife wird die Zuweisung der Zählvariable ausgeführt
- Danach wird der Schleifenrumpf solange ausgeführt, wie die Zählvariable kleiner oder gleich Ende ist
- Am Ende der Schleife wird dabei die Zählvariable automatisch um eins erhöht
- Der Schleifenrumpf ist durch Einrücken gekennzeichnet

Pseudocode und Laufzeiten

```
Beispiel3(n)
1.  j=0
2.  for i=1 to n do
3.      j=j+i
4.  output << j
```

for-Schleifen - Laufzeit

- Jede Ausführung der for-Schleife benötigt einen Zeitschritt
- Die Zeitschritte für die Durchführung des Schleifenrumpfs werden bei jedem Durchlauf benötigt

Pseudocode und Laufzeiten

```
Beispiel4(n)
1.  i=n
2.  j=0
3.  while i>0 do
4.      j=j+i
5.      i=i-1
6.  output << j
```

```
Beispiel5(n)
1.  i=n
2.  j=0
3.  repeat
4.      j=j+i
5.      i=i-1
6.  until i=0
7.  output << j
```

While und repeat-Schleifen

- Analog zur **for**-Schleife

Pseudocode und Laufzeiten

```
Beispiel6()
1.  j=100
2.  x=7+Beispiel7(j)
3.  output << j
4.  output << x
```

```
Beispiel7(j)
1.  j=j-10
2.  return j
```

j=100

x=97

Prozeduren

- Prozeduren können beliebig viele elementare Datentypen übergeben werden
- Beim Aufruf der Prozedur wird für jede übergebene Variable eine Kopie angelegt
- Die Prozedur beeinflusst also die übergebenen Variablen nicht
- Prozeduren können auch als Teil von mathematischen Ausdrücken auftreten
- Der Befehl **return** gibt einen elementaren Datentyp zurück

Pseudocode und Laufzeiten

```
Beispiel6()
1. j=100
2. x=7+Beispiel7(j)
3. output << j
4. output << x
```

```
Beispiel7(j)
1. j=j-10
2. return j
```

Prozeduren - Laufzeit

- Der Aufruf einer Prozedur kostet einen Zeitschritt plus die Zeit für die Durchführung der Prozedur
- Wird eine Prozedur innerhalb einer Zuweisung aufgerufen, so benötigt die Zuweisung einen Zeitschritt plus die Zeit für die Durchführung der Prozedur

Pseudocode und Laufzeiten

Sonstiges

- Variablen sind innerhalb Ihres Befehlsblocks sichtbar
  (und außerhalb nicht definiert)
- Die Laufzeit von **new** entspricht der Größe des reservierten Speicherbereichs
- Kommentare werden durch * gekennzeichnet

Verwendung von Pseudocode

- Pseudocode dient der Erklärung von Algorithmen
- Ziel ist Verständlichkeit nicht die Erfüllung von syntaktischen Vorgaben
- Es kann sinnvoll sein, einzelne Schritte des Algorithmus umgangssprachlich zu beschreiben
- Die Laufzeit richtet sich nach unseren Vereinbarungen zum Pseudocode

Pseudocode und Laufzeiten

Zusammenfassung

- Einzelne Pseudocode Befehle brauchen einen Rechenschritt
- Befehle, die k-mal ausgeführt werden, benötigen auch k Rechenschritte
- Tritt eine Schleife k mal in den Schleifenrumpf ein, so wird das Schleifenkonstrukt (k+1)-mal aufgerufen und der Schleifenrumpf wird k-mal ausgeführt
- Daraus ergibt sich die Anzahl der benötigten Rechenschritte
- Beim Aufruf von Prozeduren werden die übergebenen Elemente kopiert
- Pseudocode kann von diesen Definitionen abweichen, wenn es die Beschreibung des Algorithmus erleichtert

Laufzeitanalyse

Herangehensweise

- Beschreibe Laufzeit als Funktion der Eingabegröße n
- Ziel: Finde obere Schranken (Garantien) für die Laufzeit eines Algorithmus

Definition (Worst Case Analyse)

- Für jedes n definiere die Worst-Case Laufzeit T(n) durch
- T(n) = maximale Laufzeit über alle Eingaben der Größe n

Definition (Average Case Analyse)

- Für jedes n definiere die Average-Case Laufzeit T(n) durch
- T(n) = durchschnittliche Laufzeit über alle Eingaben der Größe n
- Benötigt Definition von „durchschnittlich“

Laufzeitanalyse InsertionSort

InsertionSort(A, n) Feld A der Länge n wird übergeben

1.	**for** i=2 **to** n **do**	n
2.	x = A[i]	n-1
3.	j = i -1	n-1
4.	**while** j>0 and A[j]>x **do**	$n - 1 + \sum_{i=2}^{n} t_i$
5.	A[j+1] = A[j]	$\sum_{i=2}^{n} t_i$
6.	j = j-1	$\sum_{i=2}^{n} t_i$
7.	A[j+1]=x	n-1

t_k: Anzahl Durchläufe des Schleifenrumpfs der **while**-Schleife, wenn i den Wert k hat

Laufzeitanalyse InsertionSort

InsertionSort(A, n) Feld A der Länge n wird übergeben

1.	**for** i=2 **to** n **do**	n
2.	x = A[i]	n-1
3.	j = i -1	n-1
4.	**while** j>0 and A[j]>x **do**	$n - 1 + \sum_{i=2}^{n} t_i$
5.	A[j+1] = A[j]	$\sum_{i=2}^{n} t_i$
6.	j = j-1	$\sum_{i=2}^{n} t_i$
7.	A[j+1]=x	n-1

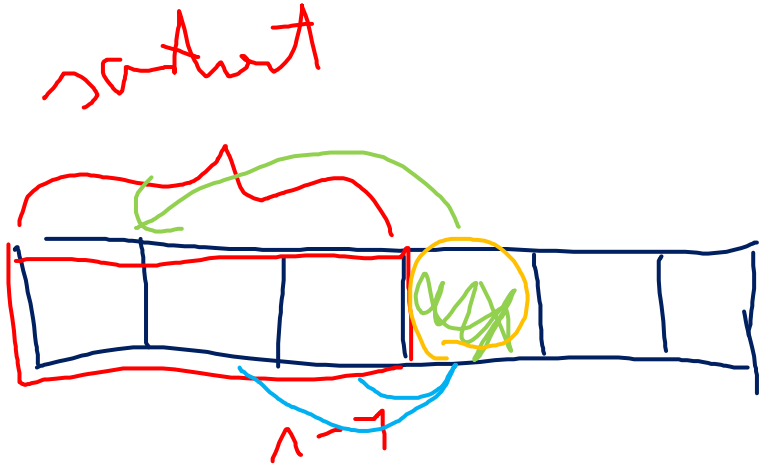
$5n + 2\sum t_i - 4$

t_k: Anzahl Durchläufe des Schleifenrumpfs der **while**-Schleife, wenn i den Wert k hat

Laufzeitanalyse InsertionSort

InsertionSort(A, n) Feld A der Länge n wird übergeben

1. **for** i=2 **to** n **do**
2. x = A[i]
3. j = i -1
4. **while** j>0 and A[j]>x **do**
5. A[j+1] = A[j]
6. j = j-1
7. A[j+1]=x



Laufzeitanalyse InsertionSort

Worst-Case Laufzeit InsertionSort

- Wir beobachten, dass t_i=i-1 gilt, wenn die Folge absteigend sortiert ist
- Es ergibt sich als Worst-Case Laufzeit
- $T(n) = 5n + 3\sum_{i=2}^{n} t_i - 4 = 2\mathrm{n} + 3\sum_{i=1}^{n} i - 4$

$$= 2n - 4 + 3\frac{n(n+1)}{2} = \frac{3n^2+7n-8}{2}$$

Laufzeitanalyse InsertionSort

Worst-Case Laufzeit InsertionSort

- Wir beobachten, dass t_i=i-1 gilt, wenn die Folge absteigend sortiert ist
- Es ergibt sich als Worst-Case Laufzeit
- $T(n) = 5n + 3\sum_{i=2}^{n} t_i - 4 = 2n + 3\sum_{i=1}^{n} i - 4$

$$= 2n - 4 + 3\frac{n(n+1)}{2} = \frac{3n^2+7n-8}{2}$$

$3\sum_{i=2}^{n} t_i = 3\sum_{i=2}^{n}(i-1) = 3\sum_{i=1}^{n-1} i = 3n - 3n + 3\cdot\sum_{i=1}^{n-1} i$

$3\cdot\sum_{i=1}^{n} i - 3n$

Laufzeitanalyse InsertionSort

Worst-Case Laufzeit InsertionSort

- Wir beobachten, dass t_i=i-1 gilt, wenn die Folge absteigend sortiert ist
- Es ergibt sich als Worst-Case Laufzeit
- $T(n) = 5n + 3\sum_{i=2}^{n} t_i - 4 = 2n + 3\sum_{i=1}^{n} i - 4$

$$= 2n - 4 + 3\frac{n(n+1)}{2} = \frac{3n^2+7n-8}{2}$$

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

$$\frac{3n^2+3n-8+4n}{2}$$

Zusammenfassung

Grundlagen der Analyse von Algorithmen

- Speicherbedarf: Eine Speicherzelle pro Variable
- Laufzeit: Ein Rechenschritt pro Pseudocodeinstruktion
- Bei Schleifen werden Pseudocodeinstruktionen mehrfach gezählt
- Worst Case Laufzeit für Eingabegröße n: Maximale Laufzeit über alle Eingaben der Größe n
- Schauen uns Laufzeitverhalten als Funktion von n an

Worst Case Speicherplatz

- Analog zu Worst Case Laufzeit