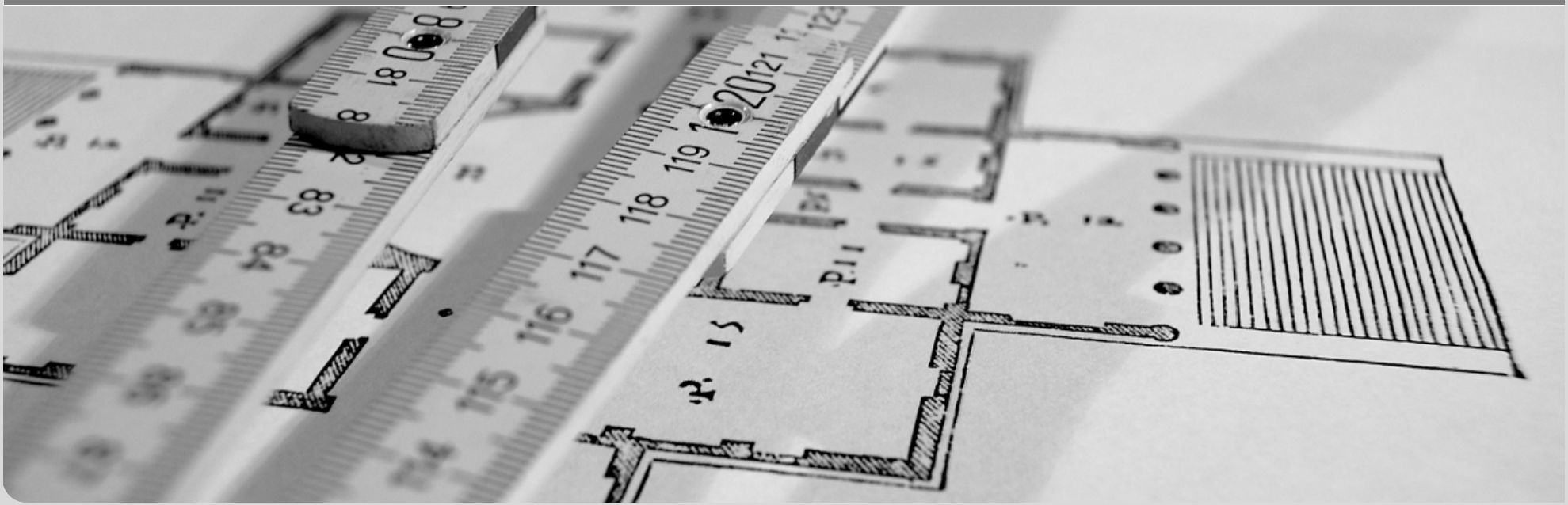
Karlsruher Institut für Technologie

Tutorium 1:
Laufzeitverhalten, O-Kalkül und Invarianten

Holger Ebhart | 22. April 2015

TUTORIUM ZUR VORLESUNG ALGORITHMEN I IM SS15



Gliederung

Organisatorisches

- Mail: **holger.ebhart@ira.uka.de**
- Folien auf github: **github.com/holger-e/Algorithmen-I-SS15**

Organisatorisches

- Mail: **holger.ebhart@ira.uka.de**
- Folien auf github: **github.com/holger-e/Algorithmen-I-SS15**

Organisatorisches

- Vorlesung: montags 15:45-17:15, Audimax
- Übung/Vorlesung: mittwochs 14:00-15:30, Audimax
- Klausur: 28. September 11:00Uhr
- Mittsemesterklausur

Organisatorisches

- Vorlesung: montags 15:45-17:15, Audimax
- Übung/Vorlesung: mittwochs 14:00-15:30, Audimax
- Klausur: 28. September 11:00Uhr
- Mittsemesterklausur

Übungsblätter

- Ausgabe: Mittwochs nach der Vorlesung
- Abgabe: Freitags bis 12:45
- Abgabe zu Zweit möglich, wenn:
  - beide aus dem gleichen Tutorium sind
  - beide Namen auf dem Blatt stehen
  - nur ein Blatt abgegeben wird

Übungsblätter

- Ausgabe: Mittwochs nach der Vorlesung
- Abgabe: Freitags bis 12:45
- Abgabe zu Zweit möglich, wenn:
  - beide aus dem gleichen Tutorium sind
  - beide Namen auf dem Blatt stehen
  - nur ein Blatt abgegeben wird

Übungsblätter

Bonuspunkte

- >25% der Punkte –>1 Bonuspunkt
- >50% der Punkte –>2 Bonuspunkt
- >75% der Punkte –>3 Bonuspunkt

Style

- bei Pseudocode stets Idee dahinter **kurz** erklären
- Erklärungen und Beschreibungen sind keine Aufsätze!!!
- lesbar Schreiben
- kein Bleistift oder Rotstift benutzen
- Name, Matrikelnummer und Tutoriumsnummer auf Blätter

Eigener Pseudocode

Schreibt man selber Pseudocode, so sollte man einige Dinge beachten:

1. Es sollten Angaben gemacht werden, welche Ein- und Ausgabe der Algorithmus hat
2. Komplexere Stellen im Pseudocode sollten auskommentiert sein
3. Bei Aufsplittung in mehrere Funktionen muss angegeben sein, welches die Funktion ist, die den Algorithmus realisiert
4. Bei langen Pseudocodeabschnitten sollte eine Beschreibung in Form von Fließtext mit angegeben werden

Eigener Pseudocode

Schreibt man selber Pseudocode, so sollte man einige Dinge beachten:

1. Es sollten Angaben gemacht werden, welche Ein- und Ausgabe der Algorithmus hat
2. Komplexere Stellen im Pseudocode sollten auskommentiert sein
3. Bei Aufsplittung in mehrere Funktionen muss angegeben sein, welches die Funktion ist, die den Algorithmus realisiert
4. Bei langen Pseudocodeabschnitten sollte eine Beschreibung in Form von Fließtext mit angegeben werden

Eigener Pseudocode

Schreibt man selber Pseudocode, so sollte man einige Dinge beachten:

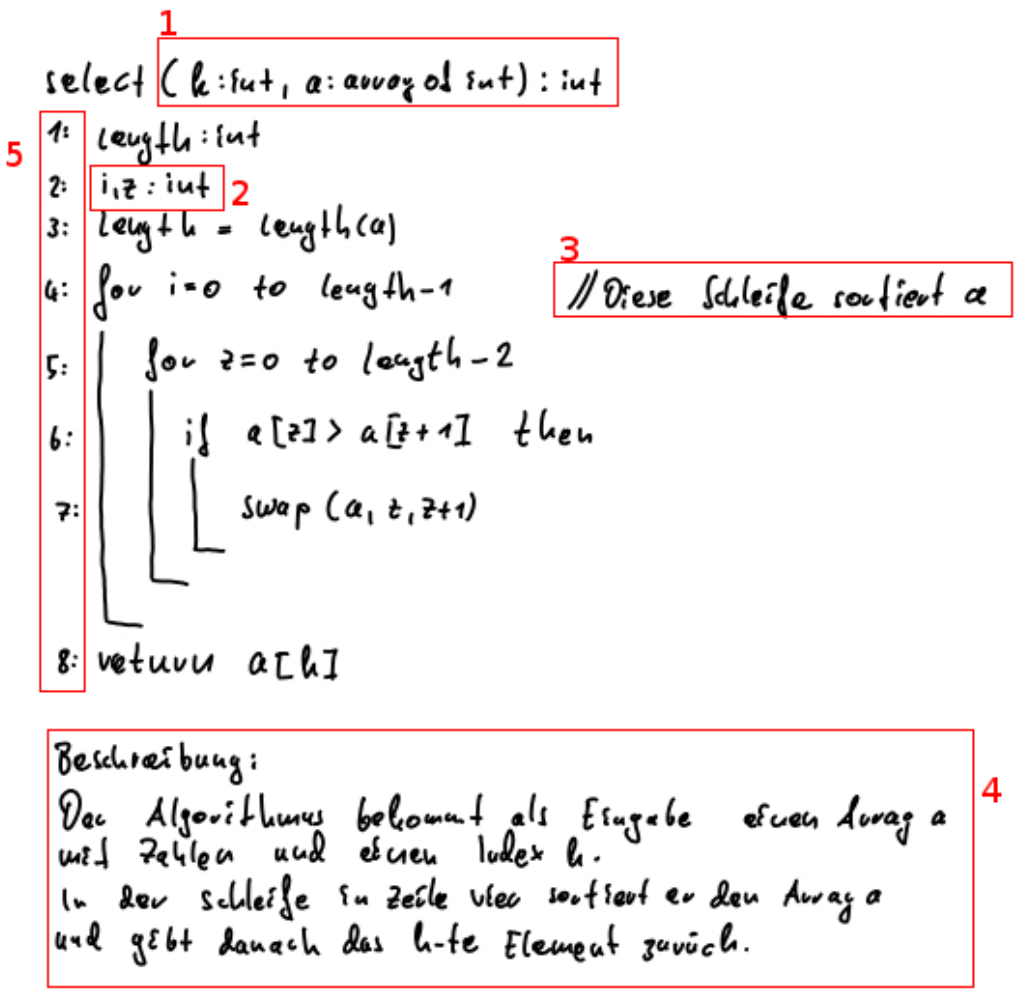
1. Es sollten Angaben gemacht werden, welche Ein- und Ausgabe der Algorithmus hat
2. Komplexere Stellen im Pseudocode sollten auskommentiert sein
3. Bei Aufsplittung in mehrere Funktionen muss angegeben sein, welches die Funktion ist, die den Algorithmus realisiert
4. Bei langen Pseudocodeabschnitten sollte eine Beschreibung in Form von Fließtext mit angegeben werden

Eigener Pseudocode

Schreibt man selber Pseudocode, so sollte man einige Dinge beachten:

1. Es sollten Angaben gemacht werden, welche Ein- und Ausgabe der Algorithmus hat
2. Komplexere Stellen im Pseudocode sollten auskommentiert sein
3. Bei Aufsplittung in mehrere Funktionen muss angegeben sein, welches die Funktion ist, die den Algorithmus realisiert
4. Bei langen Pseudocodeabschnitten sollte eine Beschreibung in Form von Fließtext mit angegeben werden

Ein Beispiel



1: Genaue Beschreibung der Eingabe und Ausgabe und deren Typen.

2: Sinnvolle Wahl von Indizes (i und j kann man meistens schlecht unterscheiden).

3: Einzelne Zeilen sind sinnvoll auskommentiert.

4: Eine kleine Beschreibung, was der Algorithmus tut, hilft beim Verständnis.

5: Wenn in der Beschreibung auf einzelne Zeilen verwiesen wird, dann sollte man die Zeilen auch durchnummerieren.

Bearbeiten der Übungsblätter

1. Abwägen ob Pseudocode oder Fließtext, wenn einem die Wahl gelassen wird
2. Mehr als zwei Seiten benötigt $\Rightarrow$ Fehler oder sehr kompliziert gelöst
3. Ist meine Lösung verständlich für andere Personen?
4. Ist meine Lösung angenehm zu lesen?

Bearbeiten der Übungsblätter

1. Abwägen ob Pseudocode oder Fließtext, wenn einem die Wahl gelassen wird
2. Mehr als zwei Seiten benötigt $\Rightarrow$ Fehler oder sehr kompliziert gelöst
3. Ist meine Lösung verständlich für andere Personen?
4. Ist meine Lösung angenehm zu lesen?

Bearbeiten der Übungsblätter

1. Abwägen ob Pseudocode oder Fließtext, wenn einem die Wahl gelassen wird
2. Mehr als zwei Seiten benötigt $\Rightarrow$ Fehler oder sehr kompliziert gelöst
3. Ist meine Lösung verständlich für andere Personen?
4. Ist meine Lösung angenehm zu lesen?

Bearbeiten der Übungsblätter

1. Abwägen ob Pseudocode oder Fließtext, wenn einem die Wahl gelassen wird
2. Mehr als zwei Seiten benötigt $\Rightarrow$ Fehler oder sehr kompliziert gelöst
3. Ist meine Lösung verständlich für andere Personen?
4. Ist meine Lösung angenehm zu lesen?

Bearbeiten der Übungsblätter

1. Abwägen ob Pseudocode oder Fließtext, wenn einem die Wahl gelassen wird
2. Mehr als zwei Seiten benötigt $\Rightarrow$ Fehler oder sehr kompliziert gelöst
3. Ist meine Lösung verständlich für andere Personen?
4. Ist meine Lösung angenehm zu lesen?

Definitionen O-Kalkül

Aufgaben zum O-Kalkül

Laufzeiten

```
1 i := 99999 :Digit;
2 while i > 0 do
3         i--;
4 end
```

Laufzeiten

```
1 i := 99999 :Digit;
2 while i < n do
3         i--;
4 end
```

Laufzeiten

```
1 for i := 0 to n do
2         for j := n down to 1 do
3                 do Operation in O(1)
4         end
5 end
```

Laufzeiten

```
1 for i := 0 to n do
2         j := 0 :int;
3         while j < n do
4                 do Operation in O(n)
5                 j++;
6         end
7 end
```

Laufzeiten und Invarianten

```
1  fib(i:Digit)
2      if i <= 2
3          return 1;
4      else
5          return fib(i-1) + fib(i-2);
```

Laufzeiten und Invarianten

```
1 int linearSearch(a:Array[0...n] of int, x:int)
2         i := 0 :int;
3         while a[i] != x do
4                 i++;
5         end
6         return i;
```

Laufzeiten und Invarianten

```
1 int linearSearch(a:Array[0...n] of int, x:int)
2         i := 0 :int;
3         while i <= n & a[i] != x do
4                 i++;
5         end
6         return i;
```

Laufzeiten und Invarianten

```
1  mult(a:Digit, b:Digit)
2    x := a :Digit;
3    y := b :Digit;
4    p := 0 :Digit;
5    // Position 1
6    while x > 0 do
7      // Position 2
8      p := p + y;
9      x := x - 1;
10   end
11   // Position 3
12   return p;
```

Invarianten

- Invarianten:
  - oft einfach, meist aber schwer zu finden
  - Vorbedingung/en
  - Schleifeninvarianten
  - Nachbedingungen
  - als Korrektheitsbeweis

Laufzeiten

```
1  function(a:Array[0...n] of Digit, x, start, end :
       Digit)
2      assert start <= end;
3      m:Digit;
4      m := (start + end) / 2;
5      if a[m] == x
6          return m;
7      else if a[m] < x:
8          return function(a, x, m + 1, end);
9      else
10         return function(a, x, start, m - 1);
```

Laufzeiten und Invarianten

```
1  int binarySearch(a:Array[0...n] of Digit, x, start,
       end :Digit)
2      assert start <= end;
3      m:Digit;
4      m := (start + end) / 2;
5      if a[m] == x
6          // a[m] = x
7          return m;
8      else if a[m] < x:
9          // a[i] < x (i <= m)
10         return binarySearch(a, x, m + 1, end);
11     else
12         // a[i] > x (i >= m)
13         return binarySearch(a, x, start, m - 1);
```

Laufzeiten und Invarianten

```
1  int binarySearch(a:Array[0...n] of Digit, x:Digit)
2          i := 0 :Digit;
3          j := length(a) :Digit;
4          while i <= j do // Position 1
5                  m := (i + j) / 2;
6                  if(x < a[m])
7                                  j := m -1;
8                  else if(x > a[m])
9                          i := m + 1;
10                 else // Position 2
11                         return m;
12         end
13         // Position 3
14         return error;
```

Laufzeiten und Invarianten

```
1  function power(a:Double, n0:Digit)
2           p := a :Double;
3           r := 1 :Double;
4           n := n0 :Digit;
5           while n > 0 do
6                   if n is odd
7                           n--;
8                           r:= r*p;
9                   else
10                          n := n/2;
11                          p := p*p;
12          end
13          return r;
```

Karatsuba-Ofman Multiplikation[1962]

Beobachtung: $(a_1 + a_0)(b_1 + b_0) = a_1 b_1 + a_0 b_0 + a_1 b_0 + a_0 b_1$

Function recMult(a, b)

assert a und b haben $n = 2k$ Ziffern, n ist Zweierpotenz

if $n = 1$ **then return** $a \cdot b$

Schreibe a als $a_1 \cdot B^k + a_0$

Schreibe b als $b_1 \cdot B^k + b_0$

c_{11}:= recMult(a_1, b_1)

c_{00}:= recMult(a_0, b_0)

return

$$c_{11} \cdot B^{2k} +$$
$$(\text{recMult}((a_1 + a_0), (b_1 + b_0)) - c_{11} - c_{00})B^k$$
$$+ c_{00}$$

Tipps

Vielen Dank für eure Aufmerksamkeit!
Bis zum nächsten Mal.