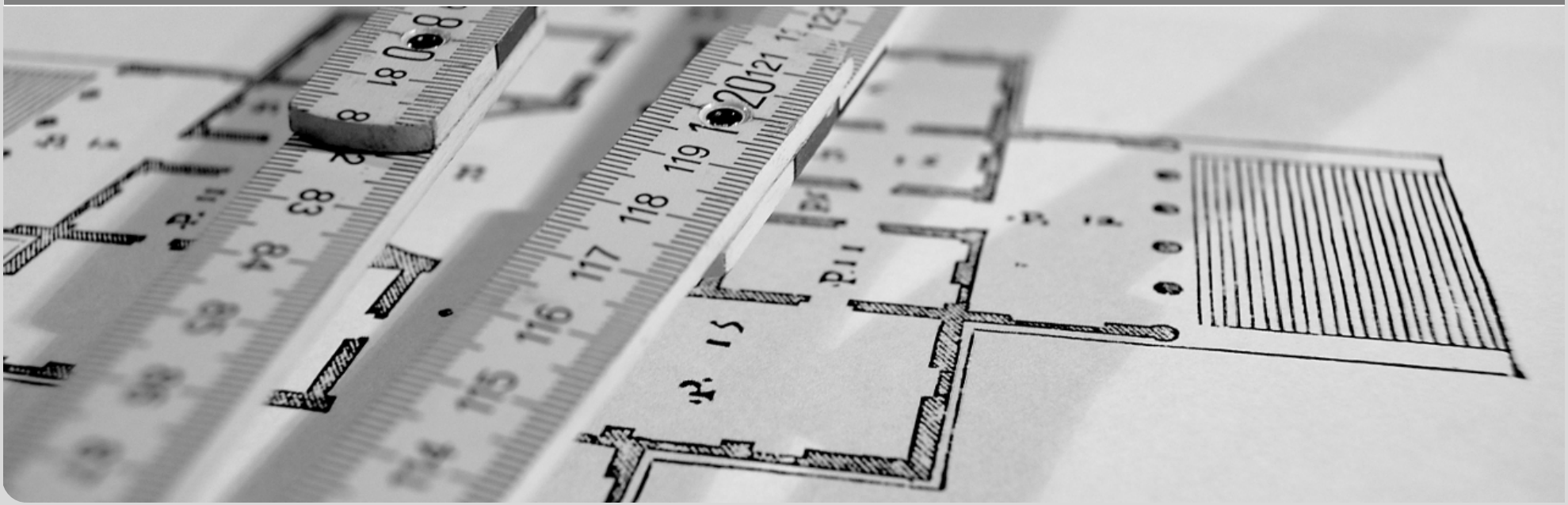
Karlsruher Institut für Technologie

Algorithmen I
Tutorium 33

Woche 1 | 27. April 2018

Daniel Jungkind (daniel.jungkind@student.kit.edu)

INSTITUT FÜR THEORETISCHE INFORMATIK



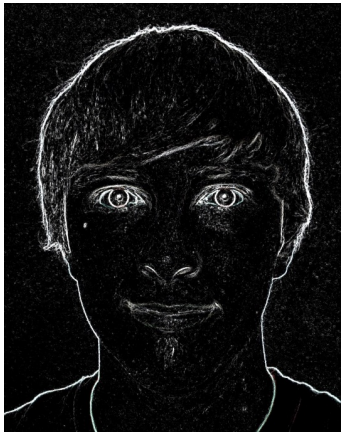
Inhalt

Orga-Kram

Pseudocode

Algorithmenanalyse

Gestatten...



Daniel Jungkind
daniel.jungkind@student.kit.edu

Informatik, 6. Fachsemester (Bachelor).

In ♡ with Haskell and
Functional Programming. :)

Kennenlernen 2.0

1	Programmieren ist noch ziemlich neu für mich.
2	Vor dem Studium habe ich nicht viel mit Programmieren zu tun gehabt, komme aber gut damit zurecht.
3	Ich konnte schon vor dem Studium einigermaßen programmieren und kannte daher vieles aus der Vorlesung schon.
4	In der Programmieren-Vorlesung habe ich eigentlich nichts neues gelernt, da ich auch so schon ganz gut programmieren konnte
5	Im Programmieren habe ich langjährige Erfahrung.
$\perp$	Nieder mit Pauschalantworten! Ich formuliere selbst!

Was machen wir hier?

- Tut-Folien bekommt ihr im ILIAS. Keine Panik.
- Inhalte der VL verstehen und anwenden
- Beispiele, Aufgaben (da kommt **ihr** ins Spiel! ☺)
- **Eure Fragen** klären!
- Den Stoff etwas weniger formal behandeln
  ⇒ Formal-Kram kann trotzdem wichtig sein!
- **Kein** Ersatz für die VL.
  Was zeitlich hier nicht mehr reinpasst, kann trotzdem wichtig sein!
- Wenn ich / meine Folien Blödsinn reden:
  Schreien! / Mail an mich! ⇒ Sonst kann ich's nicht fixen... ☹
  Verbindlich **nur** Inhalt aus der VL/Übung!

Was machen wir hier?

- Tut-Folien bekommt ihr im ILIAS. Keine Panik.
- Inhalte der VL verstehen und anwenden
- Beispiele, Aufgaben (da kommt **ihr** ins Spiel! ☺)
- **Eure Fragen** klären!
- Den Stoff etwas weniger formal behandeln
  ⇒ Formal-Kram kann trotzdem wichtig sein!
- **Kein** Ersatz für die VL
  Was zeitlich hier nicht mehr reinpasst, kann trotzdem wichtig sein!
- Wenn ich / meine Folien Blödsinn reden:
  Schreien! / Mail an mich! ⇒ Sonst kann ich's nicht fixen... ☹
  Verbindlich **nur** Inhalt aus der VL/Übung!

Was machen wir hier?

- Tut-Folien bekommt ihr im ILIAS. Keine Panik.
- Inhalte der VL verstehen und anwenden
- Beispiele, Aufgaben (da kommt **ihr** ins Spiel! ☺)
- **Eure Fragen** klären!
- Den Stoff etwas weniger formal behandeln
  $\Rightarrow$ Formal-Kram kann trotzdem wichtig sein!
- **Kein** Ersatz für die VL.
  Was zeitlich hier nicht mehr reinpasst, kann trotzdem wichtig sein!
- Wenn ich / meine Folien Blödsinn reden:
  Schreien! / Mail an mich! $\Rightarrow$ Sonst kann ich's nicht fixen... ☹
  Verbindlich **nur** Inhalt aus der VL/Übung!

Orga-Kram – Übungsblätter

- **Freiwillig**
- **Ausgabe**: Mi
  Abgabe: Mi nächster Woche, 13 Uhr (das sind 7 Tage Zeit) im Kasten im Untergeschoss des Infobaus
- Abgabe **schwerstens empfohlen**, gibt nämlich einen **Klausurbonus**:
  $\geqslant$ 25% der Gesamtpunkte $\Rightarrow$ 1 Bonuspunkt
  $\geqslant$ 50% der Gesamtpunkte $\Rightarrow$ 2 Bonuspunkte
  $\geqslant$ 75% der Gesamtpunkte $\Rightarrow$ 3 Bonuspunkte
  (Die Bonuspunkte helfen nicht beim Bestehen!)
- Nicht abgeholte Blätter landen irgendwann irgendwo beim Übungsleiter
- Abschreiben: Böhse™. Wird geahndet.
- **Aber**: Abgabe **zu zweit** erlaubt (und *erwünscht*! ☺)

Orga-Kram – Übungsblätter

- **Freiwillig**
- **Ausgabe**: Mi
  Abgabe: Mi nächster Woche, 13 Uhr (das sind 7 Tage Zeit) im Kasten im Untergeschoss des Infobaus
- Abgabe **schwerstens empfohlen**, gibt nämlich einen **Klausurbonus**:
  $\geqslant$ 25% der Gesamtpunkte $\Rightarrow$ 1 Bonuspunkt
  $\geqslant$ 50% der Gesamtpunkte $\Rightarrow$ 2 Bonuspunkte
  $\geqslant$ 75% der Gesamtpunkte $\Rightarrow$ 3 Bonuspunkte
  (Die Bonuspunkte helfen nicht beim Bestehen!)
- Nicht abgeholte Blätter landen irgendwann irgendwo beim Übungsleiter
- Abschreiben: Böhse™. Wird geahndet.
- Aber: Abgabe **zu zweit** erlaubt (und erwünscht! ☺)

Orga-Kram – Übungsblätter

- **Freiwillig**
- **Ausgabe**: Mi
  Abgabe: Mi nächster Woche, 13 Uhr (das sind 7 Tage Zeit) im Kasten im Untergeschoss des Infobaus
- Abgabe **schwerstens empfohlen**, gibt nämlich einen **Klausurbonus**:
  $\geqslant$ 25% der Gesamtpunkte $\Rightarrow$ 1 Bonuspunkt
  $\geqslant$ 50% der Gesamtpunkte $\Rightarrow$ 2 Bonuspunkte
  $\geqslant$ 75% der Gesamtpunkte $\Rightarrow$ 3 Bonuspunkte
  (Die Bonuspunkte helfen nicht beim Bestehen!)
- Nicht abgeholte Blätter landen irgendwann irgendwo beim Übungsleiter
- Abschreiben: Böhse™. Wird geahndet.
- Aber: Abgabe **zu zweit** erlaubt (und erwünscht! ☺)

Orga-Kram – Übungsblätter

- **Freiwillig**
- **Ausgabe**: Mi
  Abgabe: Mi nächster Woche, 13 Uhr (das sind 7 Tage Zeit) im Kasten im Untergeschoss des Infobaus
- Abgabe **schwerstens empfohlen**, gibt nämlich einen **Klausurbonus**:
  $\geqslant$ 25% der Gesamtpunkte $\Rightarrow$ 1 Bonuspunkt
  $\geqslant$ 50% der Gesamtpunkte $\Rightarrow$ 2 Bonuspunkte
  $\geqslant$ 75% der Gesamtpunkte $\Rightarrow$ 3 Bonuspunkte
  (Die Bonuspunkte helfen nicht beim Bestehen!)
- Nicht abgeholte Blätter landen irgendwann irgendwo beim Übungsleiter
- Abschreiben: Böhse™. Wird geahndet.
- Aber: Abgabe zu zweit erlaubt (und erwünscht! ☺)

Orga-Kram – Übungsblätter

- **Freiwillig**
- **Ausgabe**: Mi
  Abgabe: Mi nächster Woche, 13 Uhr (das sind 7 Tage Zeit) im Kasten im Untergeschoss des Infobaus
- Abgabe **schwerstens empfohlen**, gibt nämlich einen **Klausurbonus**:
  $\geqslant$ 25% der Gesamtpunkte $\Rightarrow$ 1 Bonuspunkt
  $\geqslant$ 50% der Gesamtpunkte $\Rightarrow$ 2 Bonuspunkte
  $\geqslant$ 75% der Gesamtpunkte $\Rightarrow$ 3 Bonuspunkte
  (Die Bonuspunkte helfen nicht beim Bestehen!)
- Nicht abgeholte Blätter landen irgendwann irgendwo beim Übungsleiter
- Abschreiben: Böhse™. Wird geahndet.
- **Aber**: Abgabe **zu zweit** erlaubt (und erwünscht! ☺)

Orga-Kram – Klausur

Klausur ist am **04. September 2018** um **8 Uhr**.

Details tba.

Fragen? Hilfe?

Fragen:

- **Hier** im Tut!
- Ins ILIAS. (Dann haben alle was davon. ☺)
- Organisatorischer Spezialkram? ⇒ an Iser / Sinz
  (markus.iser@kit.edu / carsten.sinz@kit.edu)
  Bei Orga-Problemen aber **Name** und **Matrikelnummer** mitangeben!
- Tut-spezifisches an **mich** (daniel.jungkind@student.kit.edu)

Content:

- VL-Folien, Ü-Blätter...: im ILIAS.

Feedback:

- Zur VL? ⇒ Gerne anonym in Zettelform in die Algo-I-Abgabekästen
  Keine Stofffragen! (Antwort unmöglich...)
- Zum Tut? ⇒ Per Mail oder persönlich an mich. ☺
  Zu schnell/langsam/leicht/schwer/viel/wenig/...? ⇒ **direkt** sagen!

Fragen? Hilfe?

Fragen:

- **Hier** im Tut!
- Ins ILIAS. (Dann haben alle was davon. ☺)
- Organisatorischer Spezialkram? ⇒ an Iser / Sinz
  (markus.iser@kit.edu / carsten.sinz@kit.edu)
  Bei Orga-Problemen aber **Name** und **Matrikelnummer** mitangeben!
- Tut-spezifisches an **mich** (daniel.jungkind@student.kit.edu)

Content:

- VL-Folien, Ü-Blätter...: im ILIAS.

Feedback:

- Zur VL? ⇒ Gerne anonym in Zettelform in die Algo-I-Abgabekästen
  Keine Stofffragen! (Antwort unmöglich...)
- Zum Tut? ⇒ Per Mail oder persönlich an mich. ☺
  Zu schnell/langsam/leicht/schwer/viel/wenig/...? ⇒ direkt sagen!

Fragen? Hilfe?

Fragen:

- **Hier** im Tut!
- Ins ILIAS. (Dann haben alle was davon. ☺)
- Organisatorischer Spezialkram? ⇒ an Iser / Sinz
  (markus.iser@kit.edu / carsten.sinz@kit.edu)
  Bei Orga-Problemen aber **Name** und **Matrikelnummer** mitangeben!
- Tut-spezifisches an **mich** (daniel.jungkind@student.kit.edu)

Content:

- VL-Folien, Ü-Blätter...: im ILIAS.

Feedback:

- Zur VL? ⇒ Gerne anonym in Zettelform in die Algo-I-Abgabekästen
  Keine Stofffragen! (Antwort unmöglich...)
- Zum Tut? ⇒ Per Mail oder persönlich an mich. ☺
  Zu schnell/langsam/leicht/schwer/viel/wenig/...? ⇒ direkt sagen!

Fragen? Hilfe?

Fragen:

- **Hier** im Tut!
- Ins ILIAS. (Dann haben alle was davon. ☺)
- Organisatorischer Spezialkram? ⇒ an Iser / Sinz
  (markus.iser@kit.edu / carsten.sinz@kit.edu)
  Bei Orga-Problemen aber **Name** und **Matrikelnummer** mitangeben!
- Tut-spezifisches an **mich** (daniel.jungkind@student.kit.edu)

Content:

- VL-Folien, Ü-Blätter...: im ILIAS.

Feedback:

- Zur VL? ⇒ Gerne anonym in Zettelform in die Algo-I-Abgabekästen
  Keine Stofffragen! (Antwort unmöglich...)
- Zum Tut? ⇒ Per Mail oder persönlich an mich. ☺
  Zu schnell/langsam/leicht/schwer/viel/wenig/...? ⇒ **direkt** sagen!

Pseudocode

Pseudocode

- Anleitung, wie man etwas macht – in einer Art „Programmiersprache“
  - Es gibt keine exakte Sprachdefinition
  - Es gibt keine exakte Sprachdefinition
  - Daher im Folgenden: Grobe *Richtlinie* für Pseudocode (ohne Anspruch auf Vollständigkeit)
  - Das **Wichtigste**: Es sollte **klar** werden, was **gemeint** ist, d.h. Pseudocode soll vor allem **übersichtlich** und **verständlich** sein
  - Kommentare mit // (wie in Java), #, o.ä.
  - Semikolon am Ende eines Befehls unnötig

Pseudocode

- Anleitung, wie man etwas macht – in einer Art „Programmiersprache“
- ✚ Es gibt keine exakte Sprachdefinition
- ▬ Es gibt keine exakte Sprachdefinition
- Daher im Folgenden: Grobe *Richtlinie* für Pseudocode (ohne Anspruch auf Vollständigkeit)
- Das **Wichtigste**: Es sollte **klar** werden, was **gemeint** ist, d.h. Pseudocode soll vor allem **übersichtlich** und **verständlich** sein
- Kommentare mit // (wie in Java), #, o.ä.
- Semikolon am Ende eines Befehls unnötig

Pseudocode

- Anleitung, wie man etwas macht – in einer Art „Programmiersprache“
- ✚ Es gibt keine exakte Sprachdefinition
- ▬ Es gibt keine exakte Sprachdefinition
- Daher im Folgenden: Grobe *Richtlinie* für Pseudocode (ohne Anspruch auf Vollständigkeit)
- Das **Wichtigste**: Es sollte **klar** werden, was **gemeint** ist, d.h. Pseudocode soll vor allem **übersichtlich** und **verständlich** sein
- Kommentare mit // (wie in Java), #, o.ä.
- Semikolon am Ende eines Befehls unnötig

Pseudocode

- Anleitung, wie man etwas macht – in einer Art „Programmiersprache“
- ✚ Es gibt keine exakte Sprachdefinition
- ▬ Es gibt keine exakte Sprachdefinition
- Daher im Folgenden: Grobe *Richtlinie* für Pseudocode (ohne Anspruch auf Vollständigkeit)
- Das **Wichtigste**: Es sollte **klar** werden, was **gemeint** ist, d.h. Pseudocode soll vor allem **übersichtlich** und **verständlich** sein
- Kommentare mit // (wie in Java), #, o.ä.
- Semikolon am Ende eines Befehls unnötig

Pseudocode

Variablendeklaration und -initialisierung

Schema: *Variablenname* $=$ *Wert* : Typ

Bsp.:

$var_1 = 42 : \text{Int}$

$var_2 = \text{"algorilla"} : \text{String}$

$var_3 := 1337$ // Typ wegelassen, falls offensichtlich

$var_4 : \mathbb{R}$ // Ohne Initialisierung (auf Papier nur)

bla := new XYZ(*Konstruktorparameter,..*)

Mögliche Typen:

- $\mathbb{R}, \mathbb{N}, \mathbb{Z}$ / Int(eger), $\mathbb{B}$ / Bool(ean), String...
- *Element* als Platzhalter für beliebigen Typ (so wie *Object* in Java)
- Weitere Datenstrukturen aus der VL (more to come!)
- array (als im Speicher zusammenhängender „Datenblock“)

Pseudocode

Variablendeklaration und -initialisierung

Schema: *Variablenname* = *Wert* : Typ

Bsp.:

$var_1 = 42 : \text{Int}$

$var_2 = \text{"algorilla"} : \text{String}$

$var_3 := 1337$ *// Typ weglassbar, falls offensichtlich*

$var_4 : \mathbb{R}$ *// Ohne Initialisierung (**aufpassen!**)*

bla := **new** XYZ(*Konstruktorparameter...*)

Mögliche Typen:

- $\mathbb{R}$, $\mathbb{N}$, $\mathbb{Z}$ / Int(eger), $\mathbb{B}$ / Bool(ean), String...
- *Element* als Platzhalter für beliebigen Typ (so wie *Object* in Java)
- Weitere Datenstrukturen aus der VL (more to come!)
- array (als im Speicher zusammenhängender „Datenblock“)

Pseudocode

Variablendeklaration und -initialisierung

Schema: *Variablenname* = *Wert* : Typ

Bsp.:

var$_1$ = 42 : Int

var$_2$ = "algorilla" : String

var$_3$:= 1337 *// Typ weglassbar, falls offensichtlich*

var$_4$: $\mathbb{R}$ *// Ohne Initialisierung (**aufpassen!**)*

bla := **new** XYZ(*Konstruktorparameter...*)

Mögliche Typen:

- $\mathbb{R}$, $\mathbb{N}$, $\mathbb{Z}$ / Int(eger), $\mathbb{B}$ / Bool(ean), String...
- *Element* als Platzhalter für beliebigen Typ (so wie *Object* in Java)
- Weitere Datenstrukturen aus der VL (more to come!)
- **array** (als im Speicher zusammenhängender „Datenblock“)

Pseudocode

Arrays

Schema: arr : **array**[$Von..Bis$] **of** T

Heißt: arr ist ein T-**array**. Der erste Index ist Von, der letzte ist Bis.

Bsp.:

$A = (1, 2, 3)$: array[42..44] of $\mathbb{Z}$

$\Rightarrow$ Dann ist $A[42] = 1$, $A[44] = 3$, $A[0]$ ist undefiniert

B : **array**[1..n] of Bool // Unterscheidet sich von Arrays in Java

Besondere Werte

- $+\infty$, $-\infty$
- $\perp$ für „undefiniert“, „bottom“

Pseudocode

Arrays

Schema: arr : **array**[$Von..Bis$] **of** T

Heißt: arr ist ein T-**array**. Der erste Index ist Von, der letzte ist Bis.

Bsp.:

$A = (1, 2, 3)$: **array**[42..44] **of** $\mathbb{Z}$

$\Rightarrow$ Dann ist $A[42] = 1$, $A[44] = 3$. $A[0]$ ist undefiniert!

Pseudocode

Arrays

Schema: arr : **array**[$Von..Bis$] **of** T

Heißt: arr ist ein T-**array**. Der erste Index ist Von, der letzte ist Bis.

Bsp.:

$A = (1, 2, 3)$: **array**[42..44] **of** $\mathbb{Z}$

$\Rightarrow$ Dann ist $A[42] = 1$, $A[44] = 3$. $A[0]$ ist undefiniert!

B : **array**[1..n] **of** Bool // *Uninitialisiert; Zugriff mit* $B[1]$ *bis* $B[n]$

Besondere Werte

- $+\infty, -\infty$
- $\perp$ für „undefiniert“, „bottom“

Pseudocode

Arrays

Schema: arr : **array**[$Von..Bis$] **of** T

Heißt: arr ist ein T-**array**. Der erste Index ist Von, der letzte ist Bis.

Bsp.:

$A = (1, 2, 3)$: **array**[42..44] **of** $\mathbb{Z}$

$\Rightarrow$ Dann ist $A[42] = 1$, $A[44] = 3$. $A[0]$ ist undefiniert!

B : **array**[1..n] **of** Bool // *Uninitialisiert; Zugriff mit* $B[1]$ *bis* $B[n]$

Besondere Werte

- $+\infty, -\infty$
- $\perp$ für „undefiniert“, „bottom“

Pseudocode

Kontrollstrukturen

```
while x ≠ y and y ≠ z do
    // Anweisungen
```

```
repeat
    // fußgesteuert
until x = y or y = z
```

```
if z = 42 then
    // Andere Anweisungen
else
    // Mehr andere Anweisungen
```

```
for i := 1 to n do
    // Noch mehr Anweisungen
```

```
for i := 1 to n step k do
    // Mit Schrittweite!
```

Pseudocode

- Und viele mehr, z.B. **do while**, **for each**, ...
- Schlüsselworte wie **continue**, **break**, **switch** natürlich auch
- Trennzeichen für Anweisungsblöcke:
  - **begin** – **end**
  - {...} (geschweifte Klammern)
  - Linien
  - Nur durch Einrückung (dann aber ordentlich!)

Pseudocode

Mathe-Bequemlichkeit

$M := \{1, 2, 3\}$ *// ungeordnete Menge*
$S := \langle 1, 2, 3 \rangle$ *// geordnete Folge (Elemente können an- und abgehängt werden)*
select any $x \in M$
foreach $x \in M$ **do**
$(a, b) := (3, 5)$ *// Tupel FTW! ☺*
$(a, b) := (b, a)$ *// bequemes Vertauschen*
div für Ganzzahl-Division, **mod** für Modulo (Divisionsrest)
...

ABER: Die Laufzeitkosten und Funktionsweise **jeder einzelnen Zeile** müssen bekannt sein!
(Heißt: Keine „magischen“ Funktionen verwenden!)

Pseudocode

Funktionen/Prozeduren

function / **procedure** name($name_1$: Typ_1, ...) : Rückgabetyp

```
    // Knorker Code
    return ...
```

- Rückgabetyp wird weggelassen, wenn nichts zurückgegeben wird
- Konvention:
  Kein Rückgabewert („`void`“) $\Rightarrow$ **procedure**/**method**,
  Rückgabe vorhanden $\Rightarrow$ **function**

Pseudocode

Anmerkungen:

- **Selbsterklärende** Bezeichner, z.B. print("...") statt *System.out*.println("...")
- Komplexere Stellen bitte **kommentieren**, sonst versteht es keiner
- Mehrere Funktionen (z.B. Hilfsfunktionen): Kenntlich machen, wo **Hauptfunktion**!
- Im „Notfall": An Java orientieren
- Für mehr Pseudocode-Details siehe Buch vom Sanders

Pseudocode

Anmerkungen:

- **Selbsterklärende** Bezeichner, z.B. print("...") statt *System*.*out*.println("...")
- Komplexere Stellen bitte **kommentieren**, sonst versteht es keiner
- Mehrere Funktionen (z.B. Hilfsfunktionen): Kenntlich machen, wo **Hauptfunktion**!
- Im „Notfall": An Java orientieren
- Für mehr Pseudocode-Details siehe Buch vom Sanders

Pseudocode

Anmerkungen:

- **Selbsterklärende** Bezeichner, z.B. print("...") statt *System*.*out*.println("...")
- Komplexere Stellen bitte **kommentieren**, sonst versteht es keiner
- Mehrere Funktionen (z.B. Hilfsfunktionen): Kenntlich machen, wo **Hauptfunktion**!
- Im „Notfall": An Java orientieren
- Für mehr Pseudocode-Details siehe Buch vom Sanders

Pseudocode

Anmerkungen:

- **Selbsterklärende** Bezeichner, z.B. print("...") statt *System*.*out*.println("...")
- Komplexere Stellen bitte **kommentieren**, sonst versteht es keiner
- Mehrere Funktionen (z.B. Hilfsfunktionen): Kenntlich machen, wo **Hauptfunktion**!
- Im „Notfall“: An Java orientieren
- Für mehr Pseudocode-Details siehe Buch vom Sanders

Pseudocode

Anmerkungen:

- **Selbsterklärende** Bezeichner, z.B. print("...") statt *System*.*out*.println("...")
- Komplexere Stellen bitte **kommentieren**, sonst versteht es keiner
- Mehrere Funktionen (z.B. Hilfsfunktionen): Kenntlich machen, wo **Hauptfunktion**!
- Im „Notfall“: An Java orientieren
- Für mehr Pseudocode-Details siehe Buch vom Sanders

Pseudocode

Anmerkungen:

- **Selbsterklärende** Bezeichner, z.B. print("...") statt *System*.*out*.println("...")
- Komplexere Stellen bitte **kommentieren**, sonst versteht es keiner
- Mehrere Funktionen (z.B. Hilfsfunktionen): Kenntlich machen, wo **Hauptfunktion**!
- Im „Notfall“: An Java orientieren
- Für mehr Pseudocode-Details siehe Buch vom Sanders

Pseudocode

Bearbeitungshinweise:

- Falls die Aufgabenstellung euch die Wahl lässt, könnt ihr selbst entscheiden, ob **Pseudocode** oder **Fließtext**
- Mehr als zwei Seiten Pseudocode $\Rightarrow$ Vermutlich viel zu kompliziert oder falsch
- Ist das Ergebnis für andere Personen verständlich?
- Ist das Ergebnis angenehm zu lesen?

Pseudocode

Bearbeitungshinweise:

- Falls die Aufgabenstellung euch die Wahl lässt, könnt ihr selbst entscheiden, ob **Pseudocode** oder **Fließtext**
- Mehr als zwei Seiten Pseudocode $\Rightarrow$ Vermutlich viel zu kompliziert oder falsch
- Ist das Ergebnis für andere Personen verständlich?
- Ist das Ergebnis angenehm zu lesen?

Pseudocode

Bearbeitungshinweise:

- Falls die Aufgabenstellung euch die Wahl lässt, könnt ihr selbst entscheiden, ob **Pseudocode** oder **Fließtext**
- Mehr als zwei Seiten Pseudocode $\Rightarrow$ Vermutlich viel zu kompliziert oder falsch
- Ist das Ergebnis für andere Personen **verständlich**?
- Ist das Ergebnis angenehm zu lesen?

Pseudocode

Bearbeitungshinweise:

- Falls die Aufgabenstellung euch die Wahl lässt, könnt ihr selbst entscheiden, ob **Pseudocode** oder **Fließtext**
- Mehr als zwei Seiten Pseudocode $\Rightarrow$ Vermutlich viel zu kompliziert oder falsch
- Ist das Ergebnis für andere Personen **verständlich**?
- Ist das Ergebnis **angenehm zu lesen**?

Pseudocode

Aufgabe 1: Pseudocode schreiben

Als Eingabe erhält der Algorithmus eine natürliche Zahl n.
Es wird ein Boolean-Array von $2..n$ angelegt und mit **false** initialisiert.

Dann wird eine Zahl i von 2 bis zur abgerundeten Wurzel von n iteriert. Falls im Schleifendurchlauf der i-te Boolean-Wert **false** ist, wird eine weitere Zahl j von $2i$ bis n durchlaufen und dabei in Schritten der Größe i inkrementiert. Darin wird jeweils der j-te Boolean-Wert auf **true** gesetzt.

Am Ende iteriert der Algorithmus erneut eine Zahl i von 2 bis n. Falls der i-te Boolean-Wert nicht **true** ist, wird ausgegeben, dass der Wert von i prima ist.

Pseudocode

(Mögliche) Lösung von Aufgabe 1

```
procedure foo(n : ℕ)
    werte = (false, ..., false) : array[2..n] of 𝔹
    for i := 2 to ⌊√n⌋ do
        if not werte[i] then
            for j := 2i to n step i do
                werte[j] := true
    for i := 2 to n do
        if not werte[i] then
            print(i + " ist prima!")
```

Pseudocode

Aufgabe 2: Mehr Pseudocode schreiben

Als Eingabe erhaltet ihr *n* Studenten, deren Matrikelnummer jeweils als Array von Ziffern gegeben ist.
Diese Studenten sollt ihr in zwei Gruppen einteilen: Eine Gruppe mit den Studenten, bei denen die Quersumme der Matrikelnummer gerade ist, und eine Gruppe für alle anderen.
Das Ergebnis wird als geordnetes Paar von Listen zurückgegeben.

Pseudocode

(Mögliche) Lösung von Aufgabe 2

```
function groupStudents(students : array[1..n] of Student) :
  (List of Student, List of Student)
    groups = (⟨⟩, ⟨⟩) : array of List of Student
    foreach student ∈ students do
        sum = 0 : Int
        nr := student.matrikelnummer
        for i := 1 to |nr| do
            sum += nr[i]
        groups[sum mod 2].add(student)
    return groups
```

ALGORITHMENANALYSE

...damit ihr nicht zu viel Spaß habt

Algorithmenanalyse

Algorithmen in Pseudocode lesen und schreiben ist (leider) nicht das Höchste der Gefühle. Sondern:

Algorithmenanalyse

Algorithmen in Pseudocode lesen und schreiben ist (leider) nicht das Höchste der Gefühle. Sondern:

- Wie schnell ist der Algorithmus?
  $\Rightarrow$ **O-Kalkül**
- Auf welcher Vorgehensweise basiert der Algorithmus?
  $\Rightarrow$ Verschiedene Kategorien von Algorithmen
- Tut der Algorithmus das, was er tun soll?
  $\Rightarrow$ Invarianten

Algorithmenanalyse

Algorithmen in Pseudocode lesen und schreiben ist (leider) nicht das Höchste der Gefühle. Sondern:

- Wie schnell ist der Algorithmus?
  $\Rightarrow$ **O-Kalkül**
- Auf welcher Vorgehensweise basiert der Algorithmus?
  $\Rightarrow$ Verschiedene **Kategorien** von Algorithmen
- Tut der Algorithmus das, was er tun soll?
  $\Rightarrow$ **Invarianten**

Algorithmenanalyse

Algorithmen in Pseudocode lesen und schreiben ist (leider) nicht das Höchste der Gefühle. Sondern:

- Wie schnell ist der Algorithmus?
  $\Rightarrow$ **O-Kalkül**
- Auf welcher Vorgehensweise basiert der Algorithmus?
  $\Rightarrow$ Verschiedene **Kategorien** von Algorithmen
- Tut der Algorithmus das, was er tun soll?
  $\Rightarrow$ **Invarianten**

Algorithmenanalyse

O-Kalkül

- Betrachtung des asymptotischen Verhaltens
- Vernachlässigung konstanter Faktoren
- Zuordnung, welches Laufzeit-Verhalten der Algorithmus für sehr große Eingaben aufweist
- Rechenregeln: Siehe GBI, ...

Algorithmenanalyse

Alte Bekannte

$$O(f(n)) := \{g(n) \mid \exists\, c, n_0 > 0 \text{ mit } 0 \leqslant g(n) \leqslant c \cdot f(n) \quad \forall n \geqslant n_0\}$$

$$\Theta(f(n)) := \{g(n) \mid \exists\, c, c', n_0 > 0 \text{ mit } 0 \leqslant c \cdot f(n) \leqslant g(n) \leqslant c' \cdot f(n) \quad \forall n \geqslant n_0\}$$

$$\Omega(f(n)) := \{g(n) \mid \exists\, c, n_0 > 0 \text{ mit } 0 \leqslant c \cdot f(n) \leqslant g(n) \quad \forall n \geqslant n_0\}$$

Die Neuen

$$o(f(n)) := \{g(n) \mid \forall c > 0\, \exists\, n_0 > 0 \text{ mit } 0 \leqslant g(n) \leqslant c \cdot f(n) \quad \forall n \geqslant n_0\}$$

$$\omega(f(n)) := \{g(n) \mid \forall c > 0\, \exists\, n_0 > 0 \text{ mit } 0 \leqslant c \cdot f(n) \leqslant g(n) \quad \forall n \geqslant n_0\}$$

Algorithmenanalyse

Alte Bekannte

$$O(f(n)) := \{g(n) \mid \exists\, c, n_0 > 0 \text{ mit } 0 \leqslant g(n) \leqslant c \cdot f(n) \quad \forall n \geqslant n_0\}$$

$$\Theta(f(n)) := \{g(n) \mid \exists\, c, c', n_0 > 0 \text{ mit } 0 \leqslant c \cdot f(n) \leqslant g(n) \leqslant c' \cdot f(n) \quad \forall n \geqslant n_0\}$$

$$\Omega(f(n)) := \{g(n) \mid \exists\, c, n_0 > 0 \text{ mit } 0 \leqslant c \cdot f(n) \leqslant g(n) \quad \forall n \geqslant n_0\}$$

Die Neuen

$$o(f(n)) := \{g(n) \mid \forall c > 0\ \exists\, n_0 > 0 \text{ mit } 0 \leqslant g(n) \leqslant c \cdot f(n) \quad \forall n \geqslant n_0\}$$

$$\omega(f(n)) := \{g(n) \mid \forall c > 0\ \exists\, n_0 > 0 \text{ mit } 0 \leqslant c \cdot f(n) \leqslant g(n) \quad \forall n \geqslant n_0\}$$

Algorithmenanalyse

Anschaulich:

$o(f(n))$	$\prec$	echt schwächer wachsende Funktionen
$O(f(n))$	$\preceq$	schwächer oder gleich stark wachsende Funktionen
$\Theta(f(n))$	$\asymp$	genau gleich stark wachsende Funktionen
$\Omega(f(n))$	$\succeq$	stärker oder gleich stark wachsende Funktionen
$\omega(f(n))$	$\succ$	echt stärker wachsende Funktionen

Algorithmenanalyse

$n \in \Theta(\sqrt{n})$. **?**

Algorithmenanalyse

$n \in \Theta(\sqrt{n})$. **Falsch.**

Algorithmenanalyse

$n \in \Theta(\sqrt{n})$. **Falsch.**

$n^2 \in o(n^3)$. **?**

Algorithmenanalyse

$n \in \Theta(\sqrt{n})$. **Falsch.**

$n^2 \in o(n^3)$. **Wahr.**

Algorithmenanalyse

$n \in \Theta(\sqrt{n})$. **Falsch.**

$n^2 \in o(n^3)$. **Wahr.**

$n^3 \in \Omega(n^2)$. **?**

Algorithmenanalyse

$n \in \Theta(\sqrt{n})$. **Falsch.**

$n^2 \in o(n^3)$. **Wahr.**

$n^3 \in \Omega(n^2)$. **Wahr.**

Algorithmenanalyse

$n \in \Theta(\sqrt{n})$. **Falsch.**

$n^2 \in o(n^3)$. **Wahr.**

$n^3 \in \Omega(n^2)$. **Wahr.**

$2^{n+1} \in \Theta(2^n)$. **?**

Algorithmenanalyse

$n \in \Theta(\sqrt{n})$. **Falsch.**

$n^2 \in o(n^3)$. **Wahr.**

$n^3 \in \Omega(n^2)$. **Wahr.**

$2^{n+1} \in \Theta(2^n)$. **Wahr.** (da $2^{n+1} = 2 \cdot 2^n$.)

Algorithmenanalyse

$n \in \Theta(\sqrt{n})$. **Falsch.**

$n^2 \in o(n^3)$. **Wahr.**

$n^3 \in \Omega(n^2)$. **Wahr.**

$2^{n+1} \in \Theta(2^n)$. **Wahr.** (da $2^{n+1} = 2 \cdot 2^n$.)

$f \approx g :\Leftrightarrow f \in O(g)$ definiert eine Äquivalenzrelation auf der Menge der Funktionen $\mathbb{N}^{\mathbb{N}}$. **?**

Algorithmenanalyse

$n \in \Theta(\sqrt{n})$. **Falsch.**

$n^2 \in o(n^3)$. **Wahr.**

$n^3 \in \Omega(n^2)$. **Wahr.**

$2^{n+1} \in \Theta(2^n)$. **Wahr.** (da $2^{n+1} = 2 \cdot 2^n$.)

$f \approx g :\Leftrightarrow f \in O(g)$ definiert eine Äquivalenzrelation auf der Menge der Funktionen $\mathbb{N}^{\mathbb{N}}$. **Falsch.**

(Aber: Ersetzt man O durch Θ, passt die Äquivalenzrelation.)

Algorithmenanalyse

$n \in \Theta(\sqrt{n})$. **Falsch.**

$n^2 \in o(n^3)$. **Wahr.**

$n^3 \in \Omega(n^2)$. **Wahr.**

$2^{n+1} \in \Theta(2^n)$. **Wahr.** (da $2^{n+1} = 2 \cdot 2^n$.)

$f \approx g :\Leftrightarrow f \in O(g)$ definiert eine Äquivalenzrelation auf der Menge der Funktionen $\mathbb{N}^{\mathbb{N}}$. **Falsch.**

(Aber: Ersetzt man O durch Θ, passt die Äquivalenzrelation.)

$o(f) = O(f) \setminus \Theta(f)$. **?**

Algorithmenanalyse

$n \in \Theta(\sqrt{n})$. **Falsch.**

$n^2 \in o(n^3)$. **Wahr.**

$n^3 \in \Omega(n^2)$. **Wahr.**

$2^{n+1} \in \Theta(2^n)$. **Wahr.** (da $2^{n+1} = 2 \cdot 2^n$.)

$f \approx g :\Leftrightarrow f \in O(g)$ definiert eine Äquivalenzrelation auf der Menge der Funktionen $\mathbb{N}^{\mathbb{N}}$. **Falsch.**

(Aber: Ersetzt man O durch Θ, passt die Äquivalenzrelation.)

$o(f) = O(f) \setminus \Theta(f)$. **Wahr.**

Algorithmenanalyse

$n \in \Theta(\sqrt{n})$. **Falsch.**

$n^2 \in o(n^3)$. **Wahr.**

$n^3 \in \Omega(n^2)$. **Wahr.**

$2^{n+1} \in \Theta(2^n)$. **Wahr.** (da $2^{n+1} = 2 \cdot 2^n$.)

$f \approx g :\Leftrightarrow f \in O(g)$ definiert eine Äquivalenzrelation auf der Menge der Funktionen $\mathbb{N}^{\mathbb{N}}$. **Falsch.**

(Aber: Ersetzt man O durch Θ, passt die Äquivalenzrelation.)

$o(f) = O(f) \setminus \Theta(f)$. **Wahr.**

$\forall c_1, c_2 \in \mathbb{N},\ f\colon \mathbb{N}_+ \longrightarrow \mathbb{N}_+ : \quad c_1 \cdot f(n) + c_2 \in O(f(n))$. **?**

Algorithmenanalyse

$n \in \Theta(\sqrt{n})$. **Falsch.**

$n^2 \in o(n^3)$. **Wahr.**

$n^3 \in \Omega(n^2)$. **Wahr.**

$2^{n+1} \in \Theta(2^n)$. **Wahr.** (da $2^{n+1} = 2 \cdot 2^n$.)

$f \approx g :\Leftrightarrow f \in O(g)$ definiert eine Äquivalenzrelation auf der Menge der Funktionen $\mathbb{N}^{\mathbb{N}}$. **Falsch.**

(Aber: Ersetzt man O durch Θ, passt die Äquivalenzrelation.)

$o(f) = O(f) \setminus \Theta(f)$. **Wahr.**

$\forall c_1, c_2 \in \mathbb{N},\ f\colon \mathbb{N}_+ \longrightarrow \mathbb{N}_+ : \quad c_1 \cdot f(n) + c_2 \in O(f(n))$. **Wahr.**

Algorithmenanalyse

$n \in \Theta(\sqrt{n})$. **Falsch.**

$n^2 \in o(n^3)$. **Wahr.**

$n^3 \in \Omega(n^2)$. **Wahr.**

$2^{n+1} \in \Theta(2^n)$. **Wahr.** (da $2^{n+1} = 2 \cdot 2^n$.)

$f \approx g :\Leftrightarrow f \in O(g)$ definiert eine Äquivalenzrelation auf der Menge der Funktionen $\mathbb{N}^{\mathbb{N}}$. **Falsch.**

(Aber: Ersetzt man O durch Θ, passt die Äquivalenzrelation.)

$o(f) = O(f) \setminus \Theta(f)$. **Wahr.**

$\forall c_1, c_2 \in \mathbb{N},\ f\colon \mathbb{N}_+ \longrightarrow \mathbb{N}_+ :\quad c_1 \cdot f(n) + c_2 \in O(f(n))$. **Wahr.**

$\forall c \in \mathbb{N},\ f : \mathbb{N}_+ \longrightarrow \mathbb{N}_+ :\quad (f(n))^c \in \omega(f(n))$. **?**

Algorithmenanalyse

$n \in \Theta(\sqrt{n})$. **Falsch.**

$n^2 \in o(n^3)$. **Wahr.**

$n^3 \in \Omega(n^2)$. **Wahr.**

$2^{n+1} \in \Theta(2^n)$. **Wahr.** (da $2^{n+1} = 2 \cdot 2^n$.)

$f \approx g :\Leftrightarrow f \in O(g)$ definiert eine Äquivalenzrelation auf der Menge der Funktionen $\mathbb{N}^{\mathbb{N}}$. **Falsch.**

(Aber: Ersetzt man O durch Θ, passt die Äquivalenzrelation.)

$o(f) = O(f) \setminus \Theta(f)$. **Wahr.**

$\forall c_1, c_2 \in \mathbb{N},\ f\colon \mathbb{N}_+ \longrightarrow \mathbb{N}_+ :\quad c_1 \cdot f(n) + c_2 \in O(f(n))$. **Wahr.**

$\forall c \in \mathbb{N},\ f : \mathbb{N}_+ \longrightarrow \mathbb{N}_+ :\quad (f(n))^c \in \omega(f(n))$. **Falsch.** (↯ für $c = 1$.)

Algorithmenanalyse

O-Kalkül: Formeln

$f(n) \in o(g(n))$	$\iff$	$\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = 0$
$f(n) \in O(g(n))$	$\iff$	$0 \leqslant \limsup\limits_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty$
$f(n) \in \Theta(g(n))$	**!** $\Longleftarrow$ **!**	$0 < \lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty$
$f(n) \in \Omega(g(n))$	$\iff$	$0 < \liminf\limits_{n\to\infty} \frac{f(n)}{g(n)} = c \leqslant \infty$
$f(n) \in \omega(g(n))$	$\iff$	$\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

Algorithmenanalyse

Lang ist's her: Logarithmus-Rechenregeln

- Für den Informatiker ist $\log n := \log_2(n)$
- $a^{\log_a(b)} = b$
- $\log_a(a) = 1, \quad \log_a(1) = 0$
- $\log(a \cdot b) = \log a + \log b, \quad \log\left(\frac{a}{b}\right) = \log a - \log b$
- $\log(a^b) = b \cdot \log a$
- $a^{\log n} = n^{\log a}$
- $x^{a \cdot b} = \left(x^a\right)^b = \left(x^b\right)^a, \quad x^{a+b} = x^a \cdot x^b$
- Beispiele:

Algorithmenanalyse

Lang ist's her: Logarithmus-Rechenregeln

- Für den Informatiker ist $\log n := \log_2(n)$
- $a^{\log_a(b)} = b$
- $\log_a(a) = 1, \quad \log_a(1) = 0$
- $\log(a \cdot b) = \log a + \log b, \quad \log\left(\frac{a}{b}\right) = \log a - \log b$
- $\log(a^b) = b \cdot \log a$
- $a^{\log n} = n^{\log a}$
- $x^{a \cdot b} = \left(x^a\right)^b = \left(x^b\right)^a, \quad x^{a+b} = x^a \cdot x^b$
- Beispiele:

Algorithmenanalyse

Lang ist's her: Logarithmus-Rechenregeln

- Für den Informatiker ist $\log n := \log_2(n)$
- $a^{\log_a(b)} = b$
- $\log_a(a) = 1, \quad \log_a(1) = 0$
- $\log(a \cdot b) = \log a + \log b, \quad \log\left(\frac{a}{b}\right) = \log a - \log b$
- $\log(a^b) = b \cdot \log a$
- $a^{\log n} = n^{\log a}$
- $x^{a \cdot b} = \left(x^a\right)^b = \left(x^b\right)^a, \quad x^{a+b} = x^a \cdot x^b$
- Beispiele:

Algorithmenanalyse

Lang ist's her: Logarithmus-Rechenregeln

- Für den Informatiker ist $\log n := \log_2(n)$
- $a^{\log_a(b)} = b$
- $\log_a(a) = 1, \quad \log_a(1) = 0$
- $\log(a \cdot b) = \log a + \log b, \quad \log(\frac{a}{b}) = \log a - \log b$
- $\log(a^b) = b \cdot \log a$
- $a^{\log n} = n^{\log a}$
- $x^{a \cdot b} = (x^a)^b = (x^b)^a, \quad x^{a+b} = x^a \cdot x^b$
- Beispiele:

Algorithmenanalyse

Lang ist's her: Logarithmus-Rechenregeln

- Für den Informatiker ist $\log n := \log_2(n)$
- $a^{\log_a(b)} = b$
- $\log_a(a) = 1, \quad \log_a(1) = 0$
- $\log(a \cdot b) = \log a + \log b, \quad \log(\frac{a}{b}) = \log a - \log b$
- $\log(a^b) = b \cdot \log a$
- $a^{\log n} = n^{\log a}$
- $x^{a \cdot b} = (x^a)^b = (x^b)^a, \quad x^{a+b} = x^a \cdot x^b$
- Beispiele:

Algorithmenanalyse

Lang ist's her: Logarithmus-Rechenregeln

- Für den Informatiker ist $\log n := \log_2(n)$
- $a^{\log_a(b)} = b$
- $\log_a(a) = 1, \quad \log_a(1) = 0$
- $\log(a \cdot b) = \log a + \log b, \quad \log(\frac{a}{b}) = \log a - \log b$
- $\log(a^b) = b \cdot \log a$
- $a^{\log n} = n^{\log a}$
- $x^{a \cdot b} = (x^a)^b = (x^b)^a, \quad x^{a+b} = x^a \cdot x^b$
- Beispiele:

Algorithmenanalyse

Lang ist's her: Logarithmus-Rechenregeln

- Für den Informatiker ist $\log n := \log_2(n)$
- $a^{\log_a(b)} = b$
- $\log_a(a) = 1, \quad \log_a(1) = 0$
- $\log(a \cdot b) = \log a + \log b, \quad \log(\frac{a}{b}) = \log a - \log b$
- $\log(a^b) = b \cdot \log a$
- $a^{\log n} = n^{\log a}$
- $x^{a \cdot b} = (x^a)^b = (x^b)^a, \quad x^{a+b} = x^a \cdot x^b$
- Beispiele:

Algorithmenanalyse

Lang ist's her: Logarithmus-Rechenregeln

- Für den Informatiker ist $\log n := \log_2(n)$
- $a^{\log_a(b)} = b$
- $\log_a(a) = 1, \quad \log_a(1) = 0$
- $\log(a \cdot b) = \log a + \log b, \quad \log(\frac{a}{b}) = \log a - \log b$
- $\log(a^b) = b \cdot \log a$
- $a^{\log n} = n^{\log a}$
- $x^{a \cdot b} = (x^a)^b = (x^b)^a, \quad x^{a+b} = x^a \cdot x^b$
- Beispiele:
  - $\log(10 \cdot n) \in O(\log n)$
  - $n^n \in \Theta(2^{n \log n})$
  - $\log_a n \in \Theta(\log_b n)$

Algorithmenanalyse

Lang ist's her: Logarithmus-Rechenregeln

- Für den Informatiker ist $\log n := \log_2(n)$
- $a^{\log_a(b)} = b$
- $\log_a(a) = 1, \quad \log_a(1) = 0$
- $\log(a \cdot b) = \log a + \log b, \quad \log(\frac{a}{b}) = \log a - \log b$
- $\log(a^b) = b \cdot \log a$
- $a^{\log n} = n^{\log a}$
- $x^{a \cdot b} = (x^a)^b = (x^b)^a, \quad x^{a+b} = x^a \cdot x^b$
- Beispiele:
  - $\log(10 \cdot n) \in O(\log n)$
  - $n^n \in \Theta(2^{n \log n})$
  - $\log_a n \in \Theta(\log_b n)$

Algorithmenanalyse

Lang ist's her: Logarithmus-Rechenregeln

- Für den Informatiker ist $\log n := \log_2(n)$
- $a^{\log_a(b)} = b$
- $\log_a(a) = 1, \quad \log_a(1) = 0$
- $\log(a \cdot b) = \log a + \log b, \quad \log(\frac{a}{b}) = \log a - \log b$
- $\log(a^b) = b \cdot \log a$
- $a^{\log n} = n^{\log a}$
- $x^{a \cdot b} = (x^a)^b = (x^b)^a, \quad x^{a+b} = x^a \cdot x^b$
- Beispiele:
  - $\log(10 \cdot n) \in O(\log n)$
  - $n^n \in \Theta(2^{n \log n})$
  - $\log_a n \in \Theta(\log_b n)$

Algorithmenanalyse

Lang ist's her: Logarithmus-Rechenregeln

- Für den Informatiker ist $\log n := \log_2(n)$
- $a^{\log_a(b)} = b$
- $\log_a(a) = 1, \quad \log_a(1) = 0$
- $\log(a \cdot b) = \log a + \log b, \quad \log(\frac{a}{b}) = \log a - \log b$
- $\log(a^b) = b \cdot \log a$
- $a^{\log n} = n^{\log a}$
- $x^{a \cdot b} = (x^a)^b = (x^b)^a, \quad x^{a+b} = x^a \cdot x^b$
- Beispiele:
  - $\log(10 \cdot n) \in O(\log n)$
  - $n^n \in \Theta(2^{n \log n})$
  - $\log_a n \in \Theta(\log_b n)$

Algorithmenanalyse

Ein mysteriöser Algorithmus

```
procedure foo(a : array of ℤ)
    n := |a|
    flag : Bool
    repeat
        flag := true
        for i := 0 to n − 2 do
            if a[i] > a[i + 1] then
                (a[i]; a[i + 1]) := (a[i + 1]; a[i])
                flag := false
    until flag
```

Algorithmenanalyse

- Das ist *Bubblesort*, der ein Array aufsteigend sortiert
  - Best case?
    $\Rightarrow O(n)$, wenn das Array schon sortiert ist
  - Worst case?
    $\Rightarrow O(n^2)$, wenn das Array absteigend („falsch rum“) sortiert ist

Algorithmenanalyse

- Das ist *Bubblesort*, der ein Array aufsteigend sortiert
- Best case?

  $\Rightarrow O(n)$, wenn das Array schon sortiert ist
- Worst case?

  $\Rightarrow O(n^2)$, wenn das Array absteigend („falsch rum“) sortiert ist

Algorithmenanalyse

- Das ist *Bubblesort*, der ein Array aufsteigend sortiert
- Best case?
  $\Rightarrow O(n)$, wenn das Array schon sortiert ist
- Worst case?
  $\Rightarrow O(n^2)$, wenn das Array absteigend („falsch rum“) sortiert ist

Algorithmenanalyse

- Das ist *Bubblesort*, der ein Array aufsteigend sortiert
- Best case?
  $\Rightarrow O(n)$, wenn das Array schon sortiert ist
- Worst case?
  $\Rightarrow O(n^2)$, wenn das Array absteigend („falsch rum“) sortiert ist

Algorithmenanalyse

- Das ist *Bubblesort*, der ein Array aufsteigend sortiert
- Best case?
  $\Rightarrow O(n)$, wenn das Array schon sortiert ist
- Worst case?
  $\Rightarrow O(n^2)$, wenn das Array absteigend („falsch rum“) sortiert ist

Algorithmenanalyse

Korrektheitsbeweise

- Korrektheitsbeweise sind **zweiteilig**:
  - 1. Teil – **Funktionalität**: Mit Invariante beweisen, dass der Algorithmus ein **korrektes** Ergebnis erzeugt
  - 2. Teil – **Terminierung**: Beweisen (ggf. anhand einer Invariante), dass der Algorithmus „irgendwann **fertig** wird". Manchmal trivial, manchmal kniffilig (und damit aufwendig)
- **Aufgabenstellung beachten**: Wenn („nur") eine Invariante angegeben/bewiesen werden soll ⇒ Terminierungsbeweis nicht nötig!

Algorithmenanalyse

Korrektheitsbeweise

- Korrektheitsbeweise sind **zweiteilig**:
  - 1. Teil – **Funktionalität:** Mit Invariante beweisen, dass der Algorithmus ein **korrektes** Ergebnis erzeugt
  - 2. Teil – **Terminierung:** Beweisen (ggf. anhand einer Invariante), dass der Algorithmus „irgendwann **fertig** wird". Manchmal trivial, manchmal kniffilig (und damit aufwendig)
  - **Aufgabenstellung beachten:** Wenn („nur") eine Invariante angegeben/bewiesen werden soll ⇒ Terminierungsbeweis nicht nötig!

Algorithmenanalyse

Korrektheitsbeweise

- Korrektheitsbeweise sind **zweiteilig**:
  - 1. Teil – **Funktionalität**: Mit Invariante beweisen, dass der Algorithmus ein **korrektes** Ergebnis erzeugt
  - 2. Teil – **Terminierung**: Beweisen (ggf. anhand einer Invariante), dass der Algorithmus „irgendwann **fertig** wird". Manchmal trivial, manchmal kniffilig (und damit aufwendig)
  - **Aufgabenstellung beachten**: Wenn („nur") eine Invariante angegeben/bewiesen werden soll ⇒ Terminierungsbeweis nicht nötig!

Algorithmenanalyse

Korrektheitsbeweise

- Korrektheitsbeweise sind **zweiteilig**:
  - 1. Teil – **Funktionalität**: Mit Invariante beweisen, dass der Algorithmus ein **korrektes** Ergebnis erzeugt
  - 2. Teil – **Terminierung**: Beweisen (ggf. anhand einer Invariante), dass der Algorithmus „irgendwann **fertig** wird“. Manchmal trivial, manchmal knifflig (und damit aufwendig)
- **Aufgabenstellung beachten**: Wenn („nur“) eine Invariante angegeben/bewiesen werden soll ⇒ Terminierungsbeweis nicht nötig!

Algorithmenanalyse

Korrektheitsbeweise

- Korrektheitsbeweise sind **zweiteilig**:
  - 1. Teil – **Funktionalität**: Mit Invariante beweisen, dass der Algorithmus ein **korrektes** Ergebnis erzeugt
  - 2. Teil – **Terminierung**: Beweisen (ggf. anhand einer Invariante), dass der Algorithmus „irgendwann **fertig** wird“. Manchmal trivial, manchmal knifflig (und damit aufwendig)
- **Aufgabenstellung beachten**: Wenn („nur“) eine Invariante angegeben/bewiesen werden soll $\Rightarrow$ Terminierungsbeweis nicht nötig!

Algorithmenanalyse

Invarianten

- Invariante finden: Manchmal offensichtlich, manchmal **Kreativität** gefragt
- Korrektheitsbeweise über Invarianten gehen im Prinzip wie **Induktion**:
- „IA“: Invariante gilt bei **Beginn** des Algorithmus / der Schleife
- „IV“: Die Invariante war beim Ende des **vorherigen** Schleifendurchlaufs gültig
- „IS“: Mithilfe der IV zeigen, dass die Invariante auch beim Ende des **aktuellen** Schleifendurchlaufs gültig ist
- **Achtung**: Invarianten müssen auch **nach Ende der Schleife** noch gelten!

Algorithmenanalyse

Invarianten

- Invariante finden: Manchmal offensichtlich, manchmal **Kreativität** gefragt
- Korrektheitsbeweise über Invarianten gehen im Prinzip wie **Induktion**:
- „IA“: Invariante gilt bei **Beginn** des Algorithmus / der Schleife
- „IV“: Die Invariante war beim Ende des **vorherigen** Schleifendurchlaufs gültig
- „IS“: Mithilfe der IV zeigen, dass die Invariante auch beim Ende des **aktuellen** Schleifendurchlaufs gültig ist
- **Achtung**: Invarianten müssen auch **nach Ende der Schleife** noch gelten!

Algorithmenanalyse

Invarianten

- Invariante finden: Manchmal offensichtlich, manchmal **Kreativität** gefragt
- Korrektheitsbeweise über Invarianten gehen im Prinzip wie **Induktion**:
  - „IA“: Invariante gilt bei **Beginn** des Algorithmus / der Schleife
  - „IV“: Die Invariante war beim Ende des **vorherigen** Schleifendurchlaufs gültig
  - „IS“: Mithilfe der IV zeigen, dass die Invariante auch beim Ende des aktuellen Schleifendurchlaufs gültig ist
  - **Achtung**: Invarianten müssen auch **nach Ende der Schleife** noch gelten!

Algorithmenanalyse

Invarianten

- Invariante finden: Manchmal offensichtlich, manchmal **Kreativität** gefragt
- Korrektheitsbeweise über Invarianten gehen im Prinzip wie **Induktion**:
- „IA“: Invariante gilt bei **Beginn** des Algorithmus / der Schleife
- „IV“: Die Invariante war beim Ende des vorherigen Schleifendurchlaufs gültig
- „IS“: Mithilfe der IV zeigen, dass die Invariante auch beim Ende des aktuellen Schleifendurchlaufs gültig ist
- **Achtung**: Invarianten müssen auch nach Ende der Schleife noch gelten!

Algorithmenanalyse

Invarianten

- Invariante finden: Manchmal offensichtlich, manchmal **Kreativität** gefragt
- Korrektheitsbeweise über Invarianten gehen im Prinzip wie **Induktion**:
- „IA“: Invariante gilt bei **Beginn** des Algorithmus / der Schleife
- „IV“: Die Invariante war beim Ende des **vorherigen** Schleifendurchlaufs gültig
- „IS“: Mithilfe der IV zeigen, dass die Invariante auch beim Ende des aktuellen Schleifendurchlaufs gültig ist
- **Achtung:** Invarianten müssen auch **nach Ende der Schleife** noch gelten!

Algorithmenanalyse

Invarianten

- Invariante finden: Manchmal offensichtlich, manchmal **Kreativität** gefragt
- Korrektheitsbeweise über Invarianten gehen im Prinzip wie **Induktion**:
- „IA“: Invariante gilt bei **Beginn** des Algorithmus / der Schleife
- „IV“: Die Invariante war beim Ende des **vorherigen** Schleifendurchlaufs gültig
- „IS“: Mithilfe der IV zeigen, dass die Invariante auch beim Ende des **aktuellen** Schleifendurchlaufs gültig ist
- **Achtung**: Invarianten müssen auch **nach Ende der Schleife** noch gelten!

Beispiel SelectionSort

SelectionSort

```
procedure SelectionSort(A : array[1..n] of Element)
    for i := 1 to n do
        minIndex := i
        for j := i + 1 to n do
            if A[j] < A[minIndex] then
                minIndex := j
        assert A[minIndex] = min(A[i..n])
        swap(A[i], A[minIndex])
```

Beispiel SelectionSort

SelectionSort

```
procedure SelectionSort(A : array[1..n] of Element)
    for i := 1 to n do
        invariant A[1 ... i − 1] is sorted and max(A[1 ... i − 1]) ⩽ min(A[i..n])
        minIndex := i
        for j := i + 1 to n do
            if A[j] < A[minIndex] then
                minIndex := j
        assert A[minIndex] = min(A[i..n])
        swap(A[i], A[minIndex])
```

SelectionSort – Beweis Invariante

Definiere $\max(()) := -\infty$ und $\min(()) := +\infty$.

Beweis Invariante:

$A[1 \ldots i-1]$ is sorted and $\max(A[1 \ldots i-1]) \leqslant \min(A[i..n])$

SelectionSort – Beweis Invariante

Definiere $\max(()) := -\infty$ und $\min(()) := +\infty$.

Beweis Invariante:

$A[1 \ldots i-1]$ **is** sorted **and** $\max(A[1 \ldots i-1]) \leqslant \min(A[i..n])$

IA. $(i = 1)$: $A[1..0] = ()$ ist sortiert und
$-\infty = \max(A[1..0]) \leqslant \min(A[1..n])$.

SelectionSort – Beweis Invariante

Definiere $\max(()) := -\infty$ und $\min(()) := +\infty$.

Beweis Invariante:

$A[1 \ldots i-1]$ **is** sorted **and** $\max(A[1 \ldots i-1]) \leqslant \min(A[i..n])$

IA. $(i = 1)$: $A[1..0] = ()$ ist sortiert und
$-\infty = \max(A[1..0]) \leqslant \min(A[1..n])$.

IV. *$(i > 1)$*: Die Invariante gilt zu Beginn des Durchlaufs i für ein *festes* i.

SelectionSort – Beweis Invariante

Definiere $\max(()) := -\infty$ und $\min(()) := +\infty$.

Beweis Invariante:

$A[1 \ldots i-1]$ **is** sorted **and** $\max(A[1 \ldots i-1]) \leqslant \min(A[i..n])$

IV. *($i > 1$)*: Die Invariante gilt zu Beginn des Durchlaufs i für ein *festes* i.

IS. ($i \rightsquigarrow i+1$): Laut IV ist $A[1 \ldots i-1]$ sortiert und

$\max(A[1 \ldots i-1]) \leqslant \min(A[i \ldots n])$ und $\text{minIndex} \in \{i, \ldots, n\}$

$\Rightarrow A[i-1] \leqslant A[\text{minIndex}]$ und $A[\text{minIndex}] \leqslant A[i]$.

$\Rightarrow A[\text{minIndex}]$ kann zur Fortsetzung der Sortierung problemlos nach $A[i]$ verschoben werden! Tauschen von $A[i]$, $A[\text{minIndex}]$.

$\Rightarrow A[1..i]$ ist sortiert,

$A[i] = \max(A[1..i]) \leqslant \min(A[i+1 \ldots n])$. $\square$

$\Rightarrow$ Nach dem n-ten Schleifendurchlauf gilt also: $A[1..n]$ ist sortiert.

SelectionSort – Beweis Invariante

Definiere $\max(()) := -\infty$ und $\min(()) := +\infty$.

Beweis Invariante:

$A[1 \ldots i-1]$ **is** sorted **and** $\max(A[1 \ldots i-1]) \leqslant \min(A[i..n])$

IV. *($i > 1$)*: Die Invariante gilt zu Beginn des Durchlaufs i für ein *festes* i.

IS. ($i \rightsquigarrow i+1$): Laut IV ist $A[1 \ldots i-1]$ sortiert und

$\max(A[1 \ldots i-1]) \leqslant \min(A[i \ldots n])$ und $\text{minIndex} \in \{i, \ldots, n\}$

$\Rightarrow A[i-1] \leqslant A[\text{minIndex}]$ und $A[\text{minIndex}] \leqslant A[i]$.

$\Rightarrow A[\text{minIndex}]$ kann zur Fortsetzung der Sortierung problemlos nach $A[i]$ verschoben werden! Tauschen von $A[i]$, $A[\text{minIndex}]$.

$\Rightarrow A[1..i]$ ist sortiert,

$A[i] = \max(A[1..i]) \leqslant \min(A[i+1 \ldots n])$. $\square$

$\Rightarrow$ Nach dem n-ten Schleifendurchlauf gilt also: $A[1..n]$ ist sortiert.

SelectionSort – Beweis Invariante

Definiere $\max(()) := -\infty$ und $\min(()) := +\infty$.

Beweis Invariante:

$A[1 \ldots i-1]$ **is** sorted **and** $\max(A[1 \ldots i-1]) \leqslant \min(A[i..n])$

IV. *($i > 1$)*: Die Invariante gilt zu Beginn des Durchlaufs i für ein *festes* i.

IS. ($i \rightsquigarrow i+1$): Laut IV ist $A[1 \ldots i-1]$ sortiert und

$\max(A[1 \ldots i-1]) \leqslant \min(A[i \ldots n])$ und $\text{minIndex} \in \{i, \ldots, n\}$

$\Rightarrow$ $A[i-1] \leqslant A[\text{minIndex}]$ und $A[\text{minIndex}] \leqslant A[i]$.

$\Rightarrow$ $A[\text{minIndex}]$ kann zur Fortsetzung der Sortierung problemlos nach $A[i]$ verschoben werden! Tauschen von $A[i]$, $A[\text{minIndex}]$:

$\Rightarrow$ $A[1..i]$ ist sortiert,

$A[i] = \max(A[1..i]) \leqslant \min(A[i+1 \ldots n])$. $\square$

$\Rightarrow$ Nach dem n-ten Schleifendurchlauf gilt also: $A[1..n]$ ist sortiert.

SelectionSort – Beweis Invariante

Definiere $\max(()) := -\infty$ und $\min(()) := +\infty$.

Beweis Invariante:

$A[1 \ldots i-1]$ **is** sorted **and** $\max(A[1 \ldots i-1]) \leqslant \min(A[i..n])$

IV. *($i > 1$)*: Die Invariante gilt zu Beginn des Durchlaufs i für ein *festes* i.

IS. ($i \rightsquigarrow i+1$): Laut IV ist $A[1 \ldots i-1]$ sortiert und

$\max(A[1 \ldots i-1]) \leqslant \min(A[i \ldots n])$ und $\text{minIndex} \in \{i, \ldots, n\}$

$\Rightarrow$ $A[i-1] \leqslant A[\text{minIndex}]$ und $A[\text{minIndex}] \leqslant A[i]$.

$\Rightarrow$ $A[\text{minIndex}]$ kann zur Fortsetzung der Sortierung problemlos nach $A[i]$ verschoben werden! Tauschen von $A[i]$, $A[\text{minIndex}]$:

$\Rightarrow$ $A[1..i]$ ist sortiert,

$A[i] = \max(A[1..i]) \leqslant \min(A[i+1 \ldots n])$. $\square$

$\Rightarrow$ Nach dem n-ten Schleifendurchlauf gilt also: $A[1..n]$ ist sortiert.

SelectionSort – Beweis Terminierung

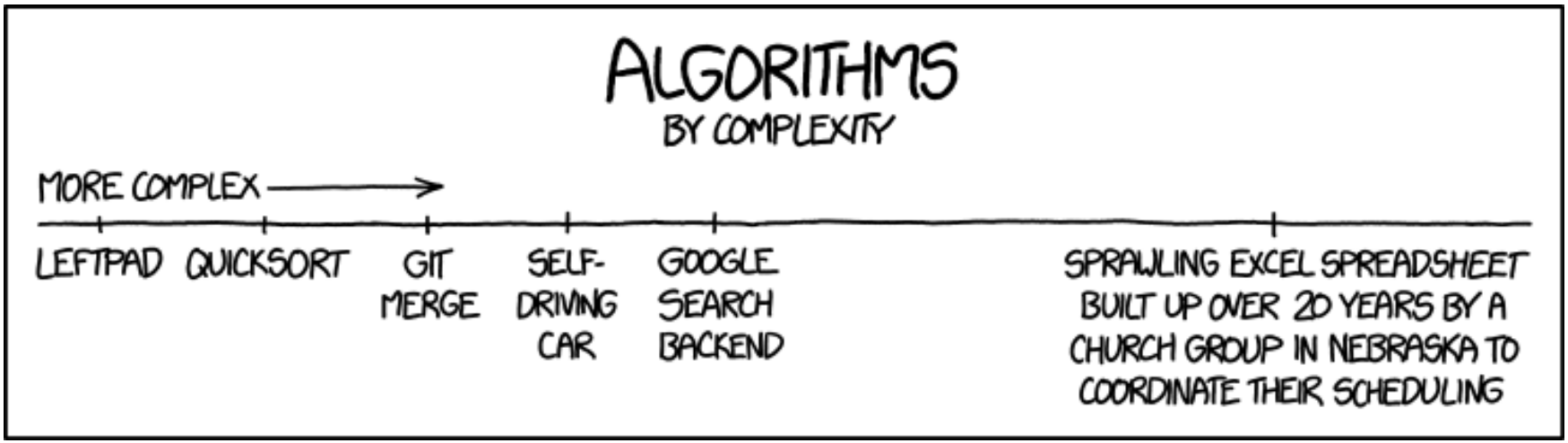
In diesem Fall trivial:

- Schleifenvariable i nach oben durch n beschränkt
- ...und wird in jedem Durchlauf inkrementiert (und sonst nicht verändert)

$\Rightarrow$ SelectionSort terminiert

$\Rightarrow$ SelectionSort funktioniert! Yay! :D

Danke für die Aufmerksamkeit! ☺



http://xkcd.com/1667

Credits

Vorgänger dieses Foliensatzes wurden erstellt von:

Christopher Hommel (urspr. Verfasser)
Daniel Jungkind