Algorithmen und Datenstrukturen

Übung 1

Stephan Friedrichs

Technische Universität Braunschweig, IBR

31. Oktober 2013

Programm für Heute

1. Organisatorisches
2. Übung
   - Probleme und Instanzen
   - Algorithmen (Pseudocode, Kontrollstrukturen, ...)
   - Problemvarianten

Organisatorisches

Diese Folien...

... braucht man nicht abzuschreiben

... stehen im Netz unter
`http://www.ibr.cs.tu-bs.de/courses/ws1314/aud/`

Apropos Vorlesungshomepage

- `http://www.ibr.cs.tu-bs.de/courses/ws1314/aud/`
- Alle Termine
- Materialien zu Vorlesung / großer Übung / kleiner Übung
- Übungsblätter
- Ankündigungen

Mailingliste

- Natürlich auf
  `http://www.ibr.cs.tu-bs.de/courses/ws1314/aud/`
- Hier sämtliche Ankündigungen!

Mailingliste

- Natürlich auf
  `http://www.ibr.cs.tu-bs.de/courses/ws1314/aud/`
- Hier sämtliche Ankündigungen!
- Fragen zum Stoff
- Auch alle Hiwis erreichbar

Mailingliste

- Natürlich auf
  `http://www.ibr.cs.tu-bs.de/courses/ws1314/aud/`
- Hier sämtliche Ankündigungen!
- Fragen zum Stoff
- Auch alle Hiwis erreichbar

1. Registrieren
2. Mail des Listenservers abwarten
3. Darauf antworten

Algorithmen und Datenstrukturen

Vorlesung
Große Übung
Kleine Übung

Vorlesung

- Neuer Inhalt
- Theorie, Beweise

Große Übung

- Aufarbeitung/Ergänzung des Vorlesungsstoffs
- Fragenstellen ist ausdrücklich erlaubt! :)

Kleine Übung

- Rückgabe/Besprechung der Hausaufgaben
- In *kleiner Runde (ca. 20/Gruppe)* über den Stoff sprechen
- Nächste Woche geht's los!

Kleine Übung – Anmeldung

- Anmeldung auf
  `http://www.ibr.cs.tu-bs.de/courses/ws1314/aud/`
- Freigeschaltet ab heute 13:00 *bis Montag 13:00*

Kleine Übung – Anmeldung

- Anmeldung auf
  `http://www.ibr.cs.tu-bs.de/courses/ws1314/aud/`
- Freigeschaltet ab heute 13:00 *bis Montag 13:00*
- Je Termin yes/ifneedbe/no
- Bitte so viele yes/ifneedbes wie möglich!

Kleine Übung – Anmeldung

- Anmeldung auf
  `http://www.ibr.cs.tu-bs.de/courses/ws1314/aud/`
- Freigeschaltet ab heute 13:00 *bis Montag 13:00*
- Je Termin yes/ifneedbe/no
- Bitte so viele yes/ifneedbes wie möglich!
- Wir werten pro Person die neueste Anmeldung
- *Kein* First come, first serve $\rightsquigarrow$ kein Stress :)
- Bei technischen Problemen: Meldet euch!
  `http://www.ibr.cs.tu-bs.de/users/sfriedr/`

Apropos Hausaufgaben...

- 7 Übungsblätter, die ersten 2 unbewertet
- 14-tägig auf
  `http://www.ibr.cs.tu-bs.de/courses/ws1314/aud/`
- Bearbeitungszeit: 14 Tage

Apropos Hausaufgaben...

- 7 Übungsblätter, die ersten 2 unbewertet
- 14-tägig auf
  `http://www.ibr.cs.tu-bs.de/courses/ws1314/aud/`
- Bearbeitungszeit: 14 Tage
- Abgabe: Bis Mittwoch 13:00, zwischen IZ 337 und IZ 338
- Rückgabe/Besprechung: 14 Tage später, kleine Übung

Apropos Hausaufgaben...

Apropos Hausaufgaben...

zusammen arbeiten, *einzeln* aufschreiben

Studienleistung: *50 % der Hausaufgabenpunkte*

Semesterübersicht

KW (ab)	VL Di+Mi	Gr. UE Do	Kl. UE Mi/Do/Fr	HA Ausgabe Mi abends	HA Abgabe Mi bis 13:00	HA Rückgabe (in kleiner UE)
43 (21.10.)						
44 (28.10.)	1, 2	1		H0a*, H0b*		
45 (4.11.)	3, 4	2	1	H1		H0a*
46 (11.11.)	5, 6					
47 (18.11.)	7, 8	3	2	H2	H1	H0b*
48 (25.11.)	9, 10					
49 (2.12.)	11, 12	4	3	H3	H2	H1
50 (9.12.)	13, 14					
51 (16.12.)	15, 16	5	4	H4	H3	H2
52 (23.12.)	Weihnachtsferien					
1 (30.12.)						
2 (6.1.)	17, 18	6	5	H5	H4	H3
3 (13.1.)	19, 20					
4 (20.1.)	21, 22	7	6		H5	H4
5 (27.1.)	23, 24					
6 (3.2.)	25, 26	8	7			H5

*) Geht nicht in die Wertung ein

Ohne Gewähr!

Klausur

Die Klausur ist am 25. 02. 2014

Fragen

Fragen sind ausdrücklich erwünscht! :)
Zum Beispiel...

- ... vor / während / nach der Vorlesung / großer Übung / kleiner Übung

Fragen

Fragen sind ausdrücklich erwünscht! :)
Zum Beispiel...

- ... vor / während / nach der Vorlesung / großer Übung / kleiner Übung
- ... auf der Mailingliste

Fragen

Fragen sind ausdrücklich erwünscht! :)
Zum Beispiel...

- ... vor / während / nach der Vorlesung / großer Übung / kleiner Übung
- ... auf der Mailingliste
- ... in Prof. Dr. Sándor P. Feketes Sprechstunde (Mittwoch 13:15 – 14:00, `http://www.ibr.cs.tu-bs.de/users/fekete/`)
- ... in meiner Sprechstunde (Nach Vereinbarung, `http://www.ibr.cs.tu-bs.de/users/sfriedr/`)

Fragerunde

Algorithmen

Probleme und Instanzen

Problem

Allgemeine Fragestellung, oft formuliert mit Hilfe von

- Eingabe (was ist bekannt?)
- Ausgabe (was ist gesucht?)

Instanz

Konkrete Eingabe eines Problems

Probleme und Instanzen

Problem

Allgemeine Fragestellung, oft formuliert mit Hilfe von

- Eingabe (was ist bekannt?)
- Ausgabe (was ist gesucht?)

Instanz

Konkrete Eingabe eines Problems

Lösung des Problems

Algorithmus

Lösung einer Instanz

Konkrete Ausgabe

Probleme und Instanzen

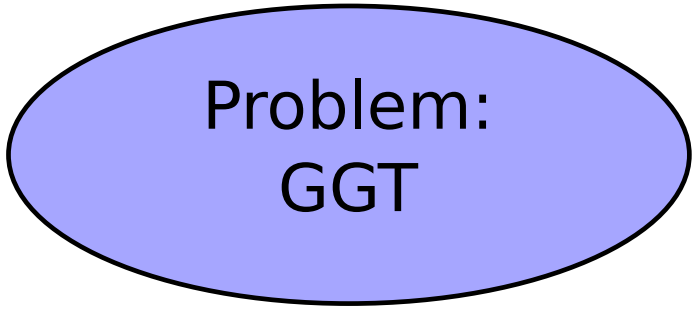
Problem und Instanzen

Lösung

Probleme und Instanzen

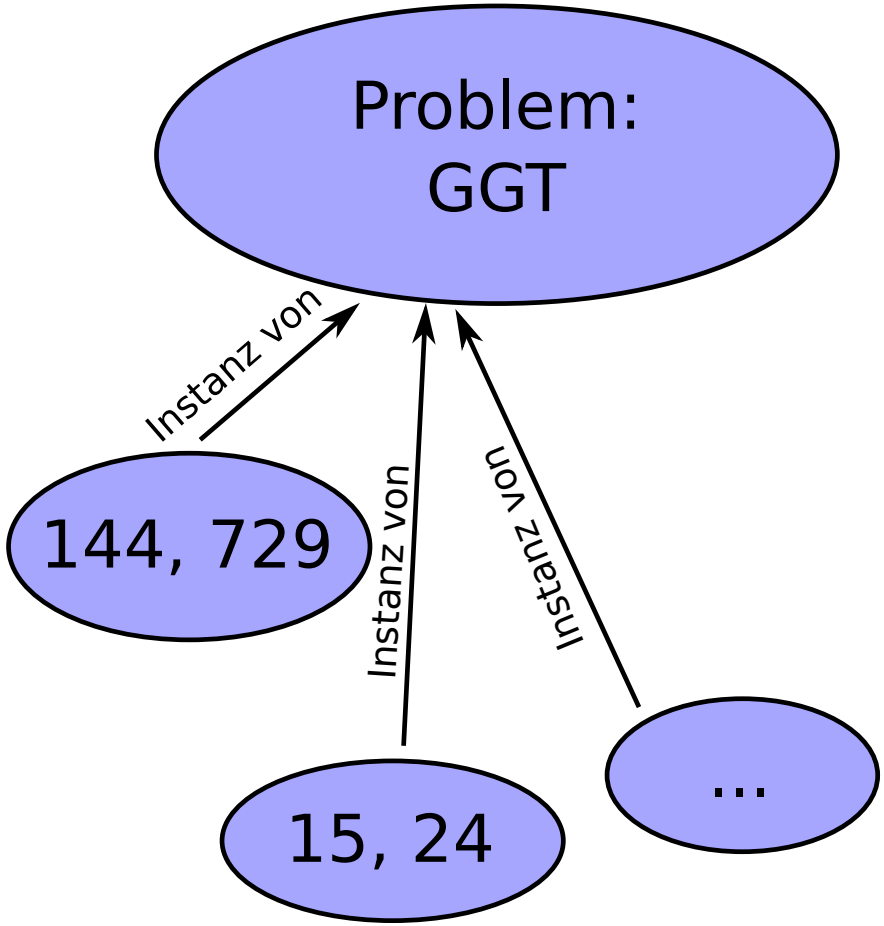
Problem und Instanzen

Lösung



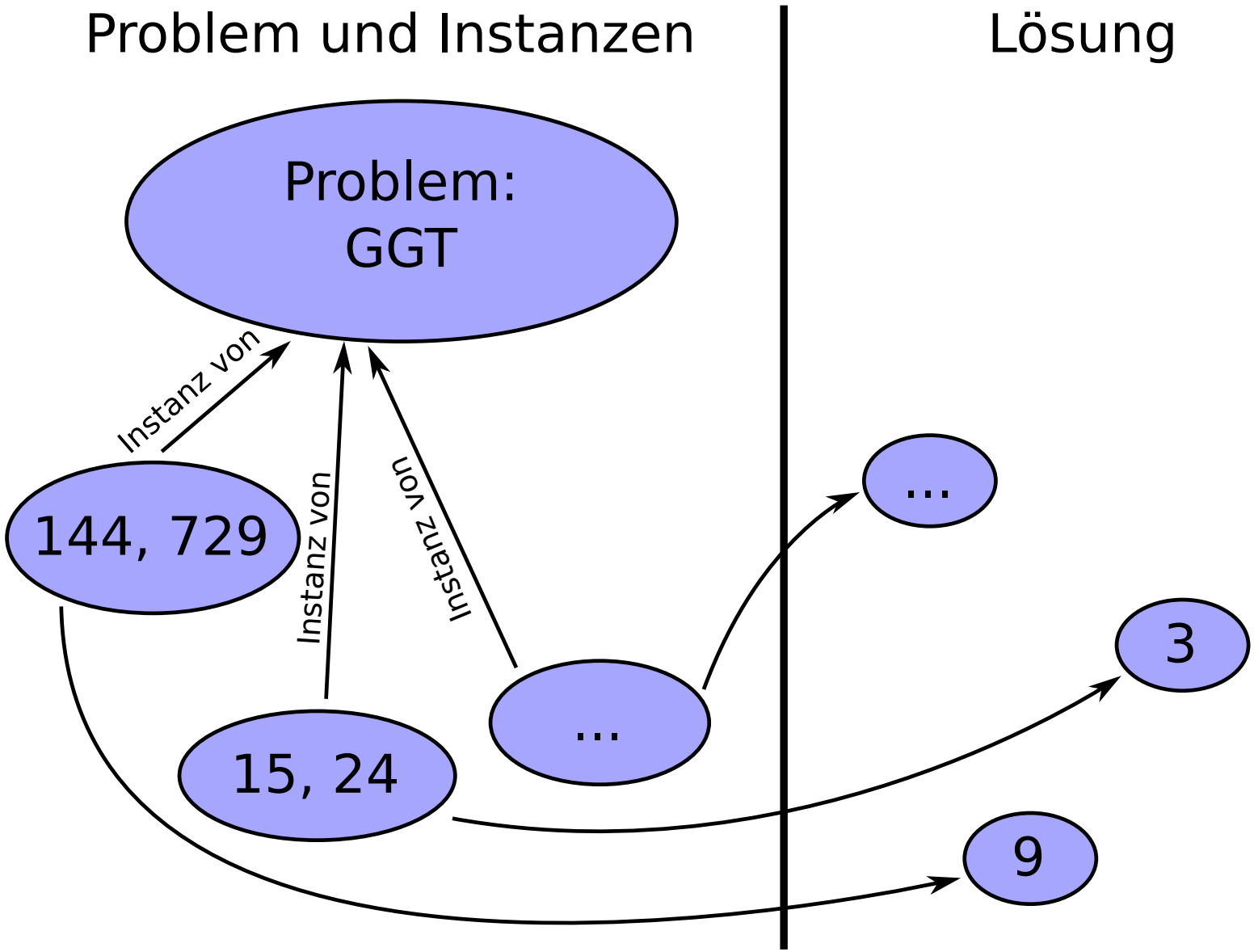
Probleme und Instanzen

Problem und Instanzen

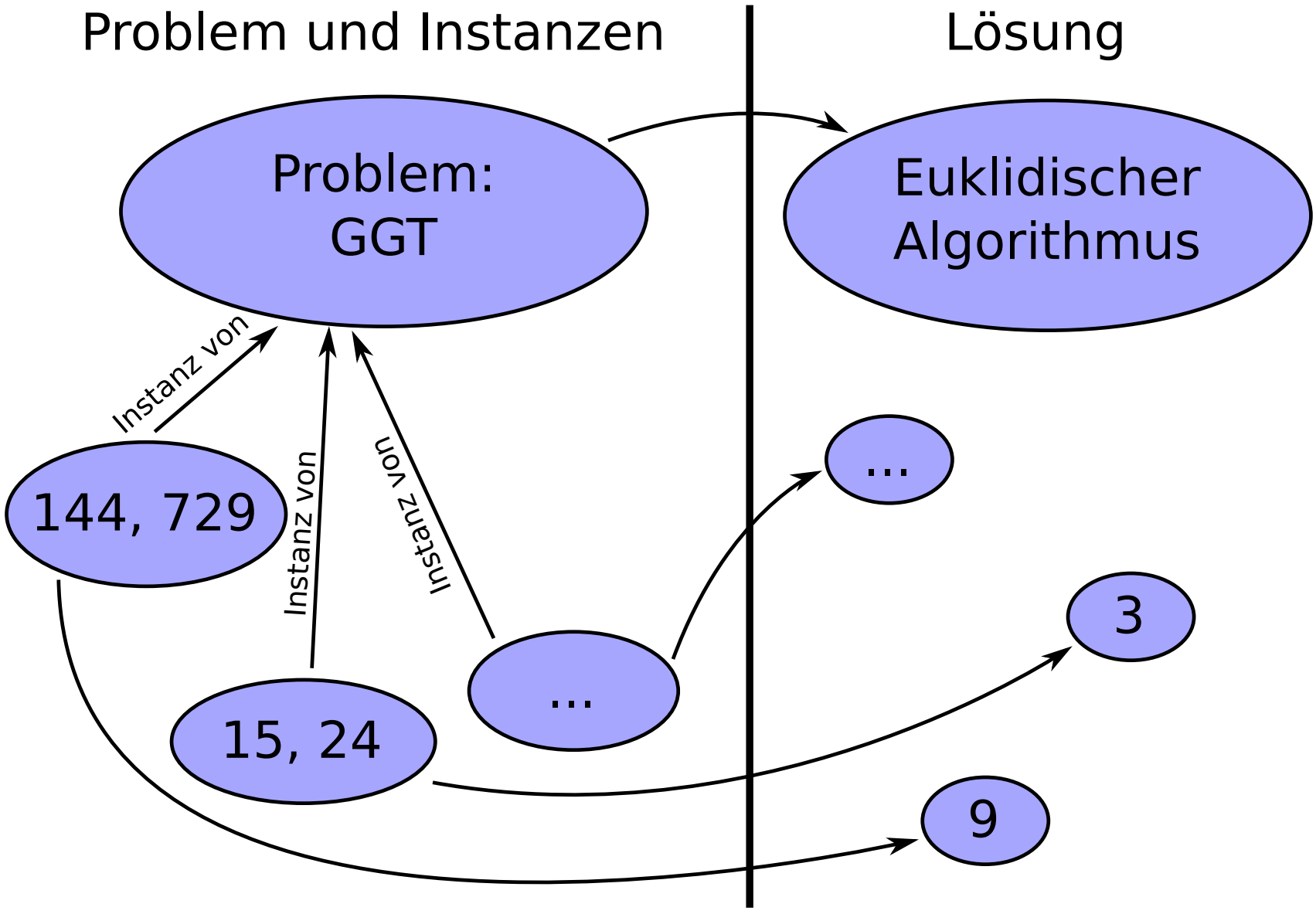


Lösung

Probleme und Instanzen



Probleme und Instanzen

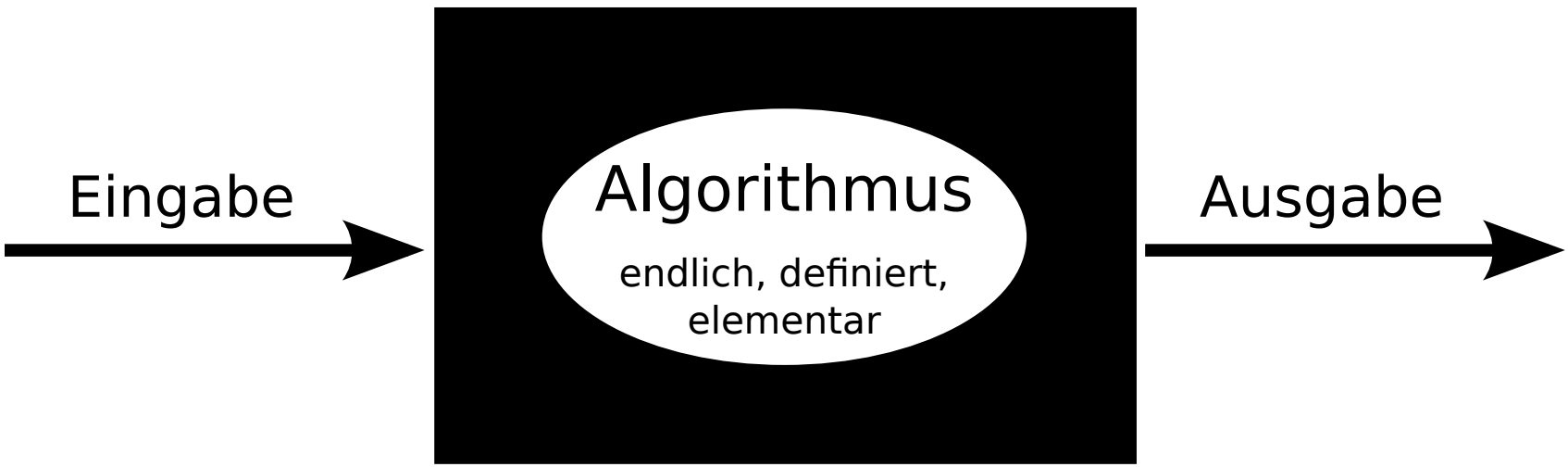


Algorithmen

Ein Algorithmus ist eine Berechnungsvorschrift, die zu *jeder Instanz eines Problems* eine Lösung liefert. Wichtig dabei:

- Endlicher Text
- Endliche Laufzeit
- Definiertheit
- Elementare Operationen

(frei nach Knuth)



Notation, Abstraktion

Wie aufschreiben?

Notation, Abstraktion

Wie aufschreiben?

Programmiersprache

- viele Details
- viel (uninteressanter) Code
- abhängig von Maschine
- (bald?) nicht mehr aktuell

$\Rightarrow$ Programmieren I und II, SEP, Teamprojekt, ...

Notation, Abstraktion

Wie aufschreiben?

Programmiersprache

- viele Details
- viel (uninteressanter) Code
- abhängig von Maschine
- (bald?) nicht mehr aktuell

$\Rightarrow$ Programmieren I und II, SEP, Teamprojekt, ...

Pseudocode

- Abstraktionsgrad frei wählbar
- Konzentration aufs Wesentliche
- Unabhängig von aktuellen Sprachen

Beispiel: Minimum zweier Zahlen

Problem

Minimum zweier Zahlen

Eingabe Zwei Zahlen $x, y \in \mathbb{R}$

Ausgabe x wenn $x \leq y$, sonst y

Instanz

$144,729.78$

Beispiel: Minimum zweier Zahlen

Problem

Minimum zweier Zahlen

Eingabe Zwei Zahlen $x, y \in \mathbb{R}$

Ausgabe x wenn $x \leq y$, sonst y

Instanz

$144,729.78$

Lösung des Problems

Nächste Folie

Lösung der Instanz

144

Beispiel: Minimum zweier Zahlen

```
1: function MIN(x, y)
2:     m ← y

3:     if x ≤ y then
4:         m ← x
5:     end if

6:     return m
7: end function
```

Lessons Learned: Variablen

Variablen liegen im Speicher und enthalten Werte (nicht nur Zahlen!).
Diese dürfen verändert werden.

1: $i \leftarrow 0$
2: $i \leftarrow 5$
3: $i \leftarrow i + 1$

4: $M \leftarrow \{0, 1, 2\}$
5: $M \leftarrow M \cup \{2, 3\}$

Lessons Learned: Fallunterscheidungen

Bool'sche Ausdrücke B_i sind entweder wahr (*true*) oder falsch (*false*).

1: **if** B_1 **then**
2: wird ausgeführt, wenn B_1 true ist
3: **end if**

Lessons Learned: Fallunterscheidungen

Bool'sche Ausdrücke B_i sind entweder wahr (*true*) oder falsch (*false*).

1: **if** B_1 **then**
2: wird ausgeführt, wenn B_1 true ist
3: **end if**

alternativ mit **else**:

1: **if** B_1 **then**
2: wird ausgeführt, wenn B_1 true ist
3: **else**
4: wird ausgeführt, wenn B_1 false ist
5: **end if**

Lessons Learned: Fallunterscheidungen

Bool'sche Ausdrücke B_i sind entweder wahr (*true*) oder falsch (*false*).

1: **if** B_1 **then**
2: wird ausgeführt, wenn B_1 true ist
3: **else if** B_2 **then**
4: wird ausgeführt, wenn B_1 false *und* B_2 true ist
5: $\vdots$
6: **else if** B_n **then**
7: wird ausgeführt, wenn alle $B_1, \ldots, B_{n-1}$ false *und* B_n true ist
8: **else**
9: wird ausgeführt, wenn alle $B_1, \ldots, B_n$ false sind
10: **end if**

Beispiel: Fakultät

Problem

Fakultät einer Zahl berechnen

Eingabe Natürliche Zahl $n \in \mathbb{N}$

Ausgabe $1 \cdot 2 \cdot \ldots \cdot (n-1) \cdot n$; oder 1 falls $n = 0$

Instanz

5

Beispiel: Fakultät

Problem

Fakultät einer Zahl berechnen

Eingabe Natürliche Zahl $n \in \mathbb{N}$

Ausgabe $1 \cdot 2 \cdot \ldots \cdot (n-1) \cdot n$; oder 1 falls $n = 0$

Instanz

5

Lösung des Problems

Nächste Folie

Lösung der Instanz

$1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$

Beispiel: Fakultät

```
1: function FACTORIAL(n)
2:     f ← 1
3:     i ← 1

4:     while i ≤ n do
5:         f ← f · i
6:         i ← i + 1
7:     end while

8:     return f
9: end function
```

Beispiel: Fakultät

```
1: function FACTORIAL(n)
2:     f ← 1
3:     i ← 1

4:     while i ≤ n do
5:         f ← f · i
6:         i ← i + 1
7:     end while

8:     return f
9: end function
```

$5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5$

Ablauf für $n = 5$

Zeitpunkt	f	i

Beispiel: Fakultät

```
1: function FACTORIAL(n)
2:     f ← 1
3:     i ← 1

4:     while i ≤ n do
5:         f ← f · i
6:         i ← i + 1
7:     end while

8:     return f
9: end function
```

$5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5$

Ablauf für $n = 5$

Zeitpunkt	f	i
Vor der Schleife	1	1

Beispiel: Fakultät

```
1: function FACTORIAL(n)
2:     f ← 1
3:     i ← 1
4:     while i ≤ n do
5:         f ← f · i
6:         i ← i + 1
7:     end while
8:     return f
9: end function
```

$5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5$

Ablauf für $n = 5$

Zeitpunkt	f	i
Vor der Schleife	1	1
Nach dem 1. Durchlauf	1	2

Beispiel: Fakultät

```
1: function FACTORIAL(n)
2:     f ← 1
3:     i ← 1
4:     while i ≤ n do
5:         f ← f · i
6:         i ← i + 1
7:     end while
8:     return f
9: end function
```

$5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5$

Ablauf für $n = 5$

Zeitpunkt	f	i
Vor der Schleife	1	1
Nach dem 1. Durchlauf	1	2
Nach dem 2. Durchlauf	2	3

Beispiel: Fakultät

```
1: function FACTORIAL(n)
2:     f ← 1
3:     i ← 1
4:     while i ≤ n do
5:         f ← f · i
6:         i ← i + 1
7:     end while
8:     return f
9: end function
```

$5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5$

Ablauf für $n = 5$

Zeitpunkt	f	i
Vor der Schleife	1	1
Nach dem 1. Durchlauf	1	2
Nach dem 2. Durchlauf	2	3
Nach dem 3. Durchlauf	6	4

Beispiel: Fakultät

```
1: function FACTORIAL(n)
2:     f ← 1
3:     i ← 1
4:     while i ≤ n do
5:         f ← f · i
6:         i ← i + 1
7:     end while
8:     return f
9: end function
```

$5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5$

Ablauf für $n = 5$

Zeitpunkt	f	i
Vor der Schleife	1	1
Nach dem 1. Durchlauf	1	2
Nach dem 2. Durchlauf	2	3
Nach dem 3. Durchlauf	6	4
Nach dem 4. Durchlauf	24	5

Beispiel: Fakultät

```
1: function FACTORIAL(n)
2:     f ← 1
3:     i ← 1
4:     while i ≤ n do
5:         f ← f · i
6:         i ← i + 1
7:     end while
8:     return f
9: end function
```

$5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5$

Ablauf für $n = 5$

Zeitpunkt	f	i
Vor der Schleife	1	1
Nach dem 1. Durchlauf	1	2
Nach dem 2. Durchlauf	2	3
Nach dem 3. Durchlauf	6	4
Nach dem 4. Durchlauf	24	5
Nach dem 5. Durchlauf	120	6

Beispiel: Fakultät

```
1: function FACTORIAL(n)
2:     f ← 1
3:     i ← 1
4:     while i ≤ n do
5:         f ← f · i
6:         i ← i + 1
7:     end while
8:     return f
9: end function
```

$5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5$

Ablauf für $n = 5$

Zeitpunkt	f	i
Vor der Schleife	1	1
Nach dem 1. Durchlauf	1	2
Nach dem 2. Durchlauf	2	3
Nach dem 3. Durchlauf	6	4
Nach dem 4. Durchlauf	24	5
Nach dem 5. Durchlauf	120	6

Nach dem 5. Durchlauf gilt $i \leq n$ nicht mehr

Lessons Learned: Schleifen

Führt einen Codeblock wiederholt aus, bis die bool'sche Bedingung B nicht mehr gilt

1: **while** B **do**
2: wird so oft ausgeführt, bis B false ist
3: **end while**

For-Notation

Typische Abkürzung für Standardschleifen:

Ursprüngliche Version:

```
1: function FACTORIAL(n)
2:     f ← 1
3:     i ← 1
4:     while i ≤ n do
5:         f ← f · i
6:         i ← i + 1
7:     end while
8:     return f
9: end function
```

For-Notation

Typische Abkürzung für Standardschleifen:

Ursprüngliche Version:

```
1: function FACTORIAL(n)
2:     f ← 1
3:     i ← 1

4:     while i ≤ n do
5:         f ← f · i
6:         i ← i + 1
7:     end while

8:     return f
9: end function
```

For-Notation

Typische Abkürzung für Standardschleifen:

Kurzform I:

```
1: function FACTORIAL(n)
2:     f ← 1
3:     for i ← 1, ..., n do
4:         f ← f · i
5:     end for
6:     return f
7: end function
```

Ursprüngliche Version:

```
1: function FACTORIAL(n)
2:     f ← 1
3:     i ← 1
4:     while i ≤ n do
5:         f ← f · i
6:         i ← i + 1
7:     end while
8:     return f
9: end function
```

For-Notation

Typische Abkürzung für Standardschleifen:

Kurzform II:

```
1: function FACTORIAL(n)
2:     f ← 1
3:     for all i ∈ {1, ..., n} do
4:         f ← f · i
5:     end for
6:     return f
7: end function
```

Ursprüngliche Version:

```
1: function FACTORIAL(n)
2:     f ← 1
3:     i ← 1
4:     while i ≤ n do
5:         f ← f · i
6:         i ← i + 1
7:     end while
8:     return f
9: end function
```

Beispiel: Minimum aus Menge

Problem

Minimum in einer Menge von Zahlen finden

Eingabe Reelle Zahlen $a_1, \ldots, a_n$ (mindestens eine, $n \geq 1$)

Ausgabe Dasjenige a_i mit $a_i \leq a_j$ für alle $j = 1, \ldots, n$

Instanz

$a_1 = 4, a_2 = 1, a_3 = 5, a_4 = -7, a_5 = 0.5$

Beispiel: Minimum aus Menge

Problem

Minimum in einer Menge von Zahlen finden

Eingabe Reelle Zahlen $a_1, \ldots, a_n$ (mindestens eine, $n \geq 1$)

Ausgabe Dasjenige a_i mit $a_i \leq a_j$ für alle $j = 1, \ldots, n$

Instanz

$a_1 = 4, a_2 = 1, a_3 = 5, a_4 = -7, a_5 = 0.5$

Lösung des Problems

Nächste Folie

Lösung der Instanz

-7

Beispiel: Minimum aus Menge

1: **function** MINIMUM$(a_1, \ldots, a_n)$
2: $\quad m \leftarrow a_1$

3: $\quad$ **for** $i \leftarrow 2, \ldots, n$ **do**
4: $\quad\quad m \leftarrow \text{MIN}(m, a_i)$ ▷ Aufruf des Algorithmus' von oben!
5: $\quad$ **end for**

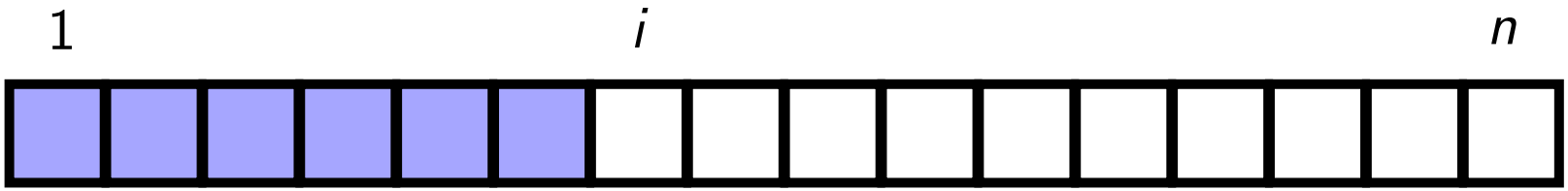
6: $\quad$ **return** m
7: **end function**

Beispiel: Minimum aus Menge

1: **function** MINIMUM$(a_1, \ldots, a_n)$
2: $\quad m \leftarrow a_1$

3: $\quad$ **for** $i \leftarrow 2, \ldots, n$ **do**
4: $\quad\quad m \leftarrow \text{MIN}(m, a_i)$ ▷ Aufruf des Algorithmus' von oben!
5: $\quad$ **end for**

6: $\quad$ **return** m
7: **end function**



$m = \min\{a_1, \ldots, a_{i-1}\}$

Lessons Learned: Funktionsaufrufe

Eben:

$\vdots$

$m \leftarrow \text{MIN}(m, a_i)$

$\vdots$

Ganz am Anfang:

```
1: function MIN(x, y)
2:     m ← y
3:     if x ≤ y then
4:         m ← x
5:     end if
6:     return m
7: end function
```

Beispiel: Zahl in Menge suchen

Problem

Herausfinden, ob sich eine Zahl x in einer gegebenen Menge von Zahlen befindet

Eingabe Reelle Zahl x, reelle Zahlen $a_1, \ldots, a_n$ (mindestens eine, $n \geq 1$)

Ausgabe $\textsc{True}$, falls $x = a_i$ für ein $1 \leq i \leq n$; $\textsc{False}$ sonst

Instanz

$x = 6, \quad a_1 = 4, a_2 = 1, a_3 = 5, a_4 = -7, a_5 = 0.5$

Beispiel: Zahl in Menge suchen

Problem

Herausfinden, ob sich eine Zahl x in einer gegebenen Menge von Zahlen befindet

Eingabe Reelle Zahl x, reelle Zahlen $a_1, \ldots, a_n$ (mindestens eine, $n \geq 1$)

Ausgabe $\textsc{True}$, falls $x = a_i$ für ein $1 \leq i \leq n$; $\textsc{False}$ sonst

Instanz

$x = 6, \quad a_1 = 4, a_2 = 1, a_3 = 5, a_4 = -7, a_5 = 0.5$

Lösung des Problems

Nächste Folie

Lösung der Instanz

$\textsc{False}$

Beispiel: Zahl in Menge suchen

```
1: function FIND(x, a_1, ..., a_n)
2:     for i ← 1, ..., n do
3:         if x = a_i then
4:             return TRUE
5:         end if
6:     end for

7:     return FALSE
8: end function
```

Beispiel: Zahl in Menge suchen

```
1: function FIND(x, a_1, ..., a_n)
2:     for i ← 1, ..., n do
3:         if x = a_i then
4:             return TRUE
5:         end if
6:     end for

7:     return FALSE
8: end function
```

Das ist langsam, im schlimmsten Fall *n Vergleiche!*

Beispiel: Zahl in Menge suchen

```
1: function FIND(x, a_1, ..., a_n)
2:     for i ← 1, ..., n do
3:         if x = a_i then
4:             return TRUE
5:         end if
6:     end for

7:     return FALSE
8: end function
```

Das ist langsam, im schlimmsten Fall *n Vergleiche!*

Geht das schneller?

Beispiel: Zahl in Menge suchen

```
1: function FIND(x, a_1, ..., a_n)
2:     for i ← 1, ..., n do
3:         if x = a_i then
4:             return TRUE
5:         end if
6:     end for

7:     return FALSE
8: end function
```

Das ist langsam, im schlimmsten Fall *n Vergleiche!*

Geht das schneller?

Nein, x könnte ja überall sein!

Problemvariante

Es geht so nicht schneller...

Problemvariante

Es geht so nicht schneller...

$\Rightarrow$ Also das Problem verändern!

Beispiel: Zahl in sortierter Menge suchen

Problem

Herausfinden, ob sich eine Zahl x in einer *sortierten* Menge von Zahlen befindet

Eingabe Reelle Zahl x, reelle Zahlen $a_1 \leq \cdots \leq a_n$ (mindestens eine, $n \geq 1$)

Ausgabe $\textsc{True}$, falls $x = a_i$ für ein $1 \leq i \leq n$; $\textsc{False}$ sonst

Instanz

$x = 6, \quad a_1 = -7, a_2 = 0.5, a_3 = 1, a_4 = 4, a_5 = 5$

Beispiel: Zahl in sortierter Menge suchen

Problem

Herausfinden, ob sich eine Zahl x in einer *sortierten* Menge von Zahlen befindet

Eingabe Reelle Zahl x, reelle Zahlen $a_1 \leq \cdots \leq a_n$ (mindestens eine, $n \geq 1$)

Ausgabe $\textsc{True}$, falls $x = a_i$ für ein $1 \leq i \leq n$; $\textsc{False}$ sonst

Instanz

$x = 6, \quad a_1 = -7, a_2 = 0.5, a_3 = 1, a_4 = 4, a_5 = 5$

Lösung des Problems

Nächste Folie

Lösung der Instanz

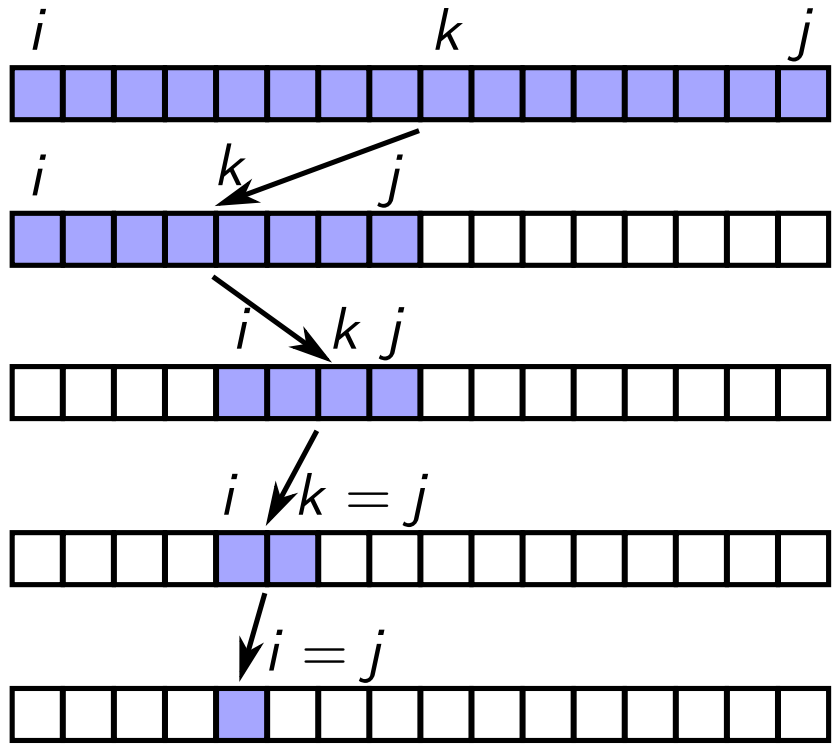
$\textsc{False}$

Beispiel: Zahl in sortierter Menge suchen

```
1:  function BINARY_SEARCH(x, a_1, ..., a_n)
2:      i ← 1
3:      j ← n
4:      while i < j do
5:          k ← ⌈(i+j)/2⌉
6:          if x < a_k then
7:              j ← k − 1
8:          else
9:              i ← k
10:         end if
11:     end while
12:     return a_i = x
13: end function
```

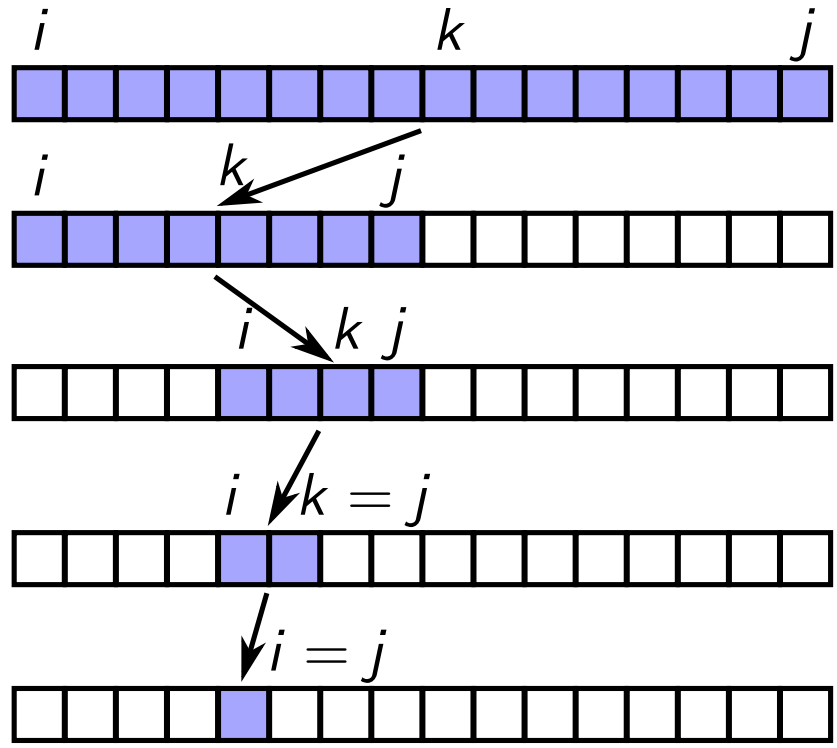
Beispiel: Zahl in sortierter Menge suchen

```
1:  function
    BINARY_SEARCH(x, a_1, ..., a_n)
2:      i ← 1
3:      j ← n
4:      while i < j do
5:          k ← ⌈(i+j)/2⌉
6:          if x < a_k then
7:              j ← k − 1
8:          else
9:              i ← k
10:         end if
11:     end while
12:     return a_i = x
13: end function
```



Beispiel: Zahl in sortierter Menge suchen

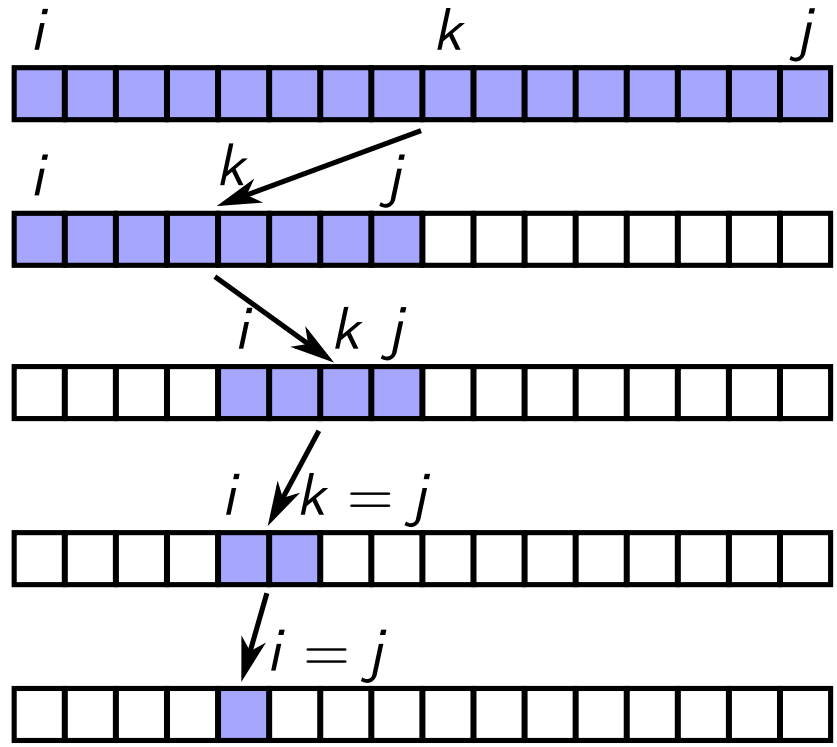
```
1:  function BINARY_SEARCH(x, a_1, ..., a_n)
2:      i ← 1
3:      j ← n
4:      while i < j do
5:          k ← ⌈(i+j)/2⌉
6:          if x < a_k then
7:              j ← k − 1
8:          else
9:              i ← k
10:         end if
11:     end while
12:     return a_i = x
13: end function
```



- $a_i \leq x \leq a_j$
  (oder $x < a_1$ oder $a_n < x$)

Beispiel: Zahl in sortierter Menge suchen

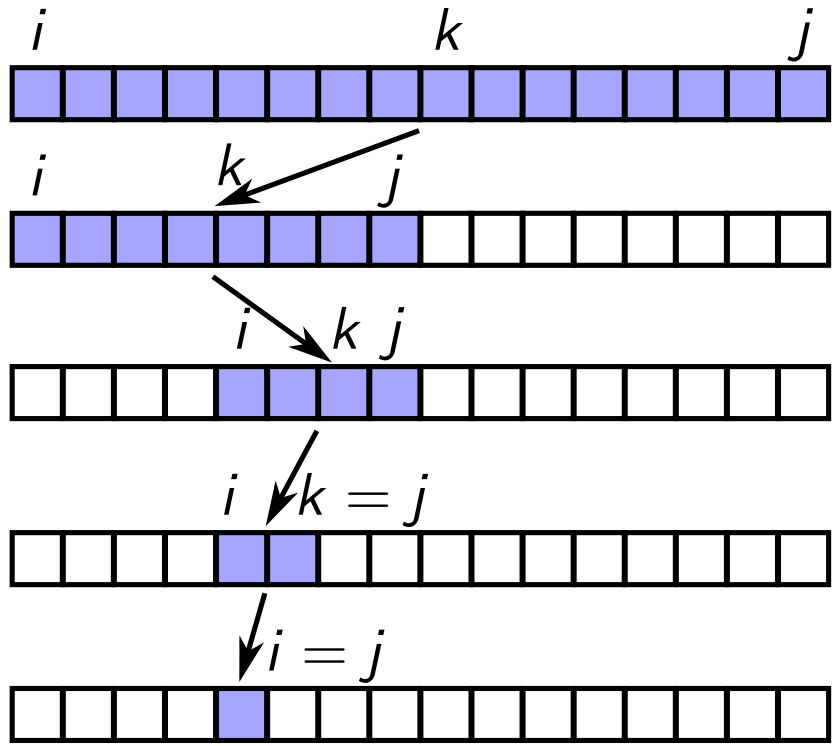
```
1:  function BINARY_SEARCH(x, a_1, ..., a_n)
2:      i ← 1
3:      j ← n
4:      while i < j do
5:          k ← ⌈(i+j)/2⌉
6:          if x < a_k then
7:              j ← k − 1
8:          else
9:              i ← k
10:         end if
11:     end while
12:     return a_i = x
13: end function
```



- $a_i \leq x \leq a_j$
  (oder $x < a_1$ oder $a_n < x$)
- $(j - i)$ halbiert sich immer

Beispiel: Zahl in sortierter Menge suchen

```
1:  function BINARY_SEARCH(x, a_1, ..., a_n)
2:      i ← 1
3:      j ← n
4:      while i < j do
5:          k ← ⌈(i+j)/2⌉
6:          if x < a_k then
7:              j ← k − 1
8:          else
9:              i ← k
10:         end if
11:     end while
12:     return a_i = x
13: end function
```



- $a_i \leq x \leq a_j$
  (oder $x < a_1$ oder $a_n < x$)
- $(j - i)$ halbiert sich immer
- Nur $\log_2(n)$ *Vergleiche!*

Lessons Learned: Problemvarianten

Geänderte Probleme ermöglichen/erfordern andere Algorithmen!

Programm für Heute

1. Organisatorisches
2. Übung
   - Probleme und Instanzen
   - Algorithmen (Pseudocode, Kontrollstrukturen, ...)
   - Problemvarianten

Fragen?