Rekursion

Martin Steffen

Autumn 2017

Rekursion

- Lateinisch: *recurrere*

> 1. rekursiv = (zu bekannten Werten) *zurückgehend*
>
> —“Deutscher Wortschatz”, U. Leipzig

Rekursion

Rückführung eines Problems auf ein einfacheres Problem

Martin Steffen

Rekursion

- Lateinisch: *recurrere*

1. rekursiv = (zu bekannten Werten) *zurückgehend*

"Deutscher Wortschatz", U. Leipzig

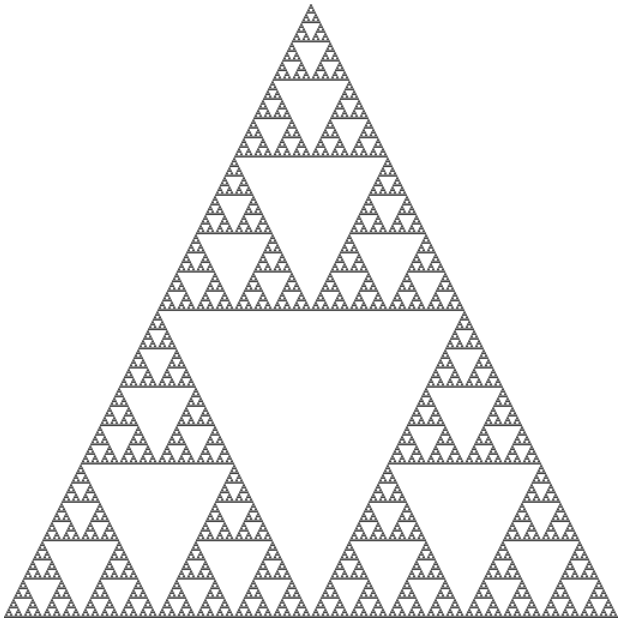
Rekursion

Rückführung eines Problems auf ein (einfachere) Version seiner selbst

- Selbsbezüglichkeit: rekursive Problemstellungen, Algorithmen, Definitionen, Datenstrukturen ...

Martin Steffen

Rekursion



Martin Steffen

Multiplikation

Martin Steffen

Aus der Grundschule: $a \times b$

"a mal b" bedeutet: addiere b zu sich selbst und zwar a-fach.

$$a \times b = \underbrace{b + b + \ldots + b}_{a \geq 0}$$

Example (4×3)

3

plus 3 gibt 6,

plus 3 gibt 9,

plus 3 gibt 12, und fertig.

Multiplikation

Martin Steffen

Aus der Grundschule: $a \times b$

"a mal b" bedeutet: addiere b zu sich selbst und zwar a-fach.

$$a \times b = 0 + \underbrace{b + b + \ldots + b}_{a \geq 0}$$

Example (4×3)

3
plus 3 gibt 6,
plus 3 gibt 9,
plus 3 gibt 12, und fertig.

Multiplikation

Martin Steffen

Aus der Grundschule: $a \times b$

"a mal b" bedeutet: addiere b zu sich selbst und zwar a-fach.

$$a \times b = 0 + \underbrace{b + b + \ldots + b}_{a \geq 0} = \sum_{i=1}^{a} b$$

Example (4×3)

3
plus 3 gibt 6,
plus 3 gibt 9,
plus 3 gibt 12, und fertig.

Als statische Methode

```
public class Timesiter {
    public static long times_iter(int a, int b) {
        int r = 0;
        for  (int i = 1; i<= a;i++) {
            r = r + b;
        };
        return r;
    }

  public static void main(String args[]) {
      System.out.println(times_iter (4, 3));
  }
}
```

Als statische Methode

```
public class Timesiter {
    public static long times_iter(int a, int b) {
        int r = 0;
        for  (int i = 1; i<= a;i++) {
            r = r + b;
        };
        return r;
    }

  public static void main(String args[]) {
      System.out.println(times_iter (4, 3));
  }
}
```

Als statische Methode (2)

```
public static long times_iter(int a, int b) {
    int r = 0;
    for  (int i = 1; i<= a;i++) {
        r = plus(r,b);
    };
    return r;
}
public static int plus(int x, int y) {
    return x + y;}
```

Der Blick auf Wesentliche

```
times_iter(a, b) {
  int r = 0;
  for  (i = 1; i<= a;i++) {
       r = plus(r,b);  };
  return r;
}

plus(x, y) { return x + y;}
```

$$a \times b = 0 + \underbrace{b + b + \ldots + b}_{a \geq 0} \quad = \quad \sum_{i=1}^{a} b$$

Multiplikation: ein weiteres Mal

$$a \times b = 0 + \underbrace{b + b + \ldots + b}_{a \geq 0} = \sum_{i=1}^{a} b$$

Martin Steffen

Multiplikation: ein weiteres Mal

$$a \times b = b + \underbrace{b + \ldots + b}_{a-1 \text{ “mal” } b} \quad = \quad b + ((a-1) \times b)$$

Selbstbezügliche Definition (= rekursiv)

Multiplikation berechnet unter Zuhilfenahme von Multiplikation (und Addition)

Martin Steffen

Multiplikation: ein weiteres Mal

$$a \times b = b + \underbrace{b + \ldots + b}_{a-1 \text{ “mal” b}} \quad = \quad b + ((a-1) \times b)$$

Selbstbezügliche Definition (= rekursiv)

Multiplikation von $a \times b$ berechnet unter Zuhilfenahme von Multiplikation von $a - 1 \times b$ (und Addition)

Martin Steffen

In Java

```
public static long times(int a, int b) {
    if (a == 0) {
        return 0;
    } else {
        return b + times (a-1, b);
    }
}
```

Martin Steffen

In Java

```
times(a, b) {
        if  (a == 0) {
            return 0;
        } else {
            return b + times (a-1, b);
        }
    }
```

$$a \times b = b + ((a-1) \times b)$$

Martin Steffen

In Java

```
times(a, b) {
        if  (a == 0) {
            return 0;
        } else {
            return b + times (a-1, b);
        }
    }
```

$$a \times b \Leftarrow b + ((a-1) \times b)$$

Martin Steffen

Und für negative Zahlen?

```
public static long times(int a, int b) {
    if (a >= 0) {
        if (a == 0) {
            return 0;
        } else {
            return b + times (a-1, b);
        }
    } else {
        return - (times (-a, b));
    }
}
```

Zum Mitschreiben

Martin Steffen

Rekursion

Eine rekursive Methode ist definiert mittels ihrer selbst.

Zum Mitschreiben

Martin Steffen

Rekursion

Eine rekursive Methode ist definiert mittels ihrer selbst.

"Vernünftige Rekursion" (= Terminierung)

Zur Lösung eines Problems:

1. Definiere eine Methode mittels einer Anwendung ihrer selbst auf eine "einfachere" Version des Problems
2. und: es gibt ein einfachstes Problem, welches direkt (= ohne weitere Rekursion) lösbar ist, sodaß die Rekursion "den Ausstieg findet".

Zum Mitschreiben

Martin Steffen

Rekursion

Eine rekursive Methode ist definiert mittels ihrer selbst.

"Vernünftige Rekursion" (= Terminierung)

Zur Lösung eines Problems:

1. Definiere eine Methode mittels ~~einer~~ Anwendung ihrer selbst auf ~~eine~~ "einfachere" Versionen des Problems
2. und: es gibt ~~ein~~ einfachste Probleme, welche~~s~~ direkt (= ohne weitere Rekursion) lösbar ~~ist~~ sind, sodaß die Rekursion "den Ausstieg findet".

Fakultät

Martin Steffen

Iterativ: “1 mal 2 mal 3 usw. bis n”

$$n! = 1 \times 2 \times \ldots \times (n-1) \times n = \prod_{i=1}^{n} i$$

Fakultät

Martin Steffen

Iterativ: "1 mal 2 mal 3 usw. bis n"

$$n! = 1 \times 2 \times \ldots \times (n-1) \times n = \prod_{i=1}^{n} i$$

```
factorial_iter(n) {
  long result = 1;
  for  (i = 1; i<= n;i++) {
     result = result * i;
  };
  return result;
```

Das Ganze nochmal rekursiv

$$n! = n \times (n-1) \times \ldots \times 2 \times 1$$

Martin Steffen

Das Ganze nochmal rekursiv

$$n! = n \times \underbrace{(n-1) \times \ldots \times 2 \times 1}_{(n-1)!}$$

```
factorial(n) {
    if (n == 1) return 1;
    return n * factorial(n-1);
}
```

Martin Steffen

Zum Vergleich

```
factorial_iter(n) {
  long result = 1;
  for (i = 1; i<= n;i++) {
     result = result * i;
  };
  return result;
}
```

- Variable `result` Schritt für Schritt (= iterativ) aktualisiert

```
factorial(n) {
   if (n == 1) return 1;
   return n * factorial(n-1);
}
```

- Variable `result` lokal im Methodenrumpf
- genau *ein* Wert pro Aufruf

Zum Vergleich

```
factorial_iter(n) {
  long result = 1;
  for (i = 1; i<= n;i++) {
    result = result * i;
  };
  return result;
}
```

- Variable `result` Schritt für Schritt (= iterativ) aktualisiert

```
factorial(n) {
  long result;
  if (n == 1) { result = 1; }
  else {
    result = n * factorial(n-1);
  };
  return result;
}
```

- Variable `result` lokal im Methodenrumpf
- genau *ein* Wert pro Aufruf

Zur Laufzeit

$fac(5)$

Martin Steffen

Zur Laufzeit

$fac(5)$

$5 \times$

Martin Steffen

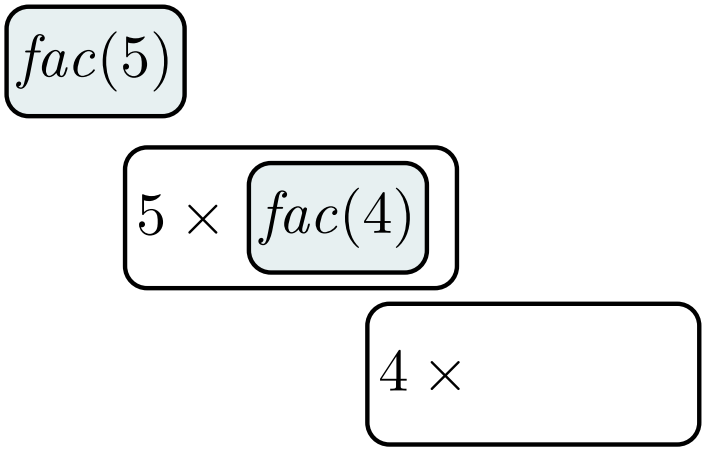
Zur Laufzeit

$$\boxed{fac(5)}$$

$$5 \times \boxed{fac(4)}$$

Martin Steffen

Zur Laufzeit



Martin Steffen

Zur Laufzeit

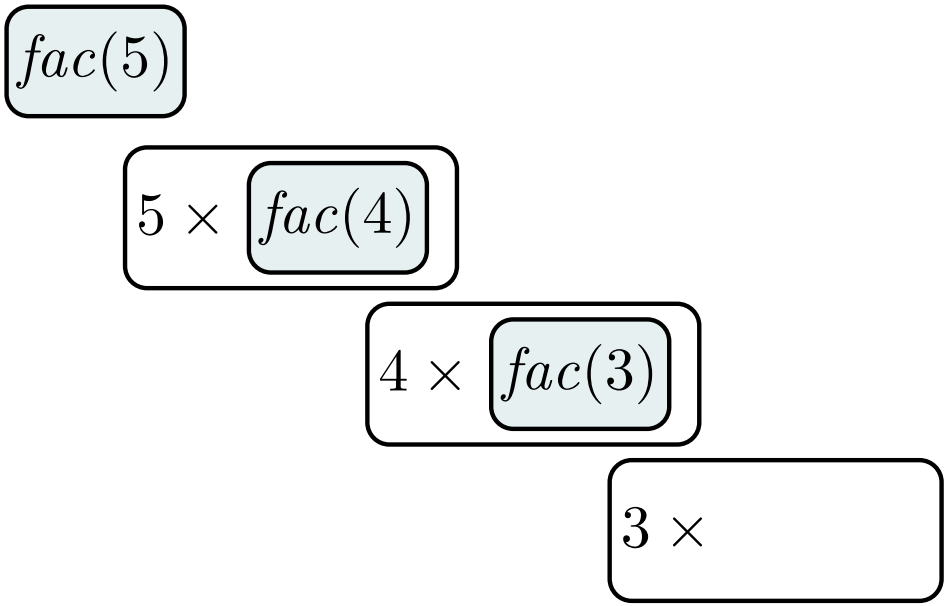
$$fac(5)$$

$$5 \times fac(4)$$

$$4 \times fac(3)$$

Martin Steffen

Zur Laufzeit



Martin Steffen

Zur Laufzeit

$fac(5)$

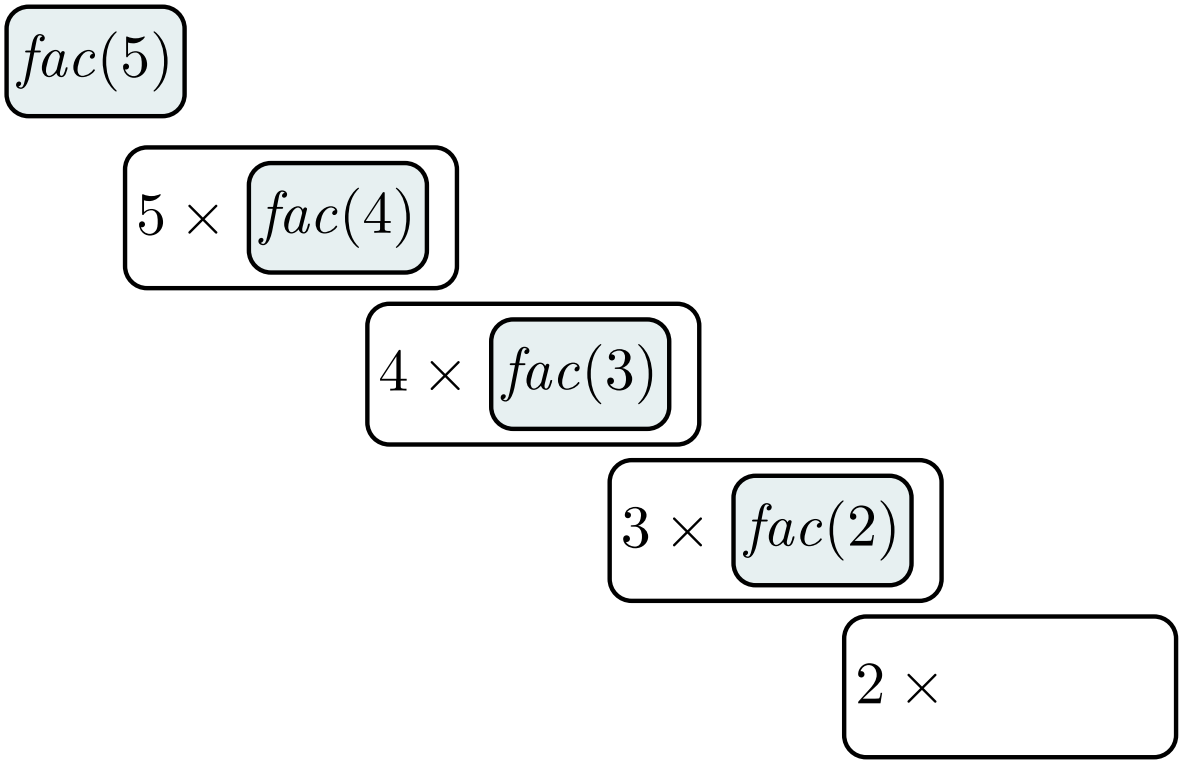
$5 \times fac(4)$

$4 \times fac(3)$

$3 \times fac(2)$

Martin Steffen

Zur Laufzeit



Martin Steffen

Zur Laufzeit

$fac(5)$

$5 \times fac(4)$

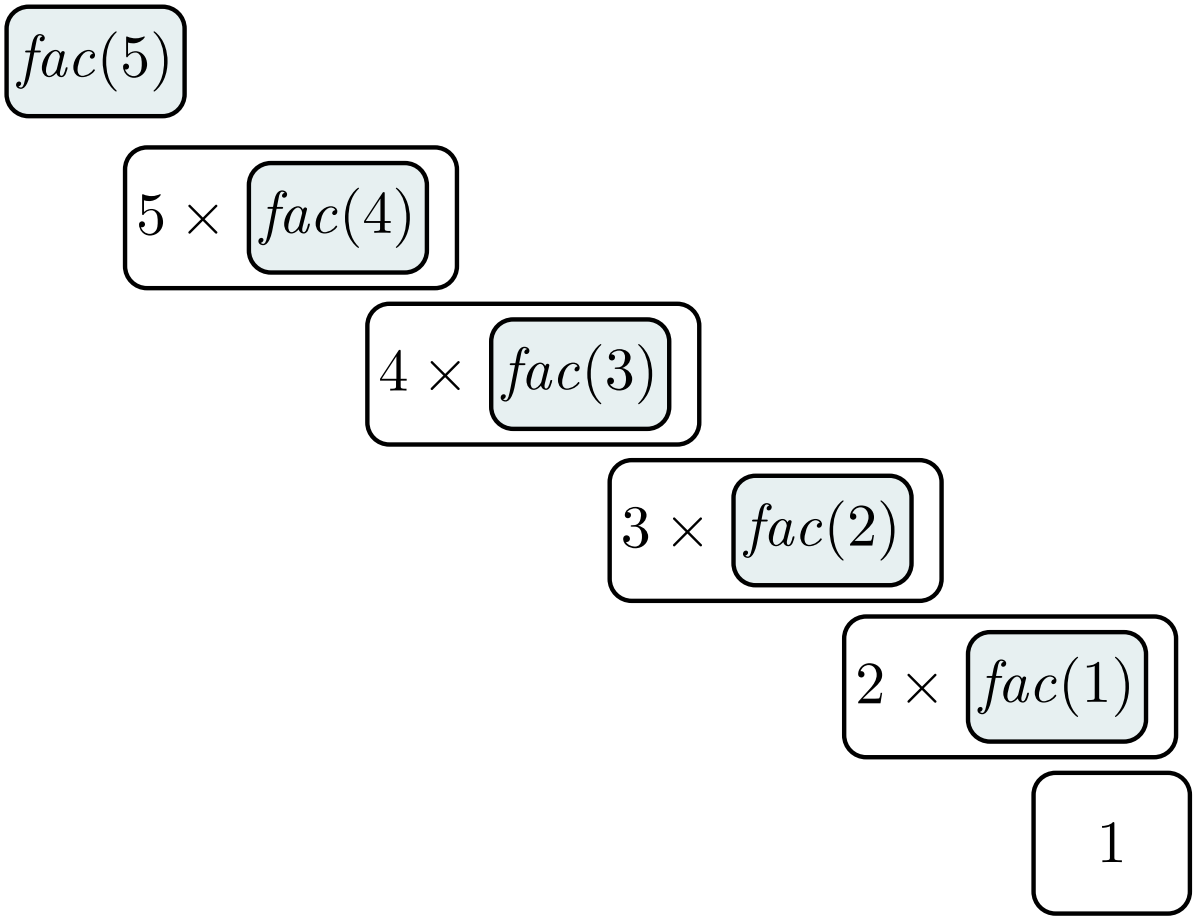
$4 \times fac(3)$

$3 \times fac(2)$

$2 \times fac(1)$

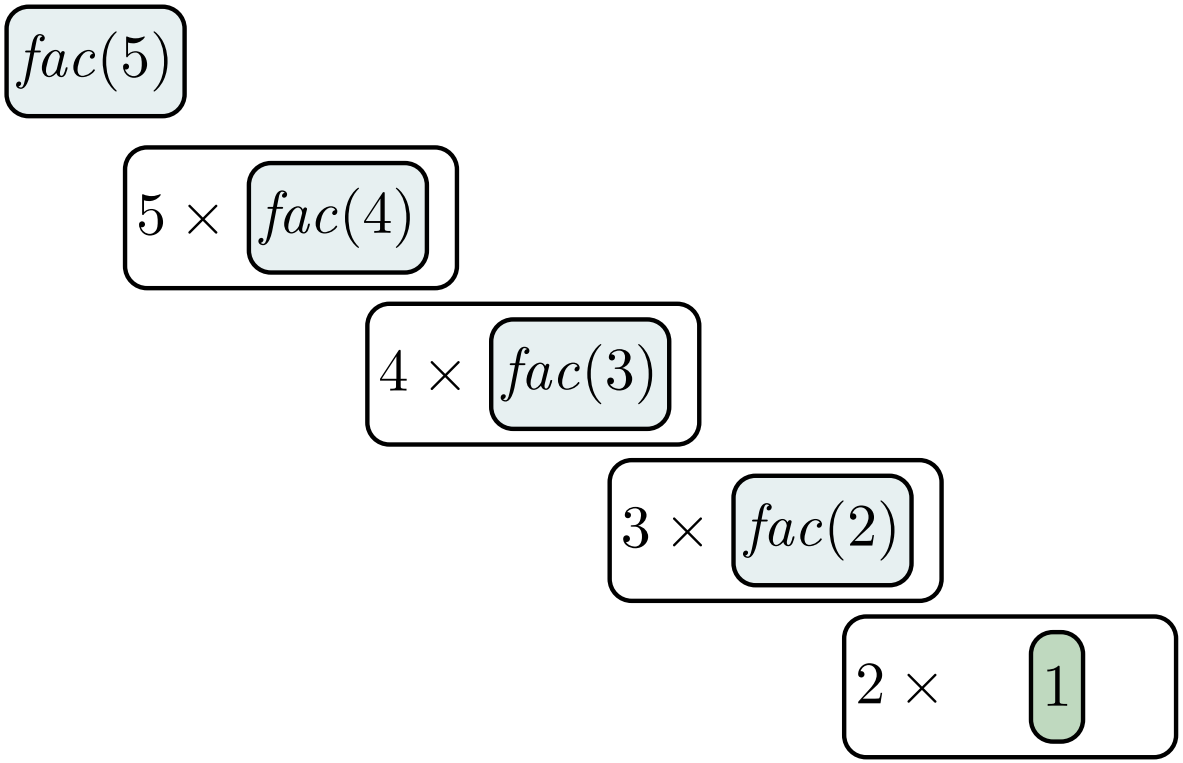
Martin Steffen

Zur Laufzeit



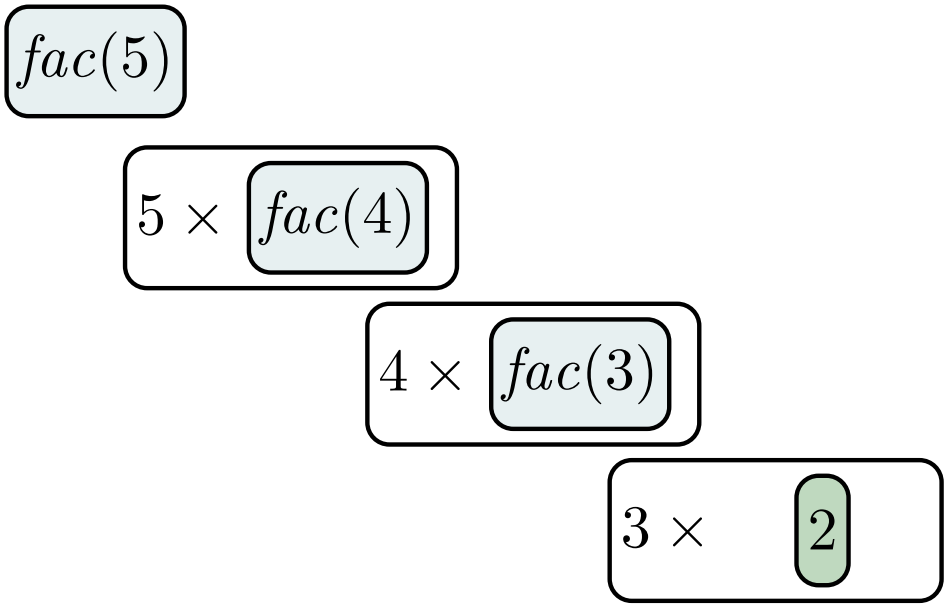
Martin Steffen

Zur Laufzeit



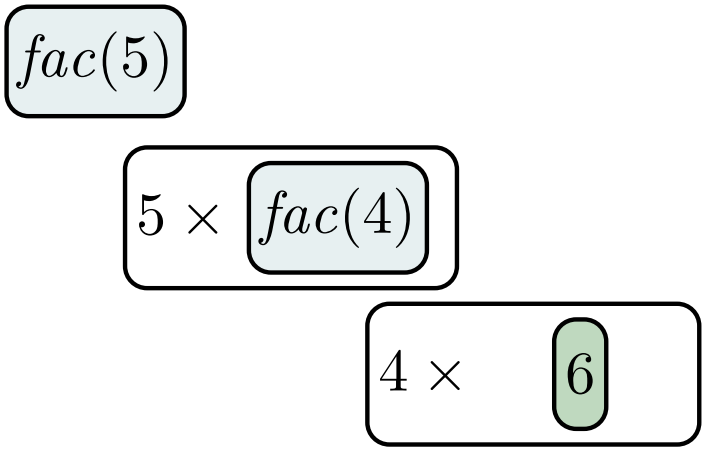
Martin Steffen

Zur Laufzeit



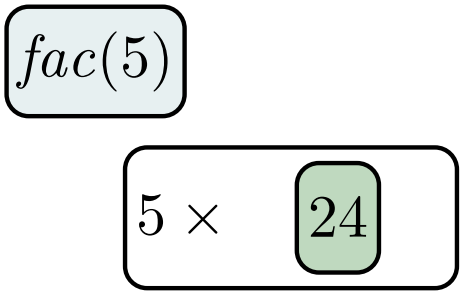
Martin Steffen

Zur Laufzeit



Martin Steffen

Zur Laufzeit



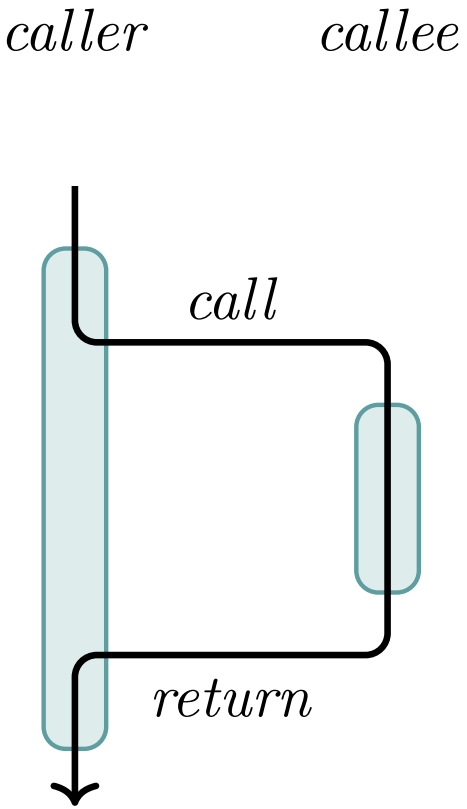
Martin Steffen

Zur Laufzeit

120

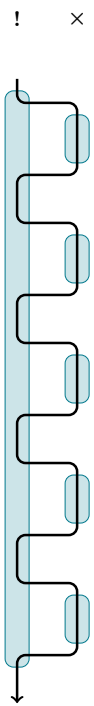
Martin Steffen

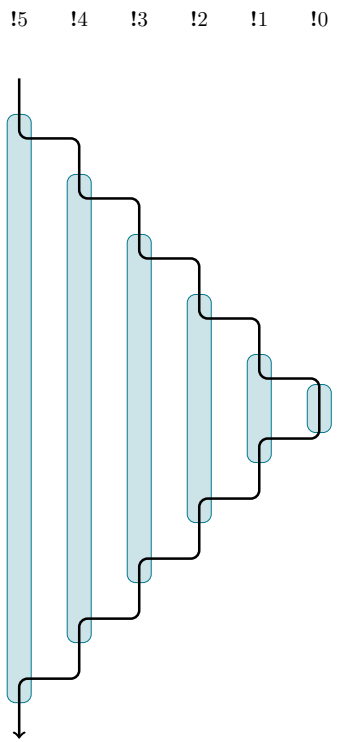
Aufrufer und Aufgerufener



Martin Steffen

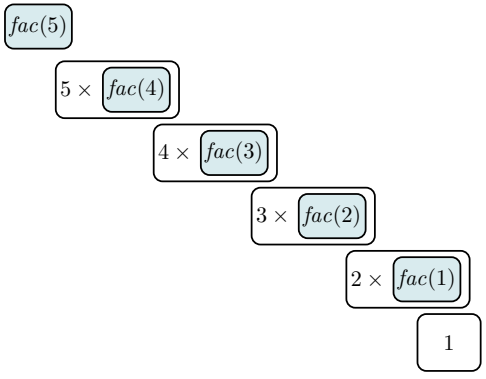
Lebenszeit lokaler Variabler





Martin Steffen

Laufzeitstapel




```
factorial(n) {
    long result;
    if (n == 1) { result = 1; }
    else {
      result = n * factorial(n-1);
    };
    return result;
}
```

- $n!$: bis zu $n+1$ (hier 6) Inkarnationen von `result`
- Allokierung/Deallokierung: LIFO $\Rightarrow$ Laufzeitstapel
- *dynamische* Speicherverwaltung
- Engl: *runtime stack*

Leonardo da Pisa

Martin Steffen

Dar Karnickelproblem

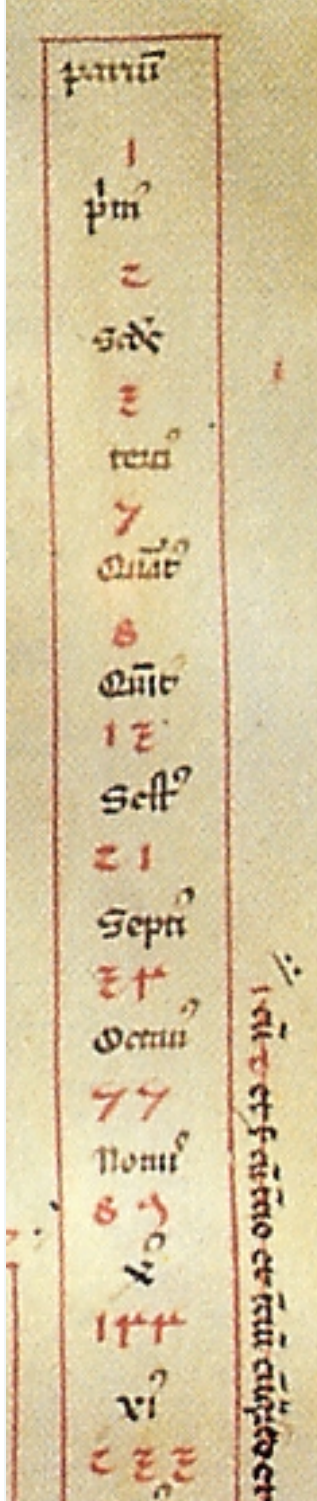
1. Ein Kaninchen wird in *einem* Monat erwachsen
2. jedes erwachsene Kaninchenpaar bringt jeden Monat ein Kaninchenpaar zur Welt

Frage

Startet man mit einem Paar, wieviele Paare hat man nach n Monaten

Fibonaccis Lösung

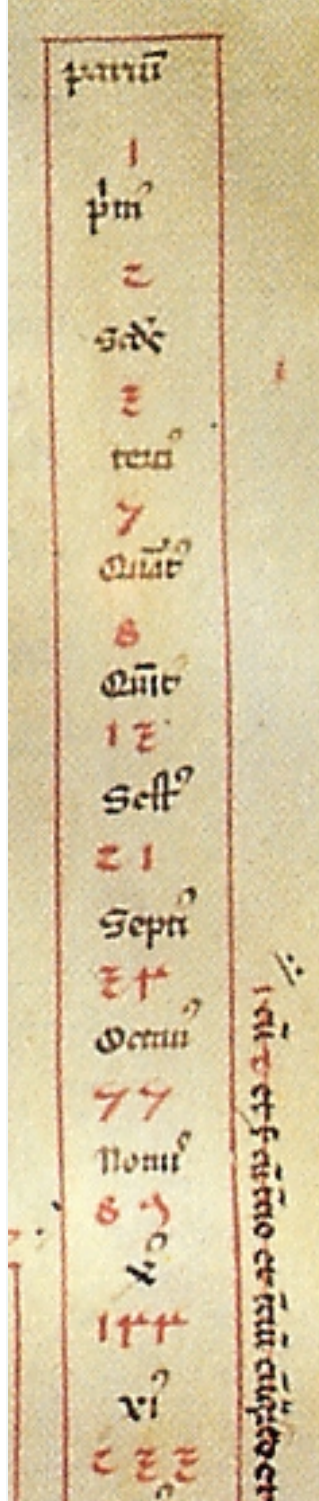
Monat	Kaninchenpaare		
	neugeboren	erwachsen	gesamt
0	1	0	1
1	0	1	1
2	1	1	2
3	1	2	3
4	2	3	5
5	3	5	8
	⋮		



Fibonaccis Lösung

Monat	Kaninchenpaare		
	neugeboren	erwachsen	gesamt
0	1	0	1
1	0	1	1
2	1	1	2
3	1	2	3
4	2	3	5
5	3	5	8
	⋮		

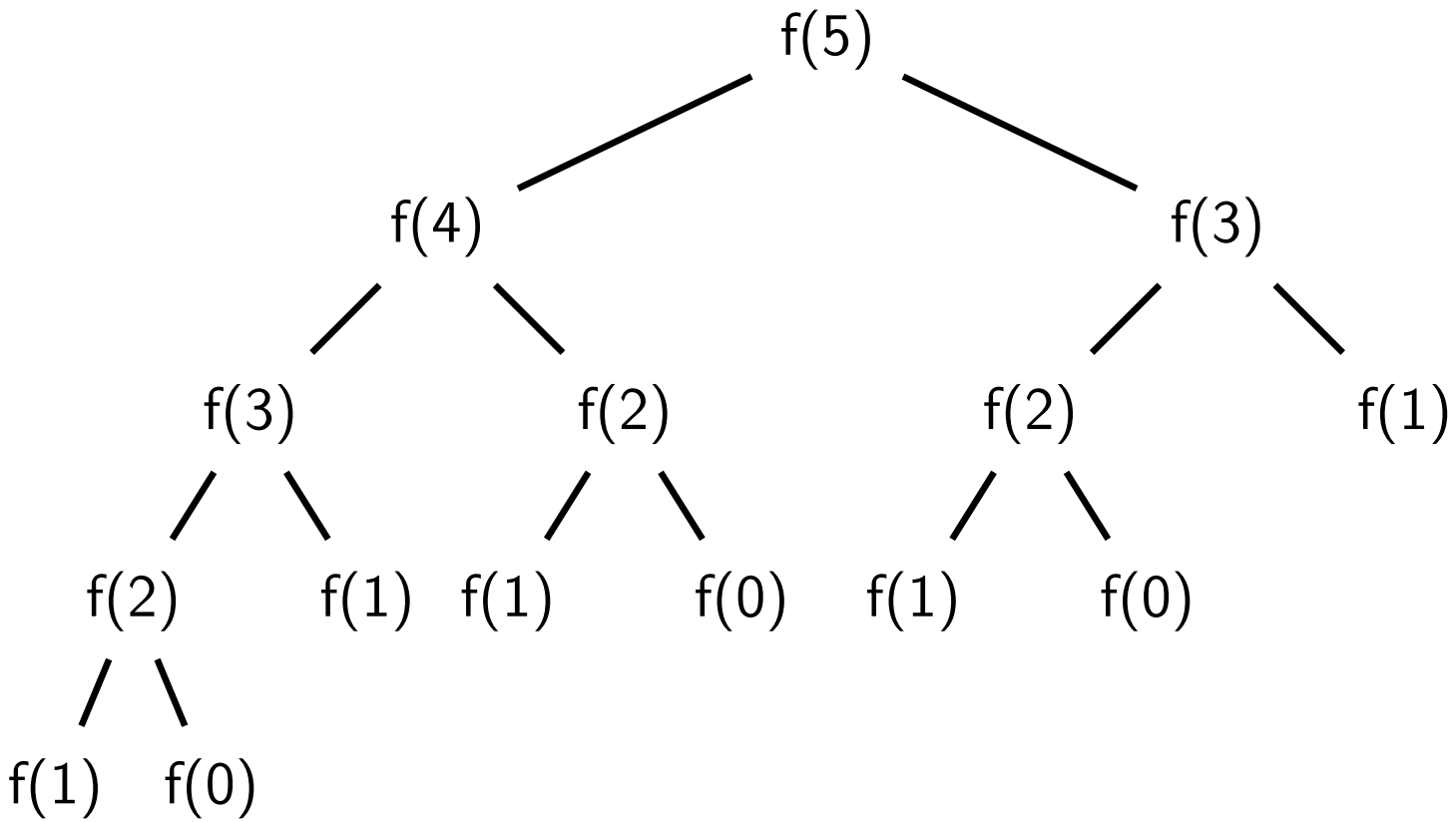
$$f_n = \begin{cases} 1 & \text{falls } n = 0 \text{ oder } n = 1 \\ f_{n-1} + f_{n-2} & \text{ansonsten} \end{cases}$$



In Java

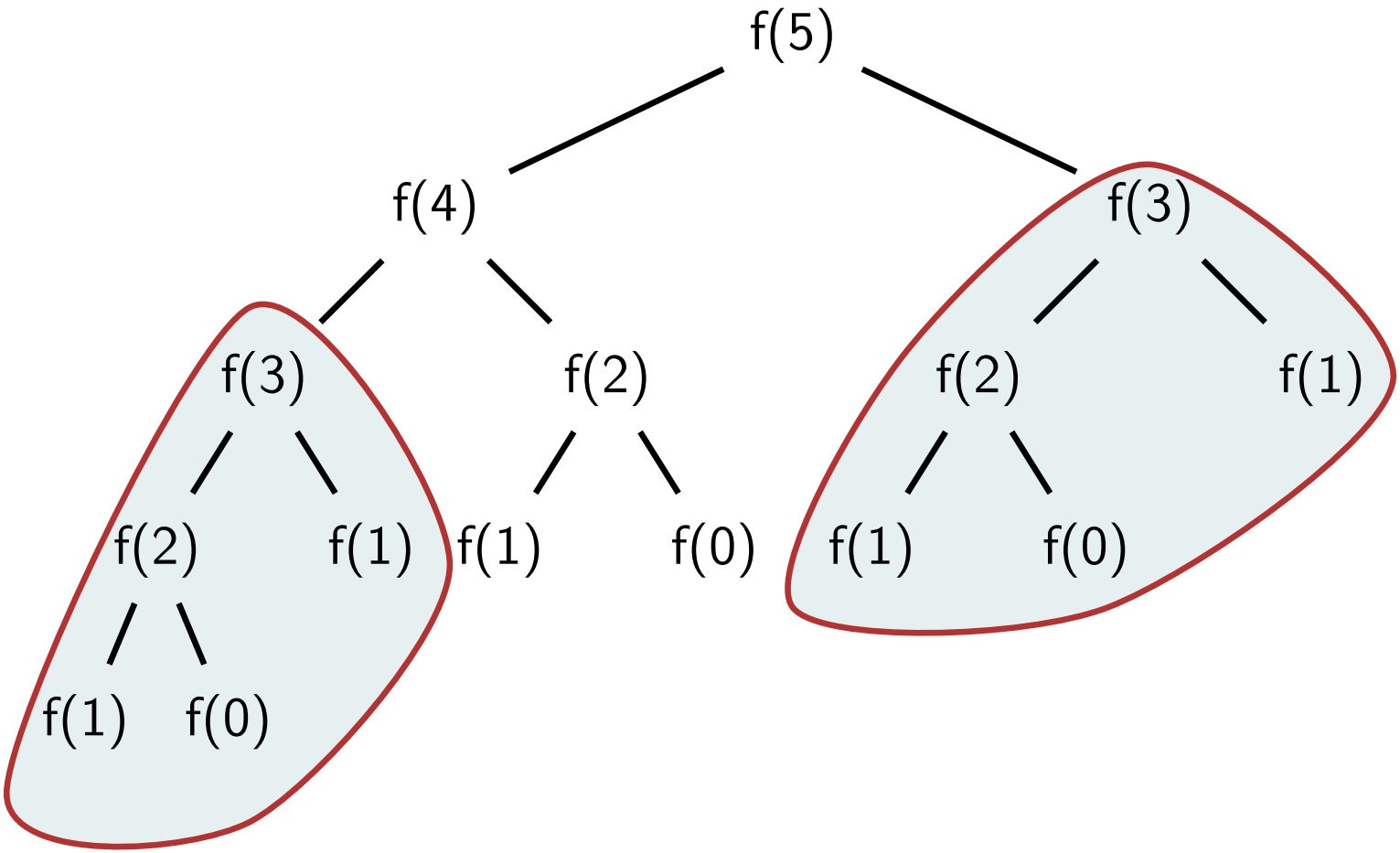
```
public static int fibonacci(int n) {
  if (n == 0) return 1;                    // Basisfall
  if (n == 1) return 1;                    // Basisfall
  return
    fibonacci (n-1) + fibonacci(n-2); // Induktionsfall
  }
```

Aufrufe



Martin Steffen

Aufrufe



Martin Steffen

Am Rande . . .

```
\node{f(5)}
child{node {f(4)}
  child{node {f(3)}
    child{node {f(2)}
      child{node {f(1)}}
      child{node {f(0)}}
    }
    child{node {f(1)}}
  }
  child{node {f(2)}
    child{node {f(1)}}
    child{node {f(0)}}
  }
}
child{node {f(3)}
  child{node {f(2)}
    child{node {f(1)}}
    child{node {f(0)}}
  }
  child{node {f(1)}}
}
;
U:---  fibonaccitree.tex
```

Baum = rek. Datenstruktur

Baumknoten =

- Knoten ohne Kinder (“Blatt”), oder
- mit n (hier 2) Baumknoten als Kinder

Martin Steffen

Bäume

```
public class TreeO {
    private TreeO left, right = null;
    private Object data;
```

in der Praxis: Baum-Datenstruktur meist komplexer

- Liste von Kindern/Unterbäumen
- anstelle `Object`: *“generics”*
- weitere Methoden, Konstruktoren, Zeiger
- vermeide null-Zeiger
- Interfaces
- ...

Martin Steffen

Das geht noch effizienter

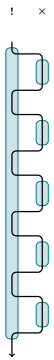
Martin Steffen


```
public static int fibonacci(int n, int a, int b) {
    if (n == 0) return b;     // let's start with 1
    return fibonacci (n-1,b,a+b);
}
```

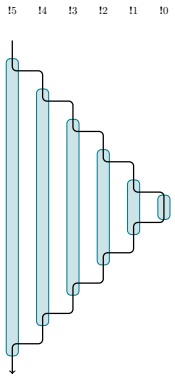
- a = "neugeboren", b = "erwachsen"
- Aufruf mit `fibonacci`(n, 0, 1)

Lebenszeit lokaler Variabler

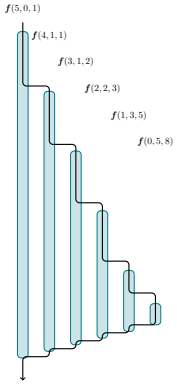
Fak. iterativ



Fak. rekursiv



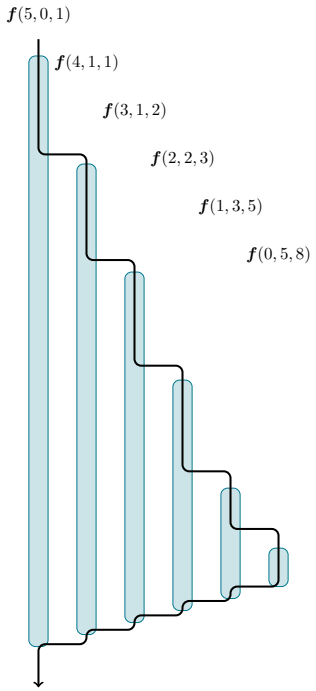
Fib. rekursiv




```
public static int fibonacci(int n, int a, int b) {
    if (n == 0) return b;     // let's start with 1
    return fibonacci (n-1,b,a+b);
}
```

Martin Steffen

Endrekursion

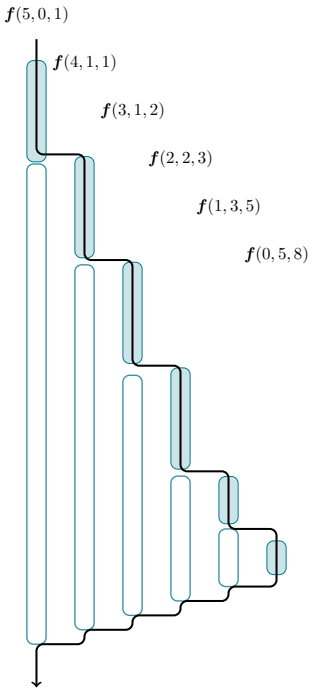


- Engl: *tail recursion*
- rekursiver Aufruf: das Letzte im Methodenrumpf

```
public static int fibonacci(int n, int a, int b) {
    if (n == 0) return b;     // let's start with 1
    return fibonacci (n-1,b,a+b);
}
```

Martin Steffen

Endrekursion



- Engl: *tail recursion*
- rekursiver Aufruf: das Letzte im Methodenrumpf
- eigentlich:
  - Stack tut nicht Not ...
  - iterativ-rekursiv
- oft: Compileroptimierung (nicht in Java, aber auf der Todo-Liste)

```
public static int fibonacci(int n, int a, int b) {
    if (n == 0) return b;     // let's start with 1
    return fibonacci (n-1,b,a+b);
}
```

“Endrekursive” Aufrufe?

```java
public static int fibonacci(int n, int a, int b) {
    if (n != 0) return fibonacci (n-1,b,a+b);
    return b;
}
```

```java
public static long factorial(int n) {
    if (n == 1) return 1;
    return n * factorial(n-1);
}
```

Martin Steffen

Rekursion vs. Iteration

LEARN ERLANG
absolute beginners

Erlang Tutorial

- Erlang - Home
- Erlang - Overview
- Erlang - Environment
- Erlang - Basic Syntax
- Erlang - Shell
- Erlang - Data Types
- Erlang - Variables
- Erlang - Operators
- Erlang - Loops
- Erlang - Decision Making
- Erlang - Functions
- Erlang - Modules
- Erlang - Recursion
- Erlang - Numbers
- Erlang - Strings
- Erlang - Lists
- Erlang - File I/O
- Erlang - Atoms
- Erlang - Maps

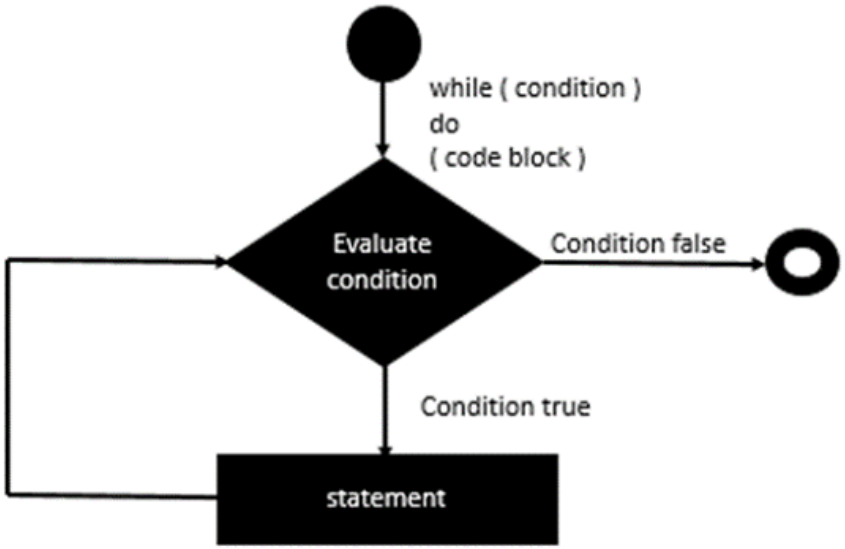
Previous Page Next Page

Erlang is a functional programming language and what needs to be remembered about all functional programming languages is that they don't offer any constructs for loops. Instead, functional programming depends on a concept called recursion.

while Statement Implementation

Since there is no direct while statement available in Erlang, one has to use the recursion techniques available in Erlang to carry out a while statement implementation.

We will try to follow the same implementation of the while loop as is followed in other programming languages. Following is the general flow which will be followed.



Martin Steffen

Rekursion vs. Iteration

- Rekursion vs. Iteration: im Prinzip *gleich ausdrucksstark*
-

Martin Steffen

Rekursion, nur etwas für "Zahlentheoretiker"?

Git: Mergen

```
[msteffen@rijkaard mmgo]$ git pull
remote: Counting objects: 9, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 9 (delta 7), reused 9 (delta 7), pack-reused 0
Unpacking objects: 100% (9/9), done.
From github.com:dfava/favasynthesis
   0d9fa6c..154dd44  master     -> origin/master
Merge made by the 'recursive' strategy.
 papers/mmgo/intro.tex | 52 ++++++++++++++++++++++++++++++----------------
 1 file changed, 30 insertions(+), 22 deletions(-)
[msteffen@rijkaard mmgo]$
```

Rekursion, nur etwas für “Zahlentheoretiker”?

Martin Steffen

Internet DNS

Recursive DNS is essentially the opposite of Dyn Standard DNS which is an authoritative DNS service that allows others to find *your* domain while Recursive DNS allows *you* to resolve other people’s domains.

The Longer Answer

Recursive DNS provides recursive DNS. Yes, that’s recursive (something which repeats or refers back to itself) and confusing. In order to make a distinction between the service we provide and the general concept of recursive DNS, here’s an explanation.

To better illustrate how recursive DNS works, let’s imagine you are sitting at a computer in your study at home. You’re connected to the Internet by a cable connection and you are surfing the web looking for widgets. You have no idea where to find widgets, so you open your web browser and type in http://www.google.com.

Rekursion, nur etwas für “Zahlentheoretiker”?

Martin Steffen

der Millionen-Raub mittels Rekursion[1]

Deconstructing theDAO Attack: A Brief Code Tour

18 JUNE 2016 on thedao, security, ethereum, solidity

TheDAO was attacked today, and the attacker seems to have made off with 3.5mm ether (at time of writing in excess of $45mm). The vulnerability was the Race To Empty or Recursive Call attack.

[1] in virtuellem Geld (Ether “blockchain”) und nur zeitweise.

Suche

Martin Steffen

Ziel

- Input: Integer-Array + Zahl
- Output: Falls die Zahl im array ist: Index *wo*

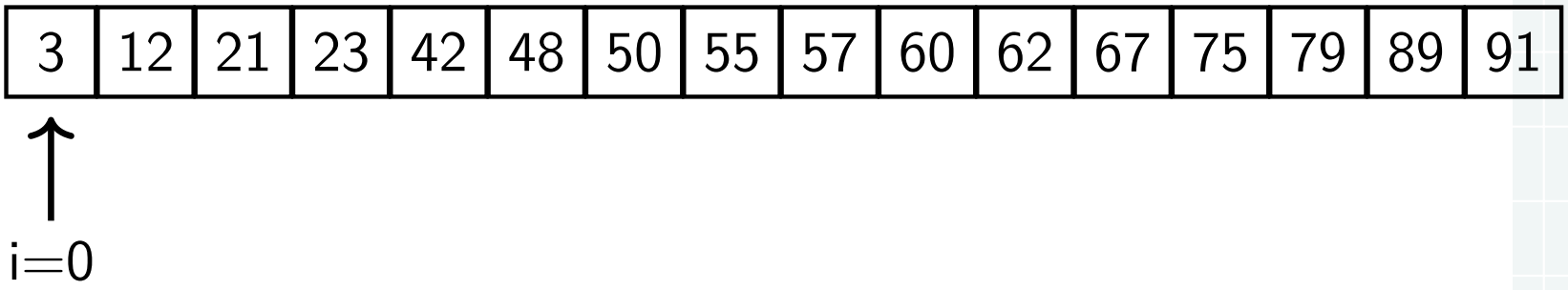
3	12	21	23	42	48	50	55	57	60	62	67	75	79	89	91

Suche

Martin Steffen

Ziel

- Input: Integer-Array + Zahl
- Output: Falls die Zahl im array ist: Index *wo*

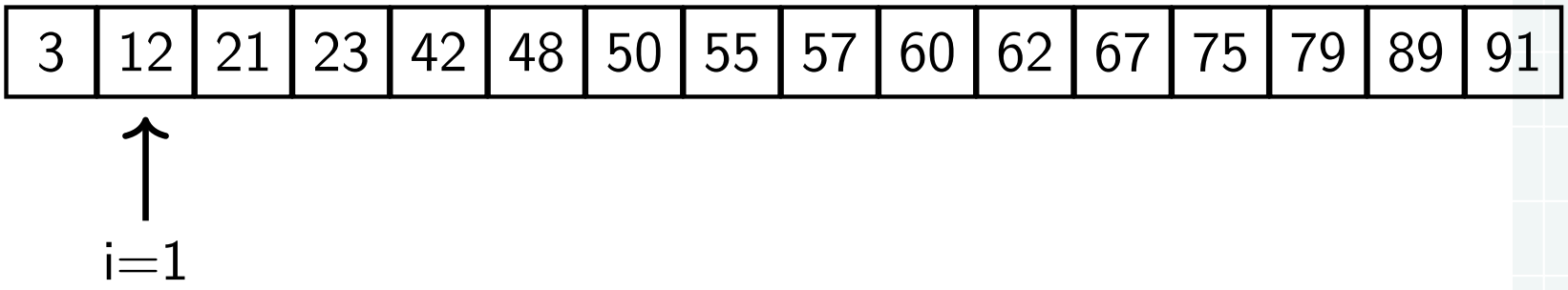


Suche

Martin Steffen

Ziel

- Input: Integer-Array + Zahl
- Output: Falls die Zahl im array ist: Index *wo*

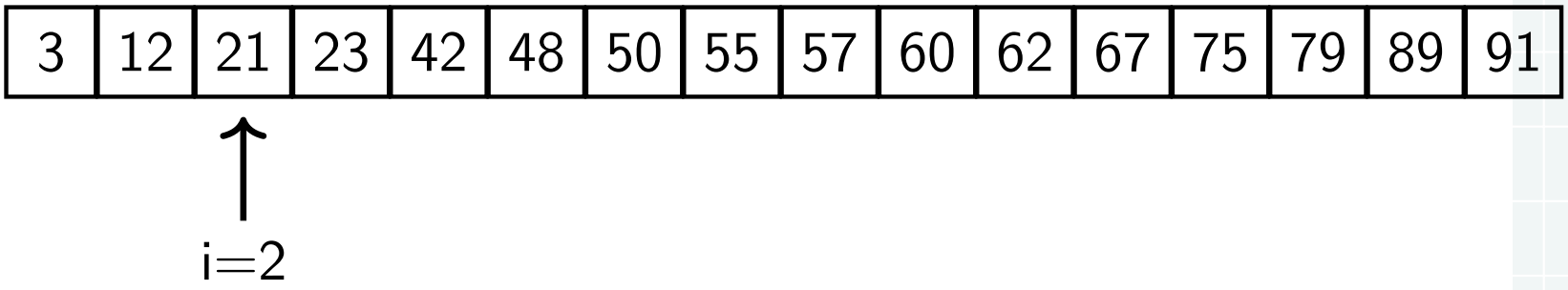


Suche

Martin Steffen

Ziel

- Input: Integer-Array + Zahl
- Output: Falls die Zahl im array ist: Index *wo*

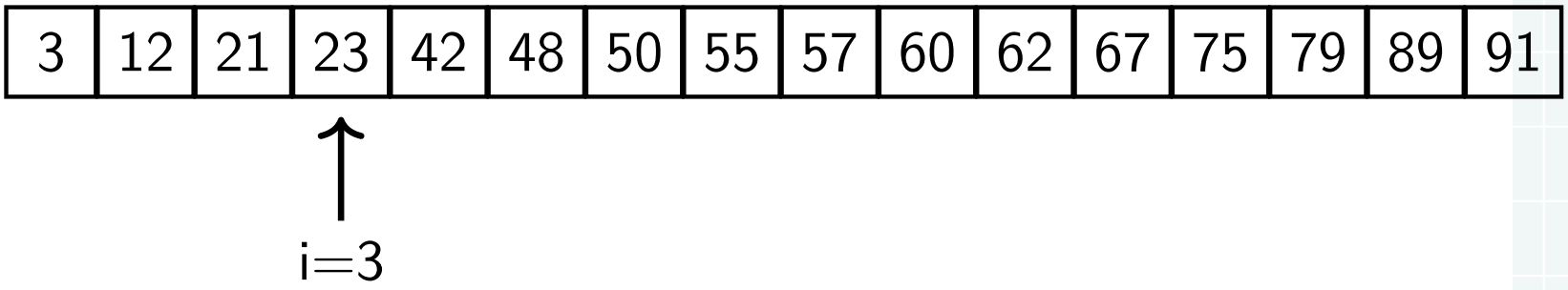


Suche

Martin Steffen

Ziel

- Input: Integer-Array + Zahl
- Output: Falls die Zahl im array ist: Index *wo*

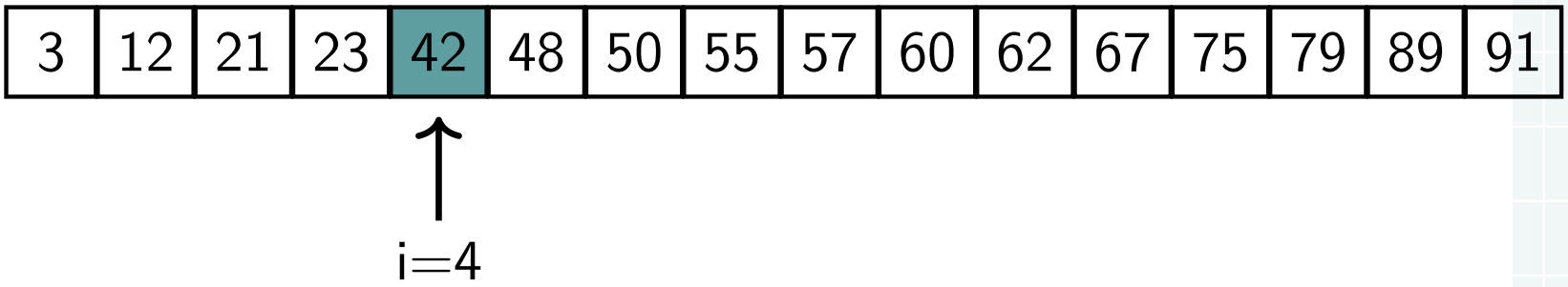


Suche

Martin Steffen

Ziel

- Input: Integer-Array + Zahl
- Output: Falls die Zahl im array ist: Index *wo*



Das geht noch besser: Binäre Suche

Martin Steffen

Ziel

- Input: sortiertes Integer-Array + Zahl
- Output: Falls die Zahl im array ist: Index *wo*

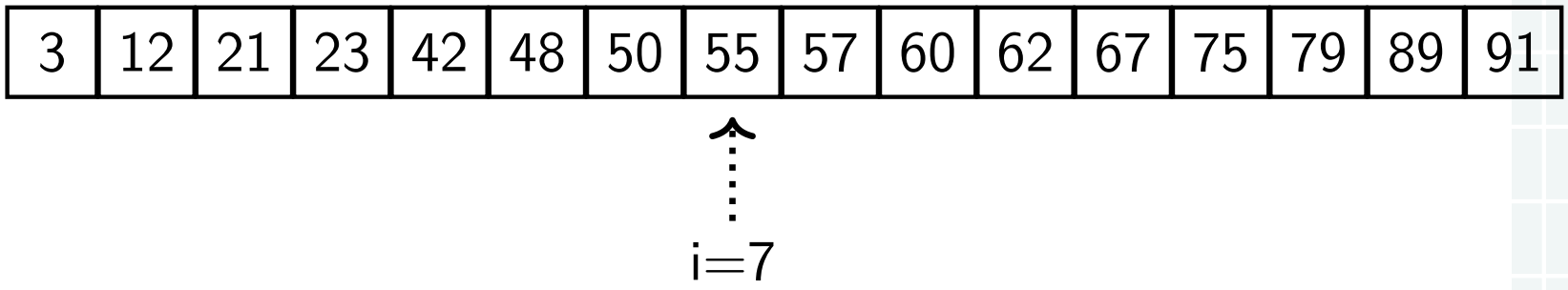
3	12	21	23	42	48	50	55	57	60	62	67	75	79	89	91

Das geht noch besser: Binäre Suche

Martin Steffen

Ziel

- Input: sortiertes Integer-Array + Zahl
- Output: Falls die Zahl im array ist: Index *wo*

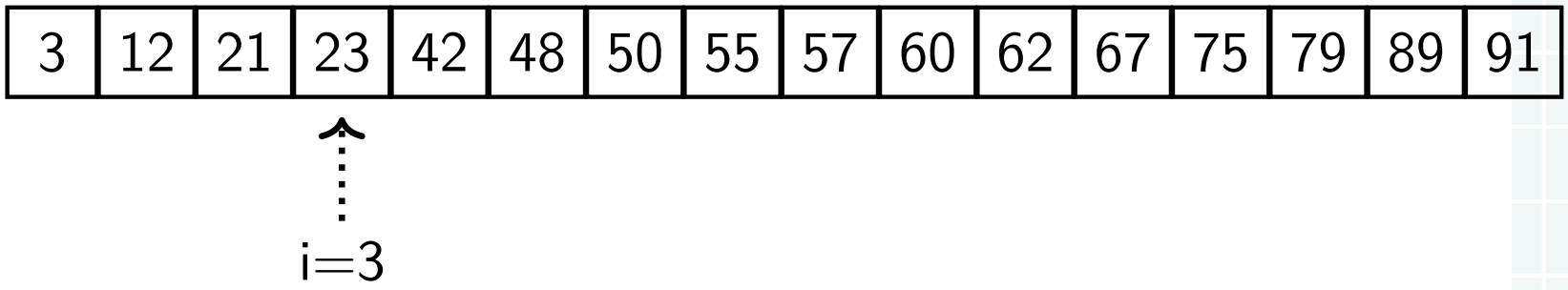


Das geht noch besser: Binäre Suche

Martin Steffen

Ziel

- Input: sortiertes Integer-Array + Zahl
- Output: Falls die Zahl im array ist: Index *wo*

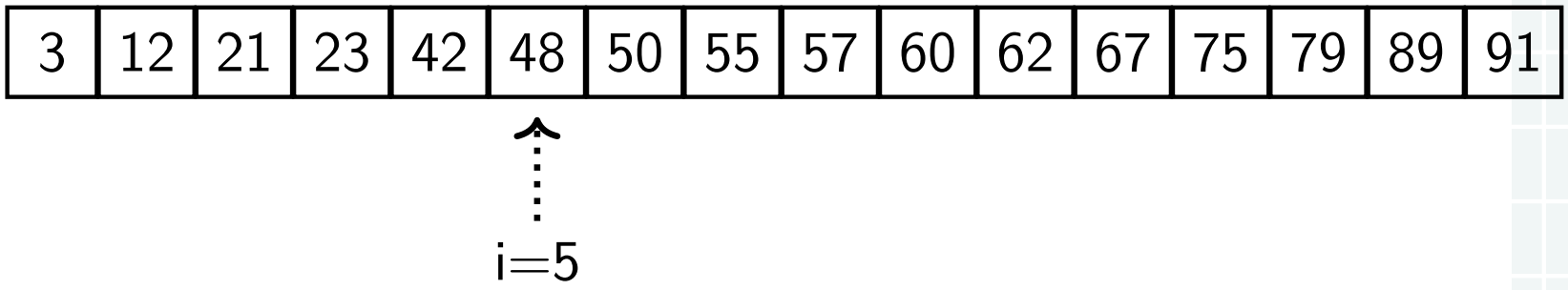


Das geht noch besser: Binäre Suche

Martin Steffen

Ziel

- Input: sortiertes Integer-Array + Zahl
- Output: Falls die Zahl im array ist: Index *wo*

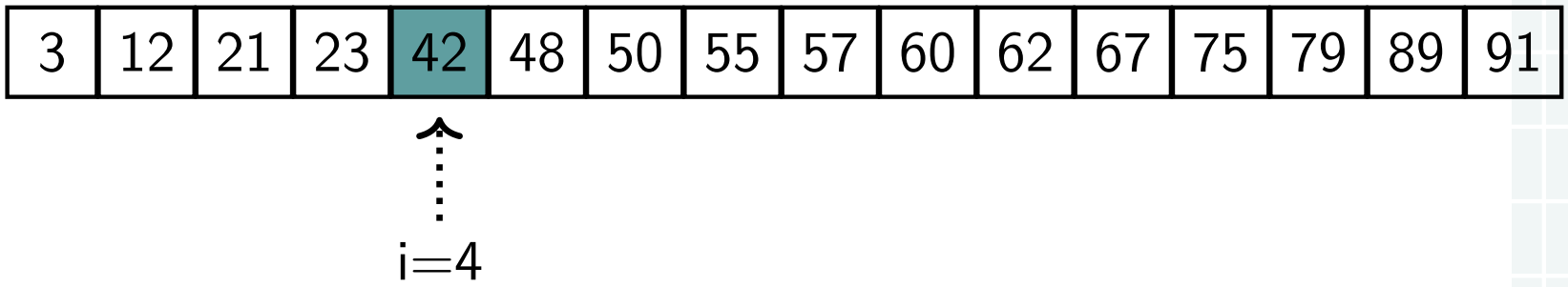


Das geht noch besser: Binäre Suche

Martin Steffen

Ziel

- Input: sortiertes Integer-Array + Zahl
- Output: Falls die Zahl im array ist: Index *wo*



Rekursiver Ansatz

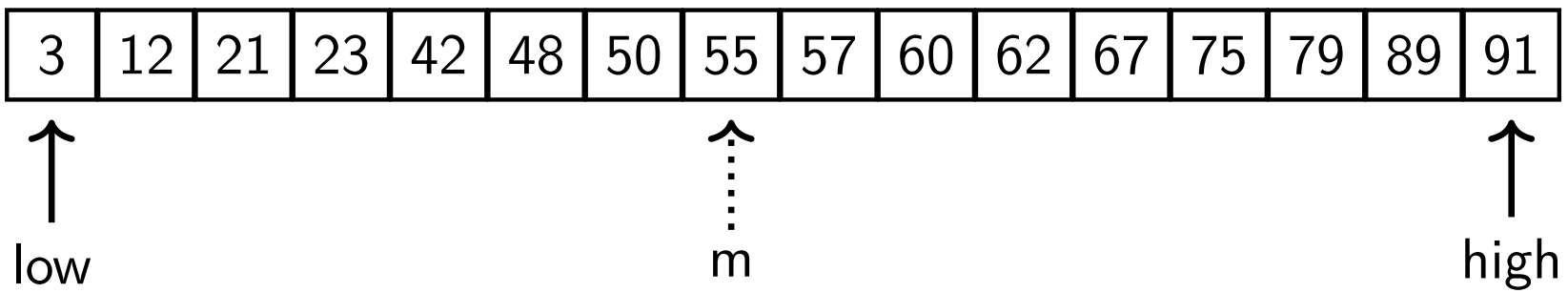
Martin Steffen

Teile & Herrsche: Suche nach `elem`

- Schlag' nach in der *Mitte* des Arrays
- falls gleich `elem` $\Rightarrow$ fertig
- falls kleiner as als `elem` $\Rightarrow$ suche rekursiv in der rechten Hälfte
- falls größer as als `elem` $\Rightarrow$ suche rekursiv in der linken Hälfte

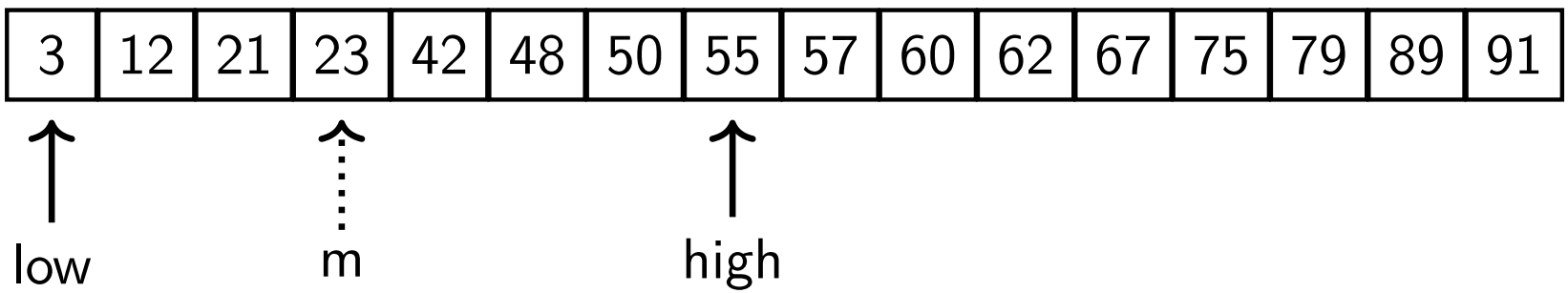
In Java

```
static int search (int elem, int[] a, int low, int high) {
    System.out.println(low);
    System.out.println(high);
    if (low == high) {
        if (elem == a[low]) {
            return low;
        } else {
            return -1;
        }
    } else {  // low ≠ high
        int m = (low + high) / 2;
        if (elem < a[m]) {
            return search(elem, a, low, m-1);
        } else {
            return search(elem, a, m+1, high);
```



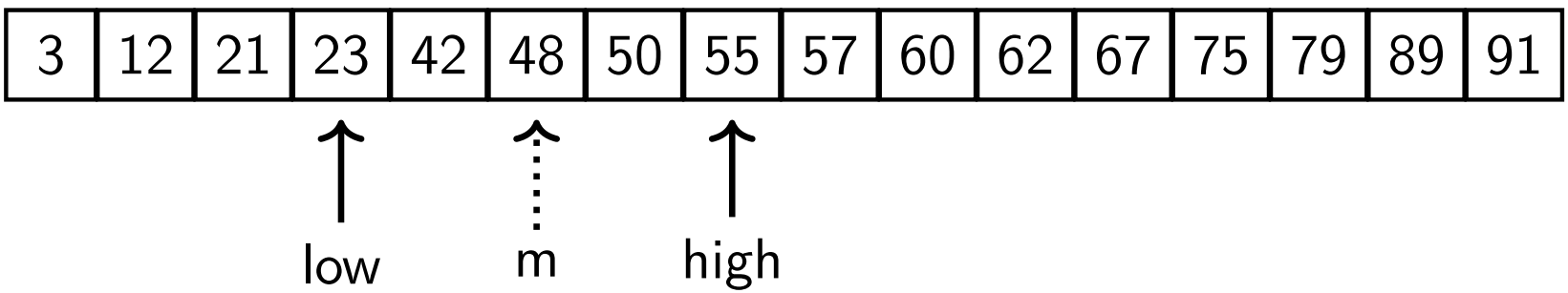
In Java

```
static int search (int elem, int[] a, int low, int high) {
    System.out.println(low);
    System.out.println(high);
    if (low == high) {
        if (elem == a[low]) {
            return low;
        } else {
            return -1;
        }
    } else {  // low ≠ high
        int m = (low + high) / 2;
        if (elem < a[m]) {
            return search(elem, a, low, m-1);
        } else {
            return search(elem, a, m+1, high);
```



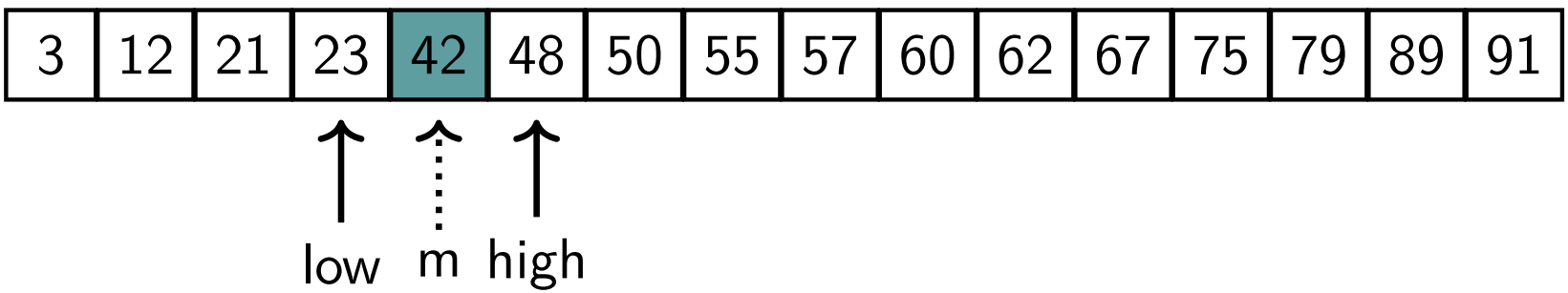
In Java

```
static int search (int elem, int[] a, int low, int high) {
    System.out.println(low);
    System.out.println(high);
    if (low == high) {
        if (elem == a[low]) {
            return low;
        } else {
            return -1;
        }
    } else {  // low ≠ high
        int m = (low + high) / 2;
        if (elem < a[m]) {
            return search(elem, a, low, m-1);
        } else {
            return search(elem, a, m+1, high);
```



In Java

```
static int search (int elem, int[] a, int low, int high) {
    System.out.println(low);
    System.out.println(high);
    if (low == high) {
        if (elem == a[low]) {
            return low;
        } else {
            return -1;
        }
    } else {  // low ≠ high
        int m = (low + high) / 2;
        if (elem < a[m]) {
            return search(elem, a, low, m-1);
        } else {
            return search(elem, a, m+1, high);
```



“Korrektheit”

Argument

1. Die “teile-und-herrsche” Idee scheint einleuchtend
2. Terminierung
   - jeder rekursive Aufruf macht das Problem echt kleiner (Induktionsfall)
   - es gibt ein kleinstes Problem (Basisfall)

“Korrektheit”

Argument

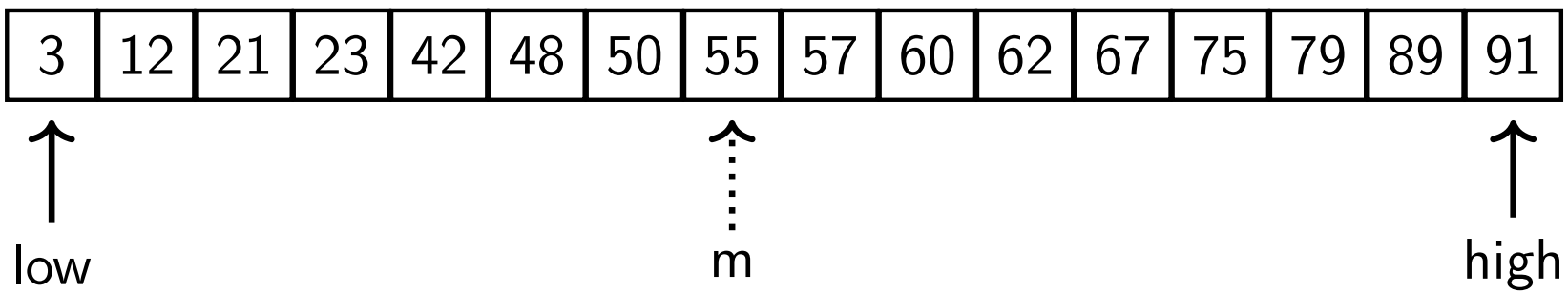
1. Die “teile-und-herrsche” Idee scheint einleuchtend
2. Terminierung
   - jeder rekursive Aufruf macht das Problem echt kleiner (Induktionsfall)
   - es gibt ein kleinstes Problem (Basisfall)
   - leider nur scheinbar . . .

```
-*- mode: compilation; default-directory: "~/javaexamples/" -*-
Compilation started at Wed Oct 11 15:58:48

java Binsearch
Exception in thread "main" java.lang.StackOverflowError
        at Binsearch.search(Binsearch.java:12)
        at Binsearch.search(Binsearch.java:12)
        at Binsearch.search(Binsearch.java:12)
        at Binsearch.search(Binsearch.java:12)
        at Binsearch.search(Binsearch.java:12)
-:%*-  *compilation*   Top L1     [(Compilation:exit [1] Abbrev)]
```

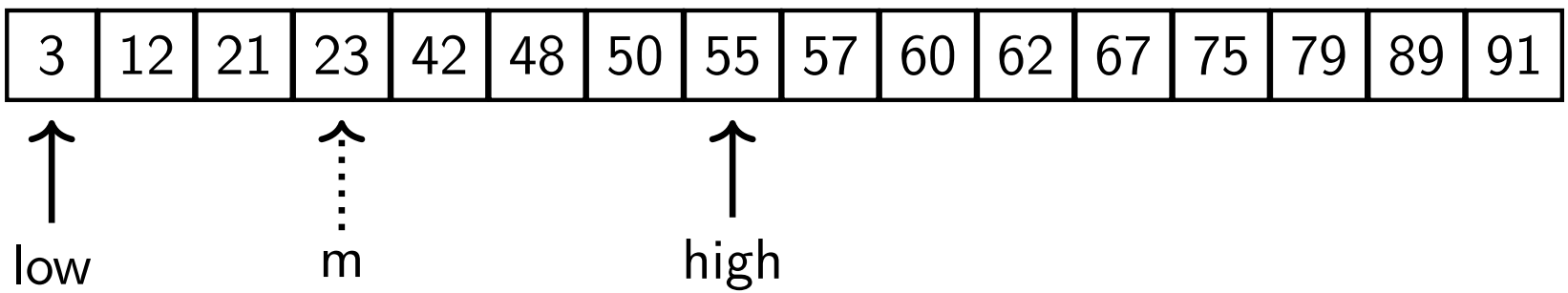
Auf den zweiten Blick . . .

```
static int search (int elem, int[] a, int low, int high) {
    if (low > high) return -1; // empty
    int m = (low + high) / 2;
    if (elem == a[m]) return m;
    if (elem < a[m]) {
        return search(elem, a, low, m-1);
    } else {
        return search(elem, a, m+1, high);
    }
}
```



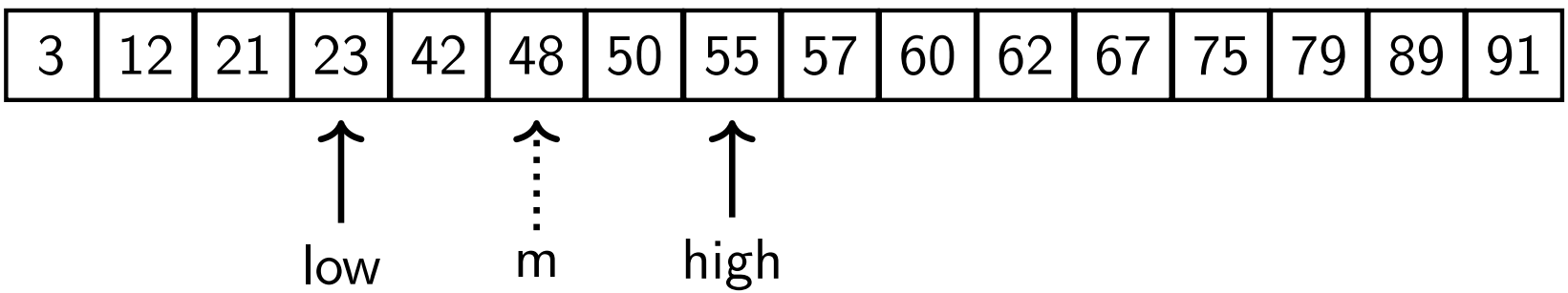
Auf den zweiten Blick . . .

```
static int search (int elem, int[] a, int low, int high) {
    if (low > high) return -1; // empty
    int m = (low + high) / 2;
    if (elem == a[m]) return m;
    if (elem < a[m]) {
        return search(elem, a, low, m-1);
    } else {
        return search(elem, a, m+1, high);
    }
}
```



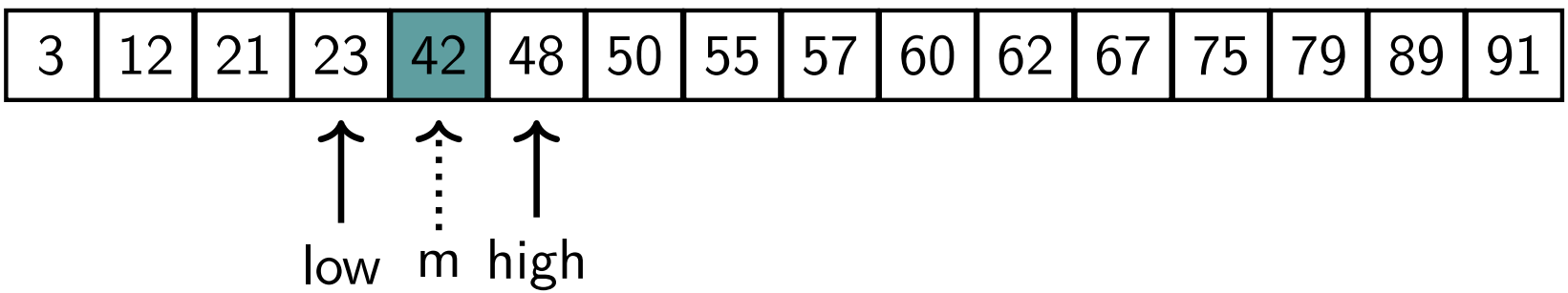
Auf den zweiten Blick . . .

```
static int search (int elem, int[] a, int low, int high) {
    if (low > high) return -1; // empty
    int m = (low + high) / 2;
    if (elem == a[m]) return m;
    if (elem < a[m]) {
        return search(elem, a, low, m-1);
    } else {
        return search(elem, a, m+1, high);
    }
}
```



Auf den zweiten Blick . . .

```
static int search (int elem, int[] a, int low, int high) {
    if (low > high) return -1; // empty
    int m = (low + high) / 2;
    if (elem == a[m]) return m;
    if (elem < a[m]) {
        return search(elem, a, low, m-1);
    } else {
        return search(elem, a, m+1, high);
    }
}
```



The bug that “fixed itself”

```
while (days > 365) {
  if (IsLeapYear(year)) {
    if (days > 366) {
      days -= 366;
      year += 1;
    }
  } else {
    days -= 365;
    year += 1;
  }
}
```

Martin Steffen

Vielleicht nur ein Anfängerfehler . . . ?

Martin Steffen

Proving that Android's, Java's and Python's sorting algorithm is broken (and showing how to fix it)

February 24, 2015 Envisage Written by Stijn de Gouw. $s

Tim Peters developed the Timsort hybrid sorting algorithm in 2002. It is a clever combination of ideas from merge sort and insertion sort, and designed to perform well on real world data. TimSort was first developed for Python, but later ported to Java (where it appears as java.util.Collections.sort and java.util.Arrays.sort) by Joshua Bloch (the designer of Java Collections who also pointed out that most binary search algorithms were broken). TimSort is today used as the default sorting algorithm for Android SDK, Sun's JDK and OpenJDK. Given the popularity of these platforms this means that the number of computers, cloud services and mobile phones that use TimSort for sorting is well into the billions.

Und die Moral . . .

- Finger weg von Rekursion?
- Basisfälle (und Spezialfälle) besonders *fehleranfällig* (“one-off” Fehler)
- funktioniert in den allermeisten Fällen $\neq$ korrekt

> “Program testing can be used to show the presence of bugs, but never to show their absence”
>
> Dijkstra 1970, p. 7

- ultimativ: sorgfältiges Argumentieren (“Korrektheitsbeweis”) angesagt.

Ausblick

Martin Steffen

- in-direkte Rekursion (“call-backs”)
- Induktion & Rekursion
- Induktive/Rekursive Datenstrukturen und korrespondierende Algorithmen (z.B. Bäume).
- Komplexität
- . . .

Kudos

Die Folien wurden mit

- gnu emacs *org-mode* ("gnu's not Unix")
- LaTeX, und
- TikZ ("TikZ ist kein Zeichenprogram")

erstellt.
Das "Design" verdankt Inspiration den elaborierten Style-files der Uni Lübeck (M. Leucker, V. Stolz).

Quellen

Das meiste ist “Allgemeinwissen” und die Vorlesung basiert auf keinem speziellen Buch oder Quelle. Ähnliche Beispiele finden sich in praktisch allen Einführungen in Java oder auch in andere Programmiersprachen. Bilder, sofern nicht selbstgemachte Grafiken, sind ebenso aus “schöpferischem Gemeingut” (creative commons). Spezielle Internetfunde sind klickbar mit eingebetteten Links.