Informatik II

Übung 4

Giuseppe Accaputo, Felix Friedrich, Patrick Gruntz, Tobias Klenze, Max Rosmannek, David Sidler, Thilo Weghorn

FS 2017

Heutiges Programm

1. Rekursion
2. Divide And Conquer und Suchen
3. Programmierübung

1. Rekursion

Rekursion in Java

$$n! = \begin{cases} 1 & \text{falls } n \leq 1 \\ n \cdot (n-1)!, & \text{andernfalls} \end{cases}$$

```
public static int fakultaet(int n) {
    if (n <= 1) {
        return 1;
    } else {
        return n * fakultaet(n−1);
    }
}
```

$n! \Leftrightarrow$ `fakultaet(n)`

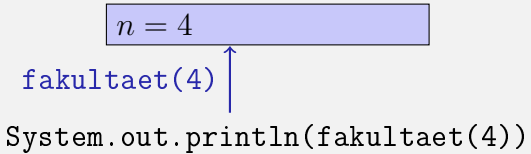
$n - 1! \Leftrightarrow$ `fakultaet(n−1)`

Der Aufrufstapel

```
System.out.println(fakultaet(4))
```

Der Aufrufstapel

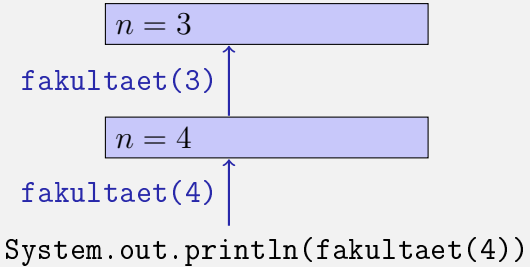
Bei jedem Funktionsaufruf:



Der Aufrufstapel

Bei jedem Funktionsaufruf:

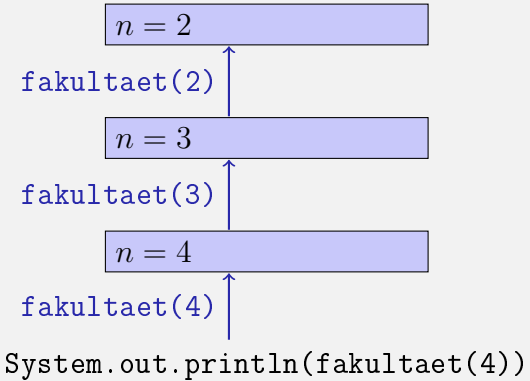
- Wert des Aufrufarguments kommt auf einen Stapel



Der Aufrufstapel

Bei jedem Funktionsaufruf:

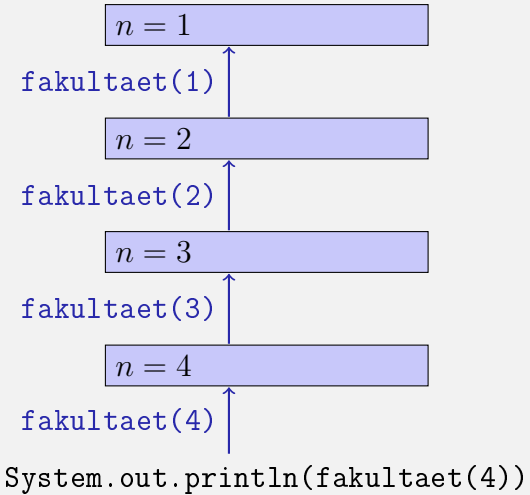
- Wert des Aufrufarguments kommt auf einen Stapel



Der Aufrufstapel

Bei jedem Funktionsaufruf:

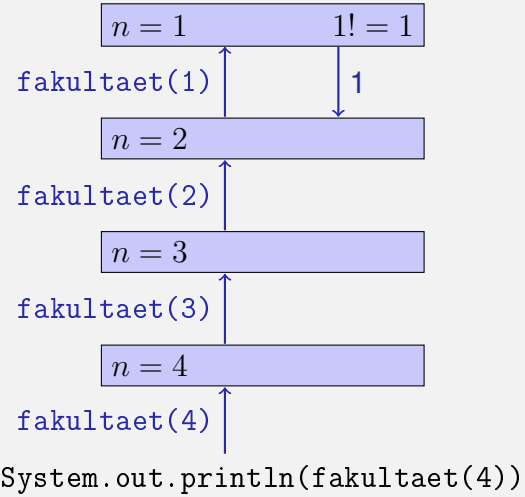
- Wert des Aufrufarguments kommt auf einen Stapel



Der Aufrufstapel

Bei jedem Funktionsaufruf:

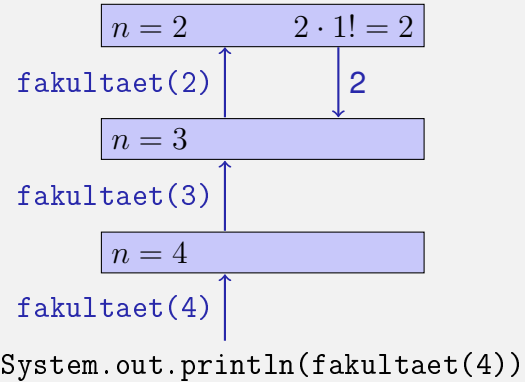
- Wert des Aufrufarguments kommt auf einen Stapel
- Es wird immer mit dem obersten Wert gearbeitet



Der Aufrufstapel

Bei jedem Funktionsaufruf:

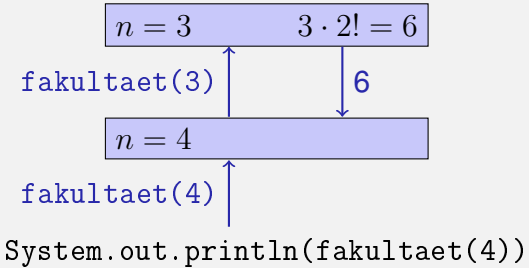
- Wert des Aufrufarguments kommt auf einen Stapel
- Es wird immer mit dem obersten Wert gearbeitet
- Am Ende des Aufrufs wird der oberste Wert wieder vom Stapel gelöscht



Der Aufrufstapel

Bei jedem Funktionsaufruf:

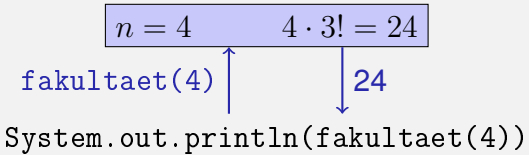
- Wert des Aufrufarguments kommt auf einen Stapel
- Es wird immer mit dem obersten Wert gearbeitet
- Am Ende des Aufrufs wird der oberste Wert wieder vom Stapel gelöscht



Der Aufrufstapel

Bei jedem Funktionsaufruf:

- Wert des Aufrufarguments kommt auf einen Stapel
- Es wird immer mit dem obersten Wert gearbeitet
- Am Ende des Aufrufs wird der oberste Wert wieder vom Stapel gelöscht



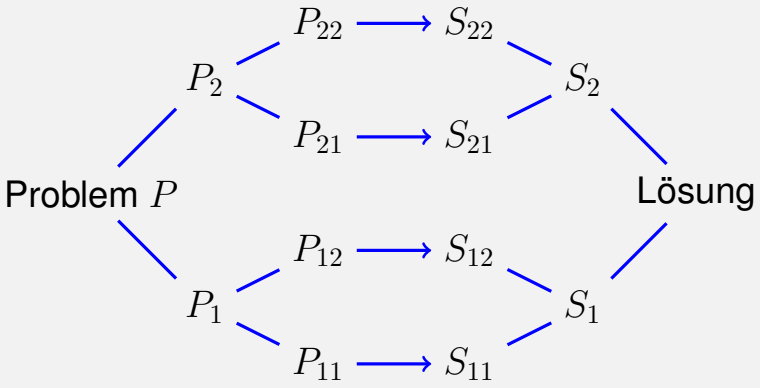
2. Divide And Conquer und Suchen

divide et impera

Teile und (be)herrsche (engl. divide and conquer)

Zerlege das Problem in Teilprobleme, deren Lösung zur vereinfachten Lösung des Gesamtproblems beitragen.

divide et impera



Binärer Suchalgorithmus `BSearch(A,b,l,r)`

Input : Sortiertes Array A von n Schlüsseln. Schlüssel b. Bereichsgrenzen $1 \leq l \leq r \leq n$ oder $l > r$ beliebig.
Output : Index des gefundenen Elements. 0, wenn erfolglos.

$m \leftarrow \lfloor (l+r)/2 \rfloor$
if $l > r$ **then** // erfolglose Suche
    return 0
else if $b = A[m]$ **then**// gefunden
    return m
else if $b < A[m]$ **then**// Element liegt links
    return BSearch$(A, b, l, m-1)$
else // $b > A[m]$: Element liegt rechts
    return BSearch$(A, b, m+1, r)$

Resultat

Theorem

Der Algorithmus zur binären sortierten Suche benötigt $\Theta(\log n)$ Elementarschritte.

Das sollten Sie beweisen können!

Fragen oder Anregungen?