

# Datenstrukturen und Algorithmen

## Übung 2

FS 2018

# Programm von heute

- 1 Feedback letzte Übung
- 2 Wiederholung Theorie
  - Induktion
  - Analyse von Programmen
- 3 Programmieraufgabe

# Landau-Notation

- Geben Sie eine korrekte, kompakte Definition der Menge  $\Theta(f)$  analog zur Definition der Mengen  $\mathcal{O}(f)$  und  $\Omega(f)$ .

# Landau-Notation

- Geben Sie eine korrekte, kompakte Definition der Menge  $\Theta(f)$  analog zur Definition der Mengen  $\mathcal{O}(f)$  und  $\Omega(f)$ .
- $\Theta(f) = \{g : \mathbb{N} \rightarrow \mathbb{R} \mid \exists a > 0, b > 0, n_0 \in \mathbb{N} : a \cdot f(n) \leq g(n) \leq b \cdot f(n) \forall n \geq n_0\}$

# Landau-Notation

- Geben Sie eine korrekte, kompakte Definition der Menge  $\Theta(f)$  analog zur Definition der Mengen  $\mathcal{O}(f)$  und  $\Omega(f)$ .
- $\Theta(f) = \{g : \mathbb{N} \rightarrow \mathbb{R} \mid \exists a > 0, b > 0, n_0 \in \mathbb{N} : a \cdot f(n) \leq g(n) \leq b \cdot f(n) \forall n \geq n_0\}$
- $\Theta(f) = \{g : \mathbb{N} \rightarrow \mathbb{R} \mid \exists c > 0, n_0 \in \mathbb{N} : \frac{1}{c} \cdot f(n) \leq g(n) \leq c \cdot f(n) \forall n \geq n_0\}$

# Landau-Notation

Beweisen oder widerlegen Sie, mit  $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$ .

(a)  $f \in \mathcal{O}(g)$  if and only if  $g \in \Omega(f)$ .

(e)  $\log_a(n) \in \Theta(\log_b(n))$  for all constants  $a, b \in \mathbb{N} \setminus \{1\}$

(g) If  $f_1, f_2 \in \mathcal{O}(g)$  and  $f(n) := f_1(n) \cdot f_2(n)$ , then  $f \in \mathcal{O}(g)$ .

# Landau-Notation

Funktionen sortieren: wenn Funktion  $f$  links der Funktion  $g$  steht, dann  $f \in \mathcal{O}(g)$ .

$2^{16}$ ,  $\log(n^4)$ ,  $\log^8(n)$ ,  $\sqrt{n}$ ,  $n \log n$ ,  $\binom{n}{3}$ ,  $n^5 + n$ ,  $\frac{2^n}{n^2}$ ,  $n!$ ,  $n^n$ .

# Summe von Elementen in zweidimensionalem Feld

Probleme / Fragen?



## **2. Wiederholung Theorie**

# Induktion: Was wird gebraucht?

- Beweise Aussagen, z.B.  $\sum_{i=1}^n i = \frac{n(n+1)}{2}$ .

# Induktion: Was wird gebraucht?

- Beweise Aussagen, z.B.  $\sum_{i=1}^n i = \frac{n(n+1)}{2}$ .
- Induktionsanfang:
  - Die gegebene Gleichung bzw. Ungleichung stimmt für einen oder mehrere Basisfälle.
  - z.B.:  $\sum_{i=1}^1 i = 1 = \frac{1(1+1)}{2}$ .

# Induktion: Was wird gebraucht?

- Beweise Aussagen, z.B.  $\sum_{i=1}^n i = \frac{n(n+1)}{2}$ .
- Induktionsanfang:
  - Die gegebene Gleichung bzw. Ungleichung stimmt für einen oder mehrere Basisfälle.
  - z.B.:  $\sum_{i=1}^1 i = 1 = \frac{1(1+1)}{2}$ .
- Induktionshypothese: Wir nehmen an, die Aussage stimmt für ein allgemeines  $n$ .

# Induktion: Was wird gebraucht?

- Beweise Aussagen, z.B.  $\sum_{i=1}^n i = \frac{n(n+1)}{2}$ .
- Induktionsanfang:
  - Die gegebene Gleichung bzw. Ungleichung stimmt für einen oder mehrere Basisfälle.
  - z.B.:  $\sum_{i=1}^1 i = 1 = \frac{1(1+1)}{2}$ .
- Induktionshypothese: Wir nehmen an, die Aussage stimmt für ein allgemeines  $n$ .
- Induktionsschritt ( $n \rightarrow n + 1$ ):
  - Aus der Gültigkeit der Aussage für  $n$  (Induktionshypothese) folgt die Gültigkeit für  $n + 1$ .
  - z.B.:  $\sum_{i=1}^{n+1} i = n + 1 + \sum_{i=1}^n i = n + 1 + \frac{n(n+1)}{2} = \frac{(n+2)(n+1)}{2}$ .

# Induktion: Beispiel

- Zu zeigen:  $\sum_{i=0}^n r^i = \frac{1-r^{n+1}}{1-r}$ .

# Induktion: Beispiel

- Zu zeigen:  $\sum_{i=0}^n r^i = \frac{1-r^{n+1}}{1-r}$ .

- Induktionsanfang:

$$n = 0: \sum_{i=0}^0 r^i = 1 = \frac{1-r^1}{1-r}.$$

# Induktion: Beispiel

■ Zu zeigen:  $\sum_{i=0}^n r^i = \frac{1-r^{n+1}}{1-r}$ .

■ Induktionsanfang:

$$n = 0: \sum_{i=0}^0 r^i = 1 = \frac{1-r^1}{1-r}.$$

■ Induktionsschritt ( $n \rightarrow n + 1$ ):

$$\begin{aligned} \sum_{i=0}^{n+1} r^i &= r^{n+1} + \sum_{i=0}^n r^i \\ &= r^{n+1} + \frac{1-r^{n+1}}{1-r} = \frac{r^{n+1} - r^{n+2} + 1 + r^{n+1}}{1-r} = \frac{1-r^{n+2}}{1-r}. \end{aligned}$$



# Analyse

Wie oft wird `f()` aufgerufen?

```
for(unsigned i = 1; i <= n/3; i += 3)
    for(unsigned j = 1; j <= i; ++j)
        f();
```

# Analyse

Wie oft wird  $f()$  aufgerufen?

```
for(unsigned i = 1; i <= n/3; i += 3)
    for(unsigned j = 1; j <= i; ++j)
        f();
```

Das Code-Fragment ruft  $f()$   $\Theta(n^2)$  mal auf: die äußere Schleife wird  $n/3$  mal durchlaufen, und die innere Schleife ruft  $f()$   $i$  mal auf.

# Analyse

Wie oft wird `f()` aufgerufen?

```
for(unsigned i = 0; i < n; ++i) {  
    for(unsigned j = 100; j*j >= 1; --j)  
        f();  
    for(unsigned k = 1; k <= n; k *= 2)  
        f();  
}
```

# Analyse

Wie oft wird `f()` aufgerufen?

```
for(unsigned i = 0; i < n; ++i) {  
    for(unsigned j = 100; j*j >= 1; --j)  
        f();  
    for(unsigned k = 1; k <= n; k *= 2)  
        f();  
}
```

Wir können die erste innere Schleife ignorieren, weil sie `f()` nur konstant oft aufruft.

# Analyse

Wie oft wird  $f()$  aufgerufen?

```
for(unsigned i = 0; i < n; ++i) {  
    for(unsigned j = 100; j*j >= 1; --j)  
        f();  
    for(unsigned k = 1; k <= n; k *= 2)  
        f();  
}
```

Wir können die erste innere Schleife ignorieren, weil sie  $f()$  nur konstant oft aufruft.

Die zweite innere Schleife ruft  $f()$   $\lfloor \log_2(n) \rfloor + 1$  mal auf, in Summe haben wir  $\Theta(n \log(n))$  Aufrufe.

# Analyse

Wie oft wird `f()` aufgerufen?

```
void g(unsigned n) {  
    for (unsigned i = 0; i < n ; ++i) {  
        g(i)  
    }  
    f();  
}
```

# Analyse

Wie oft wird  $f()$  aufgerufen?

```
void g(unsigned n) {  
    for (unsigned i = 0; i < n ; ++i) {  
        g(i)  
    }  
    f();  
}
```

$$T(0) = 1$$

# Analyse

Wie oft wird  $f()$  aufgerufen?

```
void g(unsigned n) {  
    for (unsigned i = 0; i < n ; ++i) {  
        g(i)  
    }  
    f();  
}
```

$$T(0) = 1$$

$$T(n) = 1 + \sum_{i=0}^{n-1} T(i)$$



# Analyse

Wie oft wird  $f()$  aufgerufen?

```
void g(unsigned n) {  
    for (unsigned i = 0; i < n ; ++i) {  
        g(i)  
    }  
    f();  
}
```

$$T(0) = 1$$

$$T(n) = 1 + \sum_{i=0}^{n-1} T(i)$$

$n$	0	1	2	3	4
$T(n)$	1	2	4	8	16

# Analyse

Wie oft wird  $f()$  aufgerufen?

```
void g(unsigned n) {  
    for (unsigned i = 0; i < n ; ++i) {  
        g(i)  
    }  
    f();  
}
```

$$T(0) = 1$$

$$T(n) = 1 + \sum_{i=0}^{n-1} T(i)$$

$n$	0	1	2	3	4
$T(n)$	1	2	4	8	16

Hypothese:  $T(n) = 2^n$ .

# Analyse

Wie oft wird  $f()$  aufgerufen?

```
void g(unsigned n) {  
    for (unsigned i = 0; i < n ; ++i) {  
        g(i)  
    }  
    f();  
}
```

Hypothese:  $T(n) = 2^n$ .

Induktionsschritt:

$$\begin{aligned} T(n) &= 1 + \sum_{i=0}^{n-1} 2^i \\ &= 1 + 2^n - 1 = 2^n \end{aligned}$$

# **3. Programmieraufgabe**

# Das Auswahlproblem

Eingabe

- Unsortiertes Array  $A = (A_1, \dots, A_n)$  paarweise verschiedener Werte
- Zahl  $1 \leq k \leq n$ .

Ausgabe:  $A[i]$  mit  $|\{j : A[j] < A[i]\}| = k - 1$

## Spezialfälle

$k = 1$ : Minimum: Algorithmus mit  $n$  Vergleichsoperationen trivial.

$k = n$ : Maximum: Algorithmus mit  $n$  Vergleichsoperationen trivial.

$k = \lfloor n/2 \rfloor$ : Median.

# Pivotieren



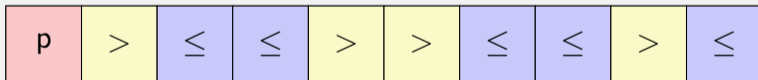
# Pivotieren

- 1 Wähle ein Element  $p$  als Pivotelement



# Pivotieren

- 1 Wähle ein Element  $p$  als Pivotelement
- 2 Teile  $A$  in zwei Teile auf, den Rang von  $p$  bestimmend.





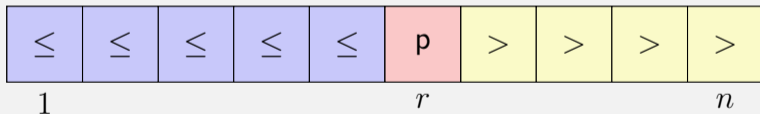
# Pivotieren

- 1 Wähle ein Element  $p$  als Pivotelement
- 2 Teile  $A$  in zwei Teile auf, den Rang von  $p$  bestimmend.



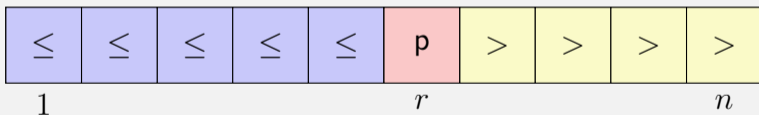
# Pivotieren

- 1 Wähle ein Element  $p$  als Pivotelement
- 2 Teile  $A$  in zwei Teile auf, den Rang von  $p$  bestimmend.



# Pivotieren

- 1 Wähle ein Element  $p$  als Pivotelement
- 2 Teile  $A$  in zwei Teile auf, den Rang von  $p$  bestimmend.
- 3 Rekursion auf dem relevanten Teil. Falls  $k = r$ , dann gefunden.



# Algorithmus Partition( $A[l..r], p$ )

**Input** : Array  $A$ , welches den Sentinel  $p$  im Intervall  $[l, r]$  mindestens einmal enthält.

**Output** : Array  $A$  partitioniert um  $p$ . Rückgabe der Position von  $p$ .

**while**  $l \leq r$  **do**

**while**  $A[l] < p$  **do**

$l \leftarrow l + 1$

**while**  $A[r] > p$  **do**

$r \leftarrow r - 1$

    swap( $A[l], A[r]$ )

**if**  $A[l] = A[r]$  **then**

$l \leftarrow l + 1$

**return**  $l - 1$

# Algorithmus Quickselect ( $A[l..r], k$ )

**Input** : Array  $A$  der Länge  $n$ . Indizes  $1 \leq l \leq k \leq r \leq n$ , so dass für alle  $x \in A[l..r]$  :  $|\{j|A[j] \leq x\}| \geq l$  und  $|\{j|A[j] \leq x\}| \leq r$ .

**Output** : Wert  $x \in A[l..r]$  mit  $|\{j|A[j] \leq x\}| \geq k$  und  $|\{j|x \leq A[j]\}| \geq n - k + 1$

**if**  $l=r$  **then**

  | return  $A[l]$ ;

$x \leftarrow \text{RandomPivot}(A[l..r])$

$m \leftarrow \text{Partition}(A[l..r], x)$

**if**  $i < m$  **then**

  | return QuickSelect( $A[l..m - 1], k$ )

**else if**  $i > m$  **then**

  | return QuickSelect( $A[m + 1..r], k$ )

**else**

  | **return**  $A[l]$

# Algorithmus RandomPivot ( $A[l..r]$ )

**Input** : Array  $A$  der Länge  $n$ . Indizes  $1 \leq l \leq i \leq r \leq n$

**Output** : Zufälliger “guter” Pivot  $x \in A[l..r]$

**repeat**

    wähle zufälligen Pivot  $x \in A[l..r]$

$p \leftarrow l$

**for**  $j = l$  **to**  $r$  **do**

**if**  $A[j] \leq x$  **then**  $p \leftarrow p + 1$

**until**  $\lfloor \frac{3l+r}{4} \rfloor \leq p \leq \lceil \frac{l+3r}{4} \rceil$

**return**  $x$

*Dieser Algorithmus ist nur von theoretischem Interesse und liefert im Erwartungswert nach 2 Durchläufen einen guten Pivot. Praktisch kann man im Algorithmus Quickselect direkt einen zufälligen Pivot uniformverteilt ziehen.*

Fragen?

Fragen?

Let's get to work.