

# Algorithmen und Datenstrukturen

Dr. M. Lüthi, Dr. G. Röger  
Frühjahrssemester 2018

Universität Basel  
Fachbereich Informatik

## Übungsblatt 2

Abgabe: 16. März 2018

### Aufgabe 2.1 (Logarithmische Laufzeit, 1 Punkt)

Beweisen Sie für beliebige  $a, b > 1$ , dass  $\log_a n \in \Theta(\log_b n)$ .

Erläuterung: Sie zeigen damit, dass die Basis des Logarithmus bei der asymptotischen Laufzeitanalyse nicht von Belang ist.

### Aufgabe 2.2 (Laufzeitanalyse Mergesort, 2 Punkte)

Zeigen Sie: Bottom-Up-Mergesort ist mindestens  $n \log_2 n$ -wachsend, d.h. es gibt Konstanten  $c > 0$  und  $n_0 > 0$ , so dass für alle  $n \geq n_0$  gilt, dass  $T(n) \geq cn \log_2 n$ .

Hinweis: Es reicht, nur Zweierpotenzen zu betrachten. Sie finden die Anzahl der Operationen pro Iteration der äusseren Schleife auf den Folien.

### Aufgabe 2.3 (Laufzeitanalyse für rekursiven Algorithmus, 4 Punkte)

Betrachten Sie den folgenden Algorithmus, bei dem Sie annehmen dürfen, dass der Aufruf von `bar` konstante Zeit benötigt.

```
1 def foo(n, val):
2     if n == 1:
3         return bar(1, 1, 1)
4     x = foo(n//2, val)
5     y = foo(n//2, val + 1)
6     z = foo(n//2, val + 2)
7     return bar(x, y, z)
```

Es ist leicht zu sehen, dass Parameter `val` keinen Einfluss auf die Laufzeit hat. Für die Abschätzung der Laufzeit in Abhängig der Eingabegrösse reicht es also, nur  $n$  zu betrachten.

Die Kosten in Zeilen 2, 3, und 7 sind jeweils konstant. Sie können also annehmen, dass alle zusammen nicht mehr als  $c$  Operationen benötigen (für irgendeine Konstante  $c$ ).

Zeigen Sie, dass die Laufzeit des Algorithmus in  $O(n^{1.6} \log_2 n)$  liegt.

Hinweis: Betrachten Sie zunächst den Fall, in dem  $n$  eine Zweierpotenz ist.

### Aufgabe 2.4 (Gross-O, 1 + 1 + 1 Punkte)

Betrachten Sie die Komplexitätsklassen  $O(1)$ ,  $O(n)$ ,  $O(n^2)$  und  $O(n^3)$ . Geben Sie jeweils die kleinste dieser Klassen an, in denen die Laufzeit der folgenden Algorithmen liegt. Begründen Sie Ihre Antwort: Warum liegt die Laufzeit in der Klasse, warum in keiner kleineren? (Kein formaler Beweis notwendig).

```
(a) 1 def spam(n, array):
2     val = 0
3     for i in range(n): # i = 0, ..., n-1
4         for j in range(i): # j = 0, ..., i-1
5             val += array[i] * array[j]
6     return val
```

$O(n)$   
 $O(n)$

Prog muss durch beide Schleifen somit ganze Methode  $O(n^2)$

(b)

```

1 def egg(n):
2     if n%2 == 0: # n is even
3         val = 0
4         for i in range(n): # i = 0, ..., n-1
5             for j in range(n) # j = 0, ..., n-1
6                 val += 2 * i * j
7     return val
8 else:
9     return -n * 3

```

$O(n)$   
 $O(n)$   
 $\boxed{O(1)}$   
 $O(n^2)$   
 $O(n^2)$   
 $O(1)$

If  $n \neq 2$  tritt genau in der Hälfte aller Fälle ein. Laufzeit  $O(n^2)$  da das Worst Case ist.

(c)

```

1 def ham(n):
2     if n < 50:
3         val = 0
4         for i in range(n): # i = 0, ..., n-1
5             for j in range(n) # j = 0, ..., n-1
6                 val += 2 * i * j
7     return val
8 else:
9     return -n * 3

```

$O(n)$   
 $O(n)$   
 $\boxed{O(1)}$   
 $O(n^2)$   
 $O(n^2)$   
 $O(1)$

If  $n < 50$  tritt nur für 50 Fälle ein ( $n$  positiv). Alle anderen  $n$  gehen in else zweig. → Methode  $O(1)$  da  $n$  beliebig gross.

Die Übungsblätter dürfen in Gruppen von zwei Studierenden bearbeitet werden. Bitte schreiben Sie beide Namen auf Ihre Lösung.

2.1)  $\log_a(n) = \frac{\log_2(n)}{\log_2(a)} = \underbrace{\frac{1}{\log_2(a)}}_{c_a} \cdot \log_2(n) = c_a \cdot \log_2(n)$

$\log_b(n) = \frac{\log_2(n)}{\log_2(b)} = \underbrace{\frac{1}{\log_2(b)}}_{c_b} \cdot \log_2(n) = c_b \cdot \log_2(n)$

→ nur Änderung der Konstante nicht aber "Wachstum"

$\log_a(n) \in \Theta(\log_b(n))$   
 $\log_a(n) = \frac{\log_b(n)}{\log_2(a)} = \underbrace{\frac{1}{\log_2(a)}}_{<1} \cdot \overbrace{\log_b(n)}^{c_b}$

## 2.2 Annahme gleichwie TopDown ( $m = 2^k$ ) Rechnung aus den Folien

Betrachte  $m = 2^k$  mit  $k \in \mathbb{N} \cup \{0\}$

$$\begin{aligned}
 T(m) &= c_1 + 2T(m/2) + c_2 m \\
 &= c_1 + 2(c_1 + 2T(m/4) + c_2(m/2)) + c_2 m \\
 &= c_1 + 2(c_1 + 2(c_1 + 2T(m/8) + c_2(m/4)) + c_2(m/2)) + c_2 m \\
 &= (1+2+4)c_1 + (m+2m/2+4m/4)c_2 + 8T(m/8) \\
 &= \dots \\
 &= \sum_{i=0}^{k-1} 2^i c_1 + kmc_2 + 2^k c_0 \\
 T(m) &= \underbrace{\sum_{i=0}^{k-1} 2^i c_1}_{\geq 0} + c_2 m \log_2 m + \underbrace{c_0 m}_{\geq 0} \\
 \Rightarrow T(m) &\geq c_2 m \log_2 m \\
 \hookrightarrow T(n) &\geq c_2 n \log_2 n
 \end{aligned}$$

Your code will always be  $O(1)$  if you hardcode everything



2.3:  $2^{k=n}$

$$\begin{aligned}
 T(n) &= c + T(n/2) + T(n/2) + T(n/2) \\
 &= c + 3T(n/2) \\
 &= c + 3(c + 3T(n/4)) \\
 &= c + 3(c + 3(c + 3T(n/8))) \\
 &= (1+3+9)c + 27T(n/8) \\
 &\stackrel{k-1}{=} \sum_{i=0}^{k-1} 3^i c \\
 &= 3^k c \quad k = \log_2(n) \text{ da } 2^k = n \\
 &\leq \log_2(n) \cdot 3^{\log_2(n)} \\
 &\leq \log_2(n) \cdot n^{\log_2(3)} \\
 &\leq \underline{c \cdot n^{\log_2(3)} \cdot c} \\
 &\leq \underline{c \cdot n^{\log_2(3)} \cdot \log_2(n)}
 \end{aligned}$$