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
```

(b)
```
def egg(n):
    if n%2 == 0:  # n is even
        val = 0
        for i in range(n):  # i = 0,..., n-1
            for j in range(n)  # j = 0,..., n-1
                val += 2 * i * j
        return val
    else:
        return -n * 3
```

(c)
```
def ham(n):
    if n < 50:
        val = 0
        for i in range(n):  # i = 0,..., n-1
            for j in range(n)  # j = 0,..., n-1
                val += 2 * i * j
        return val
    else:
        return -n * 3
```

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