max planck institut informatik

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Benjamin Doerr, Dr. Reto Spöhel **Wintersemester 2011/12**
Übungsleitung: Alexander Kobel

Übung zu Grundzüge von Algorithmen und Datenstrukturen

http://www.mpi-inf.mpg.de/departments/d1/teaching/ws11/grads/

Übungsblatt 3 Abgabe: Donnerstag, 10. November 2011, 12:00 Uhr

Hinweise zur Abgabe der Aufgaben: Schreiben Sie klar und deutlich in gebührendem Abstand zu weiterem Text Ihren **Namen** und **Matrikelnummer**, ihre **Übungsgruppe als arabische Ziffer** sowie den **Namen Ihres Tutors** auf Ihre Abgabe. **Heften** Sie mehrere Blätter geeignet zusammen. Legen Sie ihre Lösung bis zum Abgabetermin in den **unteren rechten Briefkasten** im Erdgeschoss von Gebäude E1 3 (neben Hörsaal 001). Achtung: Verwechseln Sie den Briefkasten nicht mit dem für die *Stammvorlesung* Algorithms and Data Structures!

Aufgabe 1 (*Schriftliche Übung, 2 Punkte*)

In der Vorlesung haben wir den Algorithmus Merge-Sort besprochen.

Algorithm 1: Merge-Sort

Input: array $a[1..n]$
Output: afterwards, $a[1..n]$ is sorted

```
1: if n > 1 then
2:     m := ⌊n/2⌋;
3:     Merge-Sort(a[1..m]);
4:     Merge-Sort(a[m + 1..n]);
5:     Merge(a[1..n], m);
```

Hierbei ist die Prozedur Merge wie folgt spezifiziert.

Algorithm 2: Merge

Input: array $a[1..n]$, integer t such that $a[1..t]$ and $a[t+1..n]$ are sorted
Output: afterwards, $a[1..n]$ is sorted

Beweisen Sie die Korrektheit von Merge-Sort.

Bitte wenden...

Aufgabe 2 (*Schriftliche Übung, 6 Punkte*)

Die Merge-Prozedur kann wie folgt implementiert werden.

Algorithm 3: Merge

Input: array $a[1..n]$, integer t such that $a[1..t]$ and $a[t+1..n]$ are sorted
Output: afterwards, $a[1..n]$ is sorted

```
1: i := 1; j := t + 1
2: for k = 1, ..., n do
3:    if j = n + 1 or (i ≤ t and a[i] < a[j]) then
4:       b[k] := a[i];
5:       i := i + 1;
6:    else
7:       b[k] := a[j];
8:       j := j + 1
9: a[1..n] := b[1..n];
```

(Beachten Sie die Klammer in Zeile 3; diese ist wichtig!)

Beweisen Sie die Korrektheit von Merge. Nehmen Sie dazu der Einfachkeit halber an, dass der Eingabearray n *verschiedene* Elemente enthält, und gehen Sie wie folgt vor:

a) Zeigen Sie, dass zu Beginn jeder Iteration der for-Schleife die Gleichung $i+(j-t) = k+1$ gilt. [1 P.]

b) Was können Sie über die Werte von i und j sagen, wenn keine der beiden Bedingungen in Zeile 3 erfüllt ist, also die Anweisungen in Zeilen 7-8 ausgeführt werden? [1 P.]

c) Zeigen Sie, dass zu Beginn jeder Iteration der for-Schleife die folgende Invariante gilt: Zusammen enthalten die drei Teilarrays $b[1..k-1]$, $a[i..t]$ und $a[j..n]$ genau die n Elemente des ursprünglichen Eingabearrays. [1.5 P.]

d) Zeigen Sie, dass zu Beginn jeder Iteration der for-Schleife die folgende Invariante gilt: $b[1] \leq b[2] \leq \ldots \leq b[k-1]$, und jedes Element in $b[1..k-1]$ ist kleiner als jedes Element in $a[i..t]$ oder $a[j..n]$. [1.5 P.]

e) Folgern Sie aus den vorhergehenden Invarianten die Korrektheit von Merge. [1 P.]

Hinweis: Beachten Sie, dass es grundsätzlich drei Möglichkeiten dafür gibt, wie ein Schleifendurchlauf erfolgen kann:

- *die Anweisungen in Zeilen 4-5 werden ausgeführt, weil die* erste *Bedingung in Zeile 3 erfüllt ist,*
- *—¨—, weil (nur) die* zweite *Bedingung in Zeile 3 erfüllt ist,*
- *die Anweisungen in Zeilen 7-8 werden ausgeführt, weil keine der beiden Bedingungen in Zeile 3 erfüllt ist.*

Siehe nächste Seite...

Aufgabe 3 (*Schriftliche Übung, 5 Punkte*)

Sei $c(n)$ die maximale Anzahl von Vergleichen, die Merge-Sort für Eingabearrays der Länge n benötigt. Wie in der Vorlesung diskutiert, gilt die rekursive Abschätzung

$$c(n) \leq \begin{cases} 2\,c(\lceil \frac{n}{2} \rceil) + n, & n \geq 2, \\ 1, & n = 1. \end{cases}$$

In dieser Aufgabe sehen wir, wie man das exakte Lösen einer solchen Rekursion umgehen kann und trotzdem die richtige asymptotische Schranke erhält.

a) Zeigen Sie per Induktion, dass $c(n) \leq n \cdot (\log_2 n + 1)$, falls n eine Zweierpotenz ist. [1 P.]

b) Folgern Sie daraus, dass für beliebige Werte von n Merge-Sort verwendet werden kann, um n Elemente mit $O(n \log n)$ Vergleichen zu sortieren. [2 P.]

Hinweis: Verwenden Sie "Dummy-Elemente", um die Eingabegröße auf eine Zweierpotenz zu erhöhen.

c) Beweisen Sie per Induktion, dass die durch

$$\widehat{c}(n) = \begin{cases} 2\,\widehat{c}(\lceil \frac{n}{2} \rceil) + n, & n \geq 2, \\ 1, & n = 1. \end{cases}$$

definierte Folge $\widehat{c}(n)$ monoton steigend ist, d.h., dass $\widehat{c}(n) \leq \widehat{c}(n+1)$ für alle $n \in \mathbb{N}$. Folgern Sie daraus, dass für beliebige Eingabegrößen n auch die ursprüngliche Version von Merge-Sort (ohne Dummy-Elemente) nur $O(n \log n)$ Vergleiche benötigt, um n Elemente zu sortieren. [2 P.]

Aufgabe 4 (*Schriftliche Übung, 3 Punkte*)

Wir haben in der Vorlesung gesehen, dass Select-Minimum das kleinste von n verschiedenen Elementen mit $n - 1 \in O(n)$ vielen Vergleichen findet.

a) Geben Sie einen Algorithmus an, der mit $O(n)$ vielen Vergleichen das *zweitkleinste* Element von n verschiedenen Elementen findet. [1 P.]

b) Geben Sie einen Algorithmus an, der mit höchstens $(n-1) + (\lceil \log_2 n \rceil - 1)$ vielen Vergleichen das zweitkleinste Element von n verschiedenen Elementen findet. [2 P.]

Hinweis: Denken Sie über Alternativen zu Select-Minimum nach, um das kleinste von n Elementen zu finden.

Sie brauchen die Algorithmen nicht als Pseudocode anzugeben – beschreiben Sie einfach, wie das zweitkleinste Element mit der geforderten Anzahl von Vergleichen gefunden werden kann.