# Algorithmische Mathematik I

Wintersemester 2011 / 2012
Prof. Dr. Sven Beuchler
Peter Zaspel

universitätbonn

## Übungsblatt 4.

Abgabe am **16.11.2011**.

**Aufgabe 1.** (Quick-Sort)

a) Wenden Sie den Sortieralgorithmus *Quick-Sort* auf die Zahlensequenz

$$0, 5, 7, 2, 11, 3, 9, 4, 1, 8$$

an. Geben Sie dabei jeweils alle Schritte an für den Fall dass das Pivot-Element stets das erste, mittlere und das letzte Element ist.

b) Geben Sie die exakte Laufzeit des Quick-Sort-Algorithmus für den Fall an, dass das zu sortierende Array der Länge $N = 2^n - 1$ ($N$ ungerade) bereits sortiert ist. Das Pivot-Element sei hierbei jeweils das mittlere Element.

c) Überlegen Sie sich, ob und wie der Quick-Sort-Algorithmus abgeändert werden muss, wenn nicht alle Einträge von einander verschieden sind.

(5 Punkte)

**Aufgabe 2.** (Stabile Sortierverfahren)

Ein Sortierverfahren heißt stabil, wenn es Einträge mit dem selben Wert/Schlüssel in der gleichen Reihenfolge belässt. Entscheiden Sie für die drei aus der Vorlesung bekannten Sortierverfahren, ob diese stabil sind oder nicht. Begründen Sie Ihre Entscheidung.

(5 Punkte)

**Aufgabe 3.** (Sortierverfahren)

a) Geben Sie ebenfalls die exakte Laufzeit der Algorithmen *Bubble-Sort* und *Merge-Sort* für den Fall an, dass das zu sortierende Array der Länge $N$ bereits sortiert ist. Für Merge-Sort kann die Zahl $N$ an Elementen als $N = 2^n$ vorausgesetzt werden. Die Aufteilung der Teillisten erfolgt stets in der Mitte.

b) Sei eine Folge $x_1, \dots x_N$ von Zahlen zum Sortieren gegeben. Welche Struktur muss die Folge haben, damit der Merge-Algorithmus von Merge-Sort exakt $m+l-1$ Vergleiche benötigt.

(5 Punkte)

**Aufgabe 4.** (Graphen)

Laden Sie sich die PDF-Datei mit den Beispiel-Graphen von der Vorlesungshomepage herunter. Betrachten Sie nun den ersten darin enthaltenen Graphen. Führen Sie für diesen eindeutige Bezeichner $a, b, \dots$ für alle Kanten ein. (Ignorieren Sie nachfolgend die Kantengewichte des Graphen, also die Zahlen unmittelbar neben den Kanten.)

a) Geben Sie den Graphen formal als Tupel $G = (V, E, \Psi)$ mit den entsprechenden Mengen $V, E$ bzw. der Abbildung $\Psi$ an.

b) Offensichtlich ist der Graph *einfach*. Geben Sie daher ebenfalls die verkürzte Schreibweise mit $G = (V(G), E(G))$ und die ensprechenden Mengen $V(G)$ und $E(G)$ an.

c) Notieren Sie ebenfalls für jeden Knoten $v \in V(G)$ die Vorgänger- und Nachfolger-Knoten-Mengen $\Gamma^+(v)$ und $\Gamma^-(v)$.

(5 Punkte)

**Programmieraufgabe 1.** (Effizienz-Analyse)

Wie Sie in der letzten Programmieraufgabe gesehen haben ist offensichtlich die Implementierung des Bubble-Sort-Verfahrens mit verketteten Listen nicht besonders effizient. Hier sind Arrays eindeutig besser geeignet.

a) Erweitern Sie Ihre vorangegangene Implementierung um einen alternativen Ausführungszweig, in dem Bubble-Sort auf einem Array durchgeführt wird.

b) Machen Sie sich nun mittels des Internets (google & co.) oder mittels geeigneter C/C++–Literatur mit der Funktion `gettimeofday` vertraut. Diese erlaubt es durch wiederholtes Aufrufen und geeignete Differenzenbildung die Laufzeit einzelner Routinen eines Programms zu bestimmen.

c) Erweitern Sie Ihre Implementierung derart, dass Sie die Laufzeit jeweils für Bubble-Sort auf verketteten Listen und auf Arrays bestimmen können.

d) Machen Sie sich mit dem OpenSource-Programm `gnuplot` vertraut. Dieses erlaubt es, Graphen aus Daten zu erzeugen.

e) Arbeiten Sie Ihr Programm so um, dass es automatisiert die Laufzeit für beide Bubble-Sort-Varianten für Problemgrößen von $N = 10, 100, 1000, 10000, 100000, ...$ (soweit wie Ihr Rechner es in annehmbarer Zeit durchführen kann) bestimmt und in eine Datei herausschreibt. Diese Datei lassen Sie von Gnuplot einlesen und die beiden Laufzeiten als Linien-Graphen mit geeigneten Beschriftungen an den Koordinatenachsen und Graphen zeichnen. (Die Einträge in den zu sortierenden Arrays sollen jeweils zufällig mit der Methode `rand()` (siehe Literatur / Internet) generiert werden.)

f) Verwenden Sie den Algorithmus Merge-Sort von der Vorlesungs-Webseite und bestimmen sie auch hier analog für die vorgenannten Problemgrößen die Laufzeiten. Fügen Sie diese als weitere Linie in das Diagramm der vorangegangenen Teilaufgabe ein.

(15 Punkte)

Die Abgabe der Programmieraufgaben erfolgt in den CIP-Pools in der Woche vom 14.11. bis 18.11.2011. Die Listen für die Anmeldung zu den Abgabe-Terminen hängt in der Woche vom 07.11. bis 11.11.2011 aus.