

Abgabe: 29.11.2016, 12.30 Uhr

## Übungsblatt 6

### Aufgabe 6.1: (2+4 Punkte)

- (a) Zeigen Sie, dass *Quicksort* mit der PARTITION-Funktion aus der Vorlesung nicht stabil ist.
- (b) Geben Sie an, wie *Quicksort* und auch jeder andere nicht stabile vergleichsbasierte Sortieralgorithmus stabil gemacht werden kann, ohne dessen Laufzeit und die prinzipielle Vorgehensweise zu verändern.

### Aufgabe 6.2: (2+4 Punkte)

Rekursive Funktionen können iterativ implementiert werden, wenn man den Rekursionsstack selbst verwaltet. Da bei *Quicksort* alle rekursiven Aufrufe am Ende des Funktionsrumpfes erfolgen, können wir weitere Einsparungen vornehmen. Dazu stellen wir uns vor, dass auf dem Stack Indexpaare  $(\ell, r)$  liegen, die die Teilfelder beschreiben, die noch sortiert werden müssen. Diese können wir uns als Jobs vorstellen. Um einen Job  $(\ell, r)$  abzarbeiten, führen wir PARTITION auf dem entsprechenden Teilfeld aus und müssen anschließend zwei Jobs  $(\ell, q-1)$  und  $(q+1, r)$  abarbeiten. Bevor wir diese Paare jedoch auf den Stack legen, können wir aufgrund der Endrekursivität bereits den Job  $(\ell, r)$  vom Stack entfernen. Trotz dieser Ersparnis hat die Reihenfolge, in der wir die Jobs  $(\ell, q-1)$  und  $(q+1, r)$  auf den Stack legen, einen großen Einfluss auf die maximale Stackgröße.

- (a) Als Grundlage verwenden wir die PARTITION-Funktion aus der Vorlesung. Geben Sie eine Klasse von Instanzen und eine Strategie, die Jobs auf den Stack zu legen, an, bei der die maximale Stackgröße linear in  $n$  wächst.
- (b) Geben Sie eine Strategie an, bei der die maximale Stackgröße durch  $O(\log n)$  beschränkt ist, unabhängig davon, welche PARTITION-Funktion verwendet wird und auf welcher Instanz *Quicksort* ausgeführt wird. Beweisen Sie, dass Ihre Strategie die Anforderungen erfüllt.

### Aufgabe 6.3: (6 Punkte)

Wir wollen die Laufzeiten von *Quicksort* und *Mergesort* experimentell miteinander vergleichen. Erzeugen Sie dazu für  $n = 1, \dots, 500$   $n$ -mal hintereinander ein Feld mit  $n$  zufälligen Einträgen und führen Sie *Quicksort* und *Mergesort* auf diesem Feld aus. Bestimmen Sie für jedes  $n$  die durchschnittliche Laufzeit beider Sortierverfahren und stellen Sie diese als Graphen in Abhängigkeit von  $n$  dar. Verwenden Sie dabei für *Quicksort* die PARTITION-Funktion aus der Vorlesung.

### Aufgabe 6.4: (2+4 Punkte)

Gegeben sei ein Feld  $A$  bestehend aus  $n$   $b$ -Bit-Zahlen.

- (a) Zeigen Sie, wie für ein gegebenes  $r \in \{1, \dots, b\}$  das Feld  $A$  mit einer Variante von *Radixsort* mit Laufzeit  $O\left(\frac{b}{r} \cdot (n + 2^r)\right)$  sortiert werden kann.
- (b) Wie sollte  $r$  in Abhängigkeit von  $b$  und  $n$  gewählt werden, damit die obere Schranke für die Laufzeit aus Aufgabenteil (a) asymptotisch minimal ist? Begründen Sie Ihre Antwort.

*Hinweis:* Ignorieren Sie Rundungsprobleme.

**Aufgabe 6.5:**

(4+2 Zusatzpunkte)

Gegeben sei ein zweidimensionales Feld  $A$  mit  $s := m \cdot n$  Einträgen. Wir wollen zwei Funktionen INIT und SUM bereitstellen. Die Funktion INIT wird einmal am Anfang aufgerufen und dient als Vorverarbeitungsschritt. Das Ausführen dieser Funktion soll in Zeit  $O(s)$  möglich sein. Die SUM-Funktion erhält als Parameter Indizes  $i_1, i_2$  sowie  $j_1, j_2$  mit  $1 \leq i_1 \leq i_2 \leq m$  und  $1 \leq j_1 \leq j_2 \leq n$  und soll in konstanter Zeit die Summe

$$\sum_{i=i_1}^{i_2} \sum_{j=j_1}^{j_2} A[i, j]$$

berechnen.

- (a) Geben Sie eine entsprechende INIT- und SUM-Funktion an.
- (b) Zeigen Sie, wie sich die Idee auf beliebige Dimensionen  $d$  übertragen lässt. Geben Sie die Größe der Konstante  $c$ , die in der  $O$ -Notation der SUM-Funktion steckt, in  $\Theta$ -Notation bzgl.  $d$  an.