Übungen zu Algorithmen **Sommersemester 2018**
WSI für Informatik **Universität Tübingen**
Kaufmann/Bekos/Schneck **11.05.2018**

Übungsblatt 4

Abgabe bis 17.05.2018
Besprechung: 21.05.2018 – 24.05.2018

Aufgabe 1: Countingsort (2 + 3 + 1 + 2 + 2 Punkte)

Wie Sie aus der Vorlesung wissen, haben vergleichsbasierte Sortieralgorithmen eine worst-case Laufzeit von mindestens $\mathcal{O}(n \log n)$. Es gibt allerdings auch Algorithmen, die ohne Vergleiche sortieren können, wie zum Beispiel Bucketsort. Ein anderes Beispiel ist *Countingsort*, das Sie im Folgenden betrachten sollen. Nehmen Sie dabei an, dass A ein Array der Länge n ist, das nur natürliche Zahlen zwischen 1 und k enthält.

Algorithm 1: `Countingsort`(A, k)

```
1  initialize an array C of size k with 0 at each position;
2  for i = 1, ..., n do
3  |  C[A[i]] ⟵ C[A[i]] + 1;
4  end
5  for i = 2, ..., k do
6  |  C[i] ⟵ C[i] + C[i − 1];
7  end
8  initialize an array R of size n;
9  for i = n, ..., 1 do
10 |  R[C[A[i]]] ⟵ A[i];
11 |  C[A[i]] ⟵ C[A[i]] − 1;
12 end
13 return R;
```

(a) Sei $A = [7, 1, 4, 1, 2, 5, 4, 7, 1, 5, 2]$. Stellen Sie die Funktionsweise von Countingsort an diesem Beispiel graphisch dar.

(b) Argumentieren Sie, dass `Countingsort`(A) das Array A korrekt sortiert.

(c) Analysieren Sie die Laufzeit von `Countingsort` im worst-case. Geben Sie Ihre Angaben in $\mathcal{O}$-Notation an.
Hinweis: Die Laufzeit hängt nicht nur von n ab!

(d) Ein Sortierverfahren heißt *stabil*, wenn Elemente mit gleichem Wert im Output-Array in der gleichen Reihenfolge sind wie im Input-Array. Begründen Sie, dass `Countingsort` stabil ist.

(e) Nehmen Sie an, dass die for-Schleife in Zeile 9 von `Countingsort` bei 1 startet und bei n endet (anstatt der im Pseudocode angegebenen Reihenfolge von n bis 1). Argumentieren Sie, dass diese Modifikation von `Countingsort` immer noch korrekt sortiert. Ist der modifizierte Algorithmus stabil?

Aufgabe 2: Implementation von Sortierverfahren (6 + 6 + (opt. 3) Punkte)

Laden Sie die Java-Vorlage aus dem Moodle herunter und implementieren Sie die folgenden Methoden:

(a) Mergsort in `mergeSort(double[] array)`.

(b) Quicksort in `quickSort(double[] array)`.

(c) Countingsort in `countingSort(int[] array)`.

Zählen Sie die Anzahl der Elementvergleiche für Mergesort und Quicksort und stellen Sie deren Verlauf für die in der Java-Vorlage angegebenen n grafisch dar. Bitte fügen Sie diese Grafik Ihrer pdf-Abgabe hinzu.
Verwenden Sie bei Ihrer Implementierung sinnvolle Variablennamen und kommentieren Sie Ihren Code! Laden Sie Ihre Lösung ins Moodle. Nicht kompilierende Abgaben werden **mit 0 Punkten** bewertet.

Aufgabe 3: Anwendungen von Sortieren (3 + 3 + 2 Punkte)

Gegeben eine Liste A von n nicht notwendigerweise verschiedenen Zahlen. Geben Sie Algorithmen in Pseudocode an, die mit Hilfe von Sortieren die folgenden Probleme in $\mathcal{O}(n \log n)$ lösen. Verwenden Sie in Ihrem Pseudocode sinnvolle Variablennamen und erklären Sie die Funktionsweise Ihres Pseudocodes. Begründen Sie die Korrektheit und die Laufzeit Ihrer Algorithmen.

(a) Gesucht ist das *aufeinanderfolgende* Zahlenpaar aus A mit der kleinsten Differenz. Zwei Elemente i und j aus A heißen aufeinanderfolgend, wenn kein k aus A existiert mit $i < k < j$.

(b) Gesucht ist das Element aus A, das am häufigsten vorkommt.

(c) Erklären Sie, wie sich die Laufzeit der beiden Algorithmen aus (a) und (b) verändert, wenn A nur natürliche Zahlen enthält.