Übungen zur Vorlesung

Algorithmen und Datenstrukturen

Übungsblatt 2

Arbeitsgruppe Kryptographie und Komplexitätstheorie
Prof. Dr. Marc Fischlin
Dr. Christian Janson
Patrick Harasser
Felix Rohrbach

Sommersemester 2019
Veröffentlicht am: 26.04.2019
Abgabe am: 10.05.2019, 12:00 Uhr

P1 (Gruppendiskussion)

Nehmen Sie sich etwas Zeit, um die folgenden Fachbegriffe in einer Kleingruppe zu besprechen, sodass Sie anschließend in der Lage sind, die Begriffe dem Rest der Übungsgruppe zu erklären:

(a) Asymptotische Notation (O, o, Ω, ω, Θ)
(b) Divide-and-Conquer Ansatz
(c) Bubblesort, Mergesort, Quicksort

P2 (Rechenregeln für asymptotische Notation)

Es seien $f_1, f_2, g_1, g_2, f, g, h \colon \mathbb{N} \to \mathbb{R}_{>0}$ Funktionen und $k \in \mathbb{R}_{>0}$ eine Konstante. Beweisen Sie folgende Aussagen:

(a) i) $f(n) = O(g(n))$ genau dann wenn $g(n) = \Omega(f(n))$;
ii) $f(n) = o(g(n))$ genau dann wenn $g(n) = \omega(f(n))$;
(b) $o(g(n)) \subseteq O(g(n))$, und $\omega(g(n)) \subseteq \Omega(g(n))$.
(c) $O(g(n)) \cap \Omega(g(n)) = \Theta(g(n))$, und $o(g(n)) \cap \Omega(g(n)) = \emptyset$.
(d) i) Ist $f_1(n) = O(g_1(n))$ und $f_2(n) = O(g_2(n))$, dann gilt $f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$;
ii) Ist $f_1(n) = O(g_1(n))$ und $f_2(n) = O(g_2(n))$, dann gilt $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$;
iii) Es gilt $f(n) = O(f(n))$;
iv) Ist $f(n) = O(g(n))$ und $g(n) = O(h(n))$, dann gilt $f(n) = O(h(n))$.

P3 (Rechnen mit asymptotischer Notation)

Tragen Sie für jedes Paar von Funktionen $f, g \colon \mathbb{N} \to \mathbb{R}_{>0}$ in der folgenden Tabelle ein, ob f in der Ordnung O, o, Ω, ω, und/oder Θ ist. Hier sind $k \geq 1$, $\varepsilon > 0$, $c > 1$, und $r < s$ Konstanten. Begründen Sie Ihre Wahl.

$f(n)$	$g(n)$	O	o	Ω	ω	Θ
$\log^k(n)$	n^ε					
n^k	c^n					
2^n	$2^{n/2}$					
$n^{\log(c)}$	$c^{\log(n)}$					
n^r	n^s					
$\log(n!)$	$\log(n^n)$					

P4 (Darstellung von Merge Sort)

Betrachten Sie das Array ganzer Zahlen $A = (14, 9, 5, 8, 11, 4, 21, 7, 6)$. Illustrieren Sie die Operationen von Merge Sort anhand von A.

P5 (Bubble Sort)

Bubble Sort ist ein sehr einfacher, aber ineffizienter Sortieralgorithmus. In dieser Aufgabe werden wir seine Korrektheit beweisen.

Bubblesort(A)

```
1 : n = A.length
2 : for i = 0 to n − 2 do
3 :     for j = n − 1 downto i + 1 do
4 :         if A[j] < A[j − 1] then
5 :             tmp = A[j]
6 :             A[j] = A[i]
7 :             A[i] = tmp
8 : return A
```

(a) Sei A' die Ausgabe von Bubblesort(A). Um zu beweisen, dass Bubblesort korrekt ist, müssen wir zeigen, dass der Algorithmus terminiert und dass

$$A'[0] \leq A'[1] \leq \cdots \leq A'[n-1] \tag{1}$$

gilt. Welche weiteren Eigenschaften müssen wir beweisen?

(b) Wir wollen nun mithilfe von Schleifeninvarianten zeigen, dass Ungleichung (1) gilt und dass Bubblesort terminiert. Stellen Sie dazu zunächst eine Schleifeninvariante für die Zeilen 2-7 auf, die die beiden Eigenschaften beweist.

(c) Stellen Sie nun eine weitere Schleifeninvariante für die Zeilen 3-7 auf und beweisen Sie diese.

(d) Nutzen Sie anschließend die Schleifeninvariante aus c), um die äußere Schleifenvariante von Zeile 2-7 beweisen zu können.

H1 (Sortieralgorithmen) (15 Punkte)

In dieser Übung sollen Sie lernen, Sortieralgorithmen selbst zu implementieren. **Bitte lesen Sie sich zunächst die allgemeinen Hinweise für die praktischen Übungen auf Moodle durch!** Laden Sie sich anschließend das vorgegebene Framework herunter, um die Aufgabe implementieren zu können.

In der Vorlesung haben Sie (unter anderem) Quick Sort und Insertion Sort als Sortieralgorithmen kennengelernt. Wie Sie auch gelernt haben, ist Quick Sort asymptotisch deutlich schneller als Insertion Sort im Durchschnitt. Jedoch gilt dies nicht für sehr kleine Datenmengen - durch die Komplexität von Quick Sort ist hier Insertion Sort oft schneller.

In dieser Aufgabe sollen Sie ein hybrides Sortierverfahren programmieren, welches zunächst genauso funktioniert wie Quick Sort, jedoch innerhalb von Quick Sort auf Insertion Sort wechselt, sobald die Anzahl der Elemente, die in einem Unterschritt sortiert werden sollen, kleiner als k wird.

(a) Die Elemente, die Sie sortieren sollen, sind Karten eines verallgemeinerten Kartenspiels: Jede Karte hat eine Farbe, Hearts (Herz), Diamonds (Karo), Clubs (Kreuz) oder Spades (Pik), und einen Wert, der eine beliebige, ganze Zahl sein kann. Diese Datenstruktur ist in `lab/Card.java` vorgegeben. Implementieren Sie für diese Klasse die Funktion `compareTo`, die die aktuelle Instanz mit einer anderen Instanz vergleicht. Dabei wird zunächst nach Wert verglichen und falls dieser identisch ist, nach Farbe (wobei Diamonds < Hearts < Spades < Clubs gilt). `compareTo` soll -1 zurückgeben falls `this` echt kleiner ist als `other`, 1 falls `this` echt größer ist als `other` und 0 falls beide gleich groß sind. (3 Punkte)

(b) Implementieren Sie nun die Methode `HybridSort.sort` in der Datei `lab/HybridSort.java`, die eine Liste von Karten aufsteigend sortieren soll. Nuzten Sie für die Array-Zugriffe die Klasse `frame/SortArray.java`. Wir nutzen diese Klasse, um die Lese- und Schreibzugriffe auf das Array zu zählen - versuchen Sie auf keinen Fall, diese Klasse zu umgehen! Verwenden Sie für das Pivot-Element in Quick Sort immer das erste Element im zu sortierenden Abschnitt des Arrays. (10 Punkte)

(c) Eine deterministische Wahl des Pivot-Elements (wie die Wahl des ersten, letzten oder mittleren Elements) kann immer dazu führen, dass Quick Sort eine quadratische Laufzeit auf eine bestimmte Weise formatierten Eingaben hat. Unsere Wahl des ersten Elements (in b) führt beispielsweise bei weitgehend schon sortieren Arrays zu langen Laufzeiten. Implementieren Sie daher die von `HybridSort` abgeleitete Klasse `lab/HybridSortRandomPivot.java`, welche das Pivotelement zufällig wählt. Um möglichst wenig doppelten Code zu produzieren, sollten Sie Ihren Code in `lab/HybridSort.java` so ändern, dass die Wahl des Pivotelements in eine eigene Funktion ausgelagert ist, welche Sie dann in `HybridSortRandomPivot` überschreiben können. (2 Punkte)

Testen Sie ihre Implementierung! Wir haben Ihnen mit `lab/PublicTests.java` einige Tests vorgegeben, aber Sie können und sollten auch eigene Testfälle konstruieren. Wir haben Ihnen dazu in `lab/YourTests.java` ein Template vorgegeben, in dem Sie Ihre Tests implementieren können. Wie in den allgemeinen Hinweisen beschrieben, können Sie die Formatierung Ihrer Abgabe und ob diese unter openJDK 8 läuft auf unserem Testserver überprüfen.