TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Fachbereich Informatik
AG Algorithmen und Komplexität

Ausgabedatum: 15.01.2015
Version: 2015-01-14 13:59

10. Übungsblatt für alle Tracks zur Vorlesung Entwurf und Analyse von Algorithmen, WS 14/15

Abgabe: *Bis Freitag, 23.01.2015, 12:00 Uhr, Kasten im Treppenhaus 48-6.*

Basisaufgaben

B10.1: Speicherbedarf von Sortieralgorithmen

3 Punkte

Welche Sortieralgorithmen, die Sie aus der Vorlesung kennen, brauchen wie viel Speicher (zusätzlich zur Eingabe)?

Geben Sie für die Algorithmen

- Bubblesort,
- Shellsort,
- Quicksort und
- 2-Wege-Mergesort

jeweils an, in welchen der Komplexitätsklassen $\mathcal{O}(1)$, $\mathcal{O}(\log n)$, $\mathcal{O}(n)$ und $\omega(n)$ der (zusätzliche) Speicherbedarf schlimmstenfalls liegt. Hierbei ist n die Länge des Eingabearrays.

B10.2: Inversionstafeln

3 Punkte

Gegeben sei die Inversionentafel $\mathbf{b} = b_1, b_2, \ldots, b_n$ der Permutation $\mathbf{a} = a_1, \ldots, a_n$ auf der Menge $\{1, 2, \ldots, n\}$. Beschreiben Sie einen Algorithmus, der aus der Eingabe $\mathbf{b}$ die entsprechende Permutation $\mathbf{a}$ rekonstruiert. Verdeutlichen Sie Ihr Verfahren, indem Sie seine Arbeitsweise anhand eines (nicht entarteten) Beispiels der Größe $n = 10$ illustrieren.

Lösungen, die nur Teile der Spezifikation erfüllen, geben Teilpunkte.

B10.3: Quicksort auf verketteten Listen

3 Punkte

Implementieren Sie Quicksort auf linearen Listen in verketteter Repräsentation.
(Ob einfach oder doppelt verkettet dürfen Sie sich aussuchen, aber seien Sie konsistent in Ihrer Wahl.)

Die Θ-Klasse der (erwarteten) Laufzeit soll sich nicht von der Implementierung auf Arrays im Buch unterscheiden. Die Eingabeliste muss nicht erhalten bleiben.

Hinweis: Die „faule“ Lösung, die Eingabe in ein Array zu kopieren und dann die bekannte Implementierung zu benutzen, ist explizit nicht gesucht und bringt keine Punkte.

B10.4: Stabilität I

3 Punkte

In der Praxis kommt es oft vor, dass mehrere unterschiedliche Einträge einer Wörterbuchdatenstruktur den gleichen Schlüssel haben. Ein Sortierverfahren, das die Reihenfolge der Einträge mit gleichem Schlüssel untereinander nicht ändert, heißt *stabil*. Dies ist zum Beispiel wünschenswert, wenn man sukzessive nach mehreren Schlüsseln sortieren möchte.

Untersuchen Sie die folgenden Sortieralgorithmen aus der Vorlesung auf Stabilität! Geben Sie jeweils eine präzise Begründung für Ihre Behauptung an.

Geben Sie für die nicht stabilen Verfahren außerdem an, ob und wie man sie mit kleinen Änderungen stabil machen kann.

a) Insertion-Sort

b) Quicksort

c) Mergesort

Hinweis: Untersuchen Sie genau die im Buch gegebenen Implementierungen!

Aufbauaufgaben

A10.1: Stabilität II

1 + 1 + 2 Punkte

Untersuchen Sie die Sortieralgorithmen aus der Vorlesung auf Stabilität! Geben Sie jeweils eine präzise Begründung für Ihre Behauptung an.

Geben Sie für die nicht stabilen Verfahren außerdem an, ob und wie man sie mit kleinen Änderungen stabil machen kann.

a) Shell-Sort

b) Heapsort

c) Angenommen, Sie müssen einen nicht stabilen Sortieralgorithmus A für ganze Zahlen verwenden, brauchen aber Stabilität. Können Sie die Eingabe so transformieren, dass das sortierte Ergebnis stabil ist, ohne die asymptotische Laufzeit von A zu verschlechtern? Sie können dabei annehmen, dass Sie A neben der Eingabefolge eine Vergleichsfunktion übergeben können.

   Entwerfen Sie einen entsprechenden Algorithmus, begründen Sie seine Korrektheit und analysieren Sie die Laufzeit.

Hinweis: Untersuchen Sie genau die im Buch gegebenen Implementierungen!

A10.2: Quicksort mit anderen Eingabeverteilungen

2 + 2 + 2 Punkte

In der Vorlesung haben wir Quicksort unter der Annahme analysiert, die Eingabe sei eine uniform zufällig gewählte Permutation. In der Praxis wird diese Annahme nicht immer erfüllt sein; wie sieht es dann mit der Laufzeit aus?

Bestimmen Sie die Θ-Klasse der Laufzeit unserer Quicksort-Implementierung für die folgenden Extremfälle für Eingaben. Diskutieren Sie inwiefern etwaige negative Effekte auch für weniger extreme Fälle auftreten.

Schlagen Sie – wo möglich – Verbesserungen an unserer Implementierung vor, die den beobachteten Effekten entgegenwirken.

a) Alle Elemente der Eingabe sind gleich.

b) Die Eingabe ist schon komplett sortiert.

c) Die Eingabe ist komplett sortiert, aber rückwärts.

A10.3: Shuffle

3 Punkte

Ein *Shuffle-Algorithmus* ordnet Eingabearrays beliebiger Größe $n \in \mathbb{N}$ so um, dass jede Permutation der Eingabe als Ergebnis möglich und gleich wahrscheinlich ist. Entwerfen Sie einen Shuffle-Algorithmus, der im Worst-Case in Zeit $\mathcal{O}(n)$ läuft.

Sie dürfen dabei nur die üblichen Formen von (Pseudo-) Zufallszahlen verwenden, d. h. entweder uniform in $[0, 1]$ verteilte, reelle Zahlen oder uniform in $\{1, \ldots, n\}$ verteilte, natürliche Zahlen.

Analysieren Sie neben der Laufzeit auch die Anzahl von Zufallszahlen und begründen Sie Ihre Behauptungen.