Algorithmen und Programmieren 3, WS 1999/2000 — 2. Übungsblatt

Abgabe bis Dienstag, 9. November 1999

7. (3 Punkte) Was passiert bei *Sortieren durch Einfügen*, wenn die gegebene Folge bereits (a) aufsteigend oder (b) absteigend sortiert ist? Ist es möglich, dass man bloß $O(n)$ Zeit benötigt?

8. (2 Punkte) Beantworten Sie die vorige Frage für *Sortieren durch Auswahl*.

9. (0 Punkte) Nehmen wir an, dass die Eingabefolge *fast sortiert* ist. Das heißt, dass jedes Element in der Ausgangsreihenfolge höchstens k Stellen von der endgültigen Position in der sortierten Reihenfolge entfernt ist. Geben Sie eine Schranke für die Laufzeit von *Sortieren durch Einfügen* in Abhängigkeit von n und k an.

10. (0 Punkte) Wie kann man *Sortieren durch Einfügen* so anpassen, dass es für Folgen, die absteigend oder fast absteigend sortiert sind, möglichst schnell läuft?

11. (2 Punkte) Geben Sie alle möglichen topologischen Sortierungen für folgende Eingabe an: $n = 6$ und $\{(6,1),(3,5),(1,4),(3,1),(2,4),(2,6),(5,1),(3,6)\}$.

12. (2 Punkte) Was passiert beim in der Vorlesung angegebenen Algorithmus für *Topologisches Sortieren*, wenn ein Paar (i,j) in der Eingabe mehrfach auftritt? Was passiert, wenn ein Paar (i,i) auftritt?

13. (0 Punkte) Untersuchen Sie verschiedene Möglichkeiten, wie man die Liste der freien Elemente beim topologischen Sortieren verwalten kann, im Hinblick auf ihre Effizienz. Kann man auf diese Liste auch gänzlich verzichten?

    Welche Variablen oder Felder im Programm aus der Vorlesung könnte man ohne Nachteil einsparen?

14. (8 Punkte) Erweitern Sie das Java-Programm zum topologischen Sortieren, sodass bei der Existenz eines Kreises nicht einfach mit einer Meldung abgebrochen wird, sondern auch ein Kreis (als „Beweis“) ausgegeben wird.

    Ihr Programm sollte nicht mehr als $O(m+n)$ zusätzliche Zeit und nicht mehr als $O(n)$ zusätzlichen Speicher brauchen.

15. (8 Punkte) Es gibt eine Variante von *Sortieren durch Verschmelzen* (mergesort), die von unten nach oben (*bottom-up*) arbeitet: Zunächst werden je zwei aufeinanderfolgende Elemente zu einem Paar zusammengefasst und gegebenenfalls vertauscht, sodass eine Folge von sortierten Zweierblöcken entsteht. Dann werden je zwei benachbarte Zweierblöcke zu einem sortierten Viererblock verschmolzen, diese werden wiederum paarweise zu Achterblöcken verschmolzen, usw.

    (a) (4 Punkte) Schreiben Sie ein Programm für dieses Sortierverfahren in Haskell oder Java[1].

    (b) (4 Punkte) Analysieren Sie die Laufzeit dieses Algorithmus.

    (c) (0 Punkte) Falls Sie für Aufgabe (a) nicht Haskell gewählt haben, analysieren Sie auch den *zusätzlichen* Speicherbedarf (zusätzlich zu dem Feld, in dem die zu sortierenden Elemente am Anfang stehen). Falls Sie für Aufgabe (a) Haskell gewählt haben, erklären Sie, warum Sie die obige Frage nicht beantworten müssen.

16. (0 Punkte) Entwerfen Sie eine bottom-up-Variante von Quicksort.

[1] oder in C oder C++ oder Pascal. Weitere Programmiersprachen auf Anfrage