Christian-Albrechts-Universität zu Kiel
Institut für Informatik und Praktische Mathematik
Prof. Dr. W.-P. de Roever
Dr. M. Steffen

Algorithmen & Datenstrukturen

Wintersemester 2001/2002 **Serie 2** 30. Oktober 2001

Thema : Sortieren

Ausgabetermin: 30. Oktober 2001

Abgabe: 15. November 2001

Aufgabe 1: [Heaps und Prioritätswarteschlangen]

Der Heap-Datentyp aus der Vorlesung ist nicht nur als Hilfsstruktur zum Sortieren geeignet.

- Eine Anwendung von Heaps sind Prioritätswarteschlangen (*priority queues*). Eine Prioritätswarteschlange ist ein Sammlung geordneter Elemente; das Einfügen darf in jeder beliebigen Reihenfolge geschehen, das Entfernen und der lesende Zugriff ist nur auf das Element mit der höchsten Priorität möglich. Diese Struktur kann man als *Menge* S von Elementen sehen, denen ein *Schlüssel* zugeordnet ist, wobei die Schlüssel geordnet sind. Folgende Operationen werden unterstützt:
  - $\mathit{Insert}(S, x)$: fügt x der Menge S hinzu
  - $\mathit{Maximum}(S)$: gibt ein Element mit dem größten Schlüssel zurück
  - $\mathit{Extract\_Max}(S)$: gibt ein Element mit dem größten Schlüssel zurück und entfernt dieses.

  Implementieren Sie eine Prioritätswarteschlange mit Hilfe der Heap-Datenstruktur.
- Eine Operation, die im allgemeinen *nicht* von Heaps (oder Prioritätswarteschlangen) verlangt wird, ist das *Verschmelzen* ($\mathit{Merge}(S_1, S_2)$) zweier Heaps in einen neuen. Implentieren Sie eine solche Prozedur. Was ist die *worst-case*-Komplexität Ihrer Lösung in Abhängigkeit von der Knotenanzahl?

Aufgabe 2: [Quicksort: Endrekursion und Iteration]

Rekursion ist ein mächtiges und elegantes Mittel zur Problemlösung. Rekursive Aufrufe führen in der Implentierung jedoch zu einem gewissen Overhead, der durch die Parameterübergabe und die Stack-Datenstruktur des Laufzeitsystems entsteht. Ein *iterativer* Algorithmus benutzt anstelle von rekursiven Aufrufen wie bei Divide & Conquer, *Schleifen* zum Durchlaufen einer Datenstruktur und kommt somit ohne den Overhead des Prozeduraufrufs auf.

- *Quicksort* als Beispiel eines Divide-and-Conquer Algorithmusses ist *rekursiv* definiert, für die linke Teilsequenz und danach für die rechte Teilsequenz. Der Rumpf von Quicksort ist mit dem zweiten rekursiven Aufruf für die rechte Teilsequenz *beendet.* Diese spezielle Form des rekursiven Aufrufs ist die sogenannte *Endrekursion* (*tail recursion*). Endrekursive Aufrufe lassen sich einfach iterativ, also mittels expliziter Schleifen lösen.[1]

  Ändern Sie den in der Vorlesung vorgestellten Quicksort-Algorithmus so, daß der zweite, endrekursive Aufruf vermieden wird. Erkläutern Sie informell, wie sich allgemein ein endrekursiver Algorithmus in eine iterative Form bringen läßt.

- Programmieren Sie eine rein iterative Variante des Quicksort-Algorithmus.

Aufgabe 3: [Natürliches Mischen]

Wir haben in der Vorlesung den Sortieralgorithmus durch Mischen (*merge sort*) besprochen. Es gibt eine Reihe von Verfeinerungen des Algorithmusses. Eine ist unter dem Namen *natürliches Mischen* bekannt.

Der Nachteil der banalen Mischsortierung besteht darin, daß aus *Teilsortierungen* der zu sortierenden Sequenz kein Nutzen gezogen wird. *Natürliches* Mischsortieren nutzt aus, daß das Mischen auch mit längeren Teilsequenzen starten kann, wenn sie bereits zu Beginn sortiert sind. Natürliche Mischsortierung beginnt das Mischen also mit unterschiedlich langen (maximalen) sortierten Teilsequenzen anstelle Teilsequenzen fester Länge (nämlich 1).

Programmieren Sie einen Algorithmus zum natürlichen Mischen.

[1] In der Tat sind Compiler in der Lage, Endrekursion zu erkennen und sie automatisch in eine Iteration aufzulösen. Wegen der Verwandschaft von Iteration und Prozeduren die ausschließlich Endrekursion verwenden, findet man manchmal auch endrekursive Algorithmen als iterativ bezeichnet.