Informatik II SS 2006

Übungsblatt 7

Abgabe: 20.6.06 - 23.6.06

Aufgabe 1 (Quicksort) Punkte: 2+3

Implementieren Sie die Routine *partition(A,links,rechts)* in Python.

a) Implementieren Sie eine Routine *select(A,k)*. Die Eingabe ist ein ungeordneter Datensatz A. Nach Abarbeitung der Routine erfüllt A folgende Bedingungen

- $A[k]$ enthält das k-kleinste Element von A
- $\forall i,\ 0 \leq i < k:\ A[i] \leq A[k]$ und $\forall j,\ k < j < len(A):\ A[j] \geq A[k]$

Der Datensatz A darf dazu nicht explizit sortiert werden.
Tip: Die Routine *partition(A,links,rechts)* könnte hilfreich sein.

b) Schreiben Sie ein Programm, welches einen best-case-Datensatz für Quicksort erzeugt, d.h. einen Datensatz mit N verschiedenen Elementen mit der Eigenschaft, daß jede Partitionierung zwei Teildatensätze erzeugt, die sich in der Anzahl der Einträge um maximal 1 unterscheiden. Gehen Sie von einer Quicksort-Version aus, in der als Pivot-Element einer Partition immer das Element mit dem höchsten Index gewählt wird.
Tip: Die Routine *select(A,k)* kann hilfreich sein.

Aufgabe 2 (Heaps als Priority-Queue) Punkte: 2+1+1+1.5

Aus Informatik I kennen sie Queues. Eine p-Queue ist eine Queue, in der beim Herausnehmen eines Elementes aus der Queue immer das Element mit höchstem Wert gewählt wird. Mithilfe eines Heaps lässt sich solch eine Datenstruktur effizient implementieren. Beim Einfügen eines Elementes in den Heap muss ein *Upheap()* durchgeführt werden. Beim Herausnehmen eines Elementes ein *Downheap()*. Holen Sie sich das Paket *heaps.tar.gz* von der Vorlesungshomepage. Ergängen Sie das File um

a) die Routine *Upheap()*, welche ein in den Heap eingefügtes Element an die richtige Position innerhalb des Heaps bringt.

b) die Routine *put(v)*, welche ein Element in die p-Queue einfügt.

c) die Routine *get()*, welche ein Element aus der p-Queue heraus nimmt.

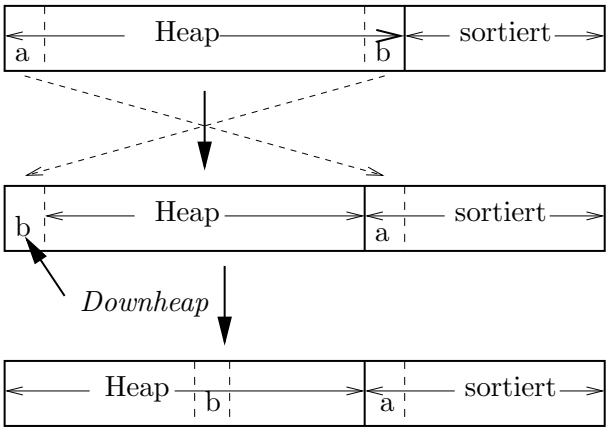
d) Geben Sie den Aufwand für die Routinen *put(v)* und *get()* an. Wie hoch wäre der Aufwand wenn man die p-Queue durch ein unsortiertes Array bzw. ein sortiertes Array realisiert. Verwenden Sie für Ihre Aufwandsangaben die $\mathcal{O}$-Notation.

Aufgabe 3 (Heapsort) Punkte: 2+2+2

Aus der Vorlesung kennen Sie *Heaps*. Mithilfe eines Heaps lässt sich ein effizientes Sortierverfahren realisieren. Die Idee ist, aus dem zu sortierenden Array einen Heap zu erstellen. Da die Blätter des Baumes bereits Heaps sind, muss die Heapeigenschaft nur für die Elemente $\lfloor n/2 \rfloor$ bis 1 mittels *Downheap* hergestellt werden.

Danach wird $n-1$ ($n := |Array|$) mal folgender Schritt wiederholt:
Nehme die Wurzel des Heaps und vertausche sie mit dem Element des Arrays mit dem grössten

Index, welches noch zum Heap gehört. Führe ein Downheap der Wurzel durch.



Dieses Sortierverfahren nennt man *Heapsort*

a) Implementieren Sie *Heapsort* in Python

Es gibt Beispiele, für die Heapsort deshalb nicht stabil ist, weil in der Routine *Downheap* (siehe *heaps.tar.gz*)

```
if v >= A[son]:
    break
```

anstatt

```
if v > A[son]:
    break
```

steht.

b) Geben Sie solch ein Beispiel an. Begründen Sie ausführlich wieso Ihr Beispiel mit der zweiten *Downheap*-Version stabil ist und mit der Ersten nicht.

c) Zeigen Sie, daß Heapsort auch unter Verwendung der unteren Anweisung nicht stabil ist.

Aufgabe 4 (Sortierverfahren) Punkte: 0.5+0.5+0.5

Sortieren Sie die Zahlenfolgen

a) 3,6,9,2,4,10,11,4,7 mit Heapsort aus 3a)

b) 8,9,6,0,5,4,3,1,7 mit Quicksort (median-of-three)

c) 5,4,9,7,1,3,0,8,4 mit Mergesort. Mergesort werden Sie in der nächsten Vorlesungsstunde kennen lernen.

Geben Sie die Zahlenfolgen nach jedem Downheap bzw. jedem Rekursionsschritt an.

Hinweis zu Übungsblatt 6: In Aufgabe 1b fehlt die Increment-Anweisung für die Variable i. Die korrigierte Version des Übungsblattes finden Sie auf der Vorlesungshomepage.