Praktikum zur Vorlesung
„Grundlagen der Informatik 1“
Studiengang E-Technik

AG Betriebssysteme, FB 3
Jan Bredereke

Datum: 7.2.2002

WS 2001/02

Vorbemerkung

Dieser Text kann auf dem Rechner lablin mit dem Kommando
`xdvi /home/jbredere/pub/aufgaben/praktikum6.dvi &`
angesehen werden. Bei Bedarf kann er gedruckt werden wie in der Beschreibung von LaTeX angegeben. Der Text ist auch im WWW verfügbar.

Sortieren eines Feldes mit Heapsort

Heapsort ist ein weiteres Sortierverfahren für Felder. Im folgenden wird die Funktionsweise kurz beschrieben, im Praktikum wird es zusätzlich an der Tafel demonstriert.

Wenn wir ein Feld A mit den Indizes 1 bis n haben, dann hat es die *Heap-Eigenschaft*, wenn gilt:

> Für alle $i \in \{2, 3, \ldots, n\}$ gilt: $A[i \operatorname{div} 2] \geq A[i]$.

Dabei ist „div“ die ganzzahlige Division. Für $n = 6$ muß also gelten: $A[1] \geq A[2]$, $A[1] \geq A[3]$, $A[2] \geq A[4]$, $A[2] \geq A[5]$, $A[3] \geq A[6]$.

Aus der Heap-Eigenschaft folgt, daß $A[1]$ das größte Element von allen ist. Der Rest des Feldes ist aber erst einmal noch nicht richtig sortiert.

Die Idee von Heapsort ist nun die folgende: Wir sorgen durch Anwendung eines geeigneten Verfahrens dafür, daß das Feld die Heap-Eigenschaft bekommt. Dann nehmen wir das erste Element $A[1]$ und vertauschen es mit dem letzten Element$A[n]$. Damit steht an der letzten Stelle n das größte Element auf dem richtigen Platz. Jetzt stellen wir durch eine geeignetes Verfahren die Heap-Eigenschaft für das restliche Feld von 1 bis $n - 1$ wieder her. Damit steht an erster Stelle das zweigrößte Element, und wir vertauschen es mit $A[n - 1]$. Dies setzen wir solange fort, bis der zu sortierende Rest des Feldes die Länge 1 hat und daher bereits trivialerweise richtig sortiert ist.

Wie stellen wir die Heap-Eigenschaft her? Dafür benötigen wir die Funktion $\text{sink}(i, t)$. Sie arbeitet auf dem Teilbereich des Feldes von 1 bis t. Sie läßt das i-te Element „absinken“ und stellt damit die Heap-Eigenschaft für die Position i her. Die Funktion $\text{sink}(i, t)$ untersucht die Elemente an den Positionen i, $2 * i$ und $2 * i + 1$. Falls nötig, tauscht sie das größte dieser Elemente an die Position i. Falls ein Tausch notwendig war, läßt sie außerdem das an die Position $2 * i$ bzw. $2 * i + 1$ getauschte Element noch weiter absinken, indem sie sich selbst rekursiv für diese Position aufruft. Bei allen diesen Operationen wird darauf geachtet, daß niemals auf eine Element jenseits der t-ten Position zugegriffen wird. Zum Beispiel muß $\text{sink}(2, 3)$ gar nichts tun, weil $2 * 2 > 3$ und $2 * 2 + 1 > 3$.

Nachdem das Heapsort-Verfahren wie oben beschrieben das größte Element von der ersten Position ans Feldende getauscht hat, kann man die Funktion $\text{sink}(1, n - 1)$ benutzen, um das „falsch“ an der ersten Position stehende Element richtig in den verbleibenden Heap einzuordnen.

Das erste Herstellen der Heap-Eigenschaft für das Gesamtfeld ist ein wenig aufwendiger: Hier kann jedes Element i in den Positionen zwischen 1 und $(n \operatorname{div} 2)$ fälschlicherweise kleiner als die Elemente an den Positionen $(i * 2)$ und $(i * 2 + 1)$ sein. Daher muß die Funktion $\text{sink}(i, n)$ für jedes dieser i aufgerufen werden. Damit sichergestellt ist, daß einer der späteren Aufrufe von sink dabei nicht das Ergebnis eines früheren Aufrufs zunichte macht, muß man dabei an der Position $i = n \operatorname{div} 2$ anfangen und rückwärts bis zur Position $i = 1$ gehen.

Heapsort hat die angenehme Eigenschaft, daß es im ungünstigsten Fall einen Zeitaufwand von

$\mathcal{O}(n \log n)$ hat, was besser ist als bei Quicksort mit $\mathcal{O}(n^2)$. Die praktische Erfahrung zeigt aber, daß Quicksort *im Mittel* schneller ist als Heapsort.

Aufgabe 1: Implementieren von Heapsort

Definiert ein Feld `A` von neun Integers mit den Werten 57, 16, 62, 30, 80, 7, 21, 78 und 41. Schreibt eine Funktion `heapsort()` und eine Hilfsfunktion `sink(int pos, int top)`, die dieses Feld in aufsteigender Reihenfolge sortieren. Laßt anschließend das sortierte Feld ausgeben.

Hinweis: Der obige Algorithmus geht davon aus, daß die Elemente des Feldes ab 1 durchnumeriert sind. In C hat das erste Element aber die Nummer 0. Am besten nehmt ihr ein Feld mit zehn Elementen und laßt einfach das Element 0 leer.