Johannes Blömer
u.v.a.m.

Paderborn, den 2.Juni 2004

Musterlösungen zur Vorlesung

Datenstrukturen und Algorithmen

SS 2004

Blatt 5

AUFGABE 1:

1. Enthält das Array A n identische Zahlen, so wird die Funktion PARTITION bei jedem Aufruf mit einem Teilarray $A[p..r]$ immer den Index $q = r$ zurückgeben. Das heisst aber auch, dass der Aufruf von Quicksort mit Teilarray $A[p..r]$ zu rekursiven Aufrufen von Quicksort mit dem Teilarray $A[p..r-1]$ und dem Teilarray $A[r]$ führt. Damit ist dann die Laufzeit $T(n)$ von Quicksort bei n identischen Zahlen gegeben durch

$$T(n) = T(n-1) + T(1) + cn,$$

wobei cn die Laufzeit für den Algorithmus PARTITION ist. Durch sukzessives Einsetzen der Gleichung in sich selbst mit $i = n-1, n-2, \dots, 2$ und unter der Annahmen, dass Quicksort für $n = 1$ Laufzeit $\leq c$ besitzt, erhalten wir

$$T(n) = nT(1) + \sum_{i=2}^{n} ci = cn + c\frac{n(n+1)}{2} - 1 = \mathcal{O}(n^2).$$

2. Sind die Eingabezahlen absteigend sortiert, so wird der Algorithmus PARTITION bei jedem Aufruf mit einem Teilarray $A[p..r]$ stets den Index $q = p$ zurückgeben. Jeder Aufruf von Quicksort mit Array $A[p..r]$ führt dann zu rekursiven Aufrufen von Quicksort mit den Arrays $A[p+1..r]$ und $A[p]$. Wie im ersten Teil erhalten wir dann für die Laufzeit $T(n)$ die Rekursionsgleichung

$$T(n) = T(n-1) + T(1) + cn.$$

Die Laufzeit ist damit auch in diesem Fall $\mathcal{O}(n^2)$.

AUFGABE 2:

Betrachten wir im Rekursionsbaum die Aufrufe der Tiefe $k, k \geq 0$. Auf Tiefe k erfolgen die rekursiven Aufrufe von Quicksort auf Mengen der Größe $\alpha^i(1-\alpha^j)n$ mit $i + j = k$. Aus $1/2 \leq \alpha < 1$ folgt nun $1 - \alpha \leq \alpha$. damit gilt für alle i, j mit $i + j = k$

$$(1-\alpha)^k n \leq \alpha^i (1-\alpha)^j n \leq \alpha^k n.$$

Damit gilt für $k \geq -\log_{1-\alpha}(n)$ und alle i, j mit $i + j = k$

$$1 \leq \alpha^i (1-\alpha)^j n.$$

Damit aber müssen Knoten mindestens Tiefe $k = -\log_{1-\alpha}(n)$ besitzen, um Blätter im Rekursionsbaum sein. Andererseits gilt für $k \leq -\log_\alpha(n)$ und alle i, j mit $i + j = k$

$$1 \geq \alpha^i (1-\alpha)^j n.$$

Damit aber existieren spätestens auf Tiefe $\log_\alpha(n)$ nur noch Blätter im Rekursionsbaum. Die Aussagen der Aufgabe folgen nun aus den Gleichungen

$$\log_{1-\alpha}(n) = \log(n)/\log(1-\alpha)$$

$$\log_\alpha(n) = \log(n)/\log(\alpha).$$

AUFGABE 3:

1. Durch die Initialisierung in Zeilen 2 und 3 sowie die Dekrementierung bzw. Inkrementierung in den Zeilen 5 und 7 sind $A[r]$ und $A[p]$ die ersten Arrayelemente auf die in Zeile 10 zugegriffen werden kann. Damit wird in Zeile 10 über den Index i nur auf Arrayelemente $A[i]$ mit $i \geq p$ und über den Index j nur auf Arrayelemente $A[j]$ mit $j \leq r$ zugegriffen. Nun stellt die Zeile 9 aber sicher, dass für Arrayelemente $A[j], A[i]$, auf die in Zeile 10 zugegriffen wird, stets $i < j$ gilt. Insgesamt wird damit in Zeile 10 nur auf Elemente im Teilarray $A[p..r]$ zugegriffen.

2. Wir müssen zeigen, dass der Index j in Zeile 5 mindestens zweimal dekrementiert wird. Einmal erfolgt dieses sicherlich beim ersten Durchlauf der while-Schleife und ersten Durchlauf der repeat-Schleife in Zeile 5. In der repeat-Schleife erfolgt sofort eine zweite Dekrementierung, es sei denn $A[r] \leq x = A[p]$. Ist daher $A[r] > x$ sind wir bereits mit dem Beweis fertig. Nehmen wir also an $A[r] \leq x$. Dann wird nach dem einmaligem Durchlauf der repeat-Schleife in Zeile 5 die repeat-Schleife in Zeile 7 durchlaufen. Da $A[p] = x$, wird die repeat-Schleife genau einmal für den Index $i = p$ durchlaufen. Da $p < r$ gilt, wird nun nach dem Vertauschen von $A[p]$ und $A[r]$ die repeat-Schleife in Zeile 5 noch einmal durchlaufen. Hier erfolgt dann die zweite Dekrementierung von j.

3. Um diese Aussage zu beweisen, betrachten wir folgende Invariante für die while-Schleife.

   Invariante: Alle Elemente in $A[p..i-1]$ sind höchstens so groß wie x und alle Elemente in $A[j+1..r]$ sind mindestens so groß wie x. Falls $i < j$, gilt ausserdem $A[i] \leq x$ und $A[j] \geq x$.

   Nun zeigen wir wie üblich Initialisierung, Erhaltung und Terminierung.

   Initialisierung Vor dem ersten Durchlauf der while-Schleife gilt $i = p-1, j = r+1$. Damit sind die Arrays $A[p..i]$ und $A[j..r]$ leer und die Invariante ist erfüllt.

   Erhaltung: Sei also die Invariante vor einem Durchlauf der while-Schleife erfüllt. D.h., wir wissen das mit dem aktuellen Wert von j und i die Invariante erfüllt ist. Sei dann j' der Index, mit dem die repeat-Schleife in Zeile 5 abbricht, und i' sei der Index mit dem die die repeat-Schleife in Zeile 7 abbricht. Nach den Abbruchbedingungen gilt dann für alle Elemente in $A[p..i'-1]$, dass sie höchstens so groß wie x sind, und für alle Elemente in $A[j'+1..r]$ gilt, dass sie mindestens so groß wie x sind. Um auch über $A[i']$ und $A[j']$ für den Fall $i' < j'$ zu sagen, beobachten wir, dass in Fall $i' < j'$ die Abbruchbedingungen in Zeilen 6 und 8 sowie die Vertauschung in Zeile 10 sicher stellen, dass $A[i'] \leq x$ und $A[j'] \geq x$.

   Terminierung: Es muss bei Terminierung für die Indizes i, j gelten, $i \geq j$. In diesem Fall aber besagt die Invariante, dass alle Elemente in $A[p..i-1]$ höchstens so groß wie x und alle Elemente in $A[j+1..r]$ mindestens so groß wie x sind. Da $i \geq j$, gilt $i-1 \geq j-1$. Wir müssen daher nur noch noch zeigen, dass auch der Eintrag $A[j]$ höchstens x ist. Da aber $i \geq j$ wurde im letzten Durchlauf der while-Schleife die Vertauschung in Zeile 10 nicht durchgeführt. Mit der Abbruchbedingung in Zeile 6 garantiert dieses $A[j] \leq x$ und der Algorithmus ist korrekt.