Technische Universität Chemnitz
Professur Theoretische Informatik
Prof. Dr. A. Goerdt, Dipl.-Inf. F. Schädlich

Wintersemester 2001/02

11. Übung „Theoretische Informatik I“
Lösungsvorschläge

Aufgabe 11.1:

Der *AVL*-Baum ist ein binärer Suchbaum. Durchläuft man diesen in `in-order` (linker Teilbaum, Knoten, rechter Teilbaum), können alle Elemente/Knoten des Baumes sortiert ausgegeben werden. Dieses Baumtraversiern kann in $O(n)$ implementiert werden. Wenn man den *AVL*-Baum nun in $O(n)$ aufbauen könnte, hat man ein auf Vergleichen basierendes Verfahren, welches in der Zeit $O(n)$ n verschiedene Elemente sortieren kann. Dies stellt jedoch einen Widerspruch zu dem Satz dar, daß solche Verfahren im Worst-Case eine Mindestlaufzeit von $\Omega(n \cdot \log n)$ haben.

Aufgabe 11.2:

Partition (A : Array, $links$**,** $rechts$**)**

// teilt das Feld A[$links \ldots rechts$] in
// - A1[$links \cdots pos - 1$]: Elemente < pivot
// - A2[$pos \cdots pos$]: Elemente = pivot
// - A3[$pos + 1 \cdots rechts$]: Elemente > pivot

$pivot$:= A[$links$]; // Pivot-Element auswählen

l := links;
r := rechts;

WHILE $l < r$ DO

    WHILE (A[l]$\leq$ $pivot$) AND ($l < rechts$) DO
        $l := l + 1$;

    WHILE (A[r]$>$ $pivot$) AND ($r > links$) DO
        $r := r - 1$

    IF $l < r$ THEN
        Vertausche (A[l], A[r]);

IF (A[l] $\leq$ $pivot$) THEN
    $pos := l$;
    ELSE
$pos := l - 1$;

A[$links$] := A[pos];
A[pos] := $pivot$;

RETURN pos;

QuickSort (A : Array, $links$**,** $rechts$**)**

```
IF (links < rechts) THEN

    pivot-pos := Partition(A : Array, links, rechts)
        // - Bestimmung des Pivot-Elementes pivot
        // - Aufteilung des Feldes A in Felder
        //      * A1 (alle Elemente kleiner als das Pivot-Element)
        //      * A2 (alle Elemente gleich dem Pivot-Element)
        //      * A3 (alle Elemente größer als das Pivot-Element)

    QuickSort(A, links, pivot-pos−1); // Sortieren von A1

    QuickSort(A, pivot-pos+1, rechts); // Sortieren von A3

RETURN;
```

Select (A : Array, $links$**,** $rechts$**,** i**)**

```
// sucht das i-t größte Element in A[links ... rechts]
// erster Aufruf: Select(A, 1, n, i);

IF (links = rechts) AND (i = 1) THEN
RETURN A[links];

IF (links ≥ rechts) THEN
RETURN;

IF (links < rechts) THEN

    pivot-pos := Partition(A : Array, links, rechts)
        // - Bestimmung des Pivot-Elementes pivot
        // - Aufteilung des Feldes A in Felder
        //      * A1 (alle Elemente kleiner als das Pivot-Element)
        //      * A2 (alle Elemente gleich dem Pivot-Element)
        //      * A3 (alle Elemente größer als das Pivot-Element)

    IF (i <pivot-pos) THEN // suche i-t größte in A1
        RETURN Select(A, links, pivot-pos−1, i);

    IF (i =pivot-pos) THEN // Pivot-Element ist das i-t größte
        RETURN A[pivot-pos];

    IF (i >pivot-pos) THEN // suche i-t größte in A3, Index anpassen
        RETURN Select(A, pivot-pos+1, rechts, i−pivot-pos);

RETURN;
```

Analog zu Quicksort tritt bei dieser Select-Variante der Worst-Case genau dann auf, wenn bei einer Eingabe von k Elementen wieder ein rekursiver Aufruf mit $k-1$ Elementen erfolgt. Dies ist z. B. dann der Fall, wenn die Daten im Feld A bereits aufsteigend sortiert sind und das Maximum gesucht wird. Damit liegt ein Laufzeitverhalten von $O(n^2)$ vor.

Aufgabe 11.3:

Bei *Quicksort* wird als Pivot-Element immer A[1] verwendet. Wäre das Pivot-Element immer der Median, so berechnet sich die Laufzeit $T(n)$ von *Quicksort* aus $T(1) = b$ bzw. $T(n) = an + T(\lceil \frac{n-1}{2} \rceil) + T(\lfloor \frac{n-1}{2} \rfloor) \leq an + 2 \cdot T(\frac{n}{2})$. Die Auflösung dieser Rekursion ergibt bekanntlich T$(n) = O(n \log n)$. Der Median kann mit der Funktion *Select* der Vorlesung in Linearzeit berechnet werden. Da die Umordnung der Elemente in *Quicksort* in jedem Schritt ebenfalls in Linearzeit erfolgt, ändert sich die obige Laufzeitkomplexität von *Quicksort* nicht, wenn der Median berechnet und als Pivot-Element verwendet wird.

Aufgabe 11.4:

Black-Box-Select (A, i) {

/* Eingabe: A[1..n], i, Ausgabe i-t größte Element */

m := Black-Box-Median (A); $O(n)$

Umorden der Elemente in A[1 .. n], so daß jedes Element in $O(n)$

- A_1=A[1 .. k] kleiner gleich dem Median m
- A_2=A[k+1 .. n] größer als der Median m

ist. Dies kann analog zu Quicksort erfolgen.

IF k=i RETURN m;

IF i<k RETURN Black-Box-Select (A[1..k-1], i);
ELSE RETURN Black-Box-Select (A[k+1..n), i-k);
}

Die Laufzeit $LZ(n)$ für ein Feld A[1..n] berechnet sich aus:

$$LZ_{\text{Black-Box-Median}}(n) + LZ_{\text{Umordnen}}(n) \leq an$$

$$\begin{aligned}
& LZ(1) \leq a \\
n > 1: \quad & LZ(n) \leq LZ_{\text{Black-Box-Median}}(n) + LZ_{\text{Umordnen}}(n) + LZ(n/2) \\
& LZ(n) \leq an + \frac{an}{2} + \frac{an}{4} + \frac{an}{8} + \frac{an}{16} + ... \\
& LZ(n) \leq an \sum_{i=0}^{\log_2 n} \frac{1}{2^i} \\
& LZ(n) \leq an \sum_{i=0}^{\infty} \frac{1}{2^i} \\
& LZ(n) \leq 2an \\
& LZ(n) = O(n) \ .
\end{aligned}$$

Aufgabe 11.5:

Die Aufgabe wird mit Hilfe der binären Suche gelöst. Bei $2n$ Elementen haben wir also eine Laufzeitkomplexität von $O(\log n)$. Es muß nun ein Test entwickelt werden, der in $O(1)$ entscheidet, ob ein X[i] (bzw. ein Y[i]) der Median ist. Hier hilft die Überlegung weiter, daß genau $(n-1)$ Elemente größer als der Median bzw. n Elemente kleiner als der Median sein müssen. Ist X[i] der Median, so gilt:

$$k = n - i + 1$$
$$X[1] \leq .. \leq X[i-1] \leq X[i] \leq X[i+1] \leq .. \leq X[n]$$
$$Y[1] \leq .. \leq Y[k] \leq \underline{X[i] \leq Y[k+1]} \leq .. \leq Y[n]$$

Ist jedoch $Y[k]$ der Median, so gilt:

$$k = n - i + 1$$
$$X[1] \leq .. \leq X[i-1] \leq X[i] \leq \underline{Y[k] \leq X[i+1]} \leq .. \leq X[n]$$
$$Y[1] \leq .. \leq Y[k-1] \leq Y[k] \leq Y[k+1] \leq .. \leq Y[n]$$

Für den Median m gilt natürlich $m \geq X[i]$ und $m \geq Y[k]$ (d. h. der Median m ist das Maximum von $X[i]$ und $Y[k]$, wenn i korrekt gewählt wurde.) Da die Felder schon sortiert sind, genügt es die entsprechende unterstrichene Ungleichung zu testen, ob X[i] bzw. Y[k] der Median ist. Hieraus ergibt sich folgender Algorithmus:

```
Median(X[1..n], Y[1..n]) {

l := 1; /* linker Rand */
r := n; /* rechter Rand */

WHILE l≠r do

    gefunden := true;

    i := ⌈(r+l)/2⌉;
    k := n-i+1;
    m := Max(X[i], Y[k]);

    IF i<n THEN
        IF m > X[i+1] THEN
            /* kann nur für m=Y[k] wahr werden */
            /* Y[k] ist nicht der Median, linken Rand anpassen */
            l :=i;
            gefunden := false;

    IF k<n THEN
        IF m > Y[k+1] THEN
            /* kann nur für m=X[i] wahr werden */
            /* X[i] ist nicht der Median, rechten Rand anpassen */
            r := i;
            gefunden := false;

    IF gefunden THEN /* WHILE-Schleife beenden */
        l := r := i;

RETURN m;
}
```

Aufgabe 11.6:

Es wird folgende *Select*-Variante mit frei wählbarer Gruppengröße k betrachtet:

Select(A, i**,** k**);**
/*
Eingabe:
A=A[1..n]
$1 \leq i \leq n$
k *.. Größe der Gruppen*

Ausgabe:
Das i-t kleinste Element aus A
*/

1. Unterteile A in Gruppen der Größe k.

2. Bestimme in jeder der $\lceil \frac{n}{k} \rceil$ k-Gruppen den Median. Hat die Gruppe eine gerade Anzahl von Elementen, so wird das größere der beiden mittleren Elemente verwendet.

3. IF $n > k$

4. Die $\lceil \frac{n}{k} \rceil$ Mediane in B=B[1..$\lceil \frac{n}{k} \rceil$] übertragen

5. x := Select $\left(\text{B}, \lceil \frac{n}{k} \rceil, \left\lceil \frac{\lceil \frac{n}{k} \rceil}{2} \right\rceil\right)$;

6. ELSE x := Median von A; /* 2. Zeile */

7. A umordnen (ähnlich Quicksort), so daß A=A', x, A" mit:
A'=A[1..$q-1$], $\forall i < q$: A'[i]$\leq x$
A[q]=x;
A"=A[$q+1$..n], $\forall i > q$: A"[i]$\geq x$

8. IF $(i = q)$ THEN RETURN x;

9. ELSE

10. IF $(i < q)$

11. THEN RETURN Select (A[1..($q-1$)], i, k);

12. ELSE RETURN Select (A[($q+1$)..n], $i - q$, k);

Analyse der Laufzeit T(n) = T(n, k):

1. $\Theta(n)$
2. $\Theta(n)$
3. $\Theta(1)$
4. $\Theta(n)$

5. T($\lceil \frac{n}{k} \rceil$)

6. $\Theta(1)$
7. $O(n)$
8. $\Theta(1)$
9. $\Theta(1)$

10. $\Theta(1)$
11. T($q-1$) (A[q]=Select(A,i,k))
12. T($n-q+1$) (A[q]=Select(A,i,k)).

Es können nun Konstanten a und b (in Abhängigkeit von der gewählten Konstante k) so gefunden werden, daß für die Laufzeit ohne die rekursiven Aufrufe gilt:

$$a \cdot n \leq \mathrm{T}(\text{Zeilen 1-4 und 6-10}) \leq b \cdot n \ .$$

Für die weiteren Betrachtungen ist zu beachten, daß die Worst-Case-Laufzeit auf Grund des Algorithmus monoton wachsend ist, d. h. je mehr Elemente im Auswahlproblem bearbeitet werden, um so größer muß die Laufzeit sein. Bevor nun die endgültige rekursive Definition von T(n) abgeleitet werden kann, muß noch die Laufzeit in den Zeilen 11 und 12 analysiert werden:

Es gibt mindestens n/k viele Gruppen bzw. Mediane, davon umfassen genau $\lfloor n/k \rfloor$ viele Gruppen k Elemente. D. h. in mindestens $1/2 \cdot \lfloor n/k \rfloor \approx n/(2k)$ Gruppen ist der Median kleiner gleich (oder größer gleich) x, dem Median der Mediane. In jeder Gruppe sind mindestens $k/2$ viele Elemente kleiner gleich (bzw. größer gleich) dem jeweiligen Median. Somit sind mindestens $(1/2 \cdot \lfloor n/k \rfloor) \cdot (k/2) \approx n/4$ viele Elemente kleiner gleich bzw. größer gleich x. Es ist $(1/2 \cdot \lfloor n/k \rfloor) \cdot (k/2) \geq (n-k+1)/4$ Im schlechtesten Fall müssen also bei der erneuten Rekursion in Zeile 11 oder 12 maximal $n - (n-k+1)/4 = 3/4n + (k-1)/4 \approx 3/4n$ viele Elemente untersucht werden. Damit ergibt sich die Rekursionsungleichung zu

$$T(n) \leq bn + T((n+k-1)/k) + T(3/4n + (k-1)/4) \ .$$

Damit $T(n) = O(n)$ gilt, also die Laufzeit linear ist, müssen die Konstanten $C > 0$ und n_0 existieren, so daß für alle $n > n_0$ gilt:

$$\begin{aligned} T((n+k-1)/k) &\leq & C \cdot (n+k-1)/k & \\ T((3/4n + (k-1)/4)) &\leq & C \cdot (3/4n + (k-1)/4) & \\ T(n) &\leq & bn + C \cdot (n+k-1)/k + C \cdot (3/4n + (k-1)/4) \leq & \quad Cn \ . \end{aligned}$$

Dies ist offensichtlich für $(n+k-1)/k + 3/4n + (k-1)/4 < n$ bzw. $1/k + (k-1)/(kn) + 3/4 + (k-1)/4n < 1$ möglich. Nun gilt $(k-1)/(kn) \to 0$ und $(k-1)/4n \to 0$. Für hinreichend große n muß also nur $1/k + 3/4 < 1$ und damit $k > 4$ erfüllt sein. Damit hat der Algorithmus für 7er-Gruppen eine lineare Laufzeit und für 3er-Gruppen ist zu vermuten, daß er schlechter ist. Dies wird im nächsten Schritt gezeigt (es wird wieder der Worst-Case betrachtet):

Die Laufzeit in Zeile 5 beträgt mindestens $T(n/3)$. Die obige Analyse zeigt, daß für eine Laufzeit im rekursiven Aufruf der Zeilen 11 bzw. 12 von mindestens $T(2/3n)$ eine nichtlineare Laufzeit erwartet werden kann. Das Ziel ist es also jetzt zu zeigen, daß eine solche Eingabe konstruiert werden kann. Dies ist offensichtlich genau dann der Fall, wenn das eine Teilfeld aus mindestens 2/3n Elementen besteht. Prinzipiell sollte dies möglich sein, da die vorangegangenen Betrachtungen gezeigt haben, daß ein Teilfeld etwa zwischen $\approx n/4$ und $\approx 3/4n$ vielen Elementen umfassen. Als Median der Mediane m soll nun das Element ausgewählt werden, welches das Feld etwa im Verhältnis 1:2 teilt. Dies ist das $\lceil n/3 \rceil$-größte. Dieses kommt in eine 3er-Gruppe, dazu ein jeweils ein kleineres und ein größeres. Nun werden etwa $1/2 \cdot n/3 = n/6$ Mediane benötigt, die kleiner als m sind. Diese kommen mit einem kleineren und einem größeren Element in eine Gruppe. Die $n/3$ vielen Elemente, die kleiner als m sind, reichen dafür (inklusive der Gruppe, die m enthält). Die restlichen Gruppen werden mit den größeren Elementen gefüllt, so daß hier der Median größer als m ist. Die Aufteilung des Feldes erfolgt nun im Verhältnis 1:2. Gegebenenfalls sind Gruppen untereinander auszutauschen bzw. Elemente zwischen den Gruppen, die die Medianbestimmung in den Gruppen nicht beeinflussen, so daß in der weiteren Rekursion immer wieder ein Teilungsverhältnis von 1:2 nach obigem Schema erreicht wird. Es ergibt nun folgende Rekorsionsungleichung:

$$T(n) \geq an + \underbrace{\underbrace{T\left(\frac{n}{3}\right)}_{\geq \frac{1}{3}an + T\left(\frac{1}{9}\right) + T\left(\frac{2}{9}\right)} + \underbrace{T\left(\frac{2}{3}n\right)}_{\geq \frac{2}{3}an + T\left(\frac{2}{9}n\right) + T\left(\frac{4}{9}n\right)}}_{\geq an + T(..) + T(..) + T(..) + T(..)} .$$

Die Bestimmung des Medians der Mediane hat eine Rekursionstiefe von $\log_3 n$. Die Rekursion von Select selbst erfolgt im Worst-Case mindestens $\log_{\frac{3}{2}} n$ mal. Somit ergibt sich eine minimale Rekursionstiefe von $\log_3 n$ im Worst-Case. In jeder Rekursionsebene tritt jedoch eine Laufzeit $\geq an$ auf:

$$\begin{aligned} T(n) &\geq \log_3 n \cdot an \\ T(n) &= \Omega(n \log n) \ . \end{aligned}$$

Die Laufzeit ist also bei der Verwendung von 3er-Gruppen nicht mehr linear.