



# Musterlösung zur Probeklausur Datenstrukturen und Algorithmen

12. August 2004

Nicola Ueffing, Evgeny Matusov, Thomas Deselaers

## Aufgabe 1: Komplexität und Wachstumsklassen

### Punkte

a) *Punktverteilung: 1 Punkte für Transitivität, 2 Punkte für die beiden anderen Behauptungen*

**Transitivität:** Sei  $f \in \Theta(g)$ . Zeige, dass dann auch gilt:  $f \in \Theta(h)$ . Dann gilt, dass  $\Theta(g) \subseteq \Theta(h)$ . Es gilt

$$f \in \Theta(g) \Rightarrow \exists c_1 > 0, \exists n_1 > 0 : \frac{1}{c_1} \cdot g(n) \leq f(n) \leq c_1 \cdot g(n) \quad \forall n \geq n_1$$

$$\text{und } g \in \Theta(h) \Rightarrow \exists c_2 > 0, \exists n_2 > 0 : \frac{1}{c_2} \cdot h(n) \leq g(n) \leq c_2 \cdot h(n) \quad \forall n \geq n_2$$

Damit gilt für  $n_0 := \max\{n_1, n_2\}$  und  $c := c_1 \cdot c_2$ ,

$$\frac{1}{c_0} \cdot h(n) \leq f(n) \leq c_0 \cdot h(n) \quad \forall n \geq n_0, \text{ also } f \in \Theta(h)$$

### Symmetrie:

$$f \in \Theta(g) \Leftrightarrow \exists c > 0, \exists n_0 > 0, \forall n \geq n_0 : \frac{1}{c} g(n) \leq f(n) \leq c \cdot g(n)$$

$$\Leftrightarrow \exists c > 0, \exists n_0 > 0, \forall n \geq n_0 : \frac{1}{c} g(n) \leq f(n) \wedge f(n) \leq c \cdot g(n)$$

$$\Leftrightarrow \exists c > 0, \exists n_0 > 0, \forall n \geq n_0 : g(n) \leq c \cdot f(n) \wedge \frac{1}{c} f(n) \leq g(n)$$

$$\Leftrightarrow \exists c > 0, \exists n_0 > 0, \forall n \geq n_0 : \frac{1}{c} f(n) \leq g(n) \leq c \cdot f(n)$$

$$\Leftrightarrow g \in \Theta(f)$$

**Umkehrsymmetrie:**

$$\begin{aligned}
f \in O(g) &\Leftrightarrow \exists c > 0, \exists n_0 > 0, \forall n \geq n_0 : 0 \leq f(n) \leq c \cdot g(n) \\
&\Leftrightarrow \exists c > 0, \exists n_0 > 0, \forall n \geq n_0 : 0 \leq \frac{1}{c} f(n) \leq g(n) \\
&\Leftrightarrow \exists c' > 0, \exists n_0 > 0, \forall n \geq n_0 : 0 \leq c' \cdot f(n) \leq g(n) \\
&\Leftrightarrow g \in \Omega(f)
\end{aligned}$$

b) *Punktverteilung: 2 Punkte pro Richtung*

Beweise  $\Theta(f) = \Omega(f) \cap O(f)$

“ $\Rightarrow$ ”: Sei  $g \in \Theta(f)$ , d.h.

$$\exists c > 0, \exists n_0 > 0 : \frac{1}{c} \cdot f(n) \leq g(n) \leq c \cdot f(n) \quad \forall n \geq n_0$$

Wähle  $c_1 := \frac{1}{c}, n_1 := n_0$  und  $c_2 = c, n_2 := n_0$ , dann gilt

$$\begin{aligned}
c_1 \cdot f(n) \leq g(n) \quad \forall n \geq n_1 &\Rightarrow g \in \Omega(f) \\
g(n) \leq c_2 \cdot f(n) \quad \forall n \geq n_2 &\Rightarrow g \in O(f)
\end{aligned}$$

“ $\Leftarrow$ ”: Sei  $g \in \Omega(f) \cap O(f)$ , d.h.

$$\begin{aligned}
\exists c_1 > 0, \exists n_1 > 0 : c_1 \cdot f(n) \leq g(n) \quad \forall n \geq n_1 \\
\exists c_2 > 0, \exists n_2 > 0 : g(n) \leq c_2 \cdot f(n) \quad \forall n \geq n_2
\end{aligned}$$

Wähle  $n_0 := \max\{n_1, n_2\}$  und  $c := \frac{c_2}{c_1} \max\{c_1, 1/c_2\}$ , dann gilt

$$\begin{aligned}
\frac{1}{c} \cdot f(n) &= \frac{c_1}{c_2 \max\{c_1, 1/c_2\}} \cdot f(n) \leq c_1 \cdot f(n) \\
&\leq g(n) \leq c_2 \cdot f(n) \leq \frac{c_2}{c_1} \max\{c_1, 1/c_2\} \cdot f(n) = c \cdot f(n) \quad \forall n \geq n_0,
\end{aligned}$$

d.h.  $g \in \Theta(f)$

**Aufgabe 2: Rekursionsgleichungen****Punkte**

a) 6 Punkte

$$T(n) = 3T(n-2) + 2, \quad T(1) = T(2) = 2$$

$$\begin{aligned}
 T(n) &= 3T(n-2) + 2 \\
 &= 3(3T(n-4) + 2) + 2 \\
 &= 3^3T(n-6) + 3^2 \cdot 2 + 3 \cdot 2 + 2 \\
 &= \dots \\
 &= 3^{\lceil \frac{n}{2} \rceil - 1} \cdot 2 + \sum_{i=0}^{\lceil \frac{n}{2} \rceil - 2} 3^i \cdot 2 \\
 &= 2 \cdot \sum_{i=0}^{\lceil \frac{n}{2} \rceil - 1} 3^i \\
 &= 2 \cdot \frac{1 - 3^{\lceil \frac{n}{2} \rceil - 1}}{1 - 3} \\
 &= 3^{\lceil \frac{n}{2} \rceil - 1} - 1 \\
 &\Rightarrow T(n) \in \Theta(3^{\frac{n}{2}})
 \end{aligned}$$

b) 4 Punkte

$$T(n) = 7T\left(\frac{n}{3}\right) + n^2, \quad T(1) = 1$$

Hier kann das Mastertheorem angewendet werden, und zwar der 3. Fall. Mit  $a = 7$ ,  $b = 3$  und  $f(n) = n^2$  ist

$$n^{\log_b a} = n^{\log_3 7} \quad \text{und} \quad \log_3 7 < 2, \quad \text{also} \quad f(n) \in \Omega(n^{\log_b a + \epsilon})$$

mit  $\epsilon = 2 - \log_3 7$ . Überprüfe noch  $af(n/b) \leq cf(n)$ :

$$7f\left(\frac{n}{3}\right) = \frac{7}{9}n^2 \leq \frac{7}{9}n^2$$

also ist die Bedingung für  $c = \frac{7}{9}$  erfüllt. Damit ergibt sich

$$T(n) \in \Theta(f(n)) = \Theta(n^2)$$

**Aufgabe 3: MergeSort****Punkte***Punkteverteilung:*

$$\frac{2 \cdot \text{richtig} - \text{falsch}}{2}$$

void mergesort(int [] a,int l, int r) {	-
if(l<r) {	B
int m=(r+l)/2;	A
mergesort(a,l,m);	B
mergesort(a,m+1,r);	C
merge(a,l,m,r);	A
}	C
}	-

void merge(int[] a,int l, int m, int r) {	-
int i,j;	B
for(i=m+1;i>l;--i) {aux[i-1]=a[i-1];}	A
for(j=m;j<r;++j) {aux[r+m-j]=a[j+1];}	C
for(int k=l;k<=r;++k) {	C
if(aux[j] < aux[i]) {	B
a[k]=aux[j];	A
--j;	A
} else {	C
a[k]=aux[i];	B
++i;	B
}}	-

**Aufgabe 4: Sortieren geht über Studieren**

**5+5 Punkte**

Sortieren Sie die Folge

85, 1, 7, 2, 83, 19, 14, 12

mit

- a) MergeSort. Geben Sie dabei nach jedem Merge die Folge und die Werte für  $l$  und  $r$  in der Tabelle an. *5 Punkte*

Wenn Sie Felder freilassen, heißt dies, dass die Zahl aus der Zeile darüber nicht geändert wird.

Position		0	1	2	3	4	5	6	7
l	r	85	1	7	2	83	19	14	12
0	1	1	85						
2	3			2	7				
0	3	1	2	7	85				
4	5					19	83		
6	7							12	14
4	7					12	14	19	83
0	7	1	2	7	12	14	19	83	85

- b) SelectionSort. Geben Sie nach jeder Vertauschung von Elementen die Folge an. Wenn Sie dabei Felder freilassen, heißt dies, dass ein Feld unverändert bleibt. *5 Punkte*

Pos	0	1	2	3	4	5	6	7
i	85	1	7	2	83	19	14	12
0	1	85						
1		2		85				
2								
3				12				85
4					14		83	
5								
6	1	2	7	12	14	19	83	85

**Aufgabe 5: Optimaler Suchbaum****10 Punkte**

*Punkteverteilung: 2 Punkte für korrekte Rekursionsgleichung, 2 für korrekte  $w_{ij}$ , 4 Punkte für korrekte Rechnung, 2 Punkte für das Zurückverfolgen des Pfades (Baumkonstruktion).*

$i$	0	1	2	3	4
$\beta_i$		3	1	7	11
$\alpha_i$	0	2	3	5	4

Zugriffskosten  $w_{ij}$  für das Intervall zwischen  $i$  und  $j$ :

$$w_{ij} = \sum_{k=i}^j \alpha_k + \sum_{k=i+1}^j \beta_k$$

Rekursionsgleichung:

$$c_{ij} = w_{ij} + \min_{i < k \leq j} (c_{i,k-1} + c_{k,j})$$

Somit folgt aus der Tabelle, dass die gewichte wie folgt repräsentiert werden können:

$w_{ij}$	0	1	2	3	4
0	0	5	9	21	36
1		2	6	18	33
2			3	15	30
3				5	20
4					4

Danach werden  $c_{ij}$  berechnet, jeweils über eine Diagonale, beginnend mit der Hauptdiagonale. An der Hauptdiagonale werden Nullen eingetragen -  $c_{ii} = 0 \forall i$ .

Für die 2-te Diagonale gibt es nur ein mögliches  $k = i$ :

$$c_{01} = w_{01} + c_{00} + c_{11} = 5 + 0 + 0 = 5 \quad (k = 1)$$

$$c_{12} = w_{12} + 0 + 0 = 6 \quad (k = 2)$$

$$c_{23} = w_{23} + 0 + 0 = 15 \quad (k = 3)$$

$$c_{34} = w_{34} + 0 + 0 = 20 \quad (k = 4)$$

Für die 3-te Diagonale ergibt sich:

$$c_{02} = w_{02} + \min(c_{00} + c_{12}, c_{01} + c_{22}) = 9 + \min(0 + 6, 5 + 0) = 9 + 5 = 14 \quad (k = 2)$$

$$c_{13} = w_{13} + \min(c_{11} + c_{23}, c_{12} + c_{33}) = 18 + \min(0 + 15, 6 + 0) = 18 + 6 = 24 \quad (k = 3)$$

$$c_{24} = w_{24} + \min(c_{22} + c_{34}, c_{23} + c_{44}) = 30 + \min(0 + 20, 15 + 0) = 30 + 15 = 45 \quad (k = 4)$$

Für die 4-te Diagonale ergibt sich:

$$c_{03} = w_{03} + \min(c_{00} + c_{13}, c_{01} + c_{23}, c_{02} + c_{33}) = 21 + \min(0 + 24, 5 + 15, 14 + 0) = 21 + 14 = 35 \quad (k = 3)$$

$$c_{14} = w_{14} + \min(c_{11} + c_{24}, c_{12} + c_{34}, c_{13} + c_{44}) = 33 + \min(0 + 45, 6 + 20, 24 + 0) = 33 + 24 = 57 \quad (k = 4)$$

Für das globale Optimum (5-te Diagonale) ergibt sich:

$$c_{04} = w_{04} + \min(c_{00} + c_{14}, c_{01} + c_{24}, c_{02} + c_{34}, c_{03} + c_{44}) = 36 + \min(0 + 57, 5 + 45, 14 + 20, 35 + 0) = 36 + 34 = 70 \quad (k = 3)$$

Somit ergibt sich für die Zugriffskosten  $c_{ij}$  und optimale  $k$  (auch  $r_{ij}$  genannt):

$c_{ij}$	0	1	2	3	4
0	0	5	14	35	70
1		0	6	24	57
2			0	15	45
3				0	20
4					0

$r_{ij}$	0	1	2	3	4
0	-	1	2	3	3
1		-	2	3	4
2			-	3	4
3				-	4
4					-

Natürlich können die Matrizen transponiert dargestellt werden. Dazu siehe folgende Tabellen:

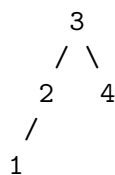
$w_{ij}$	0	1	2	3	4
0	0				
1	5	2			
2	9	6	3		
3	21	18	15	5	
4	36	33	30	20	4

$c_{ij}$	0	1	2	3	4
0	0				
1	5	0			
2	14	6	0		
3	35	24	15	0	
4	70	57	45	20	0

$r_{ij}$	0	1	2	3	4
0	-				
1	1	-			
2	2	2	-		
3	3	3	3	-	
4	3	4	4	4	-

Der Optimaler Suchbaum ist folgendermaßen aufzubauen:

$r_{04}$  gibt uns den Wurzelknoten an - 3. Also betrachte  $r_{02}$  und  $r_{34}$ , das werden jeweils der linke und der rechte Sohn sein. In  $r_{34}$  steht 4, also ist 4 der rechter Sohn. In  $r_{02}$  steht 2, also ist 2 der linke Sohn. Dann muss 1 der linke Sohn von 2 sein, sonst kann der Baum nicht aufgebaut werden. Somit ergibt sich der optimaler Suchbaum:



**Aufgabe 6: Hashing**

**Punkte**

- a) *Punkteverteilung: 1 Punkt pro korrekt beschriebenem Hashingansatz.*  
 Offenes Hashing: Beliebig erweiterbare Liste von Schlüsseln in jedem Tabellenplatz  
 (zum Auffinden eines Schlüssels muss die Liste der gespeicherten Einträge durchsucht werden)  
 Geschlossenes Hashing: Statisches Array mit einem Schlüssel pro Tabellenplatz  
 Kollisionen können auftreten  
 (Kollisionsbehandlung: Sondieren)
- b) *Punkteverteilung: 0.5 Punkte pro korrekt berechnetem Tabellenplatz, 0.5 Punkte für die korrekte Anzahl an Sondierungsschritten*

Sondierungs- schritt	Schlüssel $k$															
	1	20	35	17	9	18	25	41	50	22	21	15	49	8	28	
0	1	3	1	0	9	1	8	7	16	5	4	15	15	8	11	
1			2			2				6			16	9		
2						0							14	7		
3						5								12		

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Schlüssel $k$	17	1	35	20	21	18	22	41	25	9		28	8		49	15	50

Anzahl Sondierungsschritte: 10

**Aufgabe 7: Effiziente Suche**

**8+2 Punkte**

*Punktvergabe: Interpolationssuche mit Formel: 8 Punkte; Interpolationssuche ohne Formel: 6 Punkte; Binäre Suche: 4 Punkte; + 2 Punkte für die korrekte obere Schranke zu dem angegebenen Algorithmus.*

In dieser Aufgabe geht es um die Suche in einem sortierten Array. Da es zu erwarten ist, dass die Buchpreise gleichmäßig über das Intervall von 0 bis 1000 Euro verteilt sind, stellt sich die Interpolationssuche als besonders geeignet vor.

Interpolationssuche:  $O(\log(\log n))$

Binäre Suche:  $O(\log(n))$

Die Basis des Logarithmus spielt keine Rolle.

Suche:

```
double [] A; // Array zur Speicherung der Buecherpreise

void Search(x)      // x wird gesucht
{
    int low=1; high=A.length-1;
    int next = breakPoint(low, high);
    while((x != A[next]) && (low < high))
    {
        if(x < A[next])
            high = next - 1;
        else
            low = next + 1;
        next = breakPoint(low, high);
    }
    if(x==A[next]) System.out.println("x wurde an Position " + next + " gefunden.");
    else
        System.out.println("x wurde nicht gefunden !");
}
// Funktion breakPoint:

// Interpolationssuche:
int breakPoint(int low, int high)
{
    return
    low - 1 + obereGausklammer((high-low+1)* ((x - A[low - 1]) / (A[high+1] - A[low - 1])));

    // wichtig ist hier dass der Inhalt selbst bestimmt, wie gross der Intervall sein soll.
    // genau +-1 ist nicht so wichtig.
}

// Binaere Suche:
int breakPoint(int low, int high)
{
    return obereGausklammer( (high + low)/2 ); // obere Gaussklammer ist nicht wichtig
}
```

Zur Beschreibung des Algorithmus: Stichwörter:

Interpolationssuche: Annahme, dass Zahlen linear oder fast linear wachsen; Bestimme die "Wachstumsrate", betrachte dazu den Wertunterschied zwischen Positionen *low* and *high* in *A* und setze *next* auf die Position, wo *x* sein könnte bei linearem Wachstum der Werten in *A* zwischen *low* und *high*.

Binäre Suche: jeweils halbieren, was zu  $O(\log n)$  Schritten führt.

**Aufgabe 8: Breitendurchlauf**

**8 Punkte**

*Punkteverteilung: 1 Punkt pro korrekter Antwort, für eine falsche wird ein Punkt abgezogen. Enthaltungen sind neutral. Die Summe der Punkte dieser Aufgabe kann nicht negativ werden.*

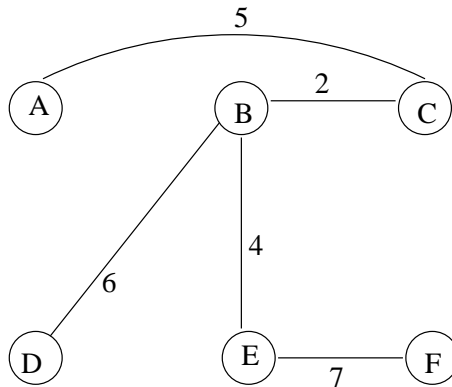
Breitendurchlauf von Graphen...	wahr	falsch
... wird mittels eines Stacks realisiert.		×
... wird mittels einer Queue realisiert.	×	
... läuft genau so ab wie ein Tiefendurchlauf.		×
... läßt sich auf Bäume anwenden.	×	
... erfordert das Markieren der schon besuchten Knoten.	×	
... besucht zunächst alle Nachfolger eines Knotens, dann deren Nachfolger usw.	×	
... ist für Graphen mit Zyklen nicht durchführbar.		×
... kann nicht iterativ gelöst werden.		×

**Aufgabe 9: Minimaler Spannbaum**

**6 Punkte**

*Punkteverteilung: 4 Punkte für korrekte Rechnung (Ausführung des Primalgorithmus), 2 Punkte für den korrekten Spannbaum.*

Der Spannbaum sieht wie folgt aus:



Dazu: falls Startknoten A:

Schritt		
1	A	$AB(6), \mathbf{AC(5)}, AE(9), AD(10)$
2	A,C	$AB(6), AE(9), AD(10), \mathbf{CB(2)}, CF(8)$
3	A,C,B	$AE(9), AD(10), CF(8), BD(6), \mathbf{BE(4)}, BF(11)$
4	A,C,B,E	$AD(10), CF(8), \mathbf{BD(6)}, BF(11), ED(8), EF(7)$
5	A,C,B,E,D	$CF(8), BF(11), \mathbf{EF(7)}$
6	A,C,B,E,D,F	

**Aufgabe 10: Editierabstand**

**8 Punkte**

*Punkteverteilung: 2 Punkte für die korrekt angegebene Rekursionsgleichung, 3 Punkte für das korrekte Rechnen (dynamische Programmierung), 2 Punkte für die korrekt angegebene Editieroperationen + 1 Punkt, falls mehrere beste Pfade zumindest erwähnt worden sind.*

Minimaler Editierabstand ist die minimaler Anzahl von Einfügungen, Löschungen und Ersetzungen, die nötig sind, um eine der Sequenzen in die andere umzuformen (“Editieroperationen”).

Vergleiche die Sequenzen  $s_1, \dots, s_j, \dots, s_J$  und  $t_1, \dots, t_i, \dots, t_I$ . Dann ist die minimaler Anzahl von Editieroperationen  $c[j, i]$ , die benötigt werden, um die Teilsequenz  $s_1, \dots, s_j$  in  $t_1, \dots, t_i$  umzuformen, gegeben durch die folgende **Rekursionsgleichung**:

$$c[j, i] = \min \{c[j - 1, i] + 1, c[j, i - 1] + 1, c[j - 1, i - 1] + 1 - \delta(s_j, t_i)\}$$

Initialisierung:  $c[0, 0] := 0, \quad c[0, i] = i, \quad c[j, 0] = j$ .

Dazu: fülle der Reihe nach die folgende Tabelle auf unter Verwendung der obigen Rekursionsgleichung:

$i \setminus j$	0	A	D	R	E	S	S	E
0	0	→ 1	→ 2	3	4	5	6	7
R	1	↘ 1	2	↘ 2	3	4	5	6
E	2	2	↘ 2	3	↘ 2	3	4	5
G	3	3	3	↘ 3	↓ 3	↘ 3	4	5
I	4	4	4	4	↘↓ 4	↘↓ 4	4	5
S	5	5	5	5	5	↘ 4	↘ 4	5
T	6	6	6	6	6	5	↘↓ 5	5
E	7	7	7	7	6	6	6	↘ 5
R	8	8	8	7	7	7	7	↓ 6

Der minimaler Abstand ist gegeben durch  $c[J, I] = c[E, R] = 6$ .

Die mögliche optimalen Pfade sind mit Pfeilen gekennzeichnet. Es gibt insgesamt 4 Pfade, einige davon bevorzugen Löschungen und Einfügungen, andere bevorzugen Ersetzungen. Die Pfade kann man von der Tabelle ablesen.

- a) Pfad: A+, D+, G → S, I-, T-, R-
- b) Pfad: A+, D+, G-, I → S, T-, R-
- c) Pfad: A+, D+, G-, I-, T → S, R-
- d) Pfad: R → A, E → D, G → R, I → E, T → S, R-

**Aufgabe 11: Maximale Teilfolge****2+3+10 Punkte**

- a) Wieviele Teilfolgen existieren überhaupt?
- 2 Punkte*

Es gibt folgende Arrays:

$n$	Arrays der Länge 1:	$[1], [2], \dots, [n]$
$n - 1$	Arrays der Länge 2:	$[1..2], [2..3], \dots, [(n - 1)..n]$
	...	
2	Arrays der Länge $n - 1$ :	$[1..(n - 1)], [2..n]$
1	Arrays der Länge $n$ :	$[1..n]$

Also ergibt sich für die Anzahl der Teilfolgen nach der Gauss-Summe unter Addition des leeren Arrays:

$$A = \frac{n(n + 1)}{2}$$

- b) Geben Sie eine untere Schranke für die Komplexität eines Verfahrens zur Bestimmung der maximalen Teilfolge an.
- 3 Punkte*

Da jedes Element als Element der maximalen Teilfolge in Frage kommt und demnach untersucht werden muß, ist der Aufwand mindestens linear in der Anzahl der Elemente, also:

$$T \in \Omega(n)$$

Da nur konstanter Speicherplatz benötigt wird (die Eingabe der Folge wird dabei nicht mitgerechnet), gilt für die Speicherplatzkomplexität:

$$M \in \Omega(1)$$

- c) Entwerfen Sie einen Algorithmus zur Bestimmung der maximalen Teilfolge mit möglichst geringer Komplexität.
- 10 Punkte*

Der Algorithmus funktioniert wie folgt:

Die gesamte Folge wird von vorne bis hinten durchlaufen und führt dabei folgende Schritte durch:

- i) solange Werte positiv sind werden diese aufsummiert
- ii) ist das bisherige Maximum kleiner, als die in Schritt 1 ermittelte Summe, so setze das Maximum auf diese Summe
- iii) solange nun negative Werte folgen, werden diese aufsummiert
- iv) ist die Summe der negative und positiven Teilfolge positiv wird diese Summe als neue positive Summe gesetzt und direkt bei Schritt 1 fortgefahren, da durch eine nun folgende positive Teilfolge der maximale Wert ggf. noch gesteigert werden kann. Andernfalls macht es keinen Sinn, die negativen Werte in die maximale Teilfolge aufzunehmen, weil dieses nur zu einer Verringerung des Wertes führt; die positive Summe wird wieder auf 0 gesetzt und mit Schritt 1 versucht, eine neue Teilfolge zu finden, die eventuell größer als das Maximum ist.

Ist die Folge bis zum Ende durchlaufen, so steht der Wert der maximalen Teilfolge fest.