

Curriculum

Mittwoch, 11. April 2018 17:09

Thema	Datum	Buchkapitel	Softwarepraktikum/Testate
Insertion sort, correctness, pseudo-code conventions, simple analysis (2.1, 2.2) (My notes: Algorithmen) + Organisatorische Ankündigungen	10.04.	2.1 2.2	09.04. Test erstes Praktikum
Simple analysis of insertion sort cont. Asymptotic notation (3.1) (My notes: Asymptotische Notation)	12.04.	3.1	
Divide and conquer/ recursion: merge sort, correctness, analysis (2.3.1, 2.3.2) (My notes: Rekursion)	17.04.		
Substitution method and master theorem. Example is Merge Sort. (4.3, 4.5) (My notes: Rekurrenzgleichungen)	19.04.	4.3 4.5	
Quicksort (7) (My notes: Quicksort)	24.04.	6	
Heapsort (6) (No notes yet)	26.04.	7	
-	01.05.		
Elementary data structures (10) (No notes yet)	03.05.	10	Programmierpraktikum I Abgabe: 06.05. Thema: Quick Sort
Hash tables (11) (My notes: Hash-Tabellen)	08.05.	11	
-	10.05.		
Binary search trees (12) (My notes: Binäre Bäume)	15.05.	12	
Binary search trees (12) (My notes: Binäre Bäume)	17.05.	12	Programmierpraktikum II Abgabe: 20.05. Thema: Hash Tables
B-trees (18) (My notes: B-Bäume)	22.05.	18	
Graphs (22.1) (My notes: Graphen)	24.05.	15	
BFS (22.2) (My notes: BFS)	29.05.	16	
-	31.05.		Programmierpraktikum III Abgabe: 03.06. Thema: B-Tree
DFS (22.3) (My notes: DFS)	05.06.	22.1	
Dijkstra (24) (My notes: Dijkstra)	07.06.	22.2	
Minimum spanning trees (23) (My notes: minimum spanning trees)	12.06.	22.3	
Maximum flow (26) (My notes: maximum flow)	14.06.	23	Programmierpraktikum IV Abgabe: 17.06. Thema: Dijkstra
Maximum flow cont. If time, start with Dijkstra	19.06		
Dynamic programming (15) (No notes yet)	21.06		
	26.06	16	

Greedy algorithms (16) (No notes yet)			
String matching (32) (No notes yet)	28.06.	32	Programmierpraktikum IV Abgabe: 01.07. Thema: Max Flow
Complexity classes (34)	03.07.	34	Testate Programmierpraktika im Zeitraum 02.07.-13.07.2018 (letzte + vorletzte VL-Woche) Testate Hausübungen im Zeitraum 09.07.-13.07.2018 (letzte VL-Woche)
NP-completeness (34)	05.07	34	
	10.07.		
	12.07.		

Topic	Chapter
Insertion sort, correctness, pseudo-code conventions, simple analysis	2.1 2.2
Simple analysis of insertion sort cont. Asymptotic notation	3.1
Divide and conquer/ recursion: merge sort, correctness, analysis	2.3.1, 2.3.2
Substitution method and master theorem. Example is Merge Sort.	4.3, 4.5
Quicksort	6
Heapsort	7
Elementary data structures	10
Hash tables	11
Binary search trees	12
B-trees	18
Graphs	15
BFS	16
DFS	22.1
Dijkstra	22.2
Minimum spanning trees	22.3
Maximum flow	23
Dynamic programming	15
Greedy algorithms	16
String matching	32
Complexity classes	34
NP-completeness	34

1 - Algorithmen

Sonntag, 13. April 2014 18:26

Diese Vorlesung handelt von Algorithmen.

Was ist ein Algorithmus?

Beispiel: Sortierproblem

Eingabe: Die Zahlenfolge

$$\langle a_1, \dots, a_n \rangle$$

Ausgabe: Eine Permutation (Umordnung)

$$\langle a'_1, \dots, a'_n \rangle$$

dieselben Folge mit der Eigenschaft

$$a'_1 \leq a'_2 \leq \dots \leq a'_n$$

Zum Beispiel

Eingabe: $\langle 31, 41, 59, 26, 41, 58 \rangle$

Ausgabe $\langle 26, 31, 41, 41, 58, 59 \rangle$

Die Eingabe ist eine Instanz des Sortierproblems.

Der Algorithmus ist korrekt, wenn er das

Der Algorithmus ist korrekt, wenn er das Problem für alle Instanzen löst.

Der folgende Algorithmus löst das Problem
Ich erläutre ihn zuerst an einem
Beispiel

1 2 3 4 5 6
31 41 59 26 41 58

$$j = 4$$

$$\text{key} = 26$$

31 41 59 41 58
 ↑ ↑ ↗

31 41 59 41 58
↑
 $\text{key} = 26$

Insertion sort

for $j \leftarrow 2$ to $\text{length}[A]$

do $\text{key} \leftarrow A[j]$

$i \leftarrow j - 1$

 while $i > 0 \wedge A[i] > \text{key}$
 do $A[i+1] \leftarrow A[i]$
 $i \leftarrow i - 1$

$A[i+1] \leftarrow \text{key}.$

Korrekt heit beweis :

Schleifen invarianten : zu Beginn der for-Schleife besteht die Teilfolge $A[1, \dots, j-1]$ aus den Elementen der Original - Teilfolge und ist sortiert.

Der Korrekt heit Beweis ist ein Induktions Beweis.

Man beweist die Invariante für die Initialisierung.

Die Invariante wird beim Durchlauf der Schleife erhalten, d.h.

Man zeigt : wenn die Invariante vor einer Iteration der Schleife korrekt ist, dann auch danach.

zum Schluss zeigt man : Bli
Beendigung der Schleife folgt aus der Invariante die Korrektheit des Algorithmus.

Initialisierung ✓

Erhaltung : erfordert bei formalem Beweis die Einführung

eines weiteren Varianten

Abschluss

✓

Analyse :
 Wie gut?
 Wie schnell?
 < Wie viel Platz? >
 < Wie viel Code? >

Man benötigt ein Berechnungsmodell.

RAM : Instruktionen werden nacheinander abgearbeitet
 Auf alle Speicherplätze kann gleich schnell zugegriffen werden.

Laufzeit : Anzahl der Schritte
 Die Ausführungszeit einer Zeile im Pseudocode ist eine Konstante.

Eingabelänge : Anzahl der zu sortierenden Objekte.

Ziel : Laufzeit als Funktion der Eingabelänge bestimmen.

Achtung : auch andere Definitionen von Laufzeit und Eingabelänge möglich.

Auch der Speicherplatz, der benötigt wird, spielt eine wichtige Rolle. Damitmidt.

Insertion sort

Kosten #

for $j \leftarrow 2$ to length [A]

$c_1 n$

do key $\leftarrow A[j]$

$c_2 n-1$

$i \leftarrow j-1$

$c_4 n-1$

while $i > 0 \wedge A[i] > \text{key}$

$c_5 \sum_{j=2}^n t_j$

$i \leftarrow i-1$

$\sum_{j=2}^n t_j$

$A[i+1] \leftarrow \text{key}$

$c_7 \sum_{j=2}^n (t_j - 1)$

$c_8 n-1$

$$T(n) = c_1 n + c_2 (n-1) + c_4 (n-1)$$

$$+ c_5 \sum_{j=2}^n t_j$$

$$+ c_6 \sum_{j=2}^n (t_j - 1)$$

$$+ c_7 \sum_{j=2}^n (t_j - 1) + c_8 (n-1)$$

Best Case: Folge geordnet: $\forall j: t_j = 1$

$$T(n) = c_1 n + c_2 (n-1) + c_4 (n-1) + c_5 (n-1)$$

$$+ c_8 (n-1)$$

$$= (c_1 + c_2 + c_4 + c_5 + c_8)h - (c_2 + c_3 + c_5$$

der Eingabellänge.

Worst case: Folge in umgekehrter Reihenfolge sortiert.

$$t_j = j : j = 2, 3, \dots, n$$

$$\sum_{j=2}^n (j-1) = \frac{n(n-1)}{2} \quad \sum_{j=2}^n j = \frac{n(n+1)}{2}$$

$$T(n) = \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2} \right) h^2 + (c_1 + c_2 + c_4 + \frac{c_5}{\alpha} - \frac{c_6}{\alpha} - \frac{c_7}{\alpha} + c_8)h - (c_2 + c_4 + c_5 + c_8)$$

quadratisch.

2 - Asymptotische

Notation

Montag, 14. April 2014 10:14

Wir führen Notation für asymptotische Schreibweise ein.

Alle Funktionen sind

$f_1, f_2, \dots : \mathbb{N} \rightarrow \mathbb{R}_{>0}$
↑
Eingabewerte Zeit.

$$\Theta(g) = \left\{ f : \exists c_1, c_2 \in \mathbb{R}_{>0}, n_0 \in \mathbb{N} \quad \forall n \geq n_0 \quad 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \right\}$$

Beispiel :

$$f(n) = \frac{1}{2}n^2 - 3n$$

Beh: $f \in \Theta(n^2)$

Beweis: $\forall n \geq 20 : \frac{1}{4}n^2 \leq f(n) \leq n^2$

Beh: $6n^3 \notin \Theta(n^2)$

Beweis: $c_1 g(n) \leq f(n) \leq c_2 g(n)$

\Rightarrow

$$c_1 \leq \frac{f(n)}{g(n)} \leq c_2$$

$$-1 \quad - \quad g(n) \quad - \quad -2$$

Aber $\lim_{n \rightarrow \infty} 6n^3/n^2 = \lim_{n \rightarrow \infty} 6n = \infty$.

$$\mathcal{O}(g) = \{ f \mid \exists n_0 \in \mathbb{N}, c \in \mathbb{R}_{>0} \quad \forall n \geq n_0 \\ f(n) \leq c g(n) \}$$

Beispiel

$$1000n^2 \in \mathcal{O}(n^2 \log n)$$

Beweis: $n_0 = 1, c = 1000$

Beispiel $g(n) = \begin{cases} n^3 & \text{für } n < 1000 \\ n^2 & \text{für } n \geq 1000 \end{cases}$

$$g(n) \in \mathcal{O}(n^2)$$

$$\underline{\Omega}(g) = \{ f : \exists c \in \mathbb{R}_{>0}, n_0 \in \mathbb{N}$$

Schreibweise: statt $f \in \underline{\Omega}(g)$ schreibt man auch $f = \underline{\Omega}(g)$.

Satz $f \in \Theta(g) \Leftrightarrow f \in \mathcal{O}(g) \wedge f \in \underline{\Omega}(g)$

Notation

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n) \\ = \Theta(n^2)$$

$$O(g) = \left\{ f : \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0 \right\}$$

$$\omega(g) = \left\{ f : \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 0 \right\}$$

Wir können zu den beschriebenen Funktionsmengen Relationen definieren:

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

Diese Relationen haben folgende Schichten:

$\Theta, O, o, \Omega, \omega$ sind transitiv

Θ, O, Ω sind reflexiv

Θ ist symmetrisch.

Die Mengen stehen auf folgendermaßen miteinander in Beziehung:

$$f = O(g) \Leftrightarrow g = o(f)$$

$$f = \Omega(g) \Leftrightarrow g = \omega(f)$$

(strong) monoton fallend, wachsend

(strong) monoton fallend, wachsend

Hier noch einige Bezeichnungen für Funktionen in gewissen Mengen:

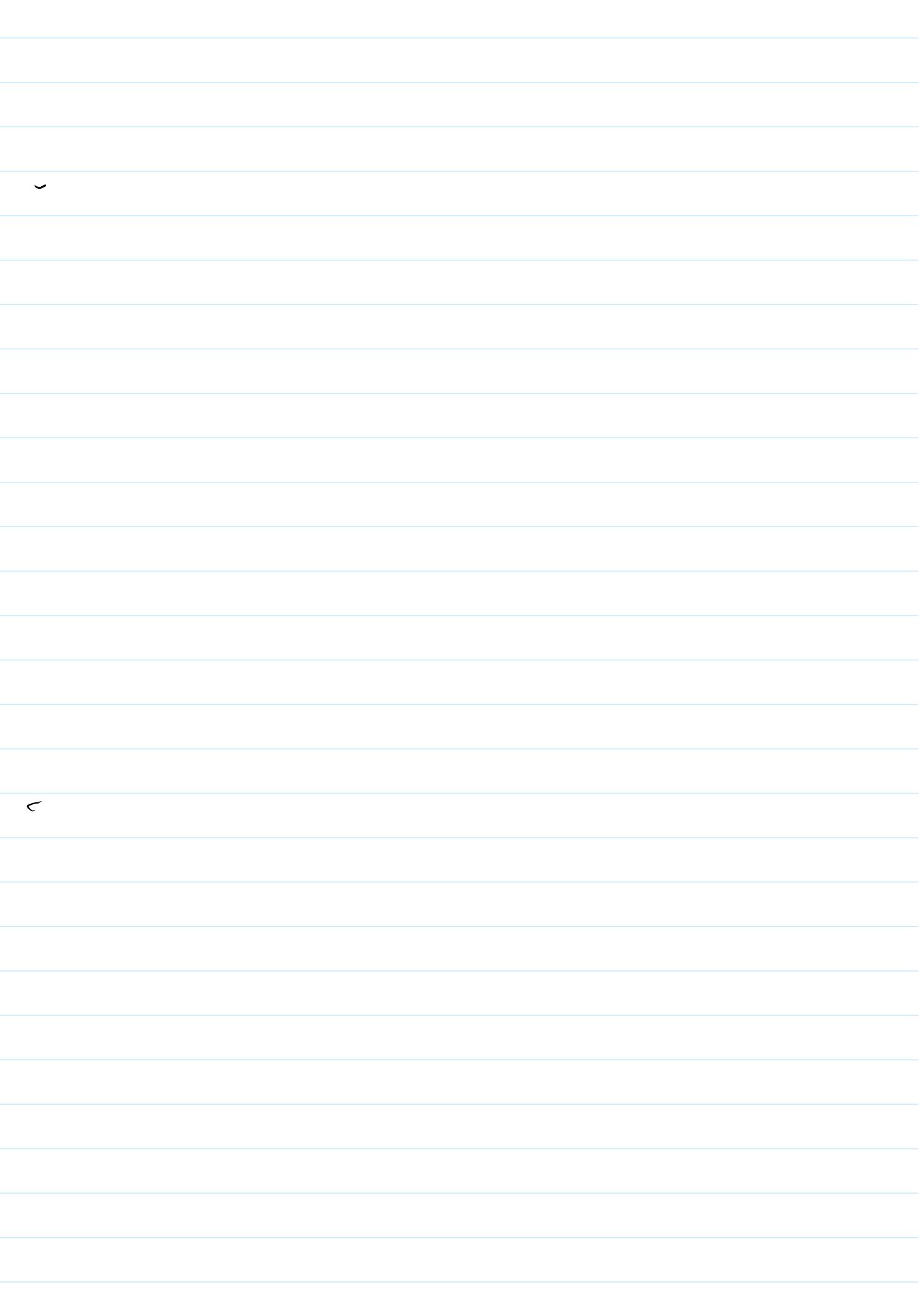
$\Theta(1)$	konstant
$\Theta(n)$	linear
$\Theta(n \lg n)$	quasi linear
$\Theta(n^2)$	quadratisch
$\Theta(n^3)$	kubisch
$\Theta(n^k)$	polynomell
$\Theta(2^n)$	exponentiell

Einige weitere Beispiele.

$n^{\lg n} = \Theta(n)$, weil für $n \geq 2$
 $\lg n \geq 1$ und darum $n^{\lg n} \geq n$. ($C=1!$)

"Übung zu asymptotischen Notationen

zeigen Sie, dass $\lg(n!)$ = $\Theta(n \lg n)$ ist
und dass $n! = O(n^n)$ ist.



Kann man effizienter sortieren?

Divide and conquer Strategie:

Divide: Zerlege das Problem in Unterprobleme

Conquer: Löse die Unterprobleme

- entweder durch Zerlegung in weitere Unterprobleme, also **rekursiv**.
- oder **nicht**, wenn das Problem klein genug ist.

Combine: Kombiniere die Teillösungen zu einer Gesamtlösung.

Anwendung:

Divide: Zerlege Folge in zwei Teilfolgen halber Länge

Conquer: Sortiere die Teilfolgen

Merge : Füge zwei sortierte Teilfolgen zusammen (Merge)

Spezifikation von Merge (A, p, q, r)

Eingabe

- * $p \leq q < r$
- * $A[p..q]$ sortiert
- * $A[q+1..r]$ sortiert

Ausgabe

$A[p..r]$ sortiert

Wenn ein solches Programm Merge existiert, kann man Merge Sort folgendermaßen implementieren:

MERGE-SORT(A, p, r)

```
1 if  $p < r$ 
2    $q = \lfloor (p + r) / 2 \rfloor$ 
3   MERGE-SORT( $A, p, q$ )
4   MERGE-SORT( $A, q + 1, r$ )
5   MERGE( $A, p, q, r$ )
```

Jetzt diskutieren wir Merge. Hier ist die Idee.



Zwei Kartenstapel.

- * Vergleiche die beiden oberen Karten und wähle die kleinere
- * Legt sie mit dem Gesicht nach unten auf den Ausgabestapel.

Optimierung:

Füge an die Teilfolgen das Element ∞ an. Dann braucht man nicht jedes mal abfragen, ob der Stapel leer ist.

Merge (A, p, q, r)

hier der Algorithmus.

MERGE(A, p, q, r)

```

1   $n_1 = q - p + 1$ 
2   $n_2 = r - q$ 
3  let  $L[1..n_1 + 1]$  and  $R[1..n_2 + 1]$  be new arrays
4  for  $i = 1$  to  $n_1$ 
5     $L[i] = A[p + i - 1]$ 
6  for  $j = 1$  to  $n_2$ 
7     $R[j] = A[q + j]$ 
8   $L[n_1 + 1] = \infty$ 
9   $R[n_2 + 1] = \infty$ 
10  $i = 1$ 
11  $j = 1$ 
12 for  $k = p$  to  $r$ 
13   if  $L[i] \leq R[j]$ 
14      $A[k] = L[i]$ 
15      $i = i + 1$ 
16   else  $A[k] = R[j]$ 
17      $j = j + 1$ 
```

Schleifeninvariante:

Am Anfang der for-Schleife in Zeile 12-17 enthält

$A[p, \dots, k-1]$ die $k-p$ kleinsten Elemente von $L[1..n_1+1], R[1..n_2+1]$ in sortierte Form.

$L[i], R[j]$ sind die kleinsten bis jetzt nicht kopierten Elemente in L und R .

Wir beweisen diese Invariante induktiv und leiten daraus die Eigenschaften des Algorithmus ab.

Erster Durchlauf:

$$k = p, i = 1, j = 1$$

$L[1]$ ist das kleinste Element in L
 $R[1]$ ist das kleinste Element in R
 $k = p$ impliziert, dass die Folge

$A[p, k-1]$ leer ist. Also ist die erste Aussage trivial und die zweite Aussage wahr.

Angenommen die Invariante gilt vor der Schleife, dann auch nachher.

Das sieht man so:

Wir wissen bereits, dass $L[i]$, $R[j]$ die beiden noch nicht eingeordneten Elemente von L bzw. R sind. Wir wissen auch, dass alle eingeordneten Elemente von L höchstens so groß wie $L[i]$ sind und dass alle eingeordneten Elemente von R höchstens so groß wie $R[j]$ sind. Das liegt daran, dass L und R sorte te Folgen sind.

Wenn $L[i] \leq R[j]$ ist, wird $L[i]$ als Element $A[k]$ eingeordnet und i um 1 erhöht. Also ist die Invariante erfüllt. Im andern Fall gilt das Entsprechende.

Terminierung:

Bei Terminierung ist $k = n+1$. Die Invariante impliziert, dass $A[p, \dots, r]$ die Elemente von L und R enthält und sortiert ist. Nach Konstruktion gilt aber

$$L[1, \dots, n_1] = A[p, \dots, q-1]$$

$$R[1, \dots, n_2] = A[q, \dots, r]$$

Also ist das neue A das alte A in sortierter Reihenfolge.

Wir analysieren Merge.

Behauptung: Merge benötigt Zeit $\Theta(n)$.
mit $n = r - p + 1$.

Um das zu beweisen, stellen wir zunächst fest, dass Zeilen 1-3 und 8-11 konstante Zeit benötigen.

Die for-Schleifen in Zeilen 4-7 benötigen Zeit $\Theta(n_1 + n_2) = \Theta(q-p+1 + r-q) = \Theta(r-p+1) = \Theta(n)$.

Die for-Schleife in Zeilen 12-17 benötigt $\Theta(p-r+1) = \Theta(n)$ Durchläufe. Die Anweisungen in dieser Schleife benötigen Zeit $\Theta(1)$. Also ist die gesamte Laufzeit $\Theta(n)$.

Jetzt wird der gesamte Algorithmus analysiert.

Die Laufzeit wird mit $T(n)$ bezeichnet.
In der Analyse benutzen wir folgendes

1. Wenn die Anzahl der zu sortierenden Objekte hinreichend klein ist, also unterhalb einer Schranke c , so ist die Laufzeit konstant ($\Theta(1)$)

2. Bei der Rekurrenz setzt sich die Laufzeit zusammen aus den Laufzeiten der kleineren Probleme (in unserem Fall sind das 2), der Laufzeit $D(n)$ für die Aufteilung des Problems und der Laufzeit $C(n)$ für die Zusammensetzung der kleineren Lösungen. Allgemein bekommt man damit die Formel

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c \\ a T(n/b) + D(n) + C(n) & \text{otherwise} \end{cases}$$

↓ ↓ ↓
 Divide Combine
 a Teilprobleme
 der Größe n/b .

Eine MergeSort ergibt sich speziell für $n = 2^k$

$$T(n) = \begin{cases} c & n=1 \\ 2T(n/2) + c \cdot n & \text{otherwise} \end{cases}$$

Daraus erhält man

$$\begin{aligned}T(n) &= 2 T(n/2) + c \cdot n \\&= 2^2 T(n/2^2) + 2c \cdot n \\&= \dots = \\&= 2^k T(1) + k \cdot c \cdot n \\&= (n + n \lg n) \in \\&= \Theta(n \lg n)\end{aligned}$$

4 - Rekurrenzgleichungen

Donnerstag, 17. April 2014 13:09

Wir haben bei der Analyse rekursive Algorithmen bereits Rekurrenzgleichungen gesehen. z.B. bei Merge-Sort

$$T(n) = \begin{cases} c_1, & \text{falls } n=1, \\ 2T(n/2) + c_2, & \text{falls } n>1. \end{cases}$$

Die Lösung solcher Gleichungen soll hier noch einmal systematisch erklärt werden.

Unter "Lösung" verstehe ich die Bestimmung einer geschlossenen Formel für T oder die Angabe einer geschlossenen Formel für eine **obere Schranke** oder eine untere Schranke.

Beispiel Merge-Sort

$$T(1) \leq C$$

$$T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + C$$

Hierin sind die Funktionen $\lceil \cdot \rceil$ und $\lfloor \cdot \rfloor$ folgendermaßen definiert. Für eine reelle Zahl x ist $\lfloor x \rfloor$ die eindeutig bestimmte ganze Zahl mit der Eigenschaft

$$x-1 < \lfloor x \rfloor \leq x.$$

Analog dazu ist $\lceil x \rceil$ die eindeutig bestimmte ganze Zahl mit

$$x \leq \lceil x \rceil < x+1.$$

Beispiele: $\lfloor 3.1 \rfloor = 3, \lceil 3.1 \rceil = 4,$
 $\lfloor -4.1 \rfloor = -5, \lceil -4.1 \rceil = -4,$

$$\lfloor -4 \cdot 1 \rfloor = -5, \lceil -4 \cdot 1 \rceil = -4,$$

$$\lfloor -4 \rfloor = \lceil -4 \rceil = -4.$$

Wir raten die Abschätzung

$$T(n) \leq 4C \cdot n \lg n.$$

und beweisen sie durch Induktion.

Hilfsbehauptung 1.

$$\left\lfloor \frac{n}{2} \right\rfloor + \left\lceil \frac{n}{2} \right\rceil = n \quad (\text{Übung})$$

für alle

$$\begin{array}{lll} \text{Beweis} & n \text{ gerade} & \left\lfloor \frac{n}{2} \right\rfloor = \frac{n}{2} = \left\lceil \frac{n}{2} \right\rceil \\ & n \text{ ungerade} & \left\lfloor \frac{n}{2} \right\rfloor = \frac{n-1}{2} \quad \left\lceil \frac{n}{2} \right\rceil = \frac{n+1}{2} \end{array}$$

Hilfsbeh. 2

$$\begin{aligned} \left\lfloor \frac{n}{2} \right\rfloor &\leq \frac{n}{2} \\ \left\lceil \frac{n}{2} \right\rceil &\leq \frac{n+1}{2} \leq \frac{2}{3}n \quad \text{für } n \geq 3 \end{aligned}$$

$$\text{Beh: } T(n) \leq 4C \cdot n \lg n, \quad n \geq 2$$

Beweis

Induktionsanfang:

$$\begin{aligned} n=1: \quad T(1) &\leq C, \quad T(2) \leq T(1) + T(1) + 2C \\ &\leq 4C \end{aligned}$$

$$4C \cdot 2 \lg 2 = 8C$$

Induktionsschritt: Sei $n \geq 2$ und sei Bch. wahr für alle $n' < n$.

$$\begin{aligned}
 T(n) &= T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + cn \\
 &\stackrel{*}{\leq} 4c \lfloor \frac{n}{2} \rfloor \lg \lfloor \frac{n}{2} \rfloor + 4c \lceil \frac{n}{2} \rceil \lg \lceil \frac{n}{2} \rceil + cn \\
 \text{Indann} &\leq 4c \log\left(\frac{2n}{3}\right) \left(\lfloor \frac{n}{2} \rfloor + \lceil \frac{n}{2} \rceil \right) + cn \\
 \text{HB 2} &= 4c \lg\left(\frac{2}{3}n\right) n + cn \\
 &= 4cn \lg n + cn \underbrace{\left(1 - 4 \lg \frac{3}{2}\right)}_{\leq 0} \\
 &\leq 4cn \lg n
 \end{aligned}$$

Wie rät man richtig?

Grundlegend ist der folgende Satz.

Satz Sei $T(n) = aT(n/b) + f(n)$ $a \geq 1$, $b > 1$.

1) Ist $f(n) = O(n^{\log_b a - \varepsilon})$ für ein $\varepsilon > 0$, dann ist $T(n) = \Theta(n^{\log_b a})$

2) Ist $f(n) = \Theta(n^{\log_b a})$, dann ist $T(n) = \Theta(n^{\log_b a} \lg n)$

3) Ist $f(n) = \Omega(n^{\log_b a + \varepsilon})$ für ein $\varepsilon > 0$. a. $f(n/b) < c f(n)$ für ein $c < 1$

und alle hinreichend großen n , dann ist $T(n) = \Theta(f(n))$.

Anwendungen:

$$1) T(n) = 9T(n/3) + n, \quad a=9, b=3, f(n)=n.$$

$$\log_b a = \log_3 9 = 2$$

$$\begin{aligned} f(n) = n &= O(n^{\frac{\log_b a - \varepsilon}{2}}) \quad \text{für ein } \varepsilon? \\ &= O(n^{2-\varepsilon}) \quad " \end{aligned}$$

$$\varepsilon = 1\%$$

$$\text{Also } T(n) = \Theta(n^2).$$

$$2) T(n) = T(2n/3) + 1.$$

$$\begin{aligned} a = 1, \quad b = \frac{3}{2}, \quad \log_{3/2} 1 &= 0. \\ f(n) = 1 & \end{aligned}$$

$$f(n) = 1 = O(n^{\log_b a}) = O(1).$$

$$\text{Also ist } T(n) = \Theta(\lg n).$$

$$3) T(n) = 3T(n/4) + n \lg n$$

$$a = 3, \quad b = 4, \quad n^{\log_b a} = n^{\log_4 3} \leq n^{0.793}$$

$$f(n) = n \lg n \geq n \geq n^{0.793+0.1} \geq n^{\log_b a}$$

$$\Rightarrow f(n) = \Omega(n^{\log_b a + 0.1})$$

Außerdem ist

$$a \cdot f(n/b) = 3 f(n/4)$$

$$\begin{aligned}
 a \cdot f(n/b) &= 3f(n/4) \\
 &= 3(n/4) \lg(n/4) \leq \left(\frac{3}{4}\right)^4 n \lg n
 \end{aligned}$$

Also ist $T(n) = \Theta(n \lg n)$

4) $T(n) = 2T(n/2) + n \lg n$

$$a = b = 2, \quad n^{\frac{\lg_b a}{2}} = n, \quad f(n) = n \lg n$$

Fall 1,2 kann nicht angewendet werden,
weil $f(n) \notin \Theta(n)$.

Fall 3? Sicher gilt $f(n) = \underline{O}(n)$.

Aber ist auch $a f(n/b) = 2f(n/2) = n \lg(n/2) < c f(n) = c n \lg(n/2)$ für ein $c < 1$?

$$\begin{aligned}
 \text{Es gilt } \lim_{n \rightarrow \infty} \frac{n \lg(n/2)}{n \lg n} &= \lim_{n \rightarrow \infty} \frac{n \lg n - n}{n \lg n} \\
 &= \lim_{n \rightarrow \infty} \frac{1 - \frac{1}{\lg n}}{1} = 1
 \end{aligned}$$

Also gibt es ein solches c nicht.

"Übung":

G 1) Zeigen Sie, dass die Lösung von
 $T(n) = T(\lceil n/2 \rceil) + 1$
 in $O(\lg n)$ liegt.

G 2) Zeigen Sie, dass die Lösung von
 $T(n) = 2T(\lfloor n/2 \rfloor) + n$
in $\Omega(n \lg n)$ liegt

H 3) Zeigen Sie, dass die Lösung von
 $T(n) = 2T(\lfloor n/2 \rfloor + 1) + n$
in $O(n \lg n)$ liegt.

Heap: nearly complete binary tree filled on all levels. Except possibly the lowest, which is filled from the left up to a point.

Two representations: Tree. Array.
See slide 2.

A.length # elements in array

A.heap-size \leq A.length
elements stored in heap

A[1] root

Functions parent, left, right
see slide 3

max-heap : $\forall i > 1 \quad A[\text{PARENT}(i)] \geq A[i]$

min-heap : $\dots \leq A[i]$

height of node = # Kanten
in Pfad von Wurzel zu Knoten

height of heap = maximum
aller height of node

n Elemente \Rightarrow height of heap
 $= \Theta(\lg n)$

MAX-HEAPIFY

Input: $A[i]$ mit der Eigenschaft,
dass Unterbäume mit Wurzeln
 $\text{LEFT}(i)$, $\text{RIGHT}(i)$ sind MAX-HEAPS.
 $A[i]$ kann aber kleiner als
seine Kinder sein.

Output: A : Die Knotenmenge

des Teilbaums mit Wurzel $A[i]$

Algorithmus siehe Folien 4 und 5

$$\text{Laufzeit : } T(n) \leq T(2n/3) + \Theta(1)$$

Weil die Größe des Teilbaums höchstens $2/3$ Größe des urspr. Baumes ist.

$$\text{Laufzeit } \cancel{\text{HEAPIFY}} \quad a = 1, b = 3/2$$

$$\text{Vergleiche } f(n) \text{ mit } n^0 = 1$$

Fall 2 des Master Theorems

$$T(n) = O(n \lg n)$$

Man kann aber sogar beweisen

$$T(n) = O(n)$$

$$\text{Laufzeit HEAP-SORT} : O(n \lg n)$$

6 - Quicksort

Donnerstag, 17. April 2014 15:08

Wir behandeln jetzt das Sortierproblem.

Eingabe: Eine Folge (ganzer) Zahlen $\langle a_1, \dots, a_n \rangle$

Ausgabe: Eine Permutation $\langle a'_1, \dots, a'_n \rangle$ der Eingabefolge mit

$$a'_1 \leq a'_2 \leq \dots \leq a'_n.$$

Anwendung auf reale Daten

In der Praxis werden nicht Zahlen geordnet, sondern komplexe Datenstrukturen, sogenannte **records**.

Jeder **record** hat einen **Schlüssel** (key), nach dem geordnet wird.

Beispiel:

Emails können nach Absender (key!), Datum (key!), subject (key!) geordnet werden. Daraan sieht man, dass die keys i.A. keine Zahlen sein müssen. Benötigt wird nur eine Folge mit einer **Totalordnung**, also je zwei Einträge sind vergleichbar.

Zwei **Sortieralgorithmen** werden schon vor gestellt:

Insertion Sort: $O(n^2)$
Merge Sort $O(n \log n)$ Laufzeit

Quick Sort

Quicksort ist ein effizienter Sortieralgorithmus mit worst case Laufzeit $O(n^2)$ aber durchschnittlicher Laufzeit $O(n \log n)$ und sehr guter praktischer Performance.

So funktioniert Quicksort

Divide : $A[p \dots r]$ wird geteilt (umgeordnet) und $q \in \{p, \dots, r\}$ wird bestimmt

$$\begin{array}{lll} \text{alle Elemente von } A[p \dots q] & \leq & A[q] \\ " & " & " \\ & & A[q+1 \dots r] & \geq & A[q] \end{array}$$

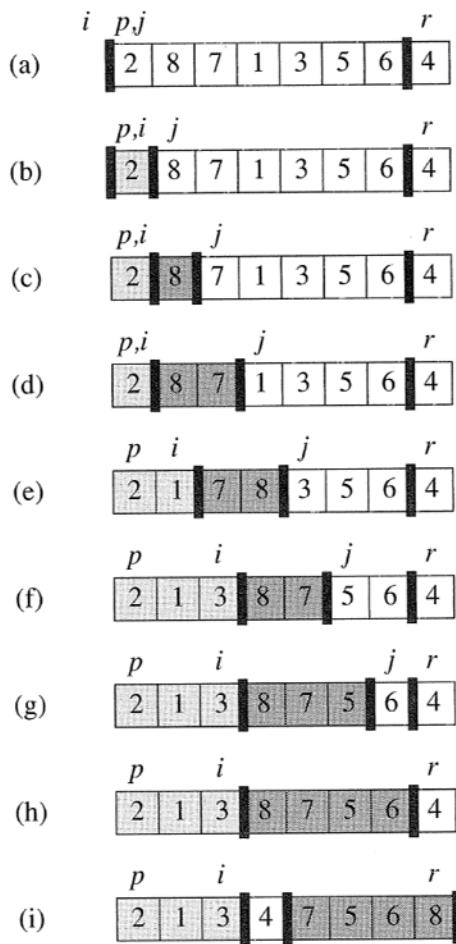
Conquer : Sortiere $A[p \dots q-1]$,
 $A[q+1 \dots r]$.

Combine : $A[p \dots r]$ ist dann sortiert.

Partition :

$$x = A[r] \quad \text{pivot}$$

Idee: gehe die Liste durch. Sammle die kleinen vorne und die großen hinten und am Schluss kommt das letzte Element in die Mitte.



QUICKSORT(A, p, r)

```

1  if  $p < r$ 
2    then  $q \leftarrow \text{PARTITION}(A, p, r)$ 
3       $\text{QUICKSORT}(A, p, q - 1)$ 
4       $\text{QUICKSORT}(A, q + 1, r)$ 

```

PARTITION(A, p, r)

```

1   $x \leftarrow A[r]$ 
2   $i \leftarrow p - 1$ 
3  for  $j \leftarrow p$  to  $r - 1$ 
4    do if  $A[j] \leq x$ 
5      then  $i \leftarrow i + 1$ 
6        exchange  $A[i] \leftrightarrow A[j]$ 
7  exchange  $A[i + 1] \leftrightarrow A[r]$ 
8  return  $i + 1$ 

```

Invariante für Zeile 3

1. Für $p \leq k \leq i$ gilt $A[k] \leq x$
2. Für $i < k \leq j-1$ gilt $A[k] \geq x$
3. $A[r] = x$.

Ableitung der Korrektheit aus der Invariante:

$$j = r ,$$

- Für $p \leq k \leq i \quad A[k] \leq A[r]$
 Für $i < k \leq r-1 \quad A[k] \geq A[r]$

nach 7 kann $A[r]$ durch $A[i+1]$ ersetzt werden und zusätzlich gilt

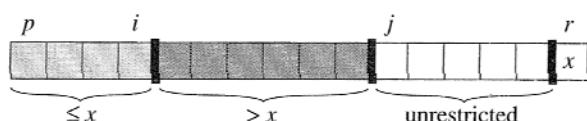
$$A[i+1] \leq A[r]$$

Beweis der Invariante

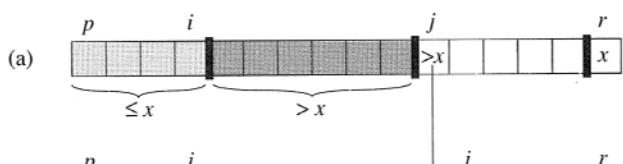
Initial:

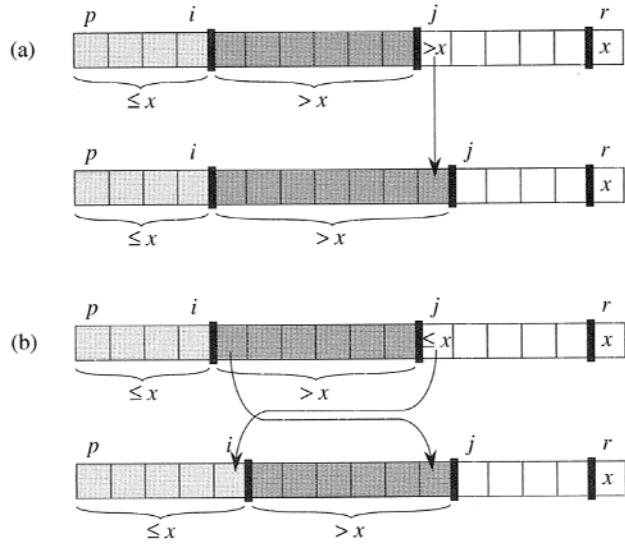
$$i = p-1, \quad j = p \quad 1., 2. \text{ leer}, 3. \checkmark$$

Induktionsabschnitt:



Der Algorithmus arbeitet mit obigen Blöcken. Es hat zwei mögliche Alternativen:





In beiden Fällen bleibt die Invariante erhalten.

Wir analysieren Quick-Sort: Partition benötigt Zeit $\Theta(n)$.
Worst case:

In jedem Aufruf wird die Aufteilung $n-1, 0$ vorgenommen

Dann

$$\begin{aligned}
 T(n) &= T(n-1) + T(0) + \Theta(n) \\
 &= T(n-1) + 2T(0) + \sum_{i=1}^n \Theta(i) \\
 &= \dots = nT(0) + \sum_{i=1}^n \Theta(i) \\
 &= \Theta(n^2).
 \end{aligned}$$

[Übung soll die genaue Abschätzung liefern].

Beispiel: Der schlechteste Fall tritt ein, wenn die Folge bereits sortiert ist.

$[1, 2, 3, 4, 5, 6, 7]$

\downarrow
 $[1, 2, 3, 4, 5, 6] \quad i+1 = 7$

\downarrow

\vdots

$[1]$

$i+1 = 2$

Dann läuft Insertion Sort ^{aber} in Zeit $\Theta(n)$

Best case:

Aufteilung $\lfloor n/2 \rfloor / \lceil n/2 \rceil - 1$

$$T(n) \leq 2T(n/2) + \Theta(n)$$

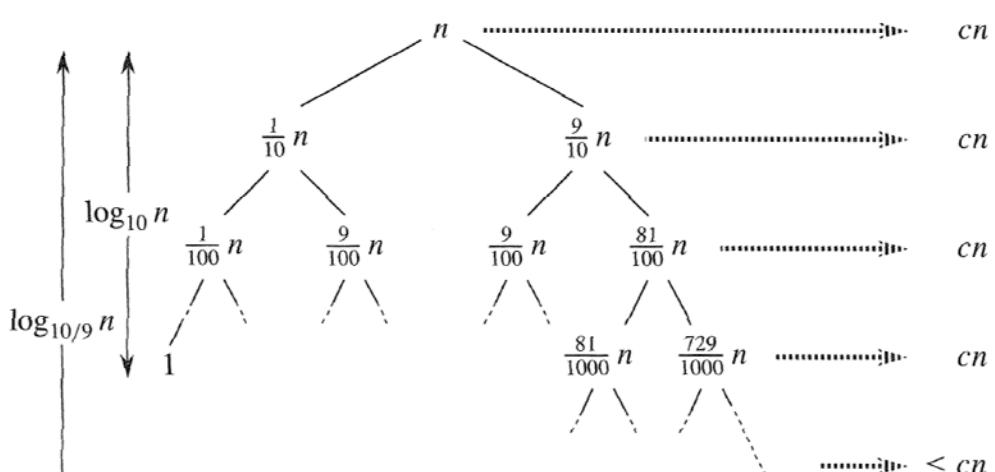
$T(n) \leq O(n \log n)$ nach dem Master Theorem.

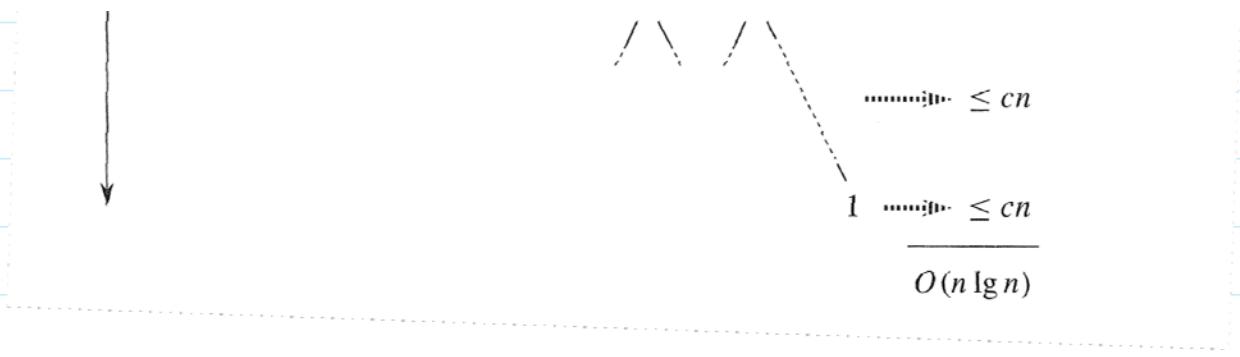
Average case:

Intuition: Selbst wenn man

$$T(n) = T(\frac{9}{10}n) + T(n/10) + cn$$

hat, bekommt man die Laufzeit $\Theta(n \log n)$





$$\left(\frac{9}{10}\right)^k n \leq 1$$

$$\Leftrightarrow n \leq \left(\frac{10}{9}\right)^k$$

$$\Leftrightarrow \log n \leq k \log \left(\frac{10}{9}\right)$$

$$\begin{aligned} k &\geq \log n / \log 9/10 \\ &= O(\log n). \end{aligned}$$

Rekursions tiefe $O(\log n)$

$$T(n) = O(n \log n).$$

Intuition für den durchschnittlichen Fall:

Nehmen wir an, gute und schlechte Aufteilungen sind gleich wahrscheinlich. Später wird das noch präzisiert. Dann folgt also eine gute Aufteilung auf eine schlechte.

Das sieht man unten im Bild.

Wenn die erste schlecht und die zweite gut ist, dann sind beide zusammen genommen gut. Das liegt daran, dass die durch die Rekursion verbleibenden Aufgaben offensichtlich gleich aufwändig sind.

Da man nicht vorher weiß, wie gut die Aufteilung ist, die sich bei Quicksort ergibt, verwendet man folgende **randomisierte** Variante von Quicksort:

Dieser Algorithmus wählt zuerst eine Zufallszahl zwischen p und r . Das macht die Funktion RANDOM. Die Verteilung ist die Gleichverteilung. Alle Werte sind also gleich wahrscheinlich. Jedes Pivot-Element wird also gleich wahrscheinlich.

Danach kommt der normale Quicksort Algorithmus.

Worst Case Analyse

Wir haben die folgende Rekurrenz für die Laufzeit $T(n)$:

$$T(n) = \max_{0 \leq q \leq n-1} (T(q) + T(n-q-1)) + \Theta(n)$$

Wir raten $T(n) \leq cn^2$.

Durch **Substitution** erhalten wir

$$\begin{aligned} T(n) &\leq \max_{0 \leq q \leq n-1} (cq^2 + c(n-q-1)^2) + \Theta(n) \\ &= c \max_{0 \leq q \leq n-1} (q^2 + (n-q-1)^2) + \Theta(n) \end{aligned}$$

Nun ist

$$q^2 + (n-q-1)^2 \leq (n-1)^2$$

$$= n^2 - 2n - 1$$

Also ist

$$\begin{aligned} T(n) &\leq cn^2 - c(2n-1) + \Theta(n) \\ &\leq cn^2. \\ &\uparrow \\ &\text{hinterreichend großes } c \end{aligned}$$

Expected time Analyse

Wir haben schon gesehen, dass die Arbeit für schlechte Aufteilungen durch die nachfolgende Arbeit kompensiert wird. Dies wird in folgendem Lemma formalisiert.

Lemma Q1

Sei X die Anzahl aller Vergleiche, die während der Sortierung eines Arrays der Länge n gemacht werden. Dann ist die Laufzeit von Quicksort $O(n + X)$.

Beweis:

In jedem Aufruf von Partition wird ein Element Pivot, das danach nicht mehr angefasst wird.

Daher ist die maximale Anzahl der Aufrufe von Partition n . Die gesamte Laufzeit ist jetzt $\sum_{i=1}^n \text{Anzahl der Aufrufe von Part.}_i$.

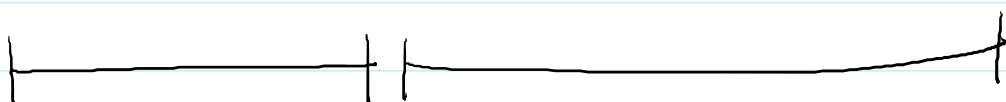
$$\stackrel{\downarrow}{O(n + X)}.$$

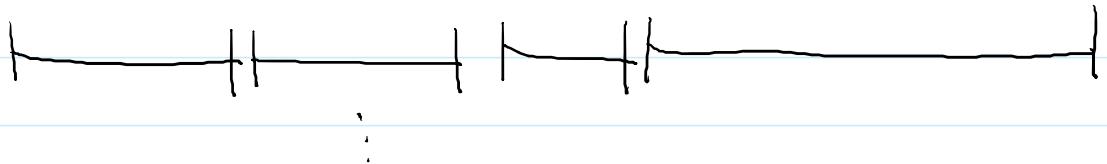
□

Jetzt kommt es darauf an, einen Erwartungswert für X zu berechnen.

Dazu schauen wir uns den Verlauf des Algorithmus an. Er gibt am Schluss das sortierte Array A aus.

Dies wird zusammengestellt aus zwei sortierten Teilen $\boxed{\quad | \quad}$ in denen ein Element als Pivot verwendet wurde, also mit allen verglichen wurde. Die beiden Teile werden ihrerseits aus zwei sortierten Teilen usw. Schaut man also vom Ende her, kann man diesen Vorgang als Zerlegung in sortierte Abschnitte auffassen.





H H H --- - - - H H

Sei also $Z = \langle z_1, \dots, z_n \rangle$ das sortierte Array A bei der Ausgabe.

Setze

$$z_{ij} = \langle z_i, \dots, z_j \rangle$$

Am Anfang ist jeder dieser Abschnitte in dem Sortierten Array enthalten. Auf einer der folgenden Ebenen kann ein solcher Abschnitt zerlegt werden und das wird er auch, weil am Schluss nur noch Arrays der Länge 1 vorhanden sind.

Abschnitte werden aber nur zerlegt, wenn ein Pivot aus ihm gewählt wurde.

Da am Schluss alle Abschnitte die Länge 1 haben, wird also aus jedem Abschnitt der Länge > 1 ein Pivot gewählt.

z_i wird genau dann mit z_j verglichen, wenn vor der Zerlegung von z_{ij} entweder z_i oder z_j als Pivot gewählt wird. Das geschieht höchstens einmal und zwar mit Wahrscheinlichkeit

$$\frac{2}{j-i+1}.$$

Damit ist die Erwartete Anzahl
von Vergleichen

$$\begin{aligned}
 & \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j-i+1} \\
 = & \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\
 < & \sum_{i=1}^{n-1} \sum_{k=1}^n \\
 = & \sum_{i=1}^{n-1} O(\lg n) \\
 = & O(n \lg n).
 \end{aligned}$$

□

"Übungsvorschläge:

7.1-1, 7.1-2, 7.1-4

7.2-1, 7.4-2, mw.

Joh Schreibe

Das ist dicker

Studien Leistung Anerkennen?

Studienordnung \rightarrow 50%

Zwei einseitig beschreibbare
A4 - Seiten?

Tesa!
Was machen wir ein
Studenten zum Zeitpunkt
der Klausur im Ausland
sind?

Algorithmen

Eingabe

Bearbeitung

Ausgabe

Beispiel Sortieren

Eingabe $\langle a_1, \dots, a_n \rangle$ Folge

Eingabe $\langle a_1, \dots, a_n \rangle$ Folge
von ganzen Zahlen.

(Umordnung)

Ausgabe Eine Permutation der

$$\langle a'_1, \dots, a'_n \rangle$$

mit

$$a'_1 \leq a'_2 \leq \dots \leq a'_n$$

For $j = 2$ to $A.length$

key $\leftarrow A[j]$

$i \leftarrow j - 1$

while $i > 0 \wedge A[i] > key$

$A[i+1] \leftarrow A[i]$

$i \leftarrow i - 1$

$A[i+1] \leftarrow key$

Korrekt?

Für jede (zulässige) Eingabe liefert Algorithmus die korrekte Ausgabe.

• Komplexität - wie aufwändig:

Laufzeit
Speicherplatz

Korrektheit Induktion

Man formuliert eine Schleifeninvariante. $j=2$

Zu Beginn der for-Schleife bestehet die Teilfolge $A[1, \dots, j-1]$ aus den entspr. Elementen der Originalfolge und ist sortiert. (Für alle j),

Letztes j : $A.length + 1$
für dieses j besagt die
Schleifeninvariante, dass
die Gesamtfolge sortiert
ist.

Induktionsbeweis

Induktionsanfang:

$$j = 2$$

Invariante: ..., Teilfolge $A[1]$

Induktionsdritt.

$$j \rightarrow j+1$$

Teilfolge $A[1, \dots, j-1]$ sortiert
vor Einstieg in Schleife.

In der Schleife wird $A[j]$
an die richtige Stelle
gesetzt.

Darum ist nach dem

Schleife auch $A[1, \dots, j]$
sortiert.

Wiederholung

Algorithmen

Pseudocode

Korrektheit - Invariante

Insertion Sort

Analyse der Zeitkomplexität

$t_j = \#$ Durchläufe der while-Schleife in Abh. von j .

Best Case : $t_j = 1$ falls Folge sortiert ist.

Worst case : $t_j = j$ falls Folge umgedreht sortiert

$$\sum_{j=2}^n t_j \quad \left\{ \begin{array}{l} \sum_{j=2}^n 1 = n-1 \quad \text{Best case} \\ \sum_{j=2}^n j = \end{array} \right. \quad \text{Worst case}$$

$$\sum_{j=1}^n = \frac{n(n+1)}{2}$$

$$\frac{n(n+1)}{2} - 1$$

Worst case

$$\begin{aligned}
 & n(c_1 + c_2 + c_4 + c_8) - c_2 - c_4 - c_8 \\
 & + \left(\frac{n(n+1)}{2} - 1 \right) (c_5 + c_6 + c_7) \\
 & \quad \underbrace{\qquad\qquad}_{= \frac{n^2 + n}{2} - 1} \quad -(n-1)(c_6 + c_7) \\
 & = \frac{n^2 + n}{2} - 1
 \end{aligned}$$

$$\begin{aligned}
 & n^2 \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2} \right) \\
 & + n \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8 \right) \\
 & - (c_2 + c_4 + c_5 + c_8) \quad \text{quadratisch}
 \end{aligned}$$

Best case : linear, optimal

$\Theta(\frac{n^2}{2})$ wurde auf Folie definiert
und "fast" gezeigt

$$\frac{1}{2} n^2 - 3n = \Theta(n^2)$$

$$6n^3 \stackrel{?}{\in} \Theta(n^2)$$

$$\lim_{n \rightarrow \infty} \frac{6n^3}{n^2} = \infty$$

$$n \rightarrow \infty \quad n^2$$

Angenommen, es gäbe ein
passendes c_2 , dann wäre

$$6n^3 \leq c_2 n^2 \quad \forall n \geq n_0$$

$$6n \leq c_2 \quad \forall n \geq n_0$$

falsch für hinreichend
großes n , nämlich $\left\{ \begin{array}{l} n \geq \frac{c_2}{6} \\ \text{für } \end{array} \right.$

$O(g)$ definiert auf Folie.

$$1000n^2 = O(n^2 \log n)$$

$$n^2 \log n = \underline{\Omega}(1000n^2)$$

$f \in O(g)$ - f wächst nicht schneller als g

$f \in \Omega(g)$ - f wächst nicht langsamer als g

$f \in o(g)$ - f wächst langsamer als g

* asymptotisch

Theorem ① $f \in \Theta(g) \Leftrightarrow f \in O(g) \wedge f \in \Omega(g)$

② $f \in O(g) \Leftrightarrow g \in \underline{\Omega}(f)$

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

die anderen Relationen analog

transitiv $O, \underline{\Omega}, \Theta, \circ^?$

reflexiv $O, \underline{\Omega}, \Theta$

Symmetrisch Θ

$\Theta(1)$ konstant

$\Theta(n)$ linear

$\Theta(n(\log n)^m)$ quasi linear

$\Theta(n^3)$ kubisch

$\Theta(n^k)$ polynomiell

$\Theta(2^n)$ exponentiell

Wiederholung

Insertion Sort

Beispiel für Korrektheitsbeweis und Laufzeitanalyse

Best Case: linear $O(n)$

Worst case: quadratisch $O(n^2)$

O, Ω, Θ, \circ

Rekurrenz - Divide and Conquer

Prinzip: Zerlege das Problem in Unterprobleme

Löse diese

- direkt

- oder durch weitere Zerlegung

Setze die Lösungen zusammen.

Beispiel: merge sort

Sortiere $A[1 \dots n]$

Zerlege in zwei Teilfolgen

Sortiere diese

Und setze zusammen

Merge².

Merge (A, p, q, r)

Eingabe } $p \leq q < r$

$A[p \dots q]$ sortiert

$A[q+1 \dots r]$ sortiert

Ausgabe $A[p \dots r]$ sortiert

Merge-Sort (A, p, r)

Eingabe $A[1, \dots, n]$

$p \leq r$

Ausgabe $A : A[p, \dots, r]$ sortiert

$$p = 3 \quad r = 6$$

$$q = \left\lfloor \frac{p+r}{2} \right\rfloor = \left\lfloor \frac{9}{2} \right\rfloor = 4$$

$$A[3, 4] \quad A[5, 6]$$

$$p = 3, \quad r = 7$$

$$q = \left\lfloor \frac{p+r}{2} \right\rfloor = \left\lfloor \frac{10}{2} \right\rfloor = 5$$

$$A[3, 4, 5] \quad A[6, 7]$$

Merge

Korrektheitsbeweis

Schleifeninvariante

Am Anfang der for-Schleife in Zeile 12 - 17 enthält

$A[p, \dots, k-1]$ sortiert, enthält die $k-p$ kleinsten Elemente aus L und R.

$L[i]$, $R[j]$ sind die kleinsten bis jetzt nicht kopierten Elemente in L und R.

Für $k = n+1$ folgt:

$A[p, \dots, n]$ sortiert und enthält die $n-p+1$ kleinsten Elemente aus L und R. Das sind aber alle Elemente aus L und R $\neq \infty$.
Daraus folgt die Korrektheit.

Induktion

Anfang $k=p$, $i=1$, $j=1$

$L[1]$ kleinstes El. in L ✓
weil L sortiert ist

$R[1]$... R R ✓

$$k=p, \quad A[p, k-1]$$
$$k-p=0$$

leere Folge
leere Aus. ✓
Schleifendurchl.

Schritt $k \rightarrow k+1$ von $k \rightarrow k+1$
Wir wissen auch, dass alle ein-
geordneten Elemente höchstens so
groß wie $L[i]$ bzw $R[j]$ sind.
Das liegt daran, dass L und
 R sortierte Folgen sind. Wenn
 $L[i] \leq R[j]$ ist wird $L[i]$
als $A[k]$ eingeordnet und i
um 1 erhöht. Also ist die
Invariante erfüllt. Andernfalls
analog.

Terminierung

Bei Terminierung ist $k=r+1$.
Dann ist die Beh. erfüllt

$A[p, \dots, r]$ sortiert.

Wie lange dauert Merge?
Beh Zeit $O(n)$ ✓

Merge Sort ?

1. Die Studienleistung
bleibt gültig

2. Bitte: alle, die sich
noch nicht in Übungen
eingetragen haben, dies
in Moodle selbst tun.

Danach: Tauschbörsen.

Systematische Analyse von rekursiven
Algorithmen

Merge Sort allgemein n

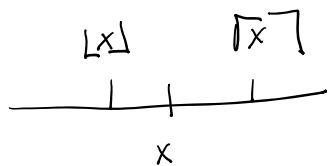
$T(n)$ Laufzeit von Merge Sort für
Folge der Längen n

$$T(1) \leq C$$

$$T(n) \leq T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + cn$$

$x \in \mathbb{R}$, $\lfloor x \rfloor$ die eindeutig bestimmte
ganze Zahl mit

$$x - 1 < \lfloor x \rfloor \leq x$$



$\lceil x \rceil$ die eindeutig bestimmte
ganze Zahl mit

$$x \leq \lceil x \rceil < x + 1$$

$$\lfloor 3,1 \rfloor = 3 \quad \lceil 3,1 \rceil = 4$$

$$\lfloor -4,1 \rfloor = -5 \quad \lceil -4,1 \rceil = -4$$

Ziel : Eine obere Abschätzung
 $T(n) \leq g(n)$, wo $g(n)$
eine Funktion ist, die durch
eine gesuchte Formel
dar gestellt werden kann.

Wir "raten" :

$$T(n) \leq 4c n \lg n \quad \lg = \log_2$$

Beweis durch Induktion

$$n=1 \quad T(1) \leq c \quad \checkmark$$

$$n=2 \quad T(2) \leq \underset{\uparrow}{T(1)} + \underset{\leq c}{T(1)} + 2c \leq c$$

Rekursionsformel

$$\leq 4c \leq 8c \quad \checkmark$$

$$4c n \lg n = 4cn = 8c$$

Induktionsschritt

Annahme: sei $n > 2$ und sei
die Behauptung wahr für alle

$$n' < n.$$

$$T(n) \leq T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + cn$$

$$\leq 4c \left\lfloor \frac{n}{2} \right\rfloor \log \left\lfloor \frac{n}{2} \right\rfloor + 4c \left\lceil \frac{n}{2} \right\rceil \log \left\lceil \frac{n}{2} \right\rceil$$

\curvearrowleft

Induktions
annahme

$$2 \cdot 1 + \left\lceil \frac{n}{2} \right\rceil = n$$

$$\leq \frac{n}{2} \leq \frac{2}{3}n + cn \leq \frac{2}{3}n$$

annunzieren

$$\begin{aligned}
 \underline{\text{Beh 1}} \quad & \lfloor \frac{n}{2} \rfloor + \lceil \frac{n}{2} \rceil = n \\
 \underline{\text{Beh 2}} \quad & \lfloor \frac{n}{2} \rfloor \leq \frac{n}{2} \quad \underline{\text{Beh 3}} \quad \lceil \frac{n}{2} \rceil \leq \frac{2}{3}n, n \geq 3
 \end{aligned}$$

$$\begin{aligned}
 & \leq 4c \log\left(\frac{2n}{3}\right) \left(\lfloor \frac{n}{2} \rfloor + \lceil \frac{n}{2} \rceil \right) \\
 & \leq 4c \log\left(\frac{2}{3}n\right) \cdot n + c \cdot n \\
 & (\log a \cdot b = \log a + \log b) \\
 & \leq 4cn \log n + cn \left(1 - 4 \log \frac{3}{2}\right) \\
 & = 4 \log \frac{2}{3} \\
 & \leq 4cn \log n. \quad \blacksquare \quad \Leftarrow 0
 \end{aligned}$$

Master Theorem erlaubt eine syst.

Ab Schätzung

$$1) T(n) = \frac{9}{3} T(n/3) + n$$

$\log_b a = \log_3 9 = 2$ $f(n)$
 $a = 9$
 $b = 3$
 $f(n) = n$

Fall 1 $n^{2-\varepsilon}$

Fall 2 n^2

Fall 3 $n^2 + \varepsilon$

Fall 1 $f(n) = O(n^{2-\varepsilon})$ wahr für $\varepsilon = 1$.

Master Theorem

$$\Rightarrow T(n) = \Theta(n^2)$$

$$2) T(n) = T(2n/3) + \underbrace{1}_{\begin{array}{l} a=1 \\ b=3/2 \\ f(n)=1 \end{array}}$$

$$\log_{3/2} 1 = 0$$

$f(n)$ mit $n^0 = 1$ vergleichen

$$\underline{\text{Fall 2}} \quad f(n) = \Theta(\underbrace{n^{\log_b a}}_1)$$

$$T(n) = \Theta(\underbrace{n^{\log_b a} \log n}_1) = \Theta(\log n)$$

$$3) T(n) = 3T(n/4) + n \log n$$

$$a=3, b=4, f(n) = n \log n$$

$$f(n) \quad n^{\log_b a} = n^{\log_4 3} \\ \leq n^{0,793}$$

$$\underline{\text{Fall 3}} \quad f(n) = n \log n \quad \forall n \geq n^{0,793+0,1}$$

$$a f(n/b) \leq c f(n)$$

$$3 f(n/4) = 3(n/4) \log(n/4) \\ = \frac{3}{4} n \log \frac{n}{4} \leq \frac{3}{4} n \log n$$

$$\text{Summt f\"ur } c = \frac{3}{4} \cdot \checkmark$$

Es folgt aus dem Master Theorem: $T(n) = \Theta(n \log n)$.

$$4) \quad T(n) = 2 T(n/2) + n \log n$$

$$a=2, b=2, f(n) = n \log n$$

Vergleiche $f(n)$ mit $n^{\frac{\log_b a}{\log b}} = n$

Gibt es ein $\varepsilon > 0$ mit:

$$n \log n = \Omega(n^{1+\varepsilon})$$

nein.

$$\lim_{n \rightarrow \infty} \frac{n^{1+\varepsilon}}{n \log n} = \infty$$

Daher ist keiner der Fälle relevant.