Grundzüge der Informatik 1

Vorlesung 6

Überblick

Überblick

- Teile & Herrsche Verfahren
- MergeSort
  - Wiederholung
  - Auflösen von Laufzeitrekursionen
- Binäre Suche
  - Idee durch Teile & Herrsche
  - Pseudocode
  - Laufzeitanalyse
  - Korrektheit

Algorithmenentwurf durch Rekursion

Teile & Herrsche Verfahren

- Idee: Teile die Eingabe in ~~mehrere~~ gleich große Teile auf
- Löse das Problem rekursiv auf den einzelnen Teilen
- Füge die Teile zu einer Lösung des Gesamtproblems zusammen

- Beispiel: Sortieren durch Aufteilen in zwei Teile

15	7	6	13	25	4	9	12

Schritt 1: Aufteilen der Eingabe

Algorithmenentwurf durch Rekursion

Teile & Herrsche Verfahren

- Idee: Teile die Eingabe in mehrere gleich große Teile auf
- Löse das Problem rekursiv auf den einzelnen Teilen
- Füge die Teile zu einer Lösung des Gesamtproblems zusammen

- Beispiel: Sortieren durch Aufteilen in zwei Teile

Schritt 1: Aufteilen der Eingabe

15	7	6	13	25	4	9	12

15	7	6	13

25	4	9	12

Algorithmenentwurf durch Rekursion

Teile & Herrsche Verfahren

- Idee: Teile die Eingabe in mehrere gleich große Teile auf
- Löse das Problem rekursiv auf den einzelnen Teilen
- Füge die Teile zu einer Lösung des Gesamtproblems zusammen

- Beispiel: Sortieren durch Aufteilen in zwei Teile

Schritt 2:
Rekursiv Sortieren

15	7	6	13	25	4	9	12

6	7	13	15

4	9	12	25

Algorithmenentwurf durch Rekursion

Teile & Herrsche Verfahren

- Idee: Teile die Eingabe in mehrere gleich große Teile auf
- Löse das Problem rekursiv auf den einzelnen Teilen
- Füge die Teile zu einer Lösung des Gesamtproblems zusammen

- Beispiel: Sortieren durch Aufteilen in zwei Teile

Schritt 3: Zusammenfügen

4	6	7	9	12	13	15	25

6	7	13	15

4	9	12	25

Algorithmenentwurf durch Rekursion

Teile & Herrsche Verfahren

- Idee: Teile die Eingabe in mehrere gleich große Teile auf
- Löse das Problem rekursiv auf den einzelnen Teilen
- Füge die Teile zu einer Lösung des Gesamtproblems zusammen

Wichtig

- Wir benötigen Rekursionsabbruch
- Beim Sortieren: Folgen der Länge 1

Laufzeitanalyse
- MergeSort

```
MergeSort(A,p,r)                    \* Sortiert A[p..r]
if p<r then
   q=⌊(p+r)/2⌋
   MergeSort(A,p,q)
   MergeSort(A,q+1,r)
   Merge(A,p,q,r)
```

Aufruf des Algorithmus

- MergeSort(A,1,r), wobei r die Länge des Feldes A ist
- Sei T(n) die Worst-Case Laufzeit von MergeSort, um ein Teilarray der Größe n=r-p+1 zu sortieren

Laufzeitanalyse - MergeSort

Laufzeit als Rekursion (n Zweierpotenz)

- $$T(n) \leq \begin{cases} 1 & \text{, falls } n=1 \\ 2\,T(n/2) + cn & \text{, falls } n>1 \end{cases}$$

- Wobei c eine genügend große Konstante ist.

Laufzeitanalyse
- MergeSort

Laufzeit als Rekursion (n Zweierpotenz)

- $T(n) \leq \begin{cases} 1 & \text{, falls } n=1 \\ 2\,T(n/2) + cn & \text{, falls } n>1 \end{cases}$

- Wobei c eine genügend große Konstante ist.

Vereinfachung

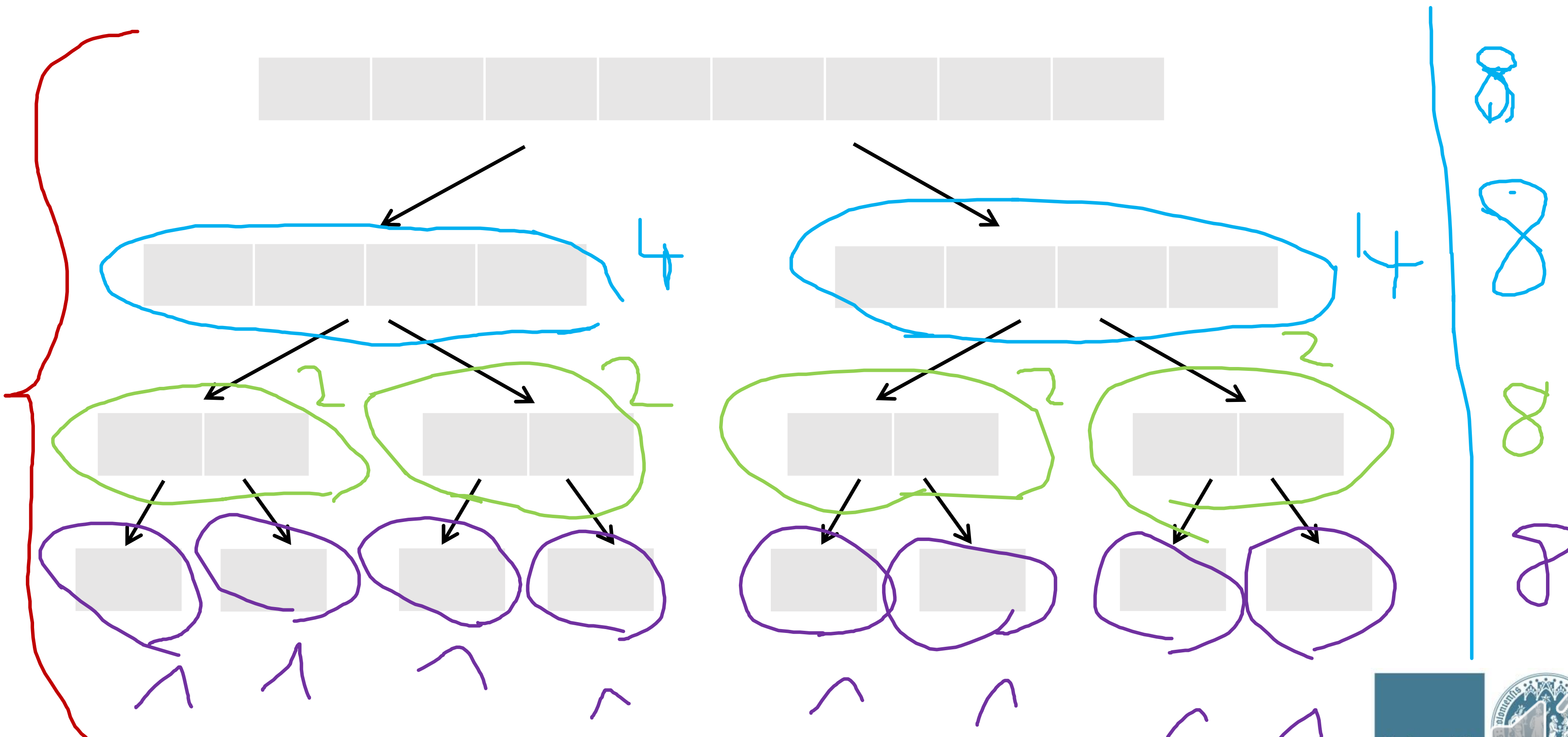
- c=1

Laufzeitanalyse - MergeSort

Rekursionsgleichung

- $T(n) = 2T(n/2)+n$
- $T(1) = 1$

Rekursionsbaum für n=8:



Laufzeitanalyse
- MergeSort

Auflösen der Rekursionsgleichung (Annahme: n ist Zweierpotenz)

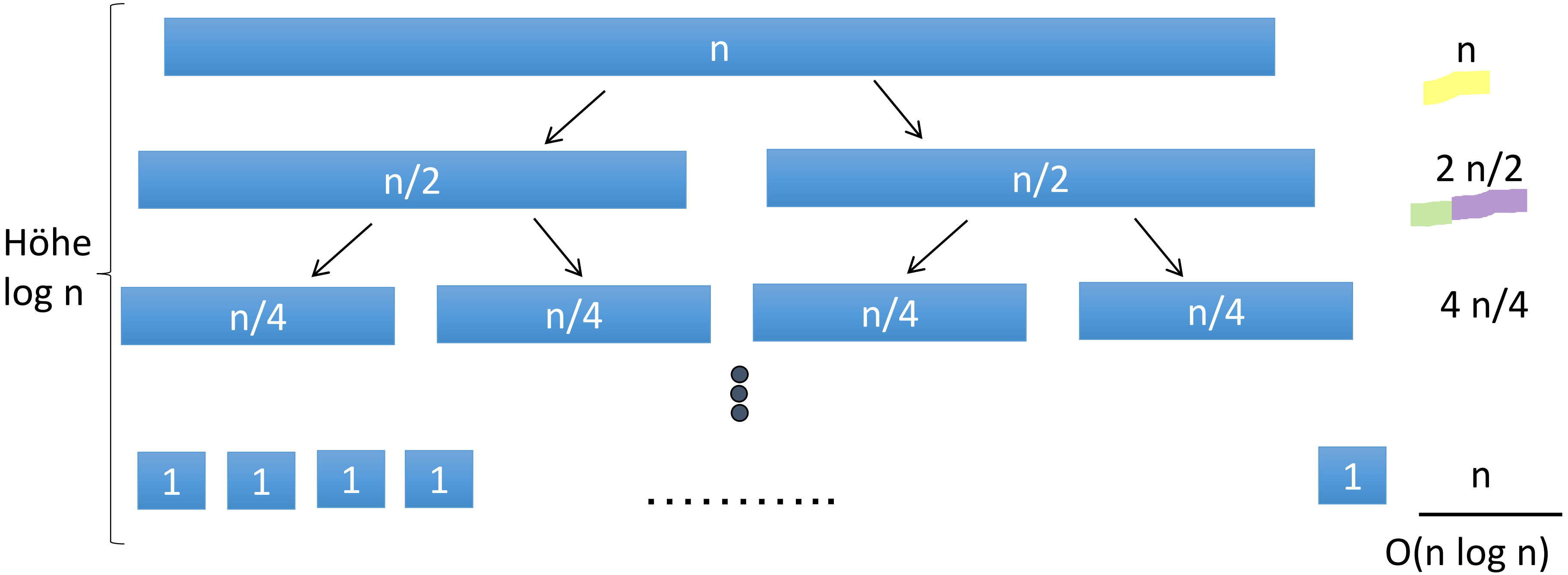
- $T(n) = 2\,T(n/2) + n$
- $2T(n/2) = 2\,(2\,T(n/4) + n/2) = 4\,T(n/4) + n$
- $4T(n/4) = 4\,(2\,T(n/8) + n/4) = 8\,T(n/8) + n$
- …
- $nT(n/n) = n\,T(1) = n$

Zusammenfügen

- Mit jeder Rekursionsstufe kommt Aufwand n hinzu
- Da n in jeder Rekursionsstufe halbiert wird, gibt es genau log n Rekursionsstufen
- Damit ist $T(n) = n \log n$

Laufzeitanalyse – grafische Darstellung - MergeSort

- Auflösen von $T(n) \leq 2\,T(n/2) + n$ (Intuition)
- $T(1) = 1$



Laufzeitanalyse
- MergeSort

Satz 6.1

- Algorithmus MergeSort(A, 1, n) hat eine Laufzeit von $O(n \log n)$.

Beweis (Teil 1)

- Sei n eine Zweierpotenz
- Wir haben bereits gezeigt, dass für eine hinreichen große Konstante $c>1$ die Laufzeit von MergeSort durch die Rekursion
- $T(n) \le 2\, T(n/2) + cn$ und
- $T(1)=1$ abgeschätzt werden kann
- Wir zeigen per Induktion: $T(n) \le 2cn \log n$ für $n \ge 2$

Laufzeitanalyse
- MergeSort

Satz 6.1

- Algorithmus MergeSort(A, 1, n) hat eine Laufzeit von O(n log n).

Beweis (Teil 2)

- Induktionsanfang (n=2):
- Die Laufzeit des Algorithmus für n=2 ist
- $T(n) = T(2) \leq 2\ T(1) + 2c = 2 + 2c \leq 4c = 2cn$ ✓

Universität zu Köln

Laufzeitanalyse
- MergeSort

Satz 6.1

- Algorithmus MergeSort(A, 1, n) hat eine Laufzeit von $O(n \log n)$.

Beweis (Teil 3)

- Induktionsannahme: Für $2 \le m < n$, m Zweierpotenz, gilt $T(m) \le 2c\, m \log m$
- Induktionsschluss:
- Sei $n>2$ eine Zweierpotenz
- Nach Induktionsannahme gilt $T(n) = 2\,T(n/2) + cn \le 2\,(2c\, n/2 \log(n/2)) + cn$
- $= 2cn\,(\log n - 1) + cn \le 2c\, n \log n$
- Nach dem Induktionsprinzip folgt $T(n) \le 2cn\log n = O(n \log n)$

Laufzeitanalyse
- MergeSort

Laufzeitvergleich

Eingabegröße n	2	16	256	4 096	65 536
n log n	2	64	2 048	49 152	1 048 576
n^2	4	256	65 536	16 777 216	4 294 967 296

Teile & Herrsche Prinzip

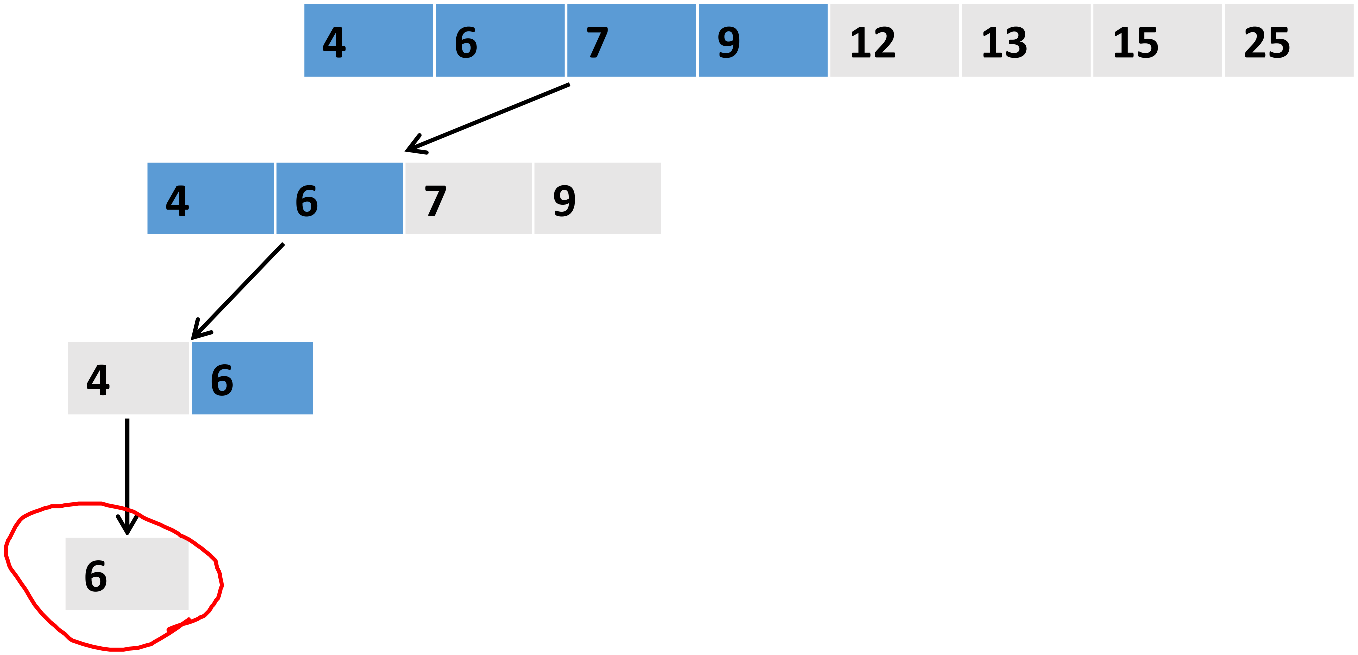
Suche in sortierten Felder

- „Telefonbuchproblem“: Wie können wir in einem sortierte Feld eine bestimmte Zahl finden?
- Die Zahl dient als „Schlüssel“ und mit ihr können weitere Informationen abgespeichert sein
- Verwende Teile & Herrsche Ansatz

Teile & Herrsche Prinzip

Teile & Herrsche Ansatz (hier: Suche nach 6):

- Teile in der Mitte und suche rekursiv, in dem eindeutigen Teil, der das gesuchte Element enthalten kann



Teile & Herrsche Algorithmen

```
BinäreSuche(A,x,p,r)                  \* Finde Zahl x in sortiertem Feld A[p..r]
1.  if p=r then return p              \* sofern vorhanden
2.  else                              \* Ausgabe: Index der gesuchten Zahl
3.      q = ⌊(p+r)/2⌋
4.      if x ≤ A[q] then return BinäreSuche(A,x,p,q)
5.      else return BinäreSuche(A,x,q+1,r)
```

Teile & Herrsche Algorithmen

```
BinäreSuche(A,x,p,r)                       \* Finde Zahl x in sortiertem Feld A[p..r]
    if p=r then return p          O(1)      \* sofern vorhanden
    else                                    \* Ausgabe: Index der gesuchten Zahl
        q = ⌊(p+r)/2⌋             O(1)
        if x ≤ A[q] then return BinäreSuche(A,x,p,q)
        else return BinäreSuche(A,x,q+1,r)
```

$O(1) + T(n/2)$

$T(n/2) + O(1)$

Laufzeitrekursion

- Sei n= p-r+1 die Größe des zu durchsuchenden Bereichs und sei n eine Zweierpotenz
- $T(n) \leq T(n/2) + c$, wobei c eine hinreichend große Konstante ist
- $T(1) = c$

Laufzeitanalyse
- Binäre Suche

Auflösen der Rekursionsgleichung (Annahme: n ist Zweierpotenz)

- $T(n) \quad = T(n/2) + c$
- $T(n/2) = T(n/4) + c$
- $T(n/4) = T(n/8) + c$
- …
- $T(n/n)= T(1) = c$

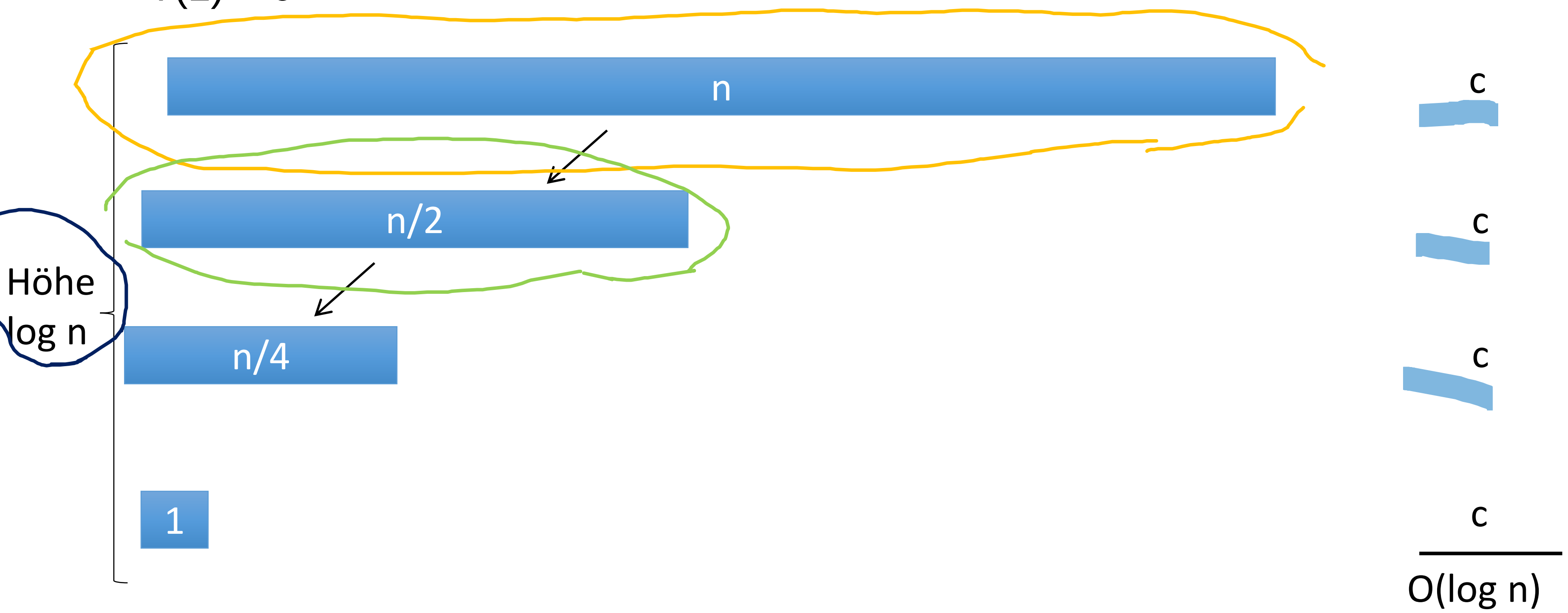
Zusammenfügen

- Mit jeder Rekursionsstufe kommt Aufwand c hinzu
- Da n in jeder Rekursionsstufe halbiert wird, gibt es genau log n Rekursionsstufen
- Damit ist $T(n) = O(\log n)$

Universität zu Köln

Laufzeitanalyse – grafische Darstellung - Binäre Suche

- Auflösen von $T(n) \leq T(n/2) + c$ (Intuition)
- $T(1) = c$



Laufzeitanalyse
- Binäre Suche

Satz 6.2

- Die Laufzeit von BinäreSuche(A,x,p,r) ist O(log n), wobei n= r-p+1 die Größe des zu durchsuchenden Bereichs ist.

Beweis (Teil 1)

- Wir nehmen an, dass n eine Zweierpotenz ist
- Die Rekursionsgleichung für die Laufzeit von BinäreSuche ist
- $T(n) \leq T(n/2) + c$ und $T(1) = c$
- Wir zeigen, dass $T(n) \leq 2c \log n$ ist für $n \geq 2$
- Induktionsanfang (n=2):
- Die Laufzeit von BinäreSuche für n=2 ist $T(n) = \underbrace{T(2) \leq T(1) + c} = c + c$
- $= 2c = 2c \log 2$

Universität zu Köln

Laufzeitanalyse
- Binäre Suche

Satz 6.2

- Die Laufzeit von BinäreSuche(A,x,p,r) ist $O(\log n)$, wobei $n= r-p+1$ die Größe des zu durchsuchenden Bereichs ist.

Beweis (Teil 2)

- Induktionsannahme: Es gelte für eine beliebige Zweierpotenz m mit $2 \le m < n$, dass $T(m) \le 2c \log m$
- Induktionsschluss:
- Aufgrund der Induktionsannahme gilt
- $T(n) \le T(n/2) + c \le 2c \log(n/2) + c = 2c (\log n - 1) + c \le 2c \log n$ ✓
- Nach dem Induktionsprinzip folgt somit $T(n) \le 2c \log n = O(\log n)$

Laufzeitanalyse
- Binäre Suche

Binäre Suche vs. Lineare Suche

Eingabegröße n	2	256	65 536	16 777 216	4 294 967 296
n	2	256	65 536	16 777 216	4 294 967 296
log n	1	8	16	24	32

Laufzeitanalyse
- Binäre Suche

Satz 6.3

- Algorithmus BinäreSuche(A,x,p,r) findet den Index einer Zahl x in einem sortierten Feld A[p..r], sofern x in A[p..r] vorhanden ist.

Beweis (Teil 1):

- Wir zeigen die Korrektheit per Induktion über n=r-p+1. Wir nehmen an, dass x in A[p..r] ist, da es sonst nichts zu zeigen gibt.
- Induktionsanfang: Für n=1, d.h. p=r, gibt der Algorithmus p zurück. Dies ist der korrekte (weil einzige) Index.
- Induktionsannahme: Für alle r,p mit m=r-p+1 und 1≤m≤n findet BinäreSuche(A,x,p,r) den Index einer Zahl x in einem sortierten Feld A[p..r], sofern x im Feld vorhanden ist.

Laufzeitanalyse
- Binäre Suche

Satz 6.3

- Algorithmus BinäreSuche(A,x,p,r) findet den Index einer Zahl x in einem sortierten Feld A[p..r], sofern x in A[p..r] vorhanden ist.

Beweis (Teil 2):

- Induktionsschluss: Wir betrachten den Aufruf von BinäreSuche für beliebige p, r mit $n+1 = r-p+1>1$.
- Da $n+1>1$ folgt $p<r$ und der Algorithmus führt den **else**-Fall aus. Dort wird q auf $\lfloor (p+r)/2 \rfloor$ gesetzt.
- Es gilt $q\geq p$ und $q<r$. Ist $x\leq A[q]$, so wird BinäreSuche rekursiv für A[p..q] aufgerufen.
- Da A[p..r] sortiert ist, liegt x in A[p..q].
- Damit folgt aus der Induktionsannahme, dass der Index von x gefunden wird.

Laufzeitanalyse
- Binäre Suche

Satz 6.3

- Algorithmus BinäreSuche(A,x,p,r) findet den Index einer Zahl x in einem sortierten Feld A[p..r], sofern x in A[p..r] vorhanden ist.

Beweis (Teil 3):

- Ist x>A[q], so wird BinäreSuche rekursiv für A[q+1..r] aufgerufen.
- Da A[p..r] sortiert ist, liegt x in A[q+1..r].
- Damit folgt aus der Induktionsannahme, dass der Index von x gefunden wird.
- Somit wird x in allen Fällen gefunden und der Satz folgt.

Teile & Herrsche Algorithmen

Wodurch unterscheiden sich Teile & Herrsche Algorithmen?

- Die Anzahl der Teilprobleme
- Die Größe der Teilprobleme
- Den Algorithmus für das Zusammensetzen der Teilprobleme
- Den Rekursionsabbruch

Wann lohnt sich Teile & Herrsche?

- Kann durch Laufzeitanalyse vorhergesagt werden

Laufzeitanalyse
- Rekursionen

Laufzeiten der Form

$$T(n) = a\,T(n/b) + f(n)$$

(und $T(1) = O(1)$)

Laufzeitanalyse
- Rekursionen

Laufzeiten der Form

$$T(n) = a\,T(n/b) + f(n)$$

(und $T(1) = O(1)$)

Laufzeitanalyse
- Rekursionen

Laufzeiten der Form

$$T(n) = a\,T(n/b) + f(n)$$

Anzahl Unterprobleme

(und $T(1) = O(1)$)

Laufzeitanalyse
- Rekursionen

Laufzeiten der Form

$$T(n) = a\,T(n/b) + f(n)$$

Anzahl Unterprobleme

(und $T(1) = O(1)$)

Größe der Unterprobleme
(bestimmt Höhe des Rekursionsbaums)

Laufzeitanalyse
- Rekursionen

Laufzeiten der Form

$$T(n) = a\,T(n/b) + f(n)$$

Anzahl Unterprobleme

Aufwand für Aufteilen und Zusammenfügen

Größe der Unterprobleme
(bestimmt Höhe des Rekursionsbaums)

(und $T(1) = O(1)$)

Zusammenfassung

Zusammenfassung

- Teile & Herrsche Verfahren
- MergeSort
  - Auflösen von Laufzeitrekursionen
- Binäre Suche
  - Idee durch Teile & Herrsche
  - Pseudocode
  - Laufzeitanalyse
  - Korrektheit

Referenzen

- T. Cormen, C. Leiserson, R. Rivest, C. Stein. Introduction to Algorithms. The MIT press. Second edition, 2001.