



# *Algorithmen und Datenstrukturen* *Teile und Herrsche* *und Rekursionsgleichungen*

---

Matthias Teschner  
Graphische Datenverarbeitung  
Institut für Informatik  
Universität Freiburg

SS 09

# *Lernziele der Vorlesung*



- Algorithmen
  - Sortieren, Suchen, Optimieren
- Datenstrukturen
  - Repräsentation von Daten
  - Listen, Stapel, Schlangen, Bäume
- Techniken zum Entwurf von Algorithmen
  - Algorithmenmuster
  - Greedy, Backtracking, Divide-and-Conquer
- Analyse von Algorithmen
  - Korrektheit, Effizienz

# Überblick



- Prinzip
- Maximale Teilsumme
  - erste Lösungsansätze
  - Teile-und-Herrsche-Ansatz
  - Laufzeit
- Rekursionsgleichungen
  - Substitutionsmethode
  - Rekursionsbaum-Methode
  - Mastertheorem
- Produkt von Polynomen nach Karatsuba
- Matrixmultiplikation nach Strassen
- Erstellung eines Spielplans

# Prinzip



- **Teile** das Gesamtproblem in kleinere Teilprobleme auf.
- **Herrsche** über die Teilprobleme durch rekursives Lösen.  
Wenn die Teilprobleme klein sind, löse sie direkt.
- **Verbinde** die Lösungen der Teilprobleme zur Lösung des Gesamtproblems.
  
- **rekursive** Anwendung des Algorithmus auf immer kleiner werdende Teilprobleme
- **direkte** Lösung eines hinreichend kleinen Teilproblems

# Prinzip



- Methode  $V$  zur Lösung des Problems  $P$  der Größe  $n$
- Methode  $V(P)$ 
  - wenn  $n < d$
  - dann
    - löse das Problem  $P$  direkt;
  - sonst
    - teile  $P$  in mehrere Teilprobleme  $P_1, P_2, \dots, P_k$  mit  $k \geq 2$ ;
    - Methode  $V(P_1)$ ; ...; Methode  $V(P_k)$ ;
    - verbinde die Teillösungen für  $P_1, P_2, \dots, P_k$  zur Gesamtlösung für  $P$ ;

# *Eigenschaften*



- kann bei konzeptionell schwierigen Problemen hilfreich sein
  - Lösung für den Trivialfall muss bekannt sein
  - Aufteilung in Teilprobleme muss möglich sein
  - Kombination der Teillösungen muss möglich sein
- kann effiziente Lösungen realisieren
  - wenn die Lösung des Trivialfalls in  $O(1)$  liegt, Aufteilung in Teilprobleme und Kombination der Teillösungen in  $O(n)$  liegt und die Zahl der Teilprobleme beschränkt ist, liegt der Algorithmus in  $O(n \log n)$
- für Parallelverarbeitung geeignet
  - Teilprobleme werden unabhängig voneinander verarbeitet

# Implementierung



- Definition des Trivialfalls
  - Möglichst kleine direkt zu lösende Teilprobleme sind elegant und einfach.
  - Andererseits wird die Effizienz verbessert, wenn schon relativ große Teilprobleme direkt gelöst werden.
  - Rekursionstiefe sollte nicht zu groß werden.
- Aufteilung in Teilprobleme
  - Wahl der Anzahl der Teilprobleme und konkrete Aufteilung kann anspruchsvoll sein.
- Kombination der Teillösungen
  - typischerweise konzeptionell anspruchsvoll

# Überblick



- Prinzip
- Maximale Teilsumme
  - erste Lösungsansätze
  - Teile-und-Herrsche-Ansatz
  - Laufzeit
- Rekursionsgleichungen
  - Substitutionsmethode
  - Rekursionsbaum-Methode
  - Mastertheorem
- Produkt von Polynomen nach Karatsuba
- Matrixmultiplikation nach Strassen
- Erstellung eines Spielplans

# Maximale Teilsumme



- Eingabe: Folge X von n ganzen Zahlen
- Ausgabe: Maximale Summe einer zusammenhängenden Teilfolge von X und deren Index-Grenzen

- Eingabe: 

Index	0	1	2	3	4	5	6	7	8	9
Wert	31	-41	59	26	-53	58	97	-93	-23	84

- Ausgabe: 187, 2, 6

# Anwendung



- Maximal möglicher Gewinn bei Kauf und Verkauf einer Aktie



Deutsche Bank  
+156% März 2009 – Mai 2009

Commerzbank  
+164% März 2009 – Mai 2009

Hypo Real Estate  
+135% März 2009 – Mai 2009

[www.comdirect.de](http://www.comdirect.de)

Deutsche Telekom, 1997 - 2009

# Naive Lösung (Brute Force)



```
maxSubArray(X)
result.sum=X(0); result.u=0; result.o=0;
  for u=0 to länge(X)-1 do           unterer, erster Index des Subarrays
    for o=u to länge(X)-1 do       oberer, letzter Index des Subarrays
      begin
        summe=0;
        for i=u to o do summe=summe+X(i);
        if result.sum<summe then
          begin
            result.sum = summe;
            result.u = u; result.o = o;
          end;
        end;
      end;
    end;
  return result.sum, result.u, result.o;
```

# Laufzeit



```
for u=0 to länge(X)-1 do
  for o=u to länge(X)-1 do
    begin
      summe=0;
      for i=u to o do summe=summe+X(i);
      if result.sum<summe then
        begin
          result.sum = summe;
          result.u = u; result.o = o;
        end;
      end;
    end;
```

n Schleifendurchläufe  $\rightarrow O(n)$

maximal n Schleifendurchläufe  $\rightarrow O(n)$

maximal n  
Schleifendurchläufe  
 $\rightarrow O(n)$

$O(1)$

- drei verschachtelte Schleifen  $O(n) \rightarrow O(n^3)$

# *Laufzeit*



u	Additionen	o
X		
n/3	n/3	n/3

- Es gibt wenigstens  $n/3$  Werte für  $u$ , für die  $n/3$  Werte von  $o$  durchlaufen werden, für die wiederum  $n/3$  Additionsschritte in der innersten Schleife durchlaufen werden.
- Damit werden mindestens  $T(n) = (n/3)^3 \in \Omega(n^3)$  Schritte benötigt.
- Aus  $T(n) \in O(n^3)$  und  $T(n) \in \Omega(n^3)$  folgt  $T(n) \in \Theta(n^3)$
- Es ist schwer, das Problem schlechter zu lösen...

# Effizientere Alternative



... aber leicht, es besser zu lösen.

- Bisher: Für jedes  $u$  und für jedes  $o$  wird  $S_{u,o} = X(u) + X(u + 1) + \dots + X(o)$  berechnet

- Effizienter:  
Inkrementelle Aktualisierung statt Schleifendurchlauf

aus  $S_{u,o} = X(u) + X(u + 1) + \dots + X(o)$

und  $S_{u,o+1} = X(u) + X(u + 1) + \dots + X(o) + X(o + 1)$

folgt  $S_{u,o+1} = S_{u,o} + X(o + 1)$   $O(1)$  statt  $O(n)$

# Bessere Lösung



```
maxSubArray(X)
result.sum=X(0); result.u=0; result.o=0;
  for u=0 to länge(X)-1 do
    begin
      summe=0;
      for o=u to länge(X)-1 do
        begin
          summe=summe+X(o);
          if result.sum<summe then
            begin
              result.sum = summe;
              result.u = u; result.o = o;
            end;
          end;
        end;
      end;
    end;
  return result.sum, result.u, result.o;  Gesamtaufwand O(n2)
```

# Überblick

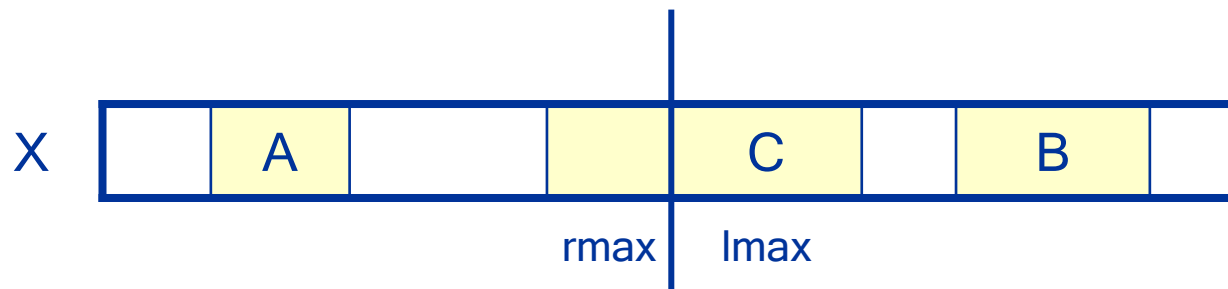


- Prinzip
- Maximale Teilsumme
  - erste Lösungsansätze
  - Teile-und-Herrsche-Ansatz
  - Laufzeit
- Rekursionsgleichungen
  - Substitutionsmethode
  - Rekursionsbaum-Methode
  - Mastertheorem
- Produkt von Polynomen nach Karatsuba
- Matrixmultiplikation nach Strassen
- Erstellung eines Spielplans

# Idee



- Löse das Problem für die linke und rechte Hälfte von X.
- Setze die Teillösungen zur Gesamtlösung zusammen.

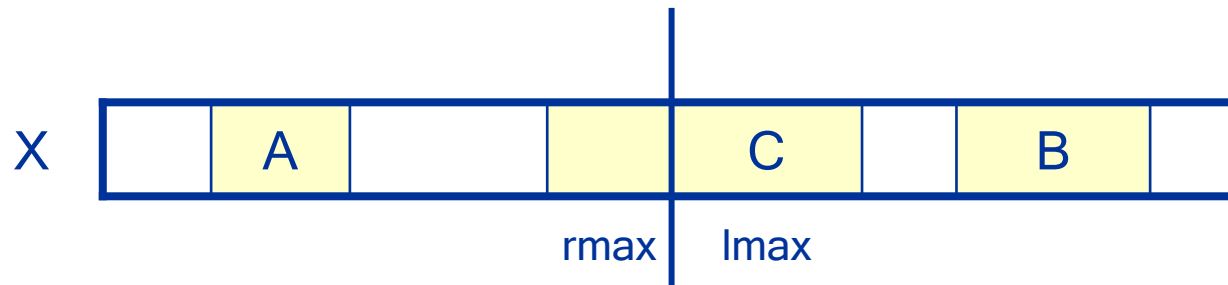


- Maximum liegt entweder in der linken Hälfte (A) oder in der rechten Hälfte (B)
- Maximum kann auch an der Grenze der Hälften liegen (C).
- Für C müssen rmax und lmax bestimmt werden.
- Gesamtlösung ist Maximum von A, B, C.

# Prinzip



- Kleine Probleme werden direkt gelöst:  $n=1 \Rightarrow \text{Max} = X(0)$
- Große Probleme werden in zwei Teilprobleme zerlegt und rekursiv gelöst. Teillösungen A und B werden geliefert.



- Um Teillösung C zu ermitteln, werden rmax und lmax für die Teilprobleme bestimmt.
- Die Gesamtlösung ergibt sich als das Maximum von A, B, C.

# Implementierung



`maxSubArray(X, u, o)` X sollte eine Referenz / ein Pointer sein.

```
if (u == o) then return X(u);
```

Trivialfall,  
Feld mit nur einem Element  
(Rekursionsabbruch)

```
m = (u+o) / 2;
```

```
A = maxSubArray (X, u, m);
```

```
B = maxSubArray (X, m+1, o);
```

Lösungen A und B für  
die beiden Teilfelder  
(Rekursion)

```
C1 = rmax (X, u, m);
```

```
C2 = lmax (X, m+1, o);
```

rmax und lmax für  
den Grenzfall C

```
return max (A, B, C1+C2);
```

Lösung ergibt sich als  
Maximum aus A, B, C

Für mehr Übersichtlichkeit wird nur das Maximum, nicht die Indexgrenzen berücksichtigt.

# Alternativer Trivialfall



```
maxSubArray(X, u, o)
```

```
if (u = o) then return X(u);  
if (u+1=o) then  
    return max(X(u), X(o), X(u)+X(o));
```

```
m = (u+o)/2;
```

```
A = maxSubArray (X, u, m);
```

```
B = maxSubArray (X, m+1, o);
```

```
C1 = rmax (X, u, m);
```

```
C2 = lmax (X, m+1, o);
```

```
return max (A, B, C1+C2);
```

# Implementierung - max



```
max (a, b, c)
```

```
if (a > b) then  
    if (a > c) then  
        return a;  
    else  
        return c;  
else  
    if (c > b) then  
        return c;  
    else  
        return b;
```

# *Alternative Implementierung max*



```
max (a, b)
```

```
    if (a > b) then  
        return a;  
    else  
        return b;
```

```
max (a, b, c)
```

```
    return max (max(a, b), c);
```

# Implementierung - lmax



```
lmax (X, u, o)
```

```
    lmax = X(u);
```

```
    summe = X(u);
```

```
    for i=u+1 to o do
```

```
        begin
```

```
            summe = summe + X(i);
```

```
            if (summe > lmax) lmax = summe;
```

```
        end
```

```
    return lmax;
```

rmax analog

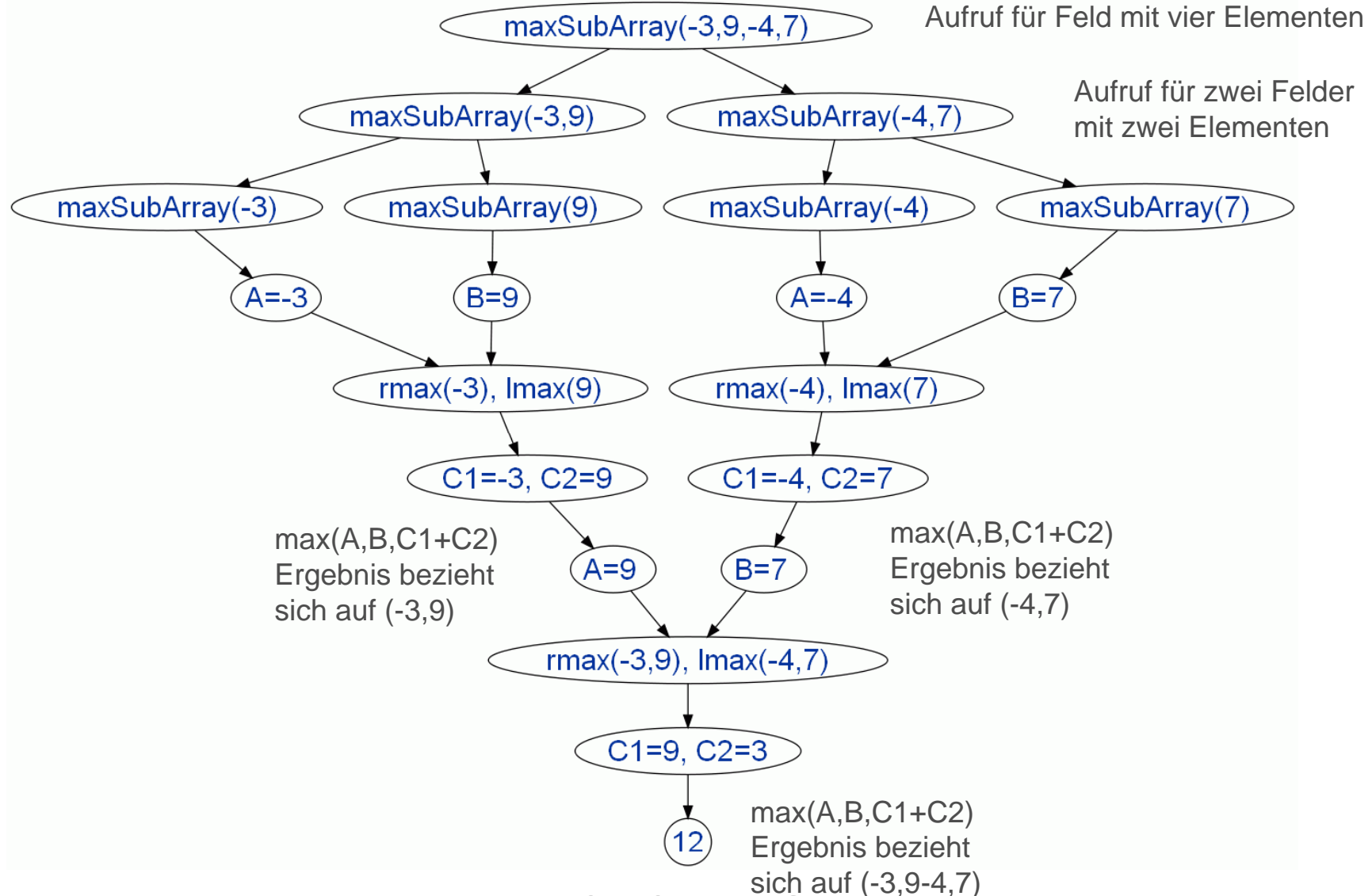
# Illustration - I<sub>max</sub>



i	u	u+1	...	...	o-1	o
X	58	-53	26	59	-41	31
summe	58	5	31	90	49	80
I <sub>max</sub>	58	58	58	90	90	90

- I<sub>max</sub> und summe werden mit X(u) initialisiert
- X wird durchlaufen von u bis o
- summe wird aktualisiert
- I<sub>max</sub> wird aktualisiert, wenn summe > I<sub>max</sub>

# Illustration - maxSubArray



# Überblick



- Prinzip
- Maximale Teilsumme
  - erste Lösungsansätze
  - Teile-und-Herrsche-Ansatz
  - Laufzeit
- Rekursionsgleichungen
  - Substitutionsmethode
  - Rekursionsbaum-Methode
  - Mastertheorem
- Produkt von Polynomen nach Karatsuba
- Matrixmultiplikation nach Strassen
- Erstellung eines Spielplans

# Laufzeit



```
maxSubArray(X, u, o)
```

$T(n)$  – Zahl der Schritte  
für Problemgröße  $n$

```
    if (u = o) then return X(u);
```

$O(1)$

```
    m = (u+o)/2;
```

$O(1)$

```
    A = maxSubArray (X, u, m);
```

$T(n/2)$

```
    B = maxSubArray (X, m+1, o);
```

$T(n/2)$

```
    C1 = rmax (X, u, m);
```

$O(n)$

```
    C2 = lmax (X, m+1, o);
```

$O(n)$

```
    return max (A, B, C1+C2);
```

$O(1)$

# Zahl der Schritte $T(n)$



$$T(n) = \begin{cases} \Theta(1) & n = 1 \\ 2 \cdot T\left(\frac{n}{2}\right) + \Theta(n) & n > 1 \end{cases}$$

Trivialfall

Lösung der Teilprobleme      Verbinden der Teillösungen

Rekursionsgleichung

- also existieren Konstanten  $a$  und  $b$  mit

$$T(n) \leq \begin{cases} a & n = 1 \\ 2 \cdot T\left(\frac{n}{2}\right) + b \cdot n & n > 1 \end{cases}$$

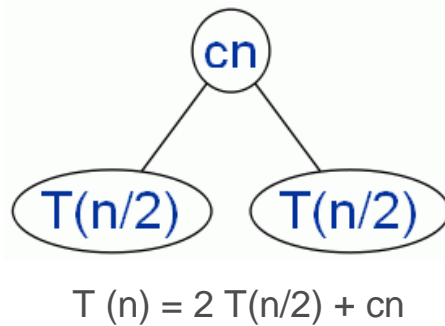
- für  $c = \max(a, b)$  gilt dann

$$T(n) \leq \begin{cases} c & n = 1 \\ 2 \cdot T\left(\frac{n}{2}\right) + c \cdot n & n > 1 \end{cases}$$

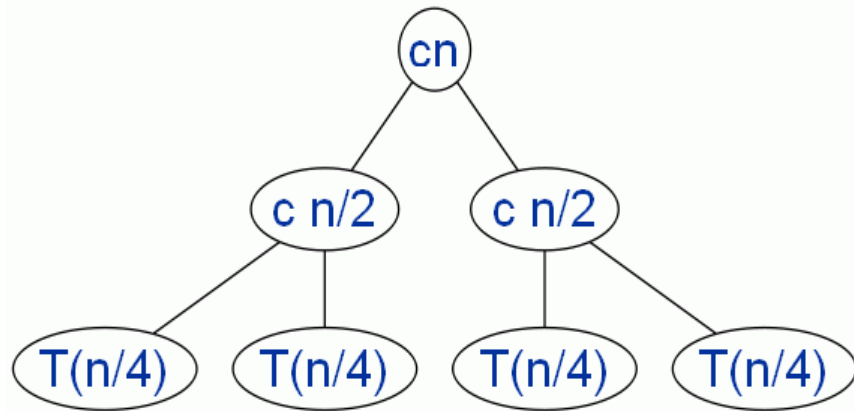
# Illustration von $T(n)$



$T(n)$

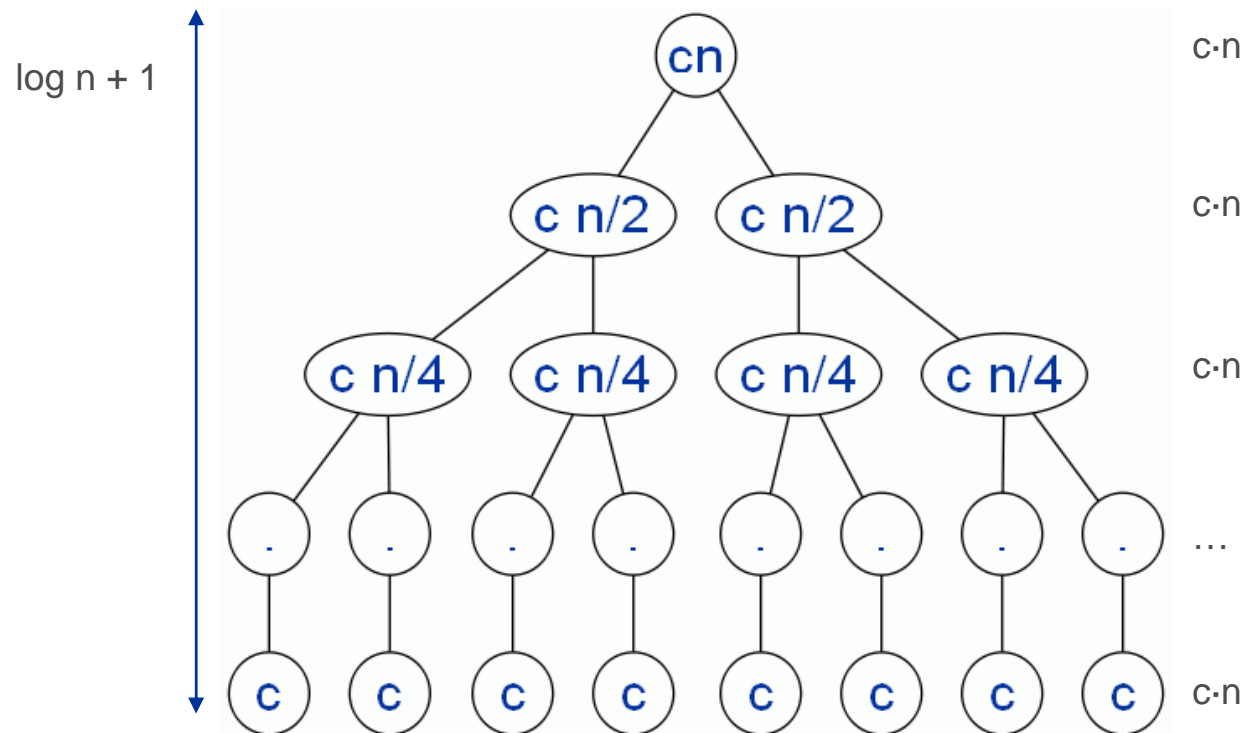


$$T(n) = 2 T(n/2) + cn$$



$$T(n/2) = 2 T(n/4) + cn/2$$

# Illustration von $T(n)$



Knoten in der  $i$ -ten Ebene berücksichtigt  $n/(2^i)$  Elemente:  
Wenn Knoten in der  $i$ -te Ebene ein Element berücksichtigt, dann  $n/(2^i) = 1$ .  
Dann  $n = 2^i$ , also  $i = \log n$

- $T(n) = cn \log n + cn \in \Theta(n \log n)$  (Rekursionsbaum-Methode)

# Überblick



- Prinzip
- Maximale Teilsumme
  - erste Lösungsansätze
  - Teile-und-Herrsche-Ansatz
  - Laufzeit
- Rekursionsgleichungen
  - Substitutionsmethode
  - Rekursionsbaum-Methode
  - Mastertheorem
- Produkt von Polynomen nach Karatsuba
- Matrixmultiplikation nach Strassen
- Erstellung eines Spielplans

# Rekursionsgleichungen



- beschreiben die Laufzeit bei Rekursionen

$$T(n) = \begin{cases} \overset{\text{Trivialfall für } n_0}{f_0(n)} & n = n_0 \\ a \cdot T\left(\frac{n}{b}\right) + f(n) & n > n_0 \end{cases}$$

Lösung von a  
Teilproblemen  
mit reduziertem  
Aufwand n/b

Verbinden der  
Teillösungen

- $n_0$  ist üblicherweise klein, oft ist  $f_0(n_0) \in \Theta(1)$
- üblicherweise  $a > 1$  und  $b > 1$
- Je nach Lösungstechnik wird  $f_0$  vernachlässigt.
- T ist nur für ganzzahlige n/b definiert, was auch gern bei der Lösung vernachlässigt wird.

# Substitutionsmethode



- Lösung raten und mit Induktion beweisen
- Beispiel:  $T(n) = \begin{cases} 1 & n = 1 \\ 2 \cdot T\left(\frac{n}{2}\right) + n & n > 1 \end{cases}$
- Vermutung:  $T(n) = n + n \log n$
- Induktionsanfang für  $n=1$ :  $T(1) = 1 + 1 \log 1 = 1$
- Induktionsschritt von  $n/2$  nach  $n$ :

$$\begin{aligned} T(n) &= 2 \cdot T\left(\frac{n}{2}\right) + n \\ &= 2 \cdot \left(\frac{n}{2} + \frac{n}{2} \log \frac{n}{2}\right) + n && \text{Induktionsvoraussetzung} \\ &= 2 \cdot \left(\frac{n}{2} + \frac{n}{2} (\log n - 1)\right) + n \\ &= n + n \log n - n + n \\ &= n + n \log n \end{aligned}$$

# Substitutionsmethode



- alternativer Lösungsansatz
- Beispiel: 
$$T(n) = \begin{cases} 1 & n = 1 \\ 2 \cdot T\left(\frac{n}{2}\right) + n & n > 1 \end{cases}$$
- Vermutung:  $T(n) \in O(n \log n)$
- Lösung: Finde  $c > 0$  mit  $T(n) \leq c \cdot n \cdot \log n$
- Induktionsschritt von  $n/2$  nach  $n$ :

$$\begin{aligned} T(n) &= 2 \cdot T\left(\frac{n}{2}\right) + n \\ &\leq 2 \cdot \left(c \cdot \frac{n}{2} \cdot \log \frac{n}{2}\right) + n && \text{Induktionsvoraussetzung} \\ &= c \cdot n \log n - c \cdot n \cdot \log 2 + n \\ &= c \cdot n \log n - c \cdot n + n \\ &\leq c \cdot n \log n && \text{für } c \geq 1 \end{aligned}$$

# Überblick



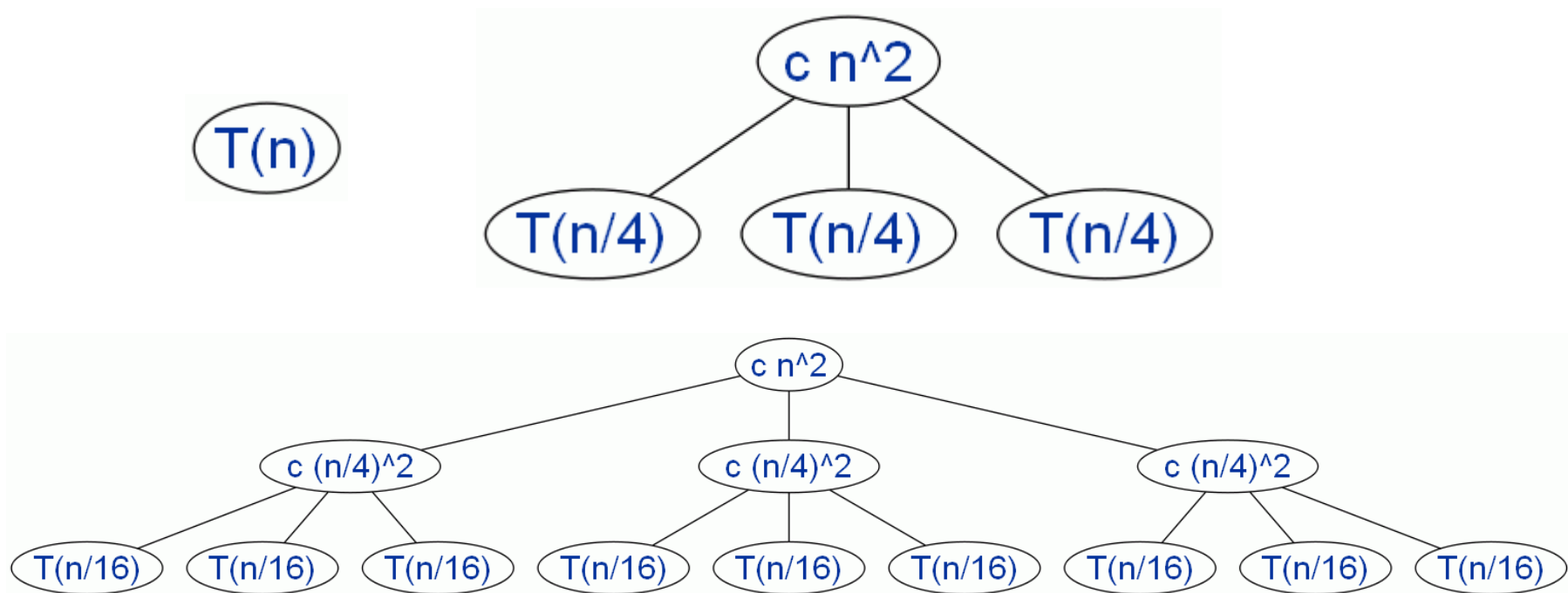
- Prinzip
- Maximale Teilsumme
  - erste Lösungsansätze
  - Teile-und-Herrsche-Ansatz
  - Laufzeit
- Rekursionsgleichungen
  - Substitutionsmethode
  - Rekursionsbaum-Methode
  - Mastertheorem
- Produkt von Polynomen nach Karatsuba
- Matrixmultiplikation nach Strassen
- Erstellung eines Spielplans

# Rekursionsbaum-Methode

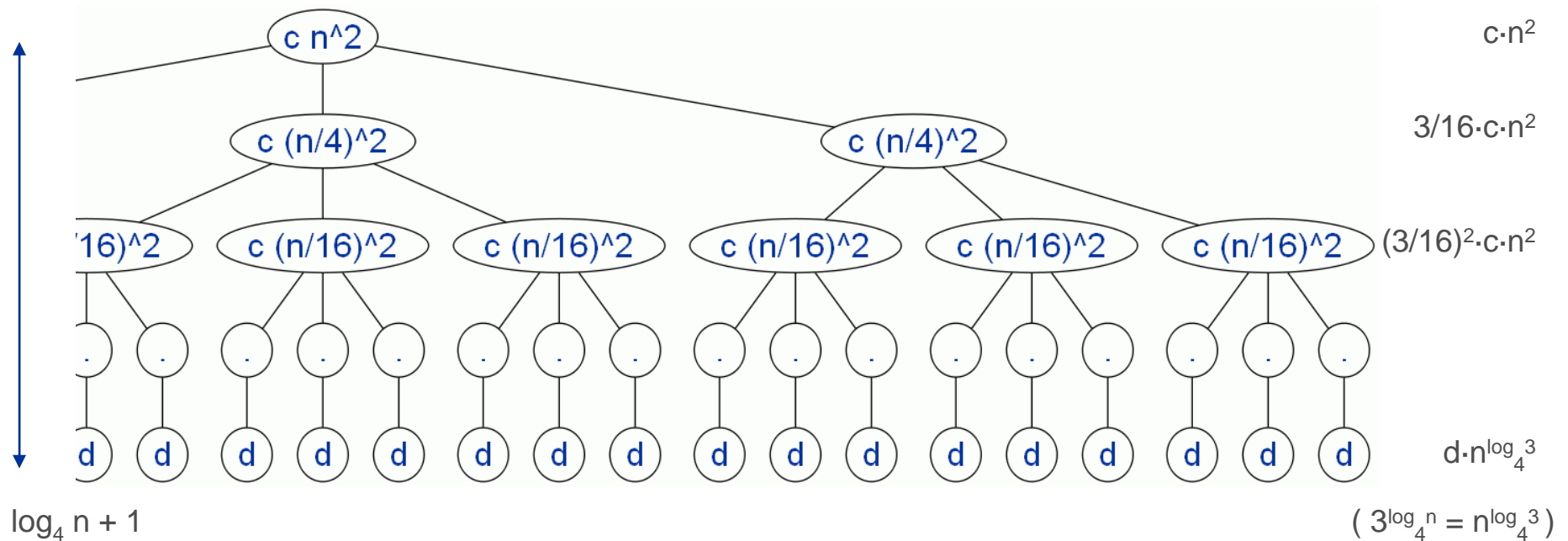
## Beispiel 1



- kann zum Aufstellen von Vermutungen verwendet werden
- Beispiel:  $T(n) = 3 T(n/4) + \Theta(n^2) \leq 3 T(n/4) + cn^2$



# Rekursionsbaum-Methode



$\log_4 n + 1$

wegen  $n = 4^{\text{Zahl der Ebenen}}$

- $\log_4 3 < 1 \Rightarrow T(n) \in O(n^2)$

# Nachweis der Vermutung



- Vermutung:  $T(n) = 3 T(n/4) + \Theta(n^2) \leq 3 T(n/4) + cn^2 \in O(n^2)$
- Substitutionsmethode
- Es existiert ein  $d > 0$  mit  $T(n) < d \cdot n^2$

$$\begin{aligned} T(n) &\leq 3 \cdot T\left(\frac{n}{4}\right) + c \cdot n^2 \\ &\leq 3d \left(\frac{n}{4}\right)^2 + c \cdot n^2 \\ &= \frac{3}{16} dn^2 + c \cdot n^2 \\ &\leq dn^2 \quad \text{für } d \geq (16/13)c \end{aligned}$$

# Rekursionsbaum-Methode

## Beispiel 2



- kann auch durch iteratives Substituieren realisiert werden
- Beispiel:  $T(1) = 1$ ,  $T(2n) = 2 T(n) + 2n$
- $n = 2^k$ ,  $k = \log n$

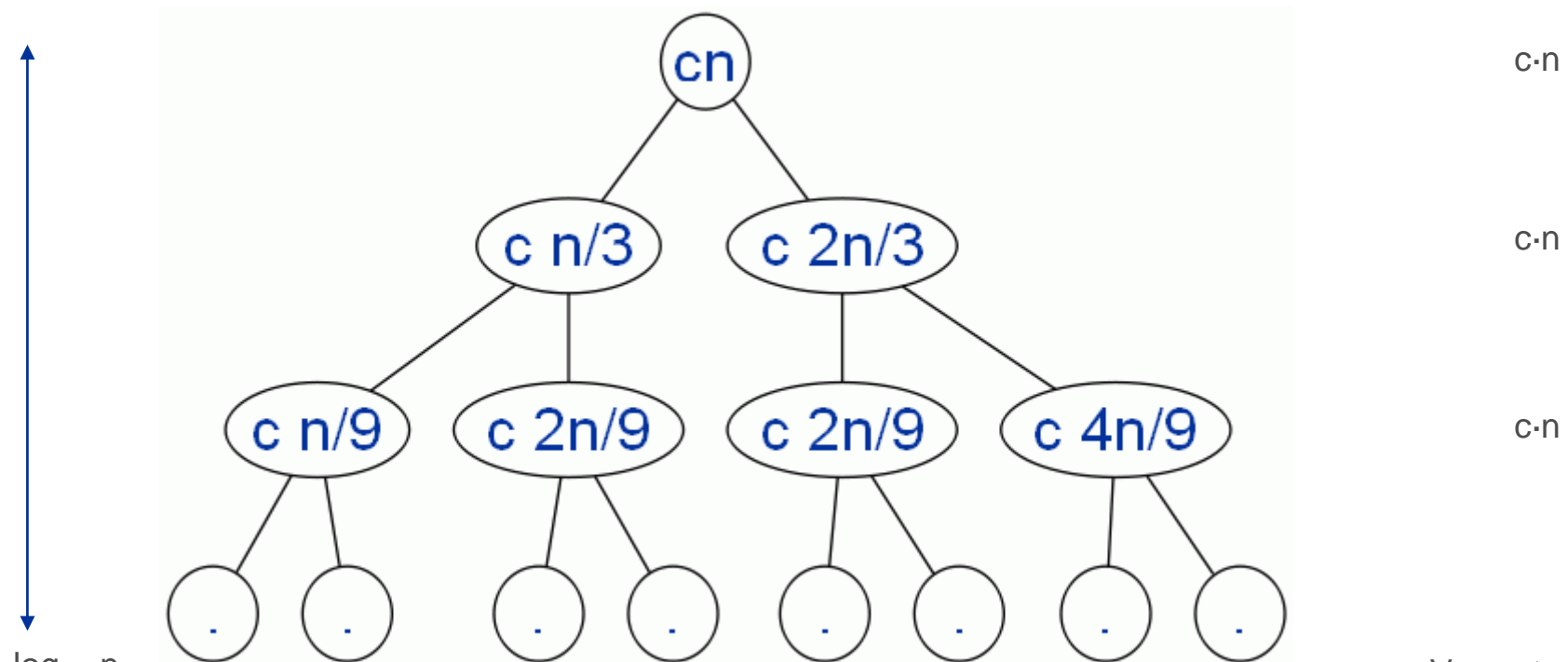
$$\begin{aligned} T(2^k) &= 2 \cdot T(2^{k-1}) + 2^k \\ &= 2 \cdot (2 \cdot T(2^{k-2}) + 2^{k-1}) + 2^k \\ &= 2^2 \cdot T(2^{k-2}) + 2^k + 2^k \\ &= 2^k \cdot T(2^{k-k}) + 2^k + \dots + 2^k \\ &\quad \quad \quad n \quad \quad T(1) = 1 \quad \quad \quad k \cdot 2^k = n \log n \\ &= n + n \log n \\ &\in n \log n \end{aligned}$$

# Rekursionsbaum-Methode

## Beispiel 3



- liefert manchmal zu pessimistische Ergebnisse
- $T(n) = T(n/3) + T(2n/3) + O(n)$



$\log_{3/2} n$   
wegen  $(2/3)^k n = 1$

Vermutung:  
 $O(n \log n)$

# Rekursionsbaum-Methode



- Aber! In Ebene  $\log_{3/2} n$  existieren maximal  $2^{\log_{3/2} n} = n^{\log_{3/2} 2}$  Blätter mit  $\log_{3/2} 2 > 1$ .
- Konstante Kosten pro Blatt führen zu  $\Theta(n^{\log_{3/2} 2}) \in \omega(n \log n)$  Kosten für die Blätter und damit  $T(n) \notin O(n \log n)$ .
- Allerdings:
  - Baum hat tatsächlich weniger Blätter.
  - In tieferen Ebenen fehlen innere Knoten, sodass die Kosten pro Ebene kleiner als  $c \cdot n$  sind.
- Motiviert durch diese Überlegungen lässt sich  $T(n) \in O(n \log n)$  zeigen.

# Nachweis der Vermutung



- Vermutung:  $T(n) = T(n/3) + T(2n/3) + O(n) \in O(n \log n)$
- Substitutionsmethode
- Es existiert ein  $d > 0$  mit  $T(n) < d \cdot n \cdot \log n$

$$\begin{aligned} T(n) &\leq T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + c \cdot n \\ &\leq d \frac{n}{3} \log \frac{n}{3} + d \frac{2n}{3} \log \frac{2n}{3} + c \cdot n \\ &= d \frac{n}{3} \log n - d \frac{n}{3} \log 3 + d \frac{2n}{3} \log n - d \frac{n}{3} \log \frac{3}{2} + c \cdot n \\ &= dn \log n - dn \left( \log 3 - \frac{2}{3} \right) + c \cdot n \\ &\leq dn \log n \quad d \geq c / (\log 3 - (2/3)) \end{aligned}$$

# Überblick



- Prinzip
- Maximale Teilsumme
  - erste Lösungsansätze
  - Teile-und-Herrsche-Ansatz
  - Laufzeit
- Rekursionsgleichungen
  - Substitutionsmethode
  - Rekursionsbaum-Methode
  - **Mastertheorem**
- Produkt von Polynomen nach Karatsuba
- Matrixmultiplikation nach Strassen
- Erstellung eines Spielplans

# Mastertheorem



- Lösungsansatz für Rekursionsgleichungen der Form

$$T(n) = a T(n/b) + f(n) \text{ mit } a \geq 1 \text{ und } b > 1$$

- $T(n)$  beschreibt die Laufzeit eines Algorithmus,
  - der ein Problem der Größe  $n$  in  $a$  Teilprobleme zerlegt,
  - der jedes der  $a$  Teilprobleme rekursiv mit der Laufzeit  $T(n/b)$  löst,
  - der  $f(n)$  Schritte benötigt, um die  $a$  Teillösungen zusammenzusetzen.

# Mastertheorem



$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

- Fall 1:  $T(n) \in \Theta\left(n^{\log_b a}\right)$  falls  $f(n) \in O\left(n^{\log_b a - \epsilon}\right)$   $\epsilon > 0$
- Fall 2:  $T(n) \in \Theta\left(n^{\log_b a} \log n\right)$  falls  $f(n) \in \Theta\left(n^{\log_b a}\right)$
- Fall 3:  $T(n) \in \Theta\left(f(n)\right)$  falls  $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$   $\epsilon > 0$   
$$af\left(\frac{n}{b}\right) \leq cf(n) \quad 0 < c < 1$$
  
$$n > n_0$$

# Beispiele



- Fall 1:  $T(n) \in \Theta(n^{\log_b a})$  falls  $f(n) \in O(n^{\log_b a - \epsilon})$   $\epsilon > 0$
- $T(n) = 8T\left(\frac{n}{2}\right) + 1000n^2$   
 $a = 8, \quad b = 2, \quad f(n) = 1000n^2, \quad \log_b a = \log_2 8 = 3$   
 $f(n) \in O(n^{\log_2 8 - 1}) \rightarrow T(n) \in \Theta(n^3)$
- $T(n) = 9T\left(\frac{n}{3}\right) + 17n$   
 $a = 9, \quad b = 3, \quad f(n) = 17n, \quad \log_b a = \log_3 9 = 2$   
 $f(n) \in O(n^{\log_3 9 - 1}) \rightarrow T(n) \in \Theta(n^2)$

# Beispiele



- Fall 2:  $T(n) \in \Theta(n^{\log_b a} \log n)$  falls  $f(n) \in \Theta(n^{\log_b a})$
- $T(n) = 2T\left(\frac{n}{2}\right) + 10n$   
 $a = 2, \quad b = 2, \quad f(n) = 10n, \quad \log_b a = \log_2 2 = 1$   
 $f(n) \in \Theta(n^{\log_2 2}) \rightarrow T(n) \in \Theta(n \log n)$
- $T(n) = T\left(\frac{2n}{3}\right) + 1$   
 $a = 1, \quad b = \frac{3}{2}, \quad f(n) = 1, \quad \log_b a = \log_{\frac{3}{2}} 1 = 0$   
 $f(n) \in \Theta\left(n^{\log_{\frac{3}{2}} 1}\right) \rightarrow T(n) \in \Theta(n^0 \log n) = \Theta(\log n)$

# Beispiele



- Fall 3:  $T(n) \in \Theta(f(n))$  falls  $f(n) \in \Omega(n^{\log_b a + \epsilon})$   $\epsilon > 0$   
 $a f\left(\frac{n}{b}\right) \leq c f(n)$   $0 < c < 1$
- $T(n) = 2T\left(\frac{n}{2}\right) + n^2$   
 $a = 2, \quad b = 2, \quad f(n) = n^2, \quad \log_b a = \log_2 2 = 1$   
 $f(n) \in \Omega(n^{\log_2 2 + 1})$   
 $2\left(\frac{n}{2}\right)^2 \leq c \cdot n^2 \quad \frac{1}{2}n^2 \leq c \cdot n^2 \quad c = \frac{1}{2}$   
 $T(n) \in \Theta(n^2)$

# Master-Theorem



- lässt sich nicht immer anwenden

- $T(n) = 2T\left(\frac{n}{2}\right) + n \log n$

$$a = 2, \quad b = 2, \quad f(n) = n \log n, \quad \log_b a = \log_2 2 = 1$$

- Fall 1:  $f(n) \notin O\left(n^{\log_2 2 - \epsilon}\right)$

- Fall 2:  $f(n) \notin \Theta\left(n^{\log_2 2}\right)$

- Fall 3:  $f(n) \notin \Omega\left(n^{\log_2 2 + \epsilon}\right)$

$n \log n$  ist asymptotisch größer als  $n$ ,  
aber nicht polynomial größer

# Überblick



- Prinzip
- Maximale Teilsumme
  - erste Lösungsansätze
  - Teile-und-Herrsche-Ansatz
  - Laufzeit
- Rekursionsgleichungen
  - Substitutionsmethode
  - Rekursionsbaum-Methode
  - Mastertheorem
- Produkt von Polynomen nach Karatsuba
- Matrixmultiplikation nach Strassen
- Erstellung eines Spielplans

# Produkt von Polynomen



- Gegeben: Polynome von Grad  $2n-1$ ,  $2n = 2^k$  Koeffizienten

$$a(x) = a_0x^0 + a_1x^1 + \dots + a_{2n-1}x^{2n-1}$$

$$b(x) = b_0x^0 + b_1x^1 + \dots + b_{2n-1}x^{2n-1}$$

- Gesucht:  $c(x) = a(x) \cdot b(x)$   
 $= x^0 \cdot (a_0b_0)$   
 $+ x^1 \cdot (a_1b_0 + a_0b_1)$   
 $+ x^2 \cdot (a_2b_0 + a_1b_1 + a_0b_2)$   
 $+ \dots$   
 $+ x^{4n-2} \cdot (a_{2n-1}b_{2n-1})$

Polynome können gegebenenfalls mit Nullen bis zum Grad  $2n-1$  aufgefüllt werden.

# Produkt von Polynomen



- Naive Implementierung benötigt  $\Omega(n^2)$  Grundoperationen. Jeder Koeffizient von a wird mit jedem Koeffizient von b multipliziert:  $2n \cdot 2n$  Multiplikationen + ? Additionen
- **Karatsuba-Algorithmus** von 1960 benötigt  $O(n^{\log_2 3})$  Grundoperationen.  
(Anatolii Alexeevich Karatsuba, Mathematische Fakultät, Lomonosov Universität Moskau)
- Anwendung: Multiplikation langer Zahlen

# Idee 1

## Zerlegung in zwei kleinere Polynome



- Zerlege das Problem in Teilprobleme halber Größe.
- Kombiniere die gelösten Teilprobleme zur Gesamtlösung.
- Beispiel:

$$a = 7x^0 + 3x^1 + 2x^2 + 6x^3$$

$$b = 5x^0 + 2x^1 + 4x^2 + 8x^3$$

Polynome mit  $2n$  Koeffizienten

$$a = 7x^0 + 3x^1 + x^2(2x^0 + 6x^1) = a_l + x^2 a_h$$

$$b = 5x^0 + 2x^1 + x^2(4x^0 + 8x^1) = b_l + x^2 b_h$$

Zerlegung in 2 Polynome  
mit  $n$  Koeffizienten

$$\begin{aligned} c &= a \cdot b = (a_l + x^2 a_h) \cdot (b_l + x^2 b_h) \\ &= a_l \cdot b_l + x^2(a_l \cdot b_h + a_h \cdot b_l) + x^4(a_h \cdot b_h) \end{aligned}$$

Kombiniere 4 Produkte von Polynomen mit  $n$  Koeffizienten  
zur Lösung des Produkts von Polynomen mit  $2n$  Koeffizienten.

# Laufzeit



- $T(2n) = 4T(n) + O(1) \rightarrow T(n) = 4T\left(\frac{n}{2}\right) + O(1)$   
 $a = 4, \quad b = 2, \quad f(n) = O(1), \quad \log_b a = \log_2 4 = 2$
- Fall 1 des Mastertheorem:  $T(n) \in \Theta\left(n^{\log_b a}\right)$   
falls  $f(n) \in O\left(n^{\log_b a - \epsilon}\right) \quad \epsilon > 0$   
 $f(n) \in O\left(n^{\log_2 4 - 2}\right) \rightarrow T(n) \in \Theta(n^2)$
- Die Umwandlung des Problems in vier Teilprobleme halber Größe führt nicht zur Effizienzsteigerung.

# Idee 2 - Wiederverwendung berechneter Produkte



$$\begin{aligned}c &= a \cdot b = (a_l + x^n a_h) \cdot (b_l + x^n b_h) \\ &= \underbrace{a_l \cdot b_l}_A + x^n (\underbrace{a_l \cdot b_h + a_h \cdot b_l}_C) + x^{2n} (\underbrace{a_h \cdot b_h}_B)\end{aligned}$$

$$\begin{aligned}(a_l + a_h) \cdot (b_l + b_h) &= a_l \cdot b_l + a_l \cdot b_h + a_h \cdot b_l + a_h \cdot b_h \\ a_l \cdot b_h + a_h \cdot b_l &= (a_l + a_h) \cdot (b_l + b_h) - a_l \cdot b_l - a_h \cdot b_h\end{aligned}$$

effiziente Formulierung von C unter Verwendung von A und B

$$A = a_l \cdot b_l$$

$$B = a_h \cdot b_h$$

$$C = (a_l + a_h) \cdot (b_l + b_h)$$

$$c = A + x^n (C - A - B) + x^{2n} B$$

3 Produkte von Polynomen  
mit n Koeffizienten

# Laufzeit



- $T(2n) = 3T(n) + O(n) \rightarrow T(n) = 3T\left(\frac{n}{2}\right) + O(n)$   
 $a_1 + a_n$  liegt in  $O(n)$   
 $a = 3, \quad b = 2, \quad f(n) = O(n), \quad \log_b a = \log_2 3 \approx 1,585$
- Fall 1 des Mastertheorem:  $T(n) \in \Theta\left(n^{\log_b a}\right)$   
falls  $f(n) \in O\left(n^{\log_b a - \epsilon}\right) \quad \epsilon > 0$   
 $f(n) \in O\left(n^{1,585 - 0,585}\right) \rightarrow T(n) \in \Theta\left(n^{1,585}\right)$
- Die Umwandlung des Problems in drei Teilprobleme halber Größe bei linearem Aufwand für die Kombination der Teilergebnisse führt zur Effizienzsteigerung.

# Pseudocode



```
■ procedure karatsuba (a, b, n)
{
    if (n<d) multiply (a, b);

    al = lowCoefficients (a);
    ah = highCoefficients (a);
    bl = lowCoefficients (b);
    bh = highCoefficients (b);

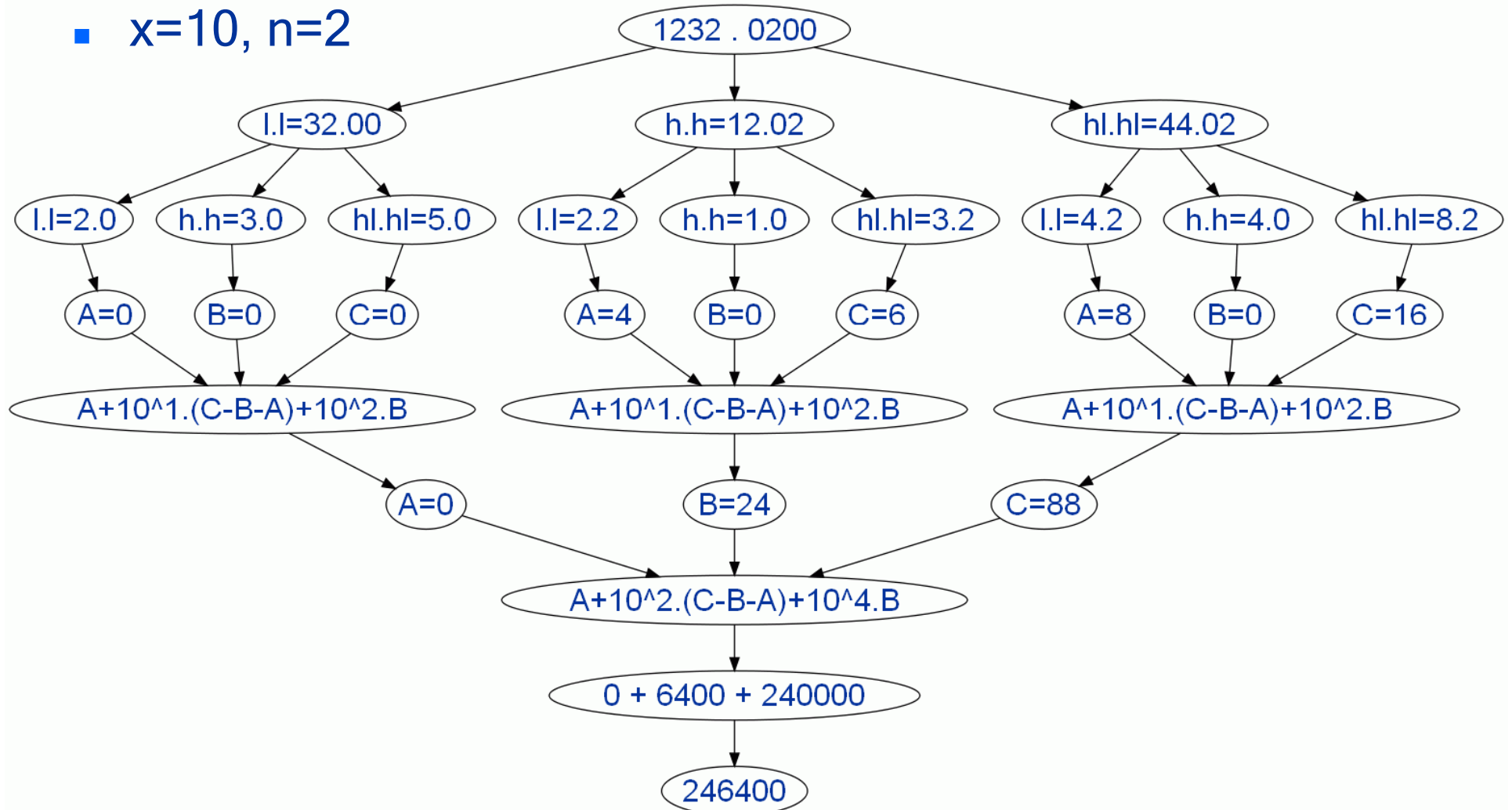
    A = karatsuba (al, bl, n/2);
    B = karatsuba (ah, bh, n/2);
    C = karatsuba (al+ah, bl+bh, n/2);

    return A + shift(C-A-B,n) + shift(B,2*n);
}
```

# Beispiel

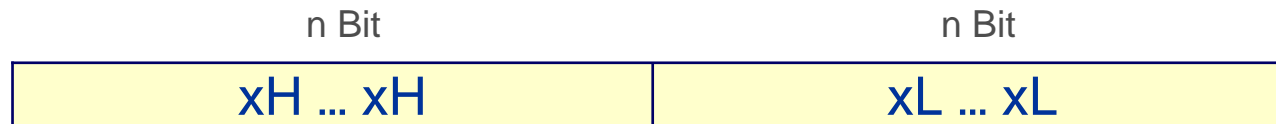


- $x=10, n=2$



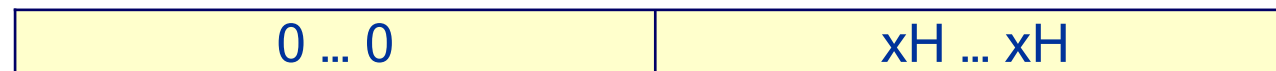


# Shift-Operationen



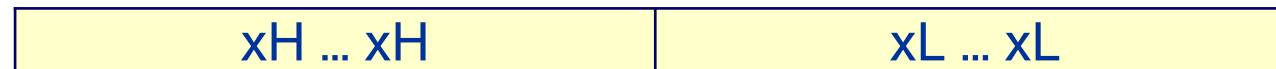
x

- `BigInteger xH = x.shiftRight (n);`

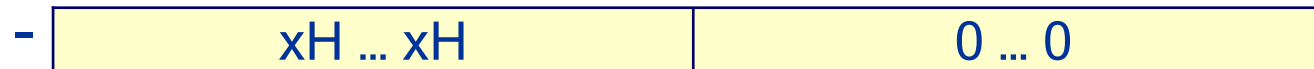


xH

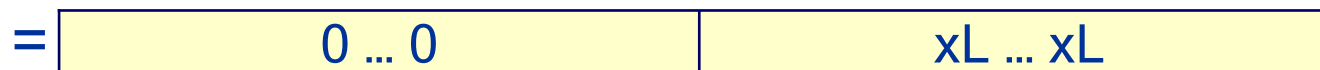
- `BigInteger xL = x.subtract (xH.shiftLeft(n));`



x



xH.shiftLeft (n)



xL

# Überblick



- Prinzip
- Maximale Teilsumme
  - erste Lösungsansätze
  - Teile-und-Herrsche-Ansatz
  - Laufzeit
- Rekursionsgleichungen
  - Substitutionsmethode
  - Rekursionsbaum-Methode
  - Mastertheorem
- Produkt von Polynomen nach Karatsuba
- **Matrixmultiplikation nach Strassen**
- Erstellung eines Spielplans

# Matrix-Multiplikation



- Matrix A mit l Zeilen und m Spalten

$$A = (a_{ij})_{i=1\dots l, j=1\dots m}$$

- Matrix B mit m Zeilen und n Spalten

$$B = (b_{ij})_{i=1\dots m, j=1\dots n}$$

- Produkt  $C = AB$  ist eine Matrix mit l Zeilen und n Spalten

$$C = A \cdot B = (c_{ij})_{i=1\dots l, j=1\dots n}$$

$$c_{ij} = \sum_{k=1}^m a_{ik} \cdot b_{kj}$$

- Element der Zeile i und Spalte j ergibt sich aus dem Skalarprodukt der Zeile i von A und der Spalte j von B.
- Spaltenzahl von A = Zeilenzahl von B
- nicht kommutativ

# Beispiel



$$\begin{aligned} & \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \cdot \begin{pmatrix} 6 & 1 \\ 3 & 2 \\ 0 & 3 \end{pmatrix} \\ &= \begin{pmatrix} 1 \cdot 6 + 2 \cdot 3 + 3 \cdot 0 & 1 \cdot 1 + 2 \cdot 2 + 3 \cdot 3 \\ 4 \cdot 6 + 5 \cdot 3 + 6 \cdot 0 & 4 \cdot 1 + 5 \cdot 2 + 6 \cdot 3 \end{pmatrix} \\ &= \begin{pmatrix} 12 & 14 \\ 39 & 32 \end{pmatrix} \end{aligned}$$

# Pseudocode - Multiplikation quadratischer Matrizen der Größe $n$



```
■ procedure matrixMultiplikation (a, b, n)
  {
O(n)  for i=1 to n do
O(n2)  for j=1 to n do
        {
O(1)    c[i][j] = 0;
O(n3)  for k=1 to n do
O(1)    c[i][j]=c[i][j]+a[i][k]*b[k][j];
        }
O(1) return c;
  }
```

Laufzeit  $\in O(n^3)$

# Idee 1

## Zerlegung in vier Blöcke



- Zerlege das Problem in Teilprobleme halber Größe.
- Kombiniere die gelösten Teilprobleme zur Gesamtlösung.
- Quadratische Matrizen der Größe  $2n = 2^k$ :

$$\begin{aligned} A \cdot B &= \begin{pmatrix} \overset{n \times n}{A_1} & \overset{n \times n}{A_2} \\ \underset{n \times n}{A_3} & \underset{n \times n}{A_4} \end{pmatrix} \cdot \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix} \quad \begin{array}{l} \text{Zerlegung in 4 Matrizen} \\ \text{mit } n \times n \text{ Elementen} \end{array} \\ &= \begin{pmatrix} A_1 \cdot B_1 + A_2 \cdot B_3 & A_1 \cdot B_2 + A_2 \cdot B_4 \\ A_3 \cdot B_1 + A_4 \cdot B_3 & A_3 \cdot B_2 + A_4 \cdot B_4 \end{pmatrix} \end{aligned}$$

Kombiniere 8 Produkte von Matrizen mit  $n \times n$  Elementen zur Lösung des Produkts von Matrizen mit  $2n \times 2n$  Elementen.

# Laufzeit



- $T(2n) = 8T(n) + O(n^2) \rightarrow T(n) = 8T\left(\frac{n}{2}\right) + O(n^2)$   
 $a = 8, \quad b = 2, \quad f(n) = O(n^2), \quad \log_b a = \log_2 8 = 3$
- Fall 1 des Mastertheorem:  $T(n) \in \Theta(n^{\log_b a})$   
falls  $f(n) \in O(n^{\log_b a - \epsilon}) \quad \epsilon > 0$   
 $f(n) \in O(n^{\log_2 8 - 1}) \rightarrow T(n) \in \Theta(n^3)$
- Und wieder nichts gewonnen.

# Idee 2

## Weniger Produkte berechnen



- Strassen-Algorithmus (1969)

Prof. em. Dr. Volker Strassen, Fak. für Mathematik und Informatik, Universität Konstanz

$$P_1 = A_1 \cdot (B_2 - B_4)$$

$$P_2 = (A_1 + A_2) \cdot B_4$$

$$P_3 = (A_3 + A_4) \cdot B_1$$

$$P_4 = A_4 \cdot (B_3 - B_1)$$

$$P_5 = (A_1 + A_4) \cdot (B_1 + B_4)$$

$$P_6 = (A_2 - A_4) \cdot (B_3 + B_4)$$

$$P_7 = (A_1 - A_3) \cdot (B_1 + B_2)$$

7 Produkte von Matrizen mit  $n \times n$  Elementen.  
Additionen sind lediglich  $O(n^2)$ .

## Idee 2

# Weniger Produkte berechnen



$$A \cdot B = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix} \cdot \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix}$$

$$8 \text{ Produkte} = \begin{pmatrix} A_1 \cdot B_1 + A_2 \cdot B_3 & A_1 \cdot B_2 + A_2 \cdot B_4 \\ A_3 \cdot B_1 + A_4 \cdot B_3 & A_3 \cdot B_2 + A_4 \cdot B_4 \end{pmatrix}$$

$$7 \text{ Produkte} = \begin{pmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_5 + P_1 - P_3 - P_7 \end{pmatrix}$$

Additionen sind lediglich  $O(n^2)$

# Laufzeit



- $T(2n) = 7T(n) + O(n^2) \rightarrow T(n) = 7T\left(\frac{n}{2}\right) + O(n^2)$   
 $a = 7, \quad b = 2, \quad f(n) = O(n^2), \quad \log_b a = \log_2 7 \approx 2,807$
- Fall 1 des Mastertheorem:  $T(n) \in \Theta\left(n^{\log_b a}\right)$   
falls  $f(n) \in O\left(n^{\log_b a - \epsilon}\right) \quad \epsilon > 0$   
 $f(n) \in O\left(n^{2,807 - 0,807}\right) \rightarrow T(n) \in \Theta\left(n^{2,807}\right)$
- Die Umwandlung des Problems in sieben Teilprobleme halber Größe bei quadratischem Aufwand für die Kombination der Teilergebnisse führt zur Effizienzsteigerung.

Der geringe Unterschied zwischen  $O(n^3)$  und  $O(n^{2,807})$  bei gleichzeitig größeren Faktoren bei den Operationen mit quadratischem Aufwand führt erst bei sehr großen Matrizen zu Laufzeitverbesserungen.

# Überblick



- Prinzip
- Maximale Teilsumme
  - erste Lösungsansätze
  - Teile-und-Herrsche-Ansatz
  - Laufzeit
- Rekursionsgleichungen
  - Substitutionsmethode
  - Rekursionsbaum-Methode
  - Mastertheorem
- Produkt von Polynomen nach Karatsuba
- Matrixmultiplikation nach Strassen
- **Erstellung eines Spielplans**

# Spielplan für ein Turnier



- $n = 2^k$  Spieler
- jeder spielt gegen jeden
- jeder soll nur einmal pro Tag spielen  
⇒ Turnier dauert  $n-1$  Tage
- Beispiel:  $T_2$  ( $2^2$  Spieler)

	Tag 1	Tag 2	Tag 3
Spieler 1	2	3	4
Spieler 2	1	4	3
Spieler 3	4	1	2
Spieler 4	3	2	1

# Rekursiver Aufbau



- $T_{k+1}$  kann aus  $T_k$  konstruiert werden.
- $T_k$  besteht aus  $n=2^k$  Zeilen und  $n-1$  Spalten
- $T_{k+1}$  besteht aus  $2n=2^{k+1}$  Zeilen und  $n-1+n$  Spalten

$$T_{k+1} = \begin{pmatrix} \overset{n \times n-1}{T_k} & \overset{n \times n}{S_k} \\ \underset{n \times n-1}{U_k} & \underset{n \times n}{Z_k} \end{pmatrix}$$

- $U_k = T_k + n$ , zu jedem Element in  $T_k$  wird  $n$  addiert
- $S_k$ , Elemente sind  $n+1 \dots 2n$ ,  
zyklisches Verschieben in den Spalten
- $Z_k$ , Elemente sind  $1 \dots n$ ,  
zyklisches Verschieben in den Zeilen

# Beispiel



$$T_2 = \begin{pmatrix} 2 & 3 & 4 \\ 1 & 4 & 3 \\ 4 & 1 & 2 \\ 3 & 2 & 1 \end{pmatrix}$$

$$U_2 = \begin{pmatrix} 6 & 7 & 8 \\ 5 & 8 & 7 \\ 8 & 5 & 6 \\ 7 & 6 & 5 \end{pmatrix}$$

$$S_2 = \begin{pmatrix} 5 & 8 & 7 & 6 \\ 6 & 5 & 8 & 7 \\ 7 & 6 & 5 & 8 \\ 8 & 7 & 6 & 5 \end{pmatrix}$$

$$Z_2 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \\ 3 & 4 & 1 & 2 \\ 4 & 1 & 2 & 3 \end{pmatrix}$$

$$T_3 = \begin{pmatrix} T_2 & S_2 \\ U_2 & Z_2 \end{pmatrix}$$

Problem wird in ein Teilproblem halber Größe zerlegt. Zusammenfügen zur Gesamtlösung liegt in  $O(n^2)$

$$T_1 = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

Trivialfall:  $2^1$  Spieler liegt in  $O(1)$

# Laufzeit



- Fall 3:  $T(n) \in \Theta(f(n))$  falls  $f(n) \in \Omega(n^{\log_b a + \epsilon})$   $\epsilon > 0$

$$af\left(\frac{n}{b}\right) \leq cf(n) \quad 0 < c < 1$$

- $T(n) = T\left(\frac{n}{2}\right) + O(n^2)$

$$a = 1, \quad b = 2, \quad f(n) = O(n^2), \quad \log_b a = \log_2 1 = 0$$

$$f(n) \in \Omega(n^{\log_2 1 + 2})$$

$$2 \cdot d \cdot \left(\frac{n}{2}\right)^2 \leq c \cdot d \cdot n^2 \quad \frac{1}{2}n^2 \leq c \cdot n^2 \quad c = \frac{1}{2}$$

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

# *Überblick*



- Zusammenfassung

# Zusammenfassung



- **Teile** das Gesamtproblem in kleinere Teilprobleme auf.
- **Herrsche** über die Teilprobleme durch rekursives Lösen.  
Wenn die Teilprobleme klein sind, löse sie direkt.
- **Verbinde** die Lösungen der Teilprobleme zur Lösung des Gesamtproblems.
  
- **rekursive** Anwendung des Algorithmus auf immer kleiner werdende Teilprobleme
- **direkte** Lösung eines hinreichend kleinen Teilproblems

# Zusammenfassung



- kann bei konzeptionell schwierigen Problemen hilfreich sein
  - Lösung für den Trivialfall muss bekannt sein
  - Aufteilung in Teilprobleme muss möglich sein
  - Kombination der Teillösungen muss möglich sein
- kann effiziente Lösungen realisieren
  - wenn die Lösung des Trivialfalls in  $O(1)$  liegt, Aufteilung in Teilprobleme und Kombination der Teillösungen in  $O(n)$  liegt und die Zahl der Teilprobleme beschränkt ist, liegt der Algorithmus in  $O(n \log n)$
- für Parallelverarbeitung geeignet
  - Teilprobleme werden unabhängig voneinander verarbeitet

# Zusammenfassung



- Definition des Trivialfalls
  - Möglichst kleine direkt zu lösende Teilprobleme sind elegant und einfach.
  - Andererseits wird die Effizienz verbessert, wenn schon relativ große Teilprobleme direkt gelöst werden.
  - Rekursionstiefe sollte nicht zu groß werden.
- Aufteilung in Teilprobleme
  - Wahl der Anzahl der Teilprobleme und konkrete Aufteilung kann anspruchsvoll sein.
- Kombination der Teillösungen
  - typischerweise konzeptionell anspruchsvoll

# Zusammenfassung



- Laufzeitabschätzung durch Lösen von Rekursionsgleichungen

$$T(n) = \begin{cases} \overset{\text{Trivialfall für } n_0}{f_0(n)} & n = n_0 \\ a \cdot T\left(\frac{n}{b}\right) + f(n) & n > n_0 \end{cases}$$

Lösung von a  
Teilproblemen  
mit reduziertem  
Aufwand n/b

Verbinden der  
Teillösungen

- Substitutionsmethode
- Rekursionsbaum-Methode
- Mastertheorem

# *Nächstes Thema*



- weitere Entwurfstechniken für Algorithmen