

Algorithmen und Datenstrukturen (ESE)
Entwurf, Analyse und Umsetzung von
Algorithmen (IEMS)
WS 2013 / 2014

Vorlesung 9, Donnerstag 19. Dezember 2013
(Teile und Herrsche, Mastertheorem)

Junior-Prof. Dr. Olaf Ronneberger
Image Analysis Lab
Institut für Informatik
Universität Freiburg

Blick über die Vorlesung heute

■ Organisatorisches

- Rückmeldungen zum Ü7 Dynamic Array

■ Teile und Herrsche

- Kurze Wiederholung: Prinzip
- Beispiel Maximale Teilsumme
- Laufzeitanalyse für Rekursionsalgorithmen
- Mastertheorem – der Hauptsatz der Laufzeitfunktionen

Rückmeldungen Ü7 (Dynamic Array)

- Schöne, sinnvolle Aufgabe
- Hat Spaß gemacht
- Mit Java war die Echtzeitaufgabe nicht lösbar – war nicht beabsichtigt, sorry, aber vielleicht haben Sie trotzdem etwas Wichtiges gelernt...
- Nutzen Sie die Weihnachtsferien entsprechend 😊

- Prinzip
- Maximale Teilsumme
 - erste Lösungsansätze
 - Teile-und-Herrsche-Ansatz
 - Laufzeit
- Rekursionsgleichungen
 - Substitutionsmethode
 - Rekursionsbaum-Methode
 - Mastertheorem

- **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

- 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 ;

- 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

■ 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

- Prinzip
- Maximale Teilsumme
 - erste Lösungsansätze
 - Teile-und-Herrsche-Ansatz
 - Laufzeit
- Rekursionsgleichungen
 - Substitutionsmethode
 - Rekursionsbaum-Methode
 - Mastertheorem

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



Naive Lösung (Brute Force)

```
maxSubArray(X)
result.sum=X(0); result.u=0; result.o=0;
  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;
    end;
  return result.sum, result.u, result.o;
```

Java-Implementierung

```
public class MaxSubArray {
    public static void main(String[] args) {
    }
    public Result maxSubArray(int[] X) {
        Result result = new Result();
        result.sum = X[0]; result.u = 0; result.o = 0;
        for (int u = 0; u < X.length; u++) {
            for (int o = u; o < X.length; o++) {
                int summe = 0;
                for (int i = u; i <= o; i++) {
                    summe = summe + X[i];
                    if (result.sum < summe) {
                        result.sum = summe; result.u = u; result.o = o;
                    }
                }
            }
        }
        return result;
    }
}

class Result {
    int sum; int u; int o;
}
```

Laufzeit obere Schranke

```
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;
```

n Schleifendurchläufe $\rightarrow O(n)$

maximal n Schleifendurchläufe $\rightarrow O(n)$

maximal n
Schleifendurchläufe
 $\rightarrow O(n)$

$O(1)$

- drei verschachtelte Schleifen $O(n)$ also insgesamt $O(n^3)$

Laufzeit untere Schranke

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

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

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 O(n)
```

```
  begin
```

```
    summe=0;
```

```
    for o=u to länge(X)-1 do O(n)
```

```
    begin
```

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

```
      if result.sum<summe then O(1)
```

```
      begin
```

```
        result.sum = summe;
```

```
        result.u = u; result.o = o;
```

```
      end;
```

```
    end;
```

```
  end;
```

```
return result.sum, result.u, result.o;
```

Gesamtaufwand $O(n^2)$

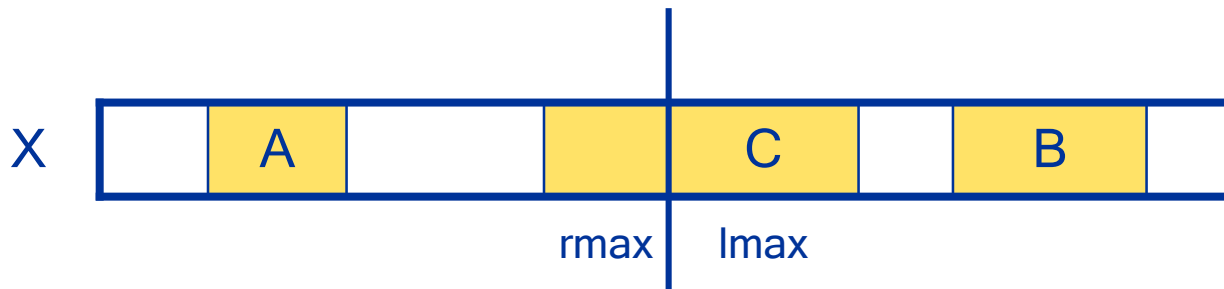
Java-Implementierung

```
public class MaxSubArray {
    public static void main(String[] args) {
    }
    public Result maxSubArray(int[] X) {
        Result result = new Result();
        result.sum = X[0]; result.u = 0; result.o = 0;
        for (int u = 0; u < X.length; u++) {
            int summe = 0;
            for (int o = u; o < X.length; o++) {
                summe = summe + X[o];
                if (result.sum < summe) {
                    result.sum = summe;
                    result.u = u;
                    result.o = o;
                }
            }
        }
        return result;
    }
}

class Result {
    int sum; int u; int o;
}
```

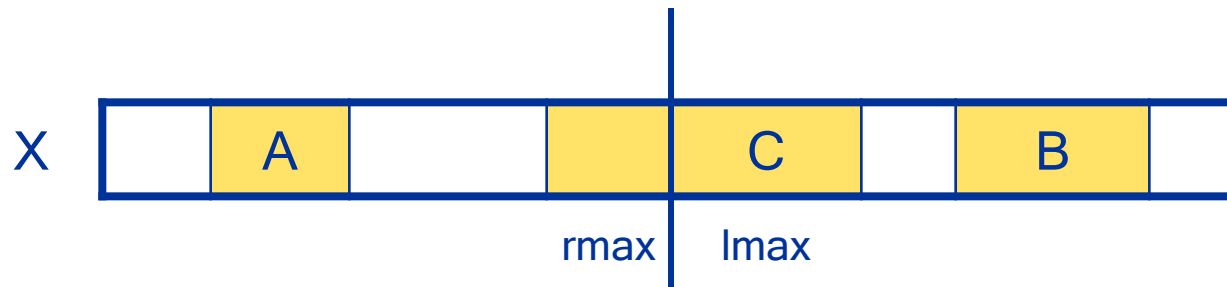
- Prinzip
- Maximale Teilsumme
 - erste Lösungsansätze
 - Teile-und-Herrsche-Ansatz
 - Laufzeit
- Rekursionsgleichungen
 - Substitutionsmethode
 - Rekursionsbaum-Methode
 - Mastertheorem

- 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 r_{\max} und l_{\max} bestimmt werden.
- Gesamtlösung ist Maximum von A, B, C.

- 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.

Java-Implementierung

```
public class MaxSubArray {  
  
    public static void main(String[] args) {  
    }  
  
    public int maxSubArray(int[] X, int u, int o) {  
  
        if (u == o) {  
            return X[u];  
        }  
  
        int m = (u + o) / 2;  
        int A = maxSubArray(X, u, m);  
        int B = maxSubArray(X, m + 1, o);  
  
        int C1 = rmax(X, u, m);  
        int C2 = lmax(X, m + 1, o);  
  
        return max(A, B, C1 + C2);  
    }  
}
```

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);
```


Java-Implementierung

```
public int maxSubArrayAlternativ(int[] X, int u, int o) {  
  
    if (u == o) {  
        return X[u];  
    }  
  
    if (u + 1 == o) {  
        return max(X[u], X[o], X[u] + X[o]);  
    }  
  
    int m = (u + o) / 2;  
    int A = maxSubArrayAlternativ(X, u, m);  
    int B = maxSubArrayAlternativ(X, m + 1, o);  
  
    int C1 = rmax(X, u, m);  
    int 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;
```

Java-Implementierung

```
public int max(int a, int b, int c) {  
  
    if (a > b) {  
        if (a > c) {  
            return a;  
        } else {  
            return c;  
        }  
    } else {  
        if (c > b) {  
            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);
```

Java-Implementierung

```
public int max2(int a, int b) {  
  
    if (a > b) {  
        return a;  
    } else {  
        return b;  
    }  
}  
  
public int max3(int a, int b, int c) {  
  
    return max2(max2(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;
```

Java-Implementierung

```
public int lmax(int[] X, int u, int o) {  
  
    int lmax = X[u];  
    int summe = X[u];  
    for (int i = u + 1; i <= o; i++) {  
        summe = summe + X[i];  
        if (summe > lmax) {  
            lmax = summe;  
        }  
    }  
    return lmax;  
}
```

```
public int rmax(int[] X, int u, int o) {  
  
    int rmax = X[o];  
    int summe = X[o];  
    for (int i = o - 1; i >= u; i--) {  
        summe = summe + X[i];  
        if (summe > rmax) {  
            rmax = summe;  
        }  
    }  
    return rmax;  
}
```

Illustration - lmax

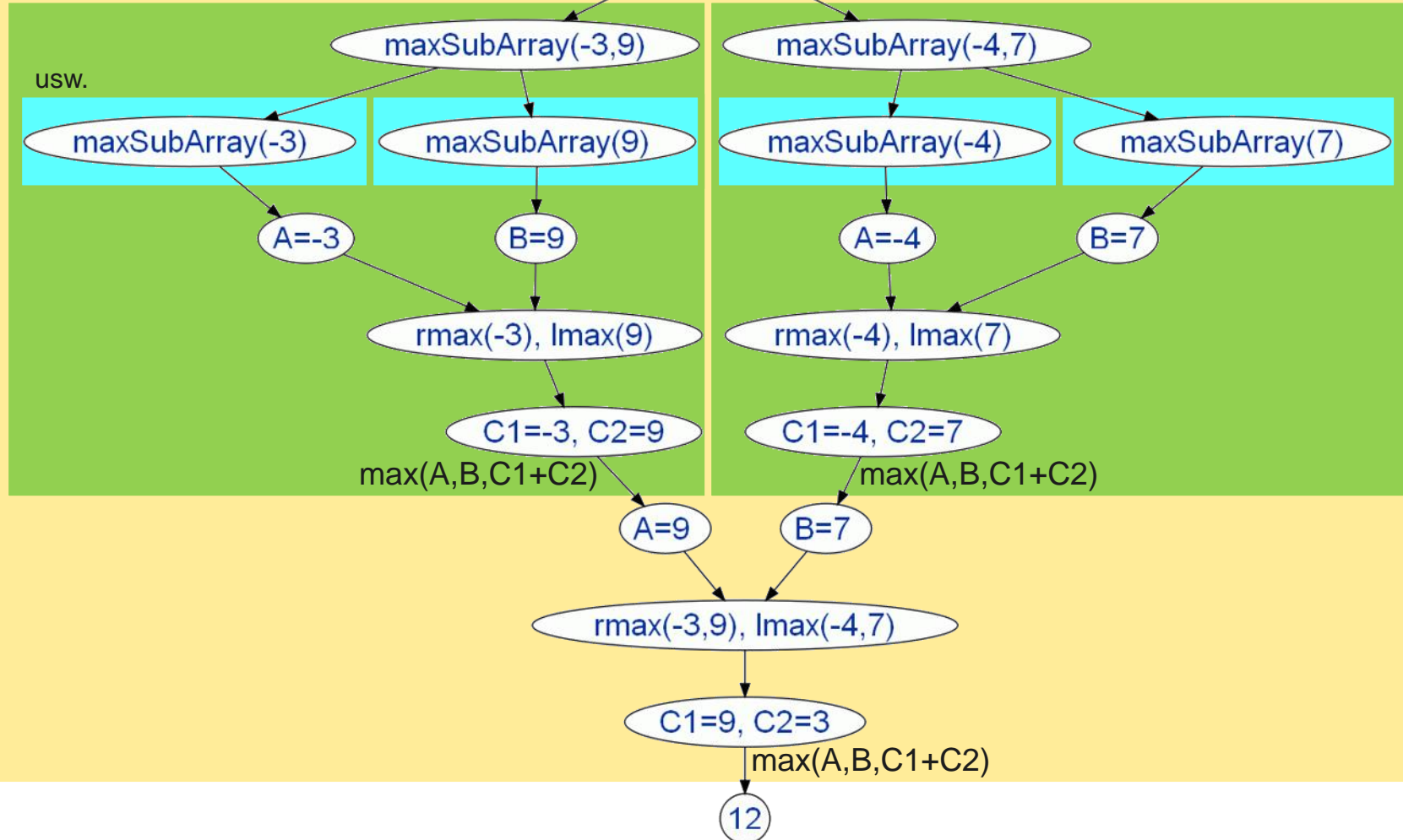
i	u	u+1	o-1	o
X	58	-53	26	59	-41	31
summe	58	5	31	90	49	80
lmax	58	58	58	90	90	90

- lmax und summe werden mit X(u) initialisiert
- X wird durchlaufen von u bis o
- summe wird aktualisiert
- lmax wird aktualisiert, wenn summe > lmax

Illustration - maxSubArray

Aufruf für Feld mit vier Elementen

Aufruf für zwei Felder mit zwei Elementen



- Prinzip
- Maximale Teilsumme
 - erste Lösungsansätze
 - Teile-und-Herrsche-Ansatz
 - Laufzeit
- Rekursionsgleichungen
 - Substitutionsmethode
 - Rekursionsbaum-Methode
 - Mastertheorem

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

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

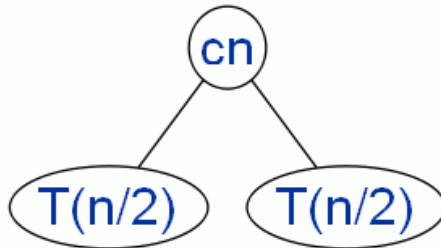
- Wir definieren $c := \max(a, b)$. dann gilt

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

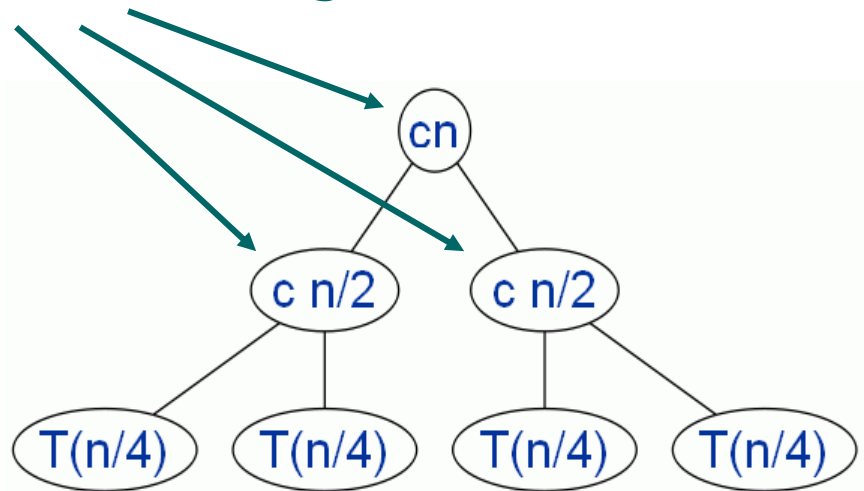
Illustration von $T(n)$

Verbinden der Teillösungen

$T(n)$



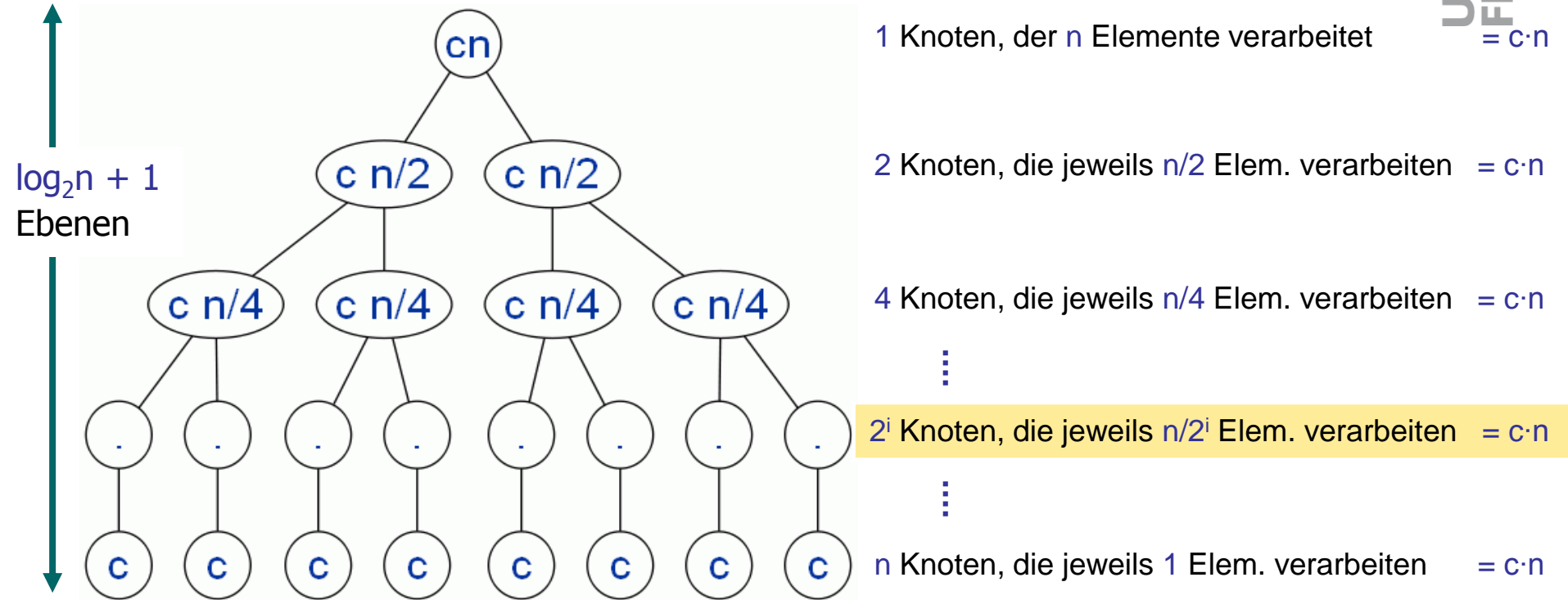
$$T(n) = 2 T(n/2) + cn$$



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

Lösen der Teilprobleme

Illustration von $T(n)$



- oberste Ebene $i=0$, unterste Ebene bei $2^i=n \rightarrow i = \log_2 n$, Also insgesamt $\log_2 n + 1$ Ebenen
- In jeder Ebene Kosten von $c \cdot n$ (hier Kosten für Verbinden der Teillösungen und für das Lösen der trivialen Probleme gleich)
- $T(n) = cn \log_2 n + cn \in \Theta(n \log n)$ (Rekursionsbaum-Methode)

Zusammenfassung Maximale Teilsumme

- Direkte Lösung war $O(n^3)$
- Bessere Lösung mit inkrementeller Aktualisierung der Teilsummen war $O(n^2)$
- Teile-und-Herrsche-Ansatz liefert $O(n \log n)$
- Es gibt auch einen Ansatz mit $O(n)$, wenn man voraussetzen kann, dass die maximale Teilsumme positiv ist. (Schauen wir uns evtl. in einer der nächsten Vorlesungen an)

- Prinzip
- Maximale Teilsumme
 - erste Lösungsansätze
 - Teile-und-Herrsche-Ansatz
 - Laufzeit
- Rekursionsgleichungen
 - Substitutionsmethode
 - Rekursionsbaum-Methode
 - Mastertheorem

Rekursionsgleichungen...

- ... beschreiben die Laufzeit bei Rekursionen

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

Trivialfall für n_0

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: $T(n) = n + n \log_2 n$
- Induktionsanfang für $n=1$: $T(1) = 1 + 1 \log_2 1 = 1$
- Induktionsschritt von $n/2$ nach n :

$$\begin{aligned} T(n) &= 2 \cdot T\left(\frac{n}{2}\right) + n \\ &= 2 \cdot \left(\frac{n}{2} + \frac{n}{2} \log_2 \frac{n}{2}\right) + n \quad \text{Induktionsvoraussetzung} \\ &= 2 \cdot \left(\frac{n}{2} + \frac{n}{2} (\log_2 n - 1)\right) + n \\ &= n + n \log_2 n - n + n \\ &= n + n \log_2 n \end{aligned}$$

Substitutionsmethode

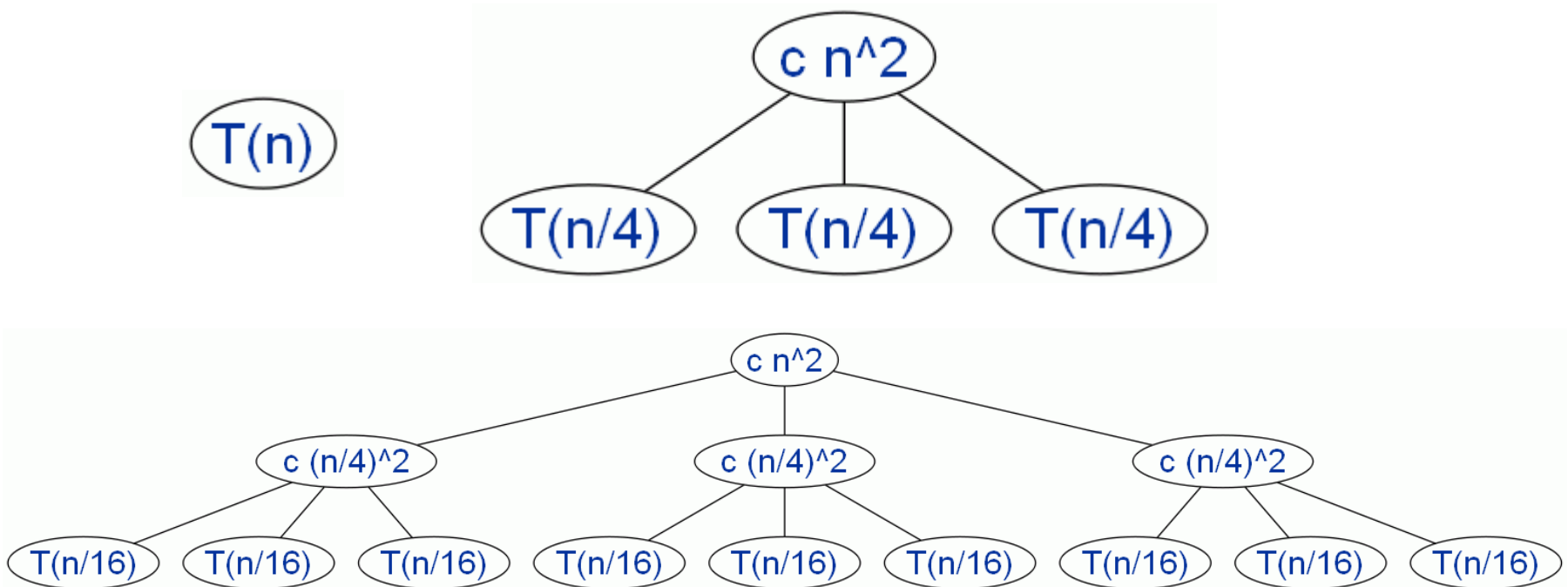
- alternativer Lösungsansatz
- Beispiel:
$$T(n) = \begin{cases} 1 & n = 1 \\ 2 \cdot T\left(\frac{n}{2}\right) + n & n > 1 \end{cases}$$
- Vermutung: $T(n) \in O(n \log n)$
- Lösung: Finde $c > 0$ mit $T(n) \leq c \cdot n \cdot \log_2 n$
- Induktionsschritt von $n/2$ nach n :

$$\begin{aligned} T(n) &= 2 \cdot T\left(\frac{n}{2}\right) + n \\ &\leq 2 \cdot \left(c \cdot \frac{n}{2} \cdot \log_2 \frac{n}{2}\right) + n && \text{Induktionsvoraussetzung} \\ &= c \cdot n \log_2 n - c \cdot n \cdot \log_2 2 + n \\ &= c \cdot n \log_2 n - c \cdot n + n \\ &\leq c \cdot n \log_2 n \quad \text{für } c \geq 1 \end{aligned}$$

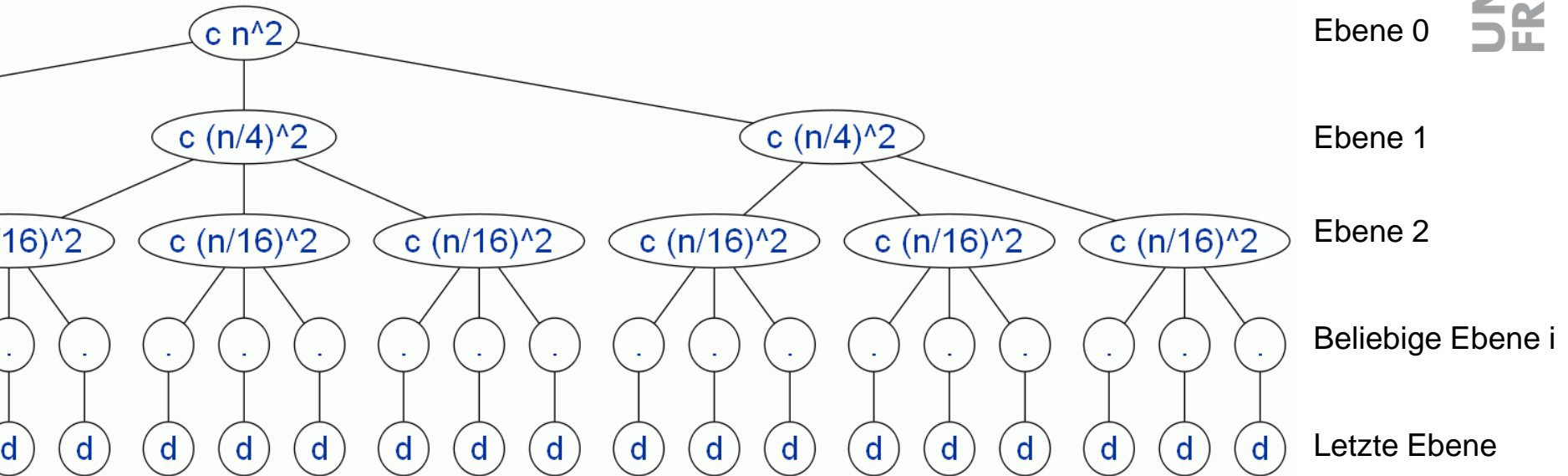
- Prinzip
- Maximale Teilsumme
 - erste Lösungsansätze
 - Teile-und-Herrsche-Ansatz
 - Laufzeit
- Rekursionsgleichungen
 - Substitutionsmethode
 - Rekursionsbaum-Methode
 - Mastertheorem

Rekursionsbaum-Methode Beispiel 1

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



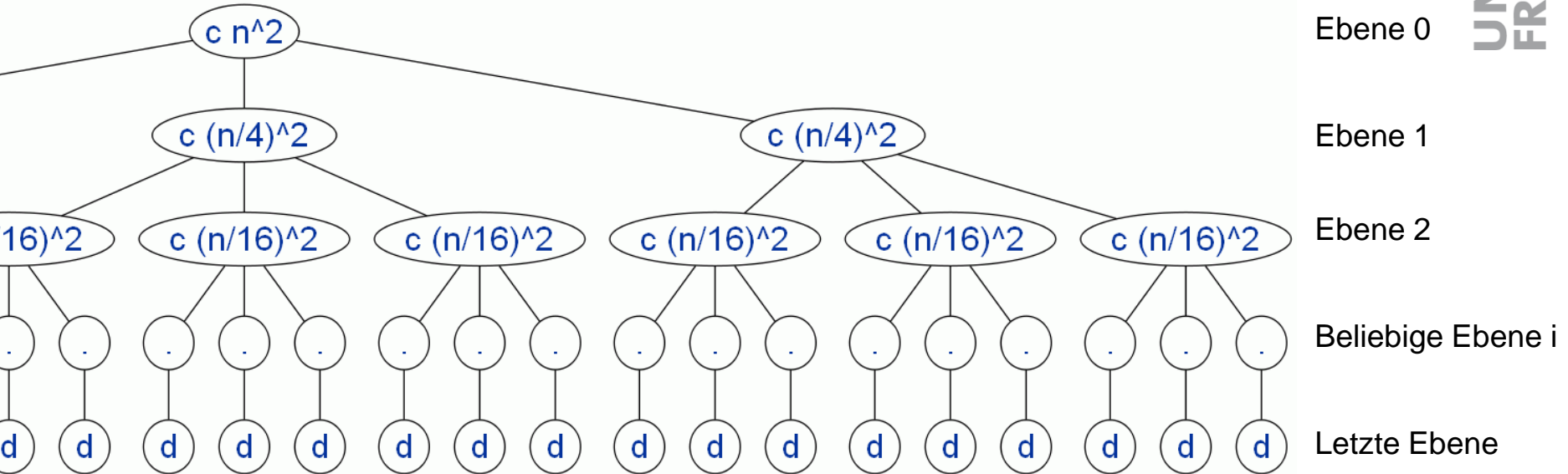
Rekursionsbaum-Methode



Kosten (Verbinden der Teillösungen) in Ebenen außer der letzten

- Größe eines Teilproblems in Ebene i : $(1/4)^i \cdot n$
- Kosten eines Teilproblems in Ebene i : $c \cdot ((1/4)^i \cdot n)^2$
- Anzahl der Teilprobleme in Ebene i : 3^i
- Kosten in Ebene i : $3^i \cdot c \cdot \left(\left(\frac{1}{4} \right)^i \cdot n \right)^2 = \left(\frac{3}{16} \right)^i \cdot c \cdot n^2$

Rekursionsbaum-Methode



Kosten für die letzte Ebene (Lösen der trivialen Probleme)

- Größe eines Teilproblems in der letzten Ebene: 1
- Somit ist die letzte Ebene bei $(1/4)^i \cdot n = 1 \rightarrow n = 4^i \rightarrow i = \log_4 n$
- Anzahl der Teilprobleme in der letzten Ebene: $3^{\log_4 n} = n^{\log_4 3}$
- Gesamtkosten in der letzten Ebene: $d \cdot n^{\log_4 3}$

Nebenrechnung

Rechenspaß mit dem Logarithmus:

$$\begin{aligned} 3^{\log_4 n} &= 3^{\frac{\log_3 n}{\log_3 4}} \\ &= 3^{\log_3 n \cdot \frac{1}{\log_3 4}} \quad \text{nutze } \frac{1}{\log_a b} = \log_b a \\ &= 3^{\log_3 n \cdot \log_4 3} \\ &= \left(3^{\log_3 n}\right)^{\log_4 3} \\ &= n^{\log_4 3} \end{aligned}$$

(Dieser Term wird uns gleich im Mastertheorem wieder begegnen)

Gesamtkosten:

- Kosten für Ebene i waren : $\left(\frac{3}{16}\right)^i \cdot c \cdot n^2$
- Kosten für letzte Ebene waren: $d \cdot n^{\log_4 3}$
- Also zusammen:

$$\sum_{i=0}^{n^{\log_4 3} - 1} \left(\frac{3}{16}\right)^i \cdot c \cdot n^2 + d \cdot n^{\log_4 3} \in \mathcal{O}(n^2)$$

Geometrische Reihe,
also konstant,
(sogar bei unendlich
vielen Ebenen)

$\log_4 3 < 1$, also wächst
viel schwächer als n^2

→ hier: Kosten für Verbinden der Teillösungen dominieren

Einschub: Geometrische Reihe

- **Geometrische Folge:** Quotient zweier benachbarter Folgenglieder ist konstant
- **Geometrische Reihe:** Die Reihe (kummulative Summe) einer geometrischen Folge
- Für $|q| < 1$ gilt:

$$\sum_{k=0}^{\infty} a_0 q^k = \frac{a_0}{1 - q}$$

- Also konstant

Nachweis der Vermutung $O(n^2)$

- 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 $k > 0$ mit $T(n) < k \cdot n^2$

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

- Prinzip
- Maximale Teilsumme
 - erste Lösungsansätze
 - Teile-und-Herrsche-Ansatz
 - Laufzeit
- Rekursionsgleichungen
 - Substitutionsmethode
 - Rekursionsbaum-Methode
 - **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 (einfache Form)

- In den Beispielen haben wir gesehen, dass
 - entweder das Verbinden der Teillösungen die Laufzeit dominiert,
 - oder das Lösen der trivialen Probleme,
 - oder beides gleich viel ausmacht
- Bei der einfachen Form des Mastertheorems kann man das anschaulich zeigen (Mehlhorn, Sanders 2.6.2)
- Hier betrachten wir den Spezialfall, dass das Verbinden der Teillösungen $f(n)$ in $O(n)$ liegt

Mastertheorem (einfache Form)

- Für eine Rekursionsgleichung der Form

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

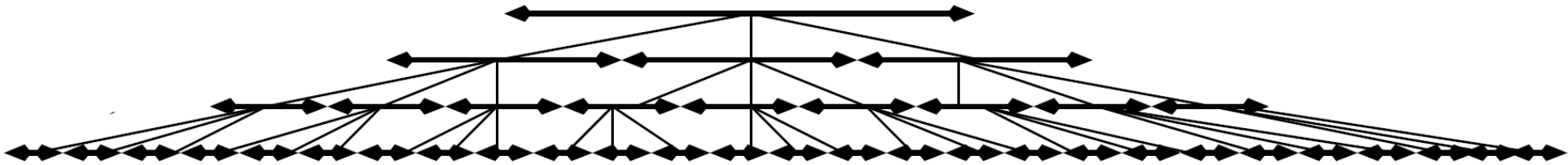
- Gilt:

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{if } a > b \\ \Theta(n \log n) & \text{if } a = b \\ \Theta(n) & \text{if } a < b \end{cases}$$

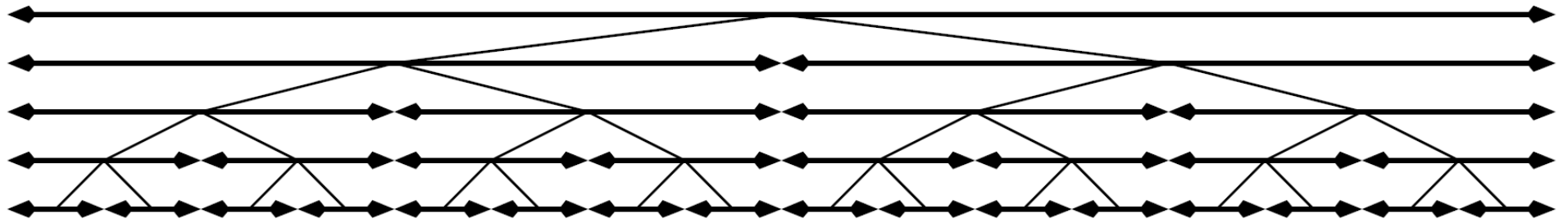
Mastertheorem (einfache Form)

■ Illustration des Aufwandes pro Ebene als Linien

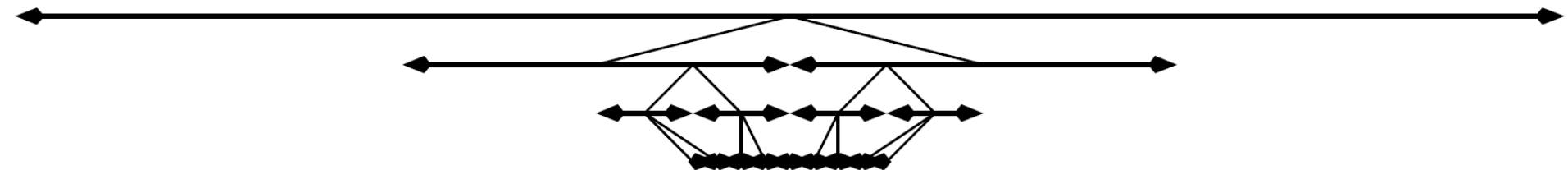
– Fall 1: $a=3, b=2$: drei Teilprobleme mit Größe $\frac{1}{2}$



– Fall 2: $a=b=2$: zwei Teilprobleme mit Größe $\frac{1}{2}$



– Fall 3: $a=2, b=4$: zwei Teilprobleme mit Größe $\frac{1}{4}$



Mastertheorem (einfache Form)

- Für eine Rekursionsgleichung der Form

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

- Gilt:

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{if } a > b \\ \Theta(n \log n) & \text{if } a = b \\ \Theta(n) & \text{if } a < b \end{cases}$$

Fall 1: Lösen der Trivialprobleme (letzte Ebene) dominiert die Laufzeit

Fall 2: Jede Ebene gleich teuer, ($\log n$ Ebenen)

Fall 3: Verbinden der Teillösungen (erste Ebene) dominiert die Laufzeit

- Beweis: über geometrische Reihe.

- Zahl der Operationen pro Ebene (vorwärts oder rückwärts) ist um konstanten Faktor kleiner

Mastertheorem (allgemeine Form)

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

- Fall 1: $T(n) \in \Theta\left(n^{\log_b a}\right)$ falls $f(n) \in \mathcal{O}\left(n^{\log_b a - \epsilon}\right)$ $\epsilon > 0$
 - Operationen in letzter Ebene (Lösen der Trivialprobleme) dominieren die Kosten
 - Fall 2: $T(n) \in \Theta\left(n^{\log_b a} \log n\right)$ falls $f(n) \in \Theta\left(n^{\log_b a}\right)$
 - Jede Ebene gleich teuer
 - Fall 3: $T(n) \in \Theta\left(f(n)\right)$ falls $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ $\epsilon > 0$
 - Operationen in erster Ebene (Verbinden der Teillösungen) dominieren die Kosten
- $$af\left(\frac{n}{b}\right) \leq cf(n) \quad 0 \leq c \leq 1$$
- $$n > n_0$$

Beispiele

■ **Fall 1:** $T(n) \in \Theta(n^{\log_b a})$ falls $f(n) \in \mathcal{O}(n^{\log_b a - \epsilon})$ $\epsilon > 0$

■ $T(n) = 8T\left(\frac{n}{2}\right) + 1000n^2$

$$a = 8, \quad b = 2, \quad f(n) = 1000n^2, \quad \log_b a = \log_2 8 = 3$$

$$f(n) \in \mathcal{O}(n^{\log_2 8 - 1}) \Rightarrow T(n) \in \Theta(n^3)$$

■ $T(n) = 9T\left(\frac{n}{3}\right) + 17n$

$$a = 9, \quad b = 3, \quad f(n) = 17n, \quad \log_b a = \log_3 9 = 2$$

$$f(n) \in \mathcal{O}(n^{\log_3 9 - 1}) \Rightarrow T(n) \in \Theta(n^2)$$

- **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 $f(n) \in \Omega(n^{\log_b a + \epsilon})$ $\epsilon > 0$

$$af\left(\frac{n}{b}\right) \leq cf(n) \quad 0 \leq c \leq 1$$
$$n > n_0$$

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

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

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

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

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

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

- 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

- Teile-und-Herrsche Ansatz:
 - **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.
- Laufzeitanalyse
 - Die rekursiven Aufrufe kann man als Baum veranschaulichen:
 - Teilen+Verbinden Operationen in den Knoten
 - Lösen der Trivialprobleme in den Blättern
 - Die Laufzeit wird entweder von den **Teilen+Verbinden** Operationen dominiert, oder von den **Lösen der Trivialprobleme** oder **beides gleich**

■ Mastertheorem

- Drei Fälle, je nachdem welcher Teil die Laufzeit dominiert.
- **Einfache Form** (lineare Kosten für Teilen+Verbinden):
$$T(n) = aT\left(\frac{n}{b}\right) + cn$$
- **Fall 1:** Lösen der Trivialprobleme (letzte Ebene) dominiert die Laufzeit
→ $\Theta(\text{Anzahl der Blätter}) = \Theta(n^{\log_b a})$
- **Fall 2:** Jede Ebene gleich teuer, ($\log n$ Ebenen)
→ $\Theta(n \log n)$
- **Fall 3:** Verbinden der Teillösungen (erste Ebene) dominiert die Laufzeit
→ $\Theta(n)$
- **Allgemeine Form:** Dieselben Fälle, aber etwas kompliziertere Ausdrücke

Übungsaufgabe

- Heute keine neue Aufgabe
- Nutzen Sie die Zeit um bei den alten Übungsaufgaben aufzuholen
- Oder um eine schnellere Programmiersprache zu lernen...

Weitere Beispiele für Rekursionsbaum- methode

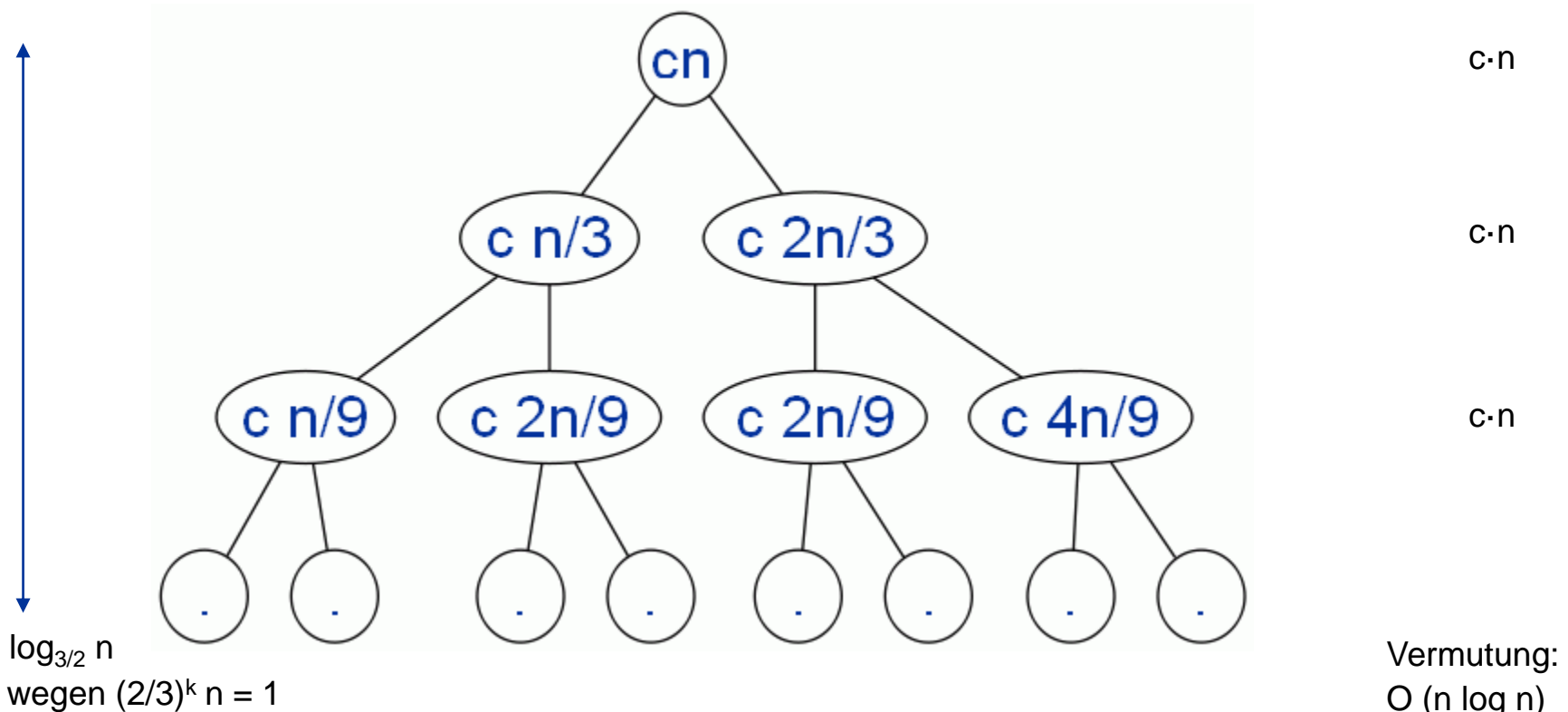
Rekursionsbaum-Methode Beispiel 2

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

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

Rekursionsbaum-Methode Beispiel 3

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



- 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 $T(n) < d \cdot n \cdot \log n$

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