Algorithmen und Datenstrukturen (ESE)
Entwurf, Analyse und Umsetzung von Algorithmen (IEMS)
WS 2014 / 2015

Vorlesung 9, Donnerstag 18. Dezember 2014
(Teile und Herrsche, Mastertheorem)

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

[basiert auf „Informatik 2" von Prof. Dr. Matthias Teschner]

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Rückmeldungen zum Ü8 Cache-Effizienz
- Teile und Herrsche
  - Kurze Wiederholung: Prinzip
  - Beispiel Maximale Teilsumme
  - Laufzeitanalyse für Rekursionsalgorithmen
  - Mastertheorem – der Hauptsatz der Laufzeitfunktionen

Rückmeldungen Ü8 (Cache Effizienz)

UNI FREIBURG

Teile und Herrsche

UNI FREIBURG

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

Prinzip

UNI FREIBURG

- 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

Überblick

UNI FREIBURG

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

Maximale Teilsumme

UNI FREIBURG

- 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

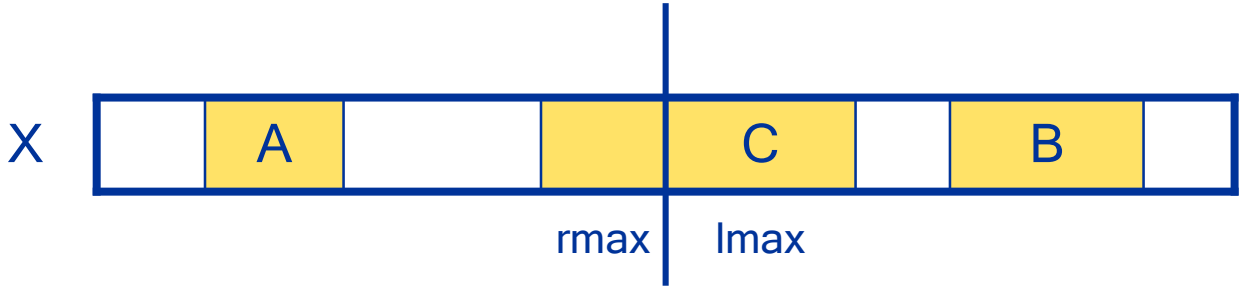
Überblick

UNI FREIBURG

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

Idee

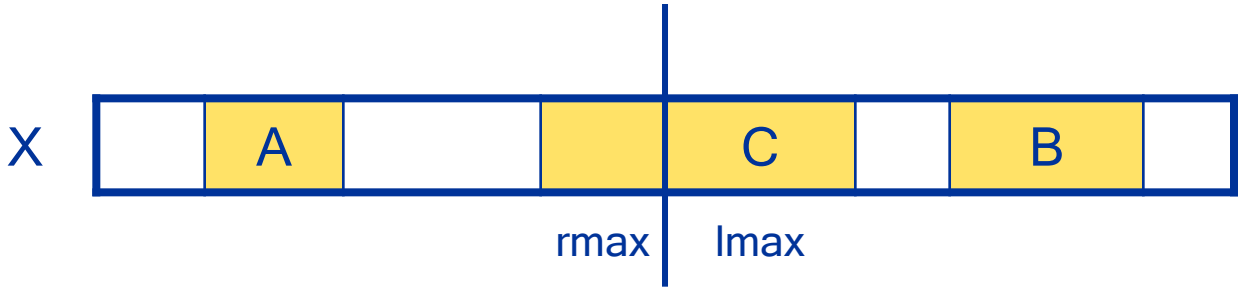
- Löse das Problem für die linke und rechte Hälfte von X.
- Setze die Teillösungen zur Gesamtlösung zusammen.



- 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 $\Rightarrow$ 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);

    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);
```

Trivialfall,
Feld mit nur einem Element
(Rekursionsabbruch)

Lösungen A und B für
die beiden Teilfelder
(Rekursion)

rmax und lmax für
den Grenzfall C

Lösung ergibt sich als
Maximum aus A, B, C

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

Java-Implementierung

UNI FREIBURG

```
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

UNI FREIBURG

```
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

UNI FREIBURG

```
public int maxSubArrayAlternative(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

UNI FREIBURG

```
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

UNI FREIBURG

```
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

UNI FREIBURG

```
max (a, b)

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

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

Java-Implementierung

UNI FREIBURG

```
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;
```

rmax analog

Java-Implementierung

UNI FREIBURG

```
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;
}
```

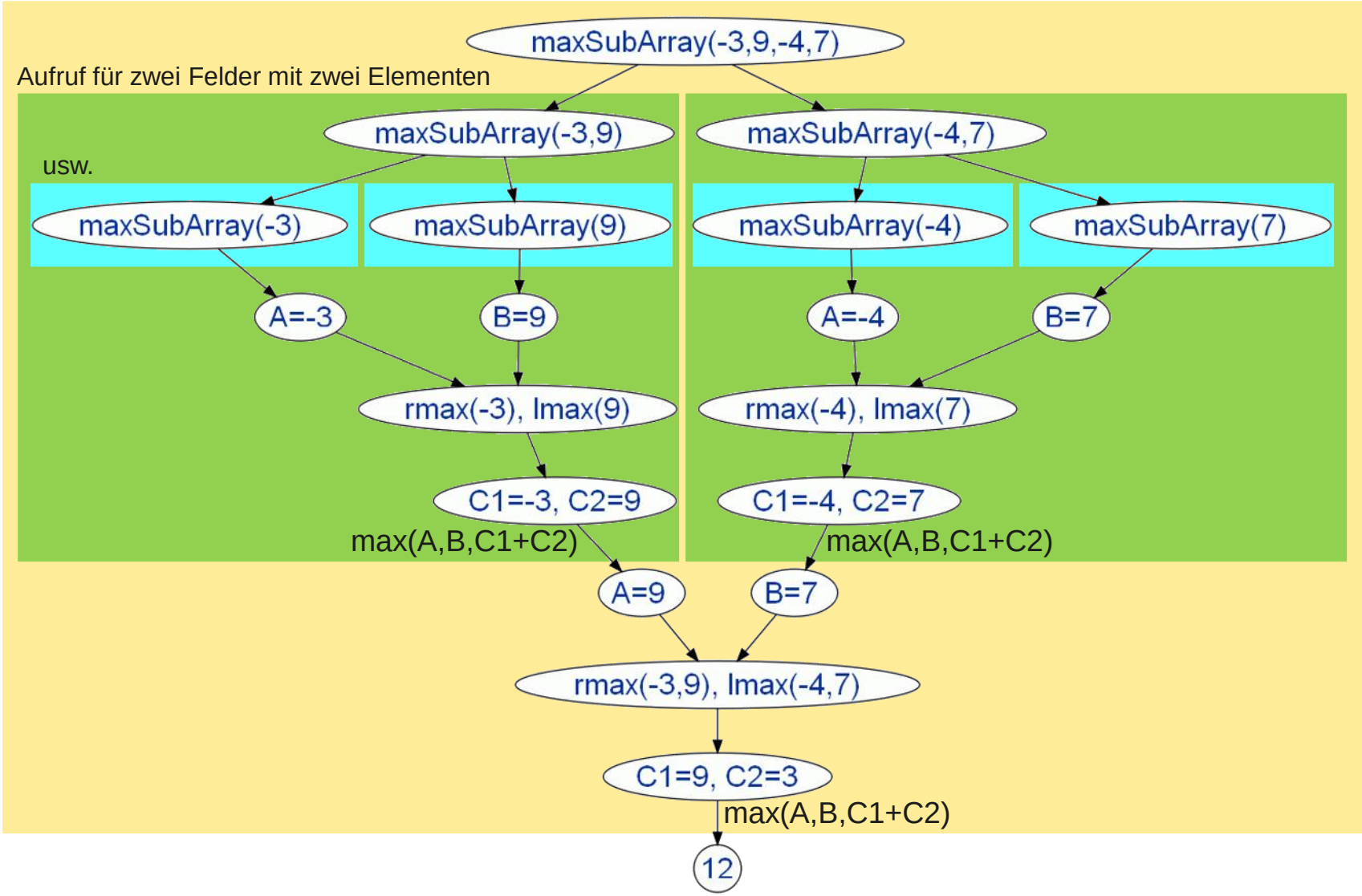
Illustration - lmax

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



Überblick

UNI FREIBURG

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

Laufzeit

UNI FREIBURG

```
maxSubArray(X, u, o)

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

    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);
```

T (n) – Zahl der Schritte für Problemgröße n

O(1)

O(1)

T(n/2)

T(n/2)

O(n)

O(n)

O(1)

Zahl der Schritte T(n)

Trivialfall

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

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)

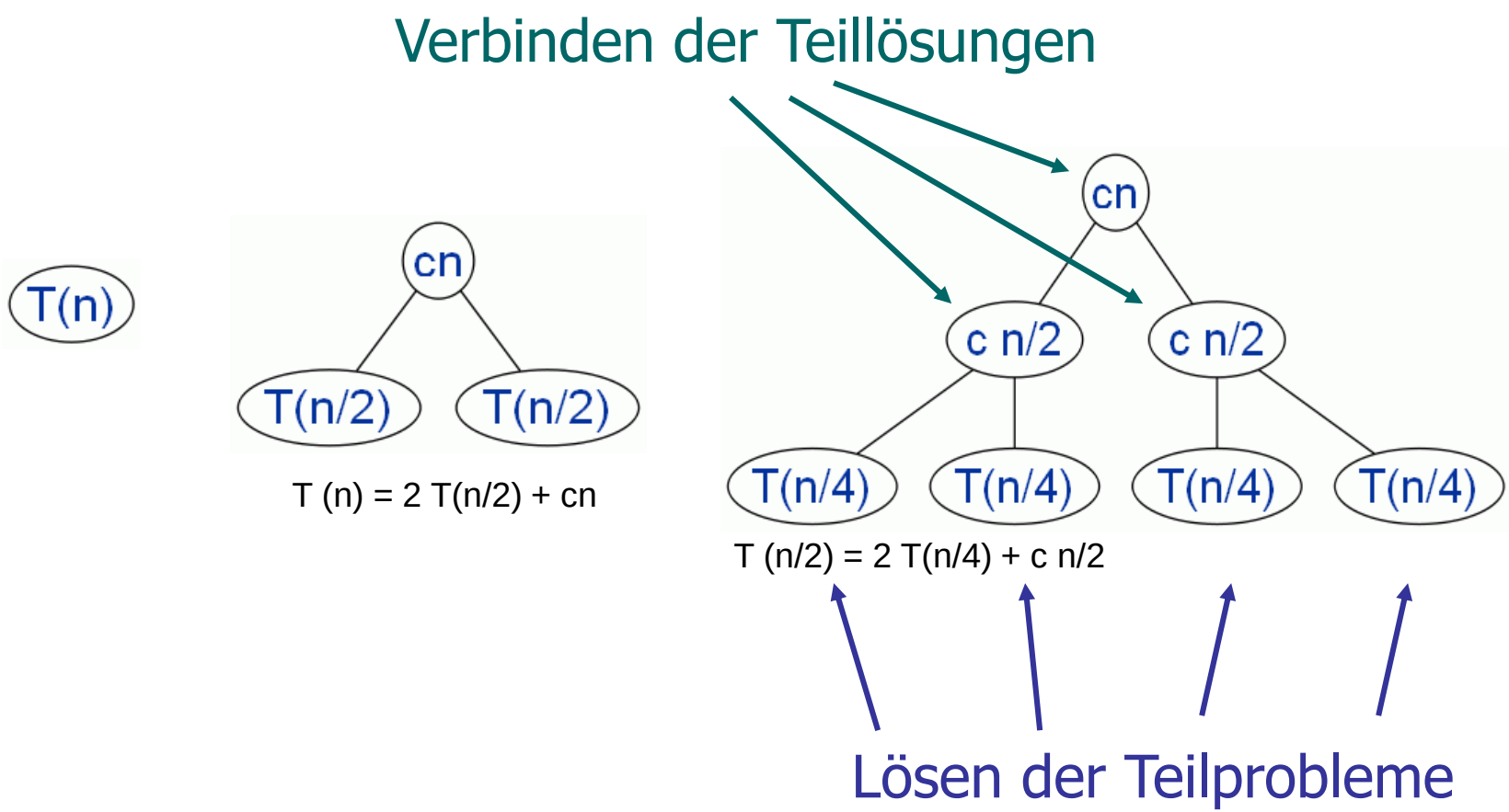
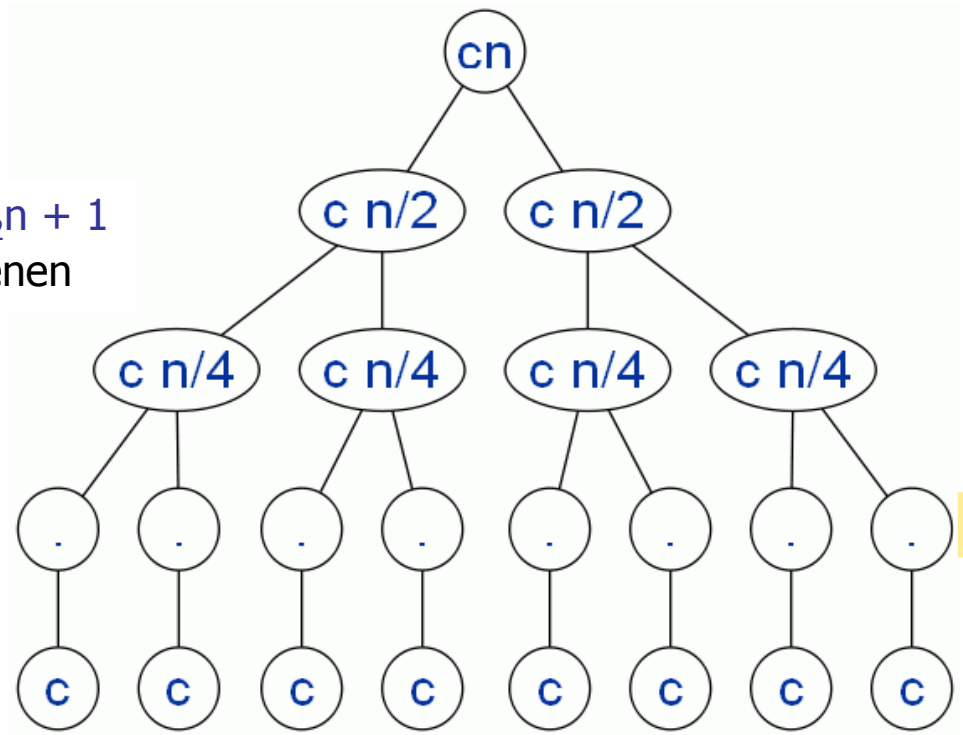


Illustration von T(n)



$\log_2 n + 1$ Ebenen

1 Knoten, der n Elemente verarbeitet = c·n

2 Knoten, die jeweils n/2 Elem. verarbeiten = c·n

4 Knoten, die jeweils n/4 Elem. verarbeiten = c·n

⋮

2^i Knoten, die jeweils $n/2^i$ Elem. verarbeiten = c·n

⋮

n Knoten, die jeweils 1 Elem. verarbeiten = c·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·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)

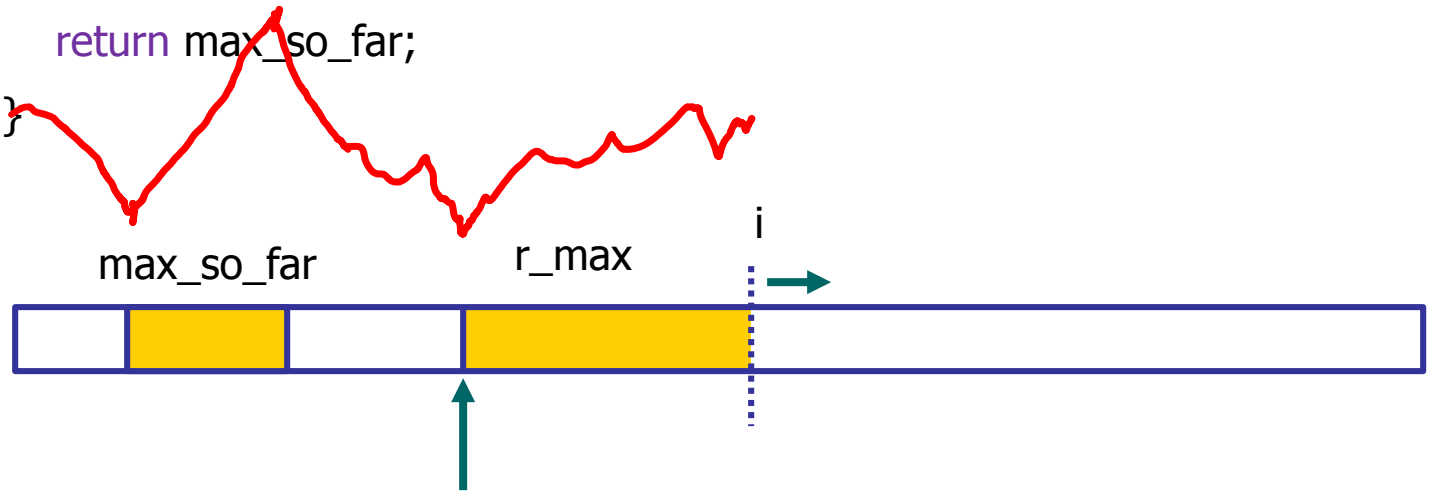
Zusammenfassung Maximale Teilsumme

UNI FREIBURG

- 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.

Maximum Subarray in O(n)

```
int maxSubarray(std::vector<int> A) {
    int r_max = 0;
    int max_so_far = 0;
    for (size_t  i = 0; i < A.size(); ++i) {
        r_max = std::max(0, r_max + A[i]);
        max_so_far = std::max(max_so_far, r_max);
    }
    return max_so_far;
}
```



Punkt, wo die Teilsumme zuletzt negativ wurde

Überblick

UNI FREIBURG

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

Rekursionsgleichungen…

- … beschreiben die Laufzeit bei Rekursionen

Trivialfall für n_0

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

Lösung von a Teilproblemen mit reduziertem Aufwand n/b

Splitten und 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:

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

Vermutung einsetzen

Substitutionsmethode

- alternative Vermutung
- 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:

$$\begin{aligned} T(n) &= 2 \cdot T\left(\tfrac{n}{2}\right) + n \\ &\leq 2 \cdot \left(c \cdot \tfrac{n}{2} \cdot \log_2 \tfrac{n}{2}\right) + n \qquad \text{Vermutung einsetzen} \\ &= c \cdot n \log_2 n - c \cdot n \cdot \log_2 2 + n \\ &= c \cdot n \log_2 n - c \cdot n + n \\ &\leq c \cdot n \log_2 n \qquad \text{für } c \geq 1 \end{aligned}$$

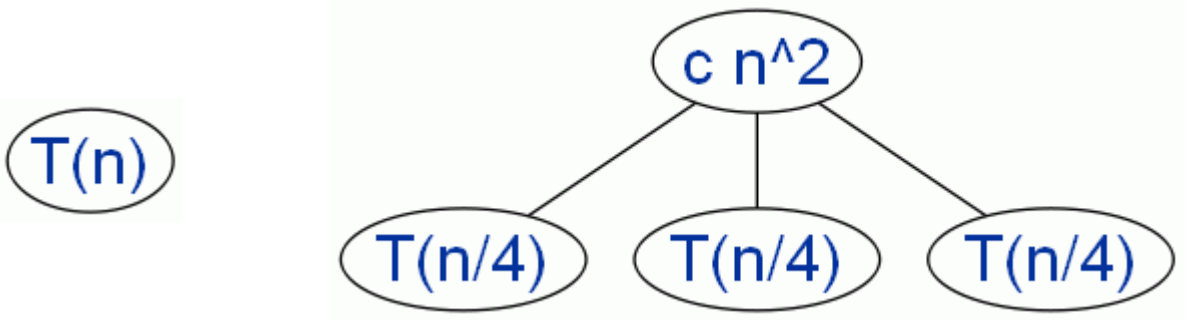
Überblick

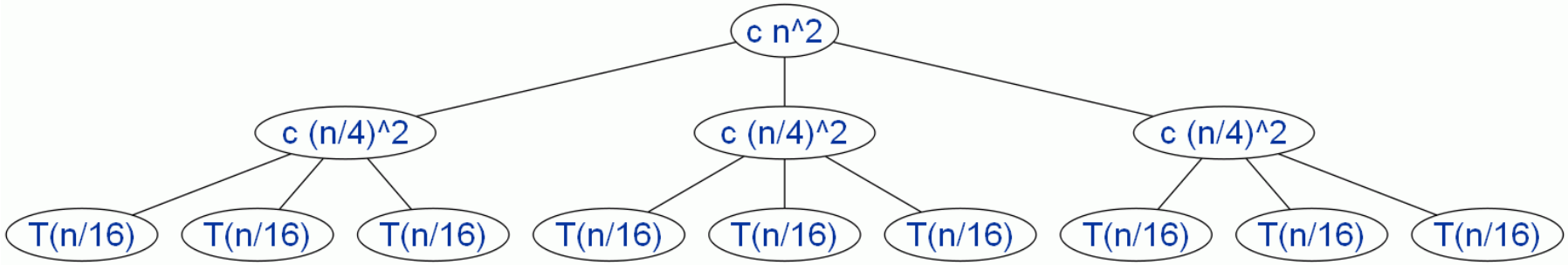
UNI FREIBURG

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

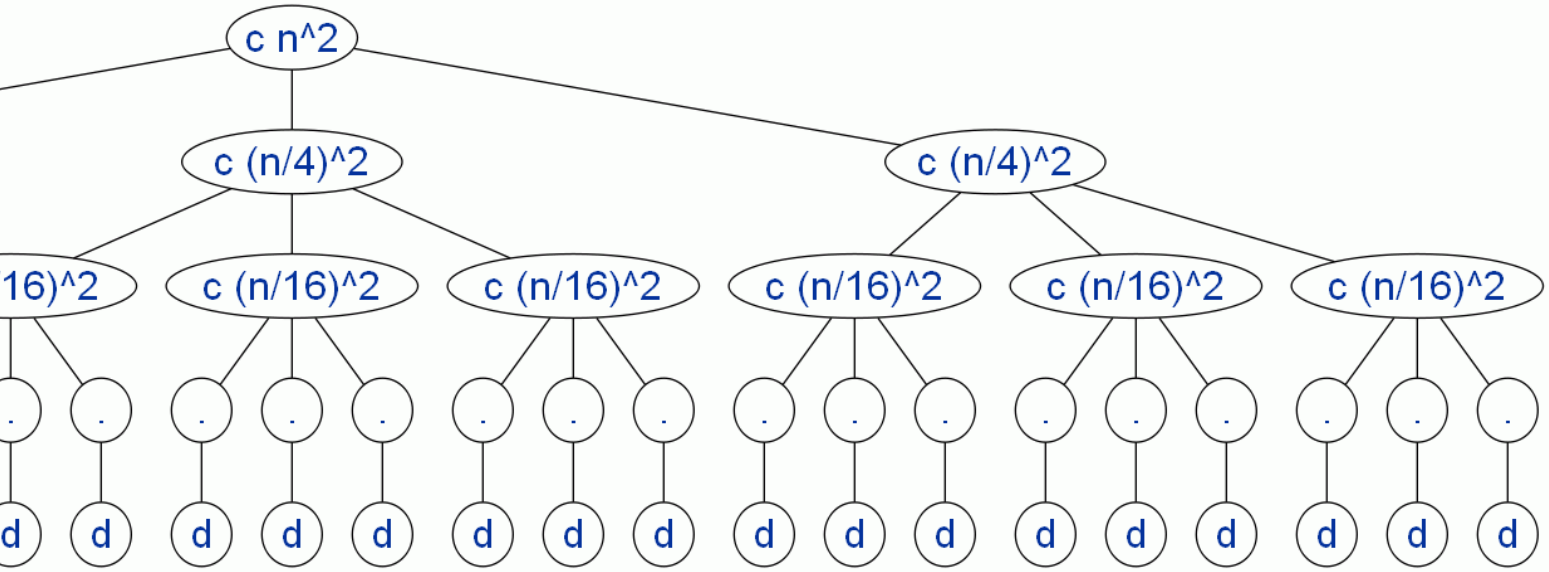
Rekursionsbaum-Methode Beispiel 1

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





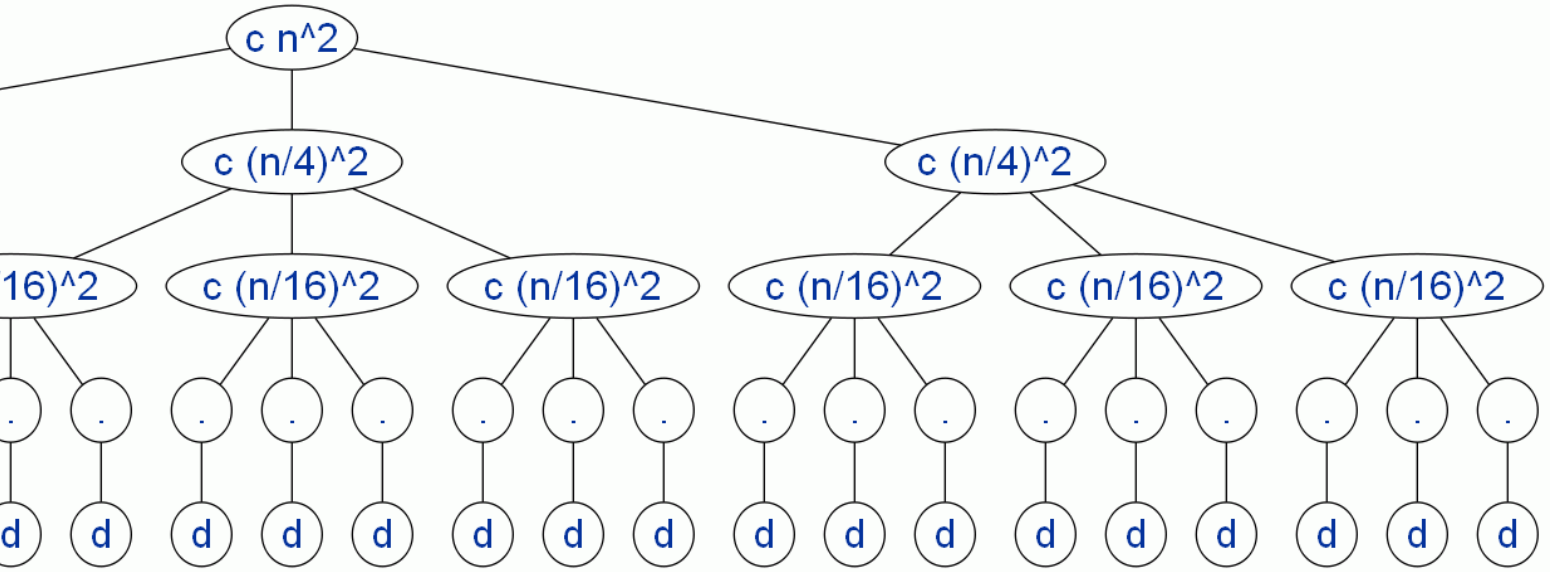
Rekursionsbaum-Methode



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: $d \cdot n^{\log_4 3}$

$\log(a^b) = b \cdot \log(a)$

Rechenspaß mit dem Logarithmus

Anzahl der Teilprobleme in der letzten Ebene (Blätter im Baum):

$$
\begin{aligned}
3^{\log_4 n} &= 3^{\log_4(3^{\log_3 n})} && \text{nutze } \log(a^b) = b \cdot \log a \\
&= 3^{(\log_3 n)\cdot \log_4 3} && \text{nutze } x^{a\cdot b} = (x^a)^b \\
&= \left(3^{\log_3 n}\right)^{\log_4 3} \\
&= n^{\log_4 3}
\end{aligned}
$$

(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:

$$\underbrace{\sum_{i=0}^{(\log_4 n)-1} \left(\frac{3}{16}\right)^i \cdot c \cdot n^2}_{\text{Geometrische Reihe, also konstant, (sogar bei unendlich vielen Ebenen)}} + \underbrace{d \cdot n^{\log_4 3}}_{\log_4 3 < 1, \text{ also wächst viel schwächer als } n^2} \quad \in \mathcal{O}(n^2)$$

→ hier: Kosten für Verbinden der Teillösungen dominieren

Einschub: Geometrische Reihe

UNI FREIBURG

- **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 $O(n^2)$

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

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

Vermutung einsetzen

Überblick

UNI FREIBURG

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

Mastertheorem

UNI FREIBURG

- Lösungsansatz für Rekursionsgleichungen der Form

  $T(n) = a\,T(n/b) + f(n)$ mit Konstanten $a \geq 1$ 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)

UNI FREIBURG

- 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)
- **Einfache Form:** Spezialfall, dass das Verbinden der Teillösungen $f(n)$ in $O(n)$ liegt

Mastertheorem (einfache Form)

- Für eine Rekursionsgleichung der Form

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

Bei der allgemeinen Form steht hier ein beliebiges f(n)

- Gilt:

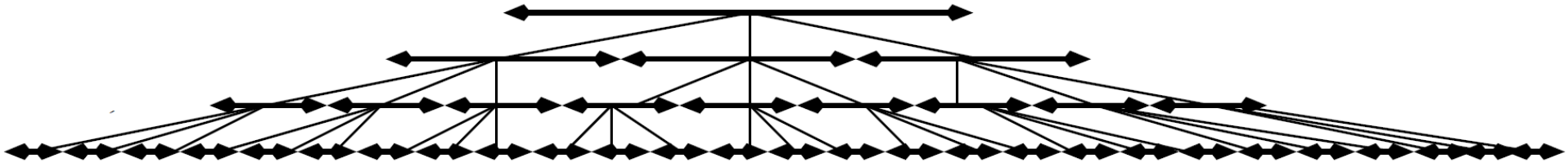
Anzahl der Blätter

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

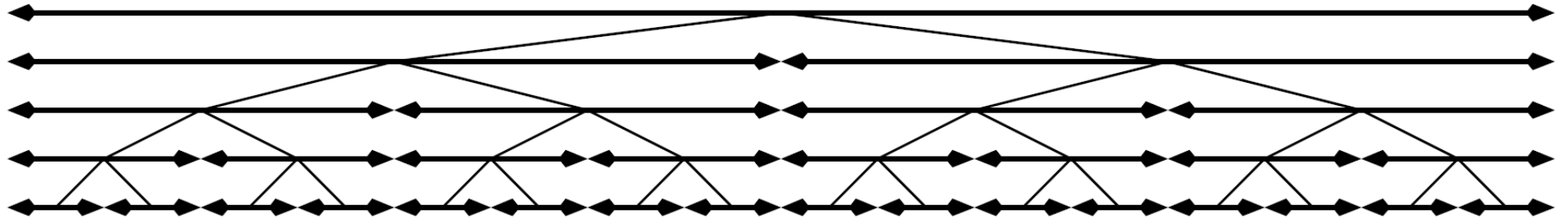
Mastertheorem (einfache Form)

UNI FREIBURG

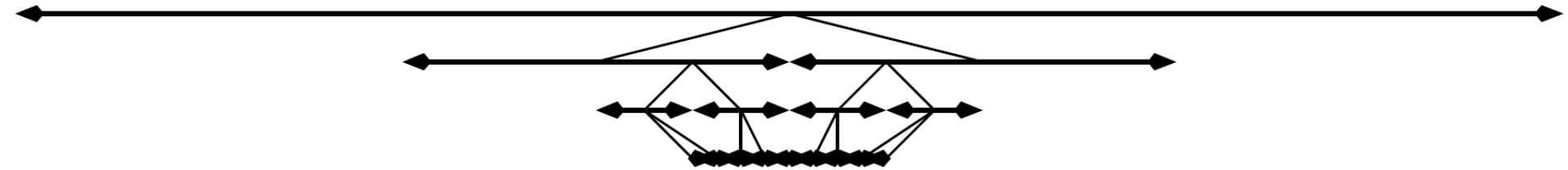
- Illustration des Aufwandes pro Ebene als Linien
  - Fall 1: a=3, b=2: drei Teilprobleme mit Größe ½



  - Fall 2: a=b=2: zwei Teilprobleme mit Größe ½



  - Fall 3: a=2, b=4: zwei Teilprobleme mit Größe ¼



Mastertheorem (einfache Form)

- Für eine Rekursionsgleichung der Form

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

- Gilt:

Anzahl der Blätter

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

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 $f(n) \in \mathcal{O}\left(n^{\log_b a - \epsilon}\right) \quad \epsilon > 0$
  - 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 $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right) \quad \epsilon > 0$
  - Operationen in erster Ebene (Verbinden der Teillösungen) dominieren die Kosten

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

$$n > n_0$$

Beispiele

UNI FREIBURG

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

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

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

  also n^3 Blätter

  $f(n) \in \mathcal{O}\left(n^{3-\epsilon}\right) \Rightarrow T(n) \in \Theta(n^3)$

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

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

  also n^2 Blätter

  $f(n) \in \mathcal{O}\left(n^{2-\epsilon}\right) \Rightarrow T(n) \in \Theta(n^2)$

Beispiele

- Fall 2: $T(n) \in \Theta\left(n^{\log_b a} \log n\right)$ falls $f(n) \in \Theta\left(n^{\log_b a}\right)$

- $T(n) = 2T\left(\frac{n}{2}\right) + 10n$

$$a = 2, \quad b = 2, \quad f(n) = 10n, \quad \log_b a = \log_2 2 = 1$$

also n^1 Blätter

$$f(n) \in \Theta\left(n^{\log_2 2}\right) \Rightarrow T(n) \in \Theta(n \log n)$$

- $T(n) = T\left(\frac{2n}{3}\right) + 1$

$$a = 1, \quad b = \tfrac{3}{2}, \quad f(n) = 1, \quad \log_b a = \log_{\frac{3}{2}} 1 = 0$$

also $n^0 = 1$ Blatt

$$f(n) \in \Theta\left(n^{\log_{\frac{3}{2}} 1}\right) \Rightarrow T(n) \in \Theta(n^0 \log n) = \Theta(\log n)$$

Beispiele

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

$$af\left(\tfrac{n}{b}\right) \leq cf(n) \qquad 0 \leq c \leq 1$$

$$n > n_0$$

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

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

also n^1 Blätter

$$f(n) \in \Omega\left(n^{1+\epsilon}\right)$$

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

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

Master-Theorem

UNI FREIBURG

- lässt sich nicht immer anwenden

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

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

also n^1 Blätter

- Fall 1: $f(n) \notin \mathcal{O}\left(n^{1-\epsilon}\right)$
- Fall 2: $f(n) \notin \Theta\left(n^1\right)$
- Fall 3: $f(n) \notin \Omega\left(n^{1+\epsilon}\right)$

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

Zusammenfassung

- Mastertheorem

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

  - Drei Fälle, je nachdem welcher Teil die Laufzeit dominiert.
  - **Fall 1:** Lösen der Trivialprobleme (letzte Ebene) ist **polynomial größer** als Verbinden der Teillösungen (erste Ebene)
    → $T(n) \in \Theta\left(n^{\log_b a}\right)$ T(n) ϵ Θ(Anzahl der Blätter)
  - **Fall 2:** Jede Ebene gleich teuer, (log n Ebenen)
    → $T(n) \in \Theta\left(n^{\log_b a} \log n\right)$
  - **Fall 3:** Verbinden der Teillösungen (erste Ebene) dominiert die Laufzeit
    → $T(n) \in \Theta\left(f(n)\right)$
  - Gilt nur für „polynomial größer", da sonst der Beweis mit der geometrischen Reihe nicht funktioniert

Übungsaufgabe

- Mastertheorem auf verschiedene Beispiele anwenden.