Kapitel 3 - Effizienz

- Laufzeit und Speicherbedarf eines Algorithmus
- Ordnung einer Funktion
- Beispiele: Suchen und Sortieren
- Untere Schranken für die Laufzeit

Effizienz
Überblick - Grundbegriffe

- Algorithmen sollen *effizient* sein, d.h. ein Problem mit wenigen "Betriebsmitteln" (Zeit, Speicher, Ein-Ausgabeeinheiten, Hilfsprogramme, Netzlast usw.) lösen
- Komplexität=Summe benötigter Betriebsmittel

Effizienz
Überblick - Komplexität

Umgangssprachliche Definition:

Die **Komplexität eines Algorithmus** ist der erforderliche Aufwand an Betriebsmitteln, den eine Implementierung des Algorithmus als Programm auf einem Computersystem benötigt.

Die **Komplexität eines Problems** ist die kleinstmögliche Komplexität eines Algorithmus, der das Problem löst.

Effizienz
Überblick - Komplexität

Sei P Algorithmus, der Problem π löst.

- Dann Komplexität von P obere Schranke für Komplexität von π.
- Umgekehrt: Komplexität von π untere Schranke für Komplexität von P.

Wichtigste Betriebsmittel: Laufzeit, Prozessorzahl, Speicherplatzbedarf, Kommunikationsbedarf.

In dieser Vorlesung: vorwiegend Laufzeitanalysen.

Im folgenden: "effizient" = "in möglichst kurzer Zeit".

Effizienz
Laufzeit und Speicherbedarf

Betrachten Problem π gegeben durch funktionale Spezifikation:

S_π: spec $f: X_\pi \rightarrow Y_\pi$ with

f(x)=y where

pre P(x)

post Q(x,y).

Effizienz
Laufzeit/Speicherbedarf - 3 Beispiele

Problem 1: Sortieren

σ sei das Problem, $n \geq 1$ ganze Zahlen $x_1, x_2, ..., x_n$ aufsteigend zu sortieren. Dann ist

$$X_\sigma = \{(x_1, ..., x_n) \mid n \geq 1, x_k \in Z, k = 1, ..., n\} \quad \text{und}$$

$$Y_\sigma = \{(x_i^1, ..., x_i^n) \mid x_i^k \in Z, n \geq 1, \text{mit } x_i^1 \leq x_i^2 \leq \ ... \ \leq x_i^n\}.$$

$f_\sigma: X_\sigma \rightarrow Y_\sigma$ liefert zu jeder Zahlenfolge $x \in X_\sigma$ die zugehörige aufsteigend sortierte Folge $f_\sigma(x) = y \in Y_\sigma$.

Effizienz
Laufzeit/Speicherbedarf - 3 Beispiele

Problem 2: Multiplikation

μ sei das Problem, zwei natürliche Zahlen a und b zu multiplizieren. Dann ist

$X_\mu = \{(a,b) \mid a,b \in IN\}$ und $Y_\mu = IN$.

f_μ liefert zu je zwei natürlichen Zahlen a,b das Produkt $f_\mu(a,b) = a*b$.

Effizienz
Laufzeit/Speicherbedarf - 3 Beispiele

Problem 3: Teilworterkennung

Sei A eine endliche Menge. η sei das Problem, zu einem Element $x \in A$ und einem Wort $w \in A^*$ festzustellen, ob x in w vorkommt oder nicht. Dann ist:

$$X_\eta = \{(x,w) \mid x \in A \text{ und } w \in A^*\}, \quad A_\eta = \{\text{"ja"}, \text{"nein"}\}.$$

f_η liefert zu jedem Paar (x,w) die Antwort "ja", falls x in w vorkommt, und sonst "nein".

Effizienz

Laufzeit/Speicherbedarf - Definition

Definition

Sei P eine Implementierung von S_π. Dann bezeichnen wir für $x \in X_\pi$ mit $\tau_P(x)$ bzw. $\sigma_P(x)$ die Laufzeit bzw. den *neben der Eingabe zusätzlichen* Speicherplatz, den P benötigt, um die korrekte Antwort auf die Eingabe x zu ermitteln. τ_P bzw. σ_P sind folglich Abbildungen der Funktionalität $X_\pi \rightarrow \mathbb{N}$.

Effizienz
Laufzeit/Speicherbedarf - Definition

- $\tau_P(x)$: Anzahl der elementaren Einzelschritte, um f(x) zu ermitteln, und für jeden Einzelschritt benötigte Zeit
- τ_P experimentell mit einer Stoppuhr bestimmen
- $\sigma_P(x)$: Zahl der Speicherzellen, die P benötigt
- Eingabe zählt nicht mit

Effizienz
Laufzeit/Speicherbedarf - Abstraktionen

4 aufeinander aufbauende Abstraktionen:

- τ_P und σ_P will man nicht für jede konkrete Eingabe exakt wissen => Zusammenfassung von Eingaben zu sinnvollen Klassen gleichschwerer Probleme
- Auswahl eines Repräsentanten, dessen Laufzeit maßgebend ist für die Klasse => Laufzeit im schlimmsten Fall (worst case)
- Übergang zu einem genormten Rechner => Einheitskostenrechnermodell
- Übergang zum qualitativen Verlauf von τ_P und σ_P => Einführung der (Größen-)Ordnung

Effizienz

Laufzeit/Speicherbedarf - Eingabenlänge

- Klassenbildung über die Länge der Eingabe
- Was ist die Länge?
  - Multiplikation: Summe der Ziffernanzahl der beiden Faktoren

Effizienz
Laufzeit/Speicherbedarf - Eingabenlänge

- Sortierproblem: zunächst unklar.
  - Anzahl der zu sortierenden Zahlen, also n?
  - Summe der Längen aller zu sortierenden Zahlen, also $|x_1|+...+|x_n|$?
  - Länge $|x_i|$ wieder abhängig von Darstellung der Zahlen:
    - a = a Striche => $|a|=a$
    - a zur Basis 2 => $|a| \approx \log_2(a)$.

Effizienz
Laufzeit/Speicher - Längenfunktion

- Formal: zu Problem π Längenfunktion

$$L_\pi: X_\pi \rightarrow \mathbb{N},$$

- die zu Eingabe $x \in X$ Länge $L_\pi(x)$ festlegt.
- Beispiele:
  - Sortierproblem σ: $L_\sigma(x_1,...,x_n)=n$,
  - Multiplikation μ: $L_\mu(a,b)$="Anzahl der Ziffern von a"+"Anzahl der Ziffern von b" +"zwei Vorzeichen"=$[\log_{10} a]+[\log_{10} b]+4$

Effizienz
Laufzeit/Speicher - Längenfunktion

- Formal: zu Problem π Längenfunktion

$$L_\pi: X_\pi \rightarrow \mathbb{N},$$

- die zu Eingabe $x \in X$ Länge $L_\pi(x)$ festlegt.
- Beispiele:
  - Sortierproblem σ: $L_\sigma(x_1$
  - Multiplikation μ: $L_\mu(a,$ ffern von a"+"Anzahl der Zi zwei Vorzeichen"=$[\log_{10} a]+[\log_{10} b]+4$

[u]: größte ganze Zahl ≤ u

Effizienz
Laufzeit/Speicherbedarf - worst case

4 aufeinander aufbauende Abstraktionen:

- T_P und σ_P will man nicht für jede konkrete Eingabe exakt wissen => Zusammenfassung von Eingaben zu sinnvollen Klassen gleichschwerer Probleme ✓
- Auswahl eines Repräsentanten, dessen Laufzeit maßgebend ist für die Klasse => Laufzeit im schlimmsten Fall (worst case)
- Übergang zu einem genormten Rechner => Einheitskostenrechnermodell
- Übergang zum qualitativen Verlauf von T_P und σ_P => Einführung der (Größen-)Ordnung

Effizienz
Laufzeit/Speicherbedarf - worst case

- Situation: Zu Eingaben gleicher Länge benötigt ein Algor. ganz unterschiedliche Laufzeiten und Speicher, z.B. beim Sortieren n unsortierte versus n bereits sortierte Zahlen
- Wähle schlimmsten Fall (worst case): Eingabe der Länge n, für die Algor. längste Zeit bzw. meisten Speicher benötigt, bestimmt Zeit und Speicher aller Eingaben der Länge n
- Alternativen:
  - bester Wert: aussagelos
  - mittlerer Wert: Häufigkeitsverteilung oft schwer zu bestimmen

Effizienz

Laufzeit/Speicherbedarf - worst case

Definition:

Sei π ein Problem mit Spezifikation S_π und P ein Programm, das S_π implementiert.

Die **Laufzeit** T_P **im schlimmsten Fall** (**worst case**) des Programms P ist eine Abbildung

$$T_P: IN \rightarrow IN \text{ mit}$$

$$T_P(n)=max\{\tau_P(x) \mid x \in X_\pi \text{ und } L_\pi(x)=n\}.$$

Effizienz

Laufzeit/Speicherbedarf - worst case

Definition (Forts.):

Der Speicherbedarf S_P im schlimmsten Fall des Programms P ist eine Abbildung

$$S_P: IN \rightarrow IN \text{ mit}$$

$$S_P(n)=\max\{\sigma_P(x) \mid x \in X_\pi \text{ und } L_\pi(x)=n\}.$$

Im folgenden immer worst case gemeint.

Effizienz

Laufzeit/Speicherbedarf - Beispiel

Programm P liest ganze Zahl x ein und stellt fest, ob in der nachfolgenden Zahlenfolge x vorkommt oder nicht (Problem η):

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x);  {Zeit C}       ende:=false;  {Zeit C'}
    while not (eof or ende) do  {Zeit C"}
    begin
        read(y);  {Zeit C}       ende:=x=y  {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Effizienz

Laufzeit/Speicherbedarf - Beispiel

Programm P liest ganze Zahl x ein und stellt fest, ob in der nachfolgenden Zahlenfolge x vorkommt oder nicht (Problem η):

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x); {Zeit C}     ende:=false; {Zeit C'}
    while not (eof or ende) do {Zeit C''}
    begin
        read(y); {Zeit C}     ende:=x=y {Zeit C'''}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden') {Zeit C''''}
end.
```

Effizienz
Laufzeit/Speicherbedarf - Beispiel

- Längenfunktion

$L_\eta: X_\eta \rightarrow IN$ mit $X_\eta = \{(x, y_1,...,y_n) | \; x,y_1,...,y_n \in Z, \; n \geq 0\}$

$$L_\eta(x, y_1,...,y_n) = n+1$$

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x);  {Zeit C}
    ende:=false;  {Zeit C'}
    while not (eof or ende) do  {Zeit C"}
    begin
        read(y);  {Zeit C}
        ende:=x=y  {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Effizienz

Laufzeit/Speicherbedarf - Beispiel

Eingabe der Länge l: $T_P(l)$

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x);  {Zeit C}
    ende:=false;  {Zeit C'}
    while not (eof or ende) do  {Zeit C"}
    begin
        read(y);  {Zeit C}
        ende:=x=y  {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x);  {Zeit C}
    ende:=false;  {Zeit C'}
    while not (eof or ende) do  {Zeit C"}
    begin
        read(y);  {Zeit C}
        ende:=x=y  {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Eingabe der Länge l: $T_P(l)$

$T_P(l)=$

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x); {Zeit C}
    ende:=false;  {Zeit C'}
    while not (eof or ende) do  {Zeit C"}
    begin
        read(y);  {Zeit C}
        ende:=x=y  {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Eingabe der Länge l: $T_P(l)$

$T_P(l)=$

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x); {Zeit C}
    ende:=false;  {Zeit C'}
    while not (eof or ende) do  {Zeit C"}
    begin
        read(y);  {Zeit C}
        ende:=x=y  {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Eingabe der Länge l: $T_P(l)$

$T_P(l) = C$

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x); {Zeit C}
    ende:=false; {Zeit C'}
    while not (eof or ende) do  {Zeit C"}
    begin
        read(y);  {Zeit C}
        ende:=x=y  {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Eingabe der Länge l: $T_P(l)$

$T_P(l) = C$

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x); {Zeit C}
    ende:=false; {Zeit C'}
    while not (eof or ende) do  {Zeit C"}
    begin
        read(y);  {Zeit C}
        ende:=x=y  {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Eingabe der Länge l: $T_P(l)$

$T_P(l) = C + C'$

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x); {Zeit C}
    ende:=false; {Zeit C'}
    while not (eof or ende) do {Zeit C"}
    begin
        read(y); {Zeit C}
        ende:=x=y {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden') {Zeit C""}
end.
```

Eingabe der Länge l: $T_P(l)$

$T_P(l) = C + C'$

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x); {Zeit C}
    ende:=false; {Zeit C'}
    while not (eof or ende) do {Zeit C"}
    begin
        read(y); {Zeit C}
        ende:=x=y {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden') {Zeit C""}
end.
```

Eingabe der Länge l: $T_P(l)$

$T_P(l) = C + C' + C''$

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x); {Zeit C}
    ende:=false; {Zeit C'}
    while not (eof or ende) do {Zeit C"}
    begin
        read(y); {Zeit C}
        ende:=x=y {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden') {Zeit C""}
end.
```

Eingabe der Länge l: $T_P(l)$

$T_P(l) = C + C' + C''$

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x); {Zeit C}
    ende:=false; {Zeit C'}
    while not (eof or ende) do {Zeit C"}
    begin
        read(y); {Zeit C}
        ende:=x=y {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden') {Zeit C""}
end.
```

Eingabe der Länge l: $T_P(l)$

$T_P(l)=$	C	+C'	+C"	+C""

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x);  {Zeit C}
    ende:=false;  {Zeit C'}
    while not (eof or ende) do  {Zeit C"}
    begin
        read(y);  {Zeit C}
        ende:=x=y  {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Effizienz

Laufzeit/Speicherbedarf - Beispiel

Eingabe der Länge 2: $T_P(2)$

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x);  {Zeit C}
    ende:=false;  {Zeit C'}
    while not (eof or ende) do  {Zeit C"}
    begin
        read(y);  {Zeit C}
        ende:=x=y  {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x);  {Zeit C}
    ende:=false;  {Zeit C'}
    while not (eof or ende) do  {Zeit C"}
    begin
        read(y);  {Zeit C}
        ende:=x=y  {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Eingabe der Länge 2: $T_P(2)$

$T_P(2)=$

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x); {Zeit C}
    ende:=false;  {Zeit C'}
    while not (eof or ende) do  {Zeit C"}
    begin
        read(y);  {Zeit C}
        ende:=x=y  {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Eingabe der Länge 2: $T_P(2)$

$T_P(2)=$

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x); {Zeit C}
    ende:=false;  {Zeit C'}
    while not (eof or ende) do  {Zeit C"}
    begin
        read(y);  {Zeit C}
        ende:=x=y  {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Eingabe der Länge 2: $T_P(2)$

$T_P(2) =$ C

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x); {Zeit C}
    ende:=false; {Zeit C'}
    while not (eof or ende) do  {Zeit C"}
    begin
        read(y);  {Zeit C}
        ende:=x=y  {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Eingabe der Länge 2: $T_P(2)$

$T_P(2) = C$

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x); {Zeit C}
    ende:=false; {Zeit C'}
    while not (eof or ende) do  {Zeit C"}
    begin
        read(y);  {Zeit C}
        ende:=x=y  {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Eingabe der Länge 2: $T_P(2)$

$T_P(2) = C + C'$

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x); {Zeit C}
    ende:=false; {Zeit C'}
    while not (eof or ende) do {Zeit C"}
    begin
        read(y); {Zeit C}
        ende:=x=y {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden') {Zeit C""}
end.
```

Eingabe der Länge 2: $T_P(2)$

$T_P(2) = C + C'$

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x); {Zeit C}
    ende:=false; {Zeit C'}
    while not (eof or ende) do {Zeit C"}
    begin
        read(y); {Zeit C}
        ende:=x=y {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden') {Zeit C""}
end.
```

Eingabe der Länge 2: $T_P(2)$

$T_P(2)=$	C	+C'	+C"

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x);  {Zeit C}
    ende:=false;  {Zeit C'}
    while not (eof or ende) do  {Zeit C"}
    begin
        read(y);  {Zeit C}
        ende:=x=y  {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Eingabe der Länge 2: $T_P(2)$

$T_P(2) = C + C' + C''$

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x); {Zeit C}
    ende:=false; {Zeit C'}
    while not (eof or ende) do {Zeit C"}
    begin
        read(y); {Zeit C}
        ende:=x=y  {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Eingabe der Länge 2: $T_P(2)$

$T_P(2)=$	C	+C'	+C"	+C

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x);  {Zeit C}
    ende:=false;  {Zeit C'}
    while not (eof or ende) do  {Zeit C"}
    begin
        read(y);  {Zeit C}
        ende:=x=y  {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Eingabe der Länge 2: $T_P(2)$

$T_P(2)=$	C	+C'	+C"	+C

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x); {Zeit C}
    ende:=false; {Zeit C'}
    while not (eof or ende) do {Zeit C"}
    begin
        read(y); {Zeit C}
        ende:=x=y {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Eingabe der Länge 2: $T_P(2)$

$T_P(2) = C + C' + C'' + C''' + C$

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x);  {Zeit C}
    ende:=false;  {Zeit C'}
    while not (eof or ende) do  {Zeit C"}
    begin
        read(y);  {Zeit C}
        ende:=x=y  {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Eingabe der Länge 2: $T_P(2)$

$T_P(2) = C + C' + C" + C + C'"$

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x);                          {Zeit C}
    ende:=false;                      {Zeit C'}
    while not (eof or ende) do        {Zeit C"}
    begin
        read(y);                      {Zeit C}
        ende:=x=y                     {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')  {Zeit C""}
end.
```

Eingabe der Länge 2: $T_P(2)$

$T_P(2) = C + C' + C" + C + C'" + C"$

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x);                          {Zeit C}
    ende:=false;                      {Zeit C'}
    while not (eof or ende) do        {Zeit C"}
    begin
        read(y);                      {Zeit C}
        ende:=x=y                     {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden')   {Zeit C""}
end.
```

Eingabe der Länge 2: $T_P(2)$

$T_P(2) = C + C' + C" + C + C'" + C"$

Effizienz

Laufzeit/Speicherbedarf - Beispiel

```
program P(input,output);
var x,y: integer;
    ende: boolean;
begin
    read(x); {Zeit C}
    ende:=false; {Zeit C'}
    while not (eof or ende) do {Zeit C"}
    begin
        read(y); {Zeit C}
        ende:=x=y {Zeit C'"}
    end;
    if ende then writeln ('Zahl ist vorhanden')
        else writeln ('Zahl ist nicht vorhanden') {Zeit C""}
end.
```

Eingabe der Länge 2: $T_P(2)$

$T_P(2)$= C +C' +C" +C +C'" +C" +C""

Effizienz

Laufzeit/Speicherbedarf - Beispiel

- Eingabe der Länge 3
  - gesuchte Zahl x als erste der beiden Zahlen

    $$2C+C'+2C''+C'''+C'''',$$
  - gesuchte Zahl x als zweite der beiden Zahlen: weiterer Schleifendurchlauf und Zeit C+C"+C'" kommt hinzu. Gesamtzeit

    $$3C+C'+3C''+2C'''+C''''.$$
  - Definition: Laufzeit im schlimmsten Fall definiert definiert Laufzeit für Länge 3:

    $$T_P(3)=3C+C'+3C''+2C'''+C''''.$$

Effizienz
Laufzeit/Speicherbedarf - Beispiel

- Allgemein: Eingabe der Länge $n \geq 2$
  - 2C+C'+2C"+C'"+C"", falls x an 1. Stelle
  - 3C+C'+3C"+2C'"+C"", falls x an 2. Stelle
  - 4C+C'+4C"+3C'"+C"", falls x an 3. Stelle
  - ...
  - nC+C'+nC"+(n-1)C'"+C"", falls x an letzter Stelle oder gar nicht vorkommt.
- Schlimmster Fall: Maximum aller Zeiten:

$$T_P(n)=nC+C'+nC"+(n-1)C'"+C""$$

- Der Speicherbedarf ist konstant 3, also $S_P(n)=3$ für alle n.

Effizienz

Laufzeit/Speicherbedarf - Verbesserung

- Unhandlich: viele Konstanten C, C', C'', ... für jede Elementaranweisung und jeden Vergleich
- $T_P(n)$ wird für größere Programme unleserlich
- Konstanten wegabstrahieren !

Effizienz
Laufzeit/Speicherbedarf - unit cost

4 aufeinander aufbauende Abstraktionen:

- T_P und σ_P will man nicht für jede konkrete Eingabe exakt wissen => Zusammenfassung von Eingaben zu sinnvollen Klassen gleichschwerer Probleme ✓
- Auswahl eines Repräsentanten, dessen Laufzeit maßgebend ist für die Klasse => Laufzeit im schlimmsten Fall (worst case) ✓
- Übergang zu einem genormten Rechner => Einheitskostenrechnermodell
- Übergang zum qualitativen Verlauf von T_P und σ_P => Einführung der (Größen-)Ordnung

Effizienz
Einheitskostenrechnermodell

- Genormter Rechner, auf dem Laufzeit und Speicherbedarf aller Programme ermittelt werden
  - je Elementaroperation eine Zeiteinheit unabhängig von Größe der Operanden
  - je elementarem Datum eine Speicherzelle unabhängig von Größe der Daten
- Analogien: DIN-Verbrauch beim Auto, Einheitswert von Grundstücken, Warenkorb für Inflation, ...

Effizienz
Einheitskostenrechnermodell

- Genormter Rechner, auf dem Laufzeit und Speicherbedarf aller Programme ermittelt werden
  - je Elementaroperation eine Zeiteinheit unabhängig von Größe der Operanden
  - je elementarem Datum eine Speicherzelle unabhängig von Größe der Daten
- Analogien: DIN-Verbrauch beim Auto, Einheitswert von Grundstücken, Warenkorb für Inflation, ...

Effizienz
Einheitskostenrechnermodell

- Genormter Rechner, auf d[illegible] und Speicherbedarf aller Pro[illegible]elt werden

  - je Elementaroperation eine Zeiteinheit unabhängig von Größe der Operanden
  - je elementarem Datum eine Speicherzelle unabhängig von Größe der Daten

- Analogien: DIN-Verbrauch beim Auto, Einheitswert von Grundstücken, Warenkorb für Inflation, ...

Effizienz
Einheitskostenrechnermodell

- Das Einheitskostenmodell ist realistisch
  - solange auftretende Operanden (z.B. in Zwischenrechnungen) nicht beliebig groß werden, sondern sich immer in der Größenordnung der Länge der Eingabe bewegen
- Ausnahme: arithmetische Algorithmen, wie z.B. Multiplikationsalgorithmen
- hier: logarithmisches Kostenmodell

Effizienz
log. Kostenmodell

- Längenfunktion:

$$L(n)=[\log_2 n]+1.$$

- elementare arithmetische Operationen mit Operanden a und b nicht ein Rechenschritt, sondern

$$L(a)+L(b)=[\log_2 a]+[\log_2 b]+2$$

  Rechenschritte

- Hier in der Folge nur Einheitskostenmodell.

Effizienz

Laufzeit/Speicherbedarf - obiges Beispiel

- Durchsuchen einer Folge:
  - Schlimmster Fall:

    $$T_P(n)=nC+C'+nC''+(n-1)C'''+C''''$$

  - Einheitskostenmodell: $C=C'=C'''=1$, $C''=C''''=2$
  - Laufzeit im schlimmsten Falle folgt dann

    $$T_P(n)=4n+2$$

Effizienz

Laufzeit/Speicherbedarf - unit cost

4 aufeinander aufbauende Abstraktionen:

- τ_P und σ_P will man nicht für jede konkrete Eingabe exakt wissen => Zusammenfassung von Eingaben zu sinnvollen Klassen gleichschwerer Probleme ✓
- Auswahl eines Repräsentanten, dessen Laufzeit maßgebend ist für die Klasse => Laufzeit im schlimmsten Fall (worst case) ✓
- Übergang zu einem genormten Rechner => Einheitskostenrechnermodell ✓
- Übergang zum qualitativen Verlauf von τ_P und σ_P => Einführung der (Größen-)Ordnung

Effizienz
Laufzeit/Speicherbedarf - Ordnung

- Motivation: 4 Algorithmen A, B, C, D
  - $T_A(n)=100n+30$
  - $T_B(n)=100n \cdot \log_2 n$,
  - $T_C(n)=10n^2$
  - $T_D(n)=2^n$.
- Welcher ist der beste?

Effizienz
Laufzeit/Speicherbedarf - Ordnung

- Motivation: 4 Algorithmen A, B, C, D
  - $T_A(n)=100n+30$
  - $T_B(n)=100n \cdot \log_2 n$,
  - $T_C(n)=10n^2$
  - $T_D(n)=2^n$.

← $2 \leq n \leq 9$

- Welcher ist der beste?

Effizienz
Laufzeit/Speicherbedarf - Ordnung

- Motivation: 4 Algorithmen A, B, C, D
  - $T_A(n)=100n+30$
  - $T_B(n)=100n \cdot \log_2 n$,
  - $T_C(n)=10n^2$
  - $T_D(n)=2^n$.
- Welcher ist der beste?

Effizienz
Laufzeit/Speicherbedarf - Ordnung

- Motivation: 4 Algorithmen A, B, C, D
  - $T_A(n)=100n+30$
  - $T_B(n)=100n \cdot \log_2 n$,
  - $T_C(n)=10n^2$
  - $T_D(n)=2^n$.
- Welcher ist der beste?

← n=10

Effizienz
Laufzeit/Speicherbedarf - Ordnung

- Motivation: 4 Algorithmen A, B, C, D
  - $T_A(n)=100n+30$
  - $T_B(n)=100n \cdot \log_2 n$,
  - $T_C(n)=10n^2$
  - $T_D(n)=2^n$.
- Welcher ist der beste?

Effizienz
Laufzeit/Speicherbedarf - Ordnung

- Motivation: 4 Algorithmen A, B, C, D
  - $T_A(n)=100n+30$ ← n>10
  - $T_B(n)=100n \cdot \log_2 n$,
  - $T_C(n)=10n^2$
  - $T_D(n)=2^n$.
- Welcher ist der beste?

Effizienz
Laufzeit/Speicherbedarf - Ordnung

- Motivation: 4 Algorithmen A, B, C, D
  - $T_A(n)=100n+30$
  - $T_B(n)=100n \cdot \log_2 n$,
  - $T_C(n)=10n^2$
  - $T_D(n)=2^n$.
- Welcher ist der beste?

Effizienz
Laufzeit/Speicherbedarf - Ordnung

- Motivation: 4 Algorithmen A, B, C, D
  - $T_A(n)=100n+30$
  - $T_B(n)=100n \cdot \log_2 n$, ← nie
  - $T_C(n)=10n^2$
  - $T_D(n)=2^n$.
- Welcher ist der beste?

Effizienz
Laufzeit/Speicherbedarf - Ordnung

- Motivation: 4 Algorithmen A, B, C, D
  - $T_A(n)=100n+30$
  - $T_B(n)=100n \cdot \log_2 n$,
  - $T_C(n)=10n^2$
  - $T_D(n)=2^n$.
- Welcher ist der beste?

Effizienz
Laufzeit/Speicherbedarf - Ordnung

- Motivation: 4 Algorithmen A, B, C, D
  - $T_A(n)=100n+30$
  - $T_B(n)=100n \cdot \log_2 n$,
  - $T_C(n)=10n^2$
  - $T_D(n)=2^n$.
- Welcher ist der beste?

Bester Algorithmus ist der, der ab einer bestimmten Eingabelänge immer der beste ist.

Effizienz

Laufzeit - asymptotisch schneller

Definition:

Gegeben zwei Algorithmen/Programme A und B.
Dann gilt:

a) A ist **schneller** als B, falls

$$T_A(n) \leq T_B(n) \text{ für alle } n \in \mathbb{N} \text{ ist.}$$

b) A ist **asymptotisch schneller** als B, falls

$$\lim_{n \to \infty} \frac{T_A(n)}{T_B(n)} = 0.$$

Analog definiert man für den Speicherplatz.

Effizienz

Laufzeit - asymptotisch schneller

Beispiel:

Von den Algorithmen A, B, C, D ist keiner schneller als einer der anderen, jedoch ist A asymptotisch schneller als B, B asymptotisch schneller als C und C asymptotisch schneller als D, z.B.:

$$\lim_{n\to\infty} \frac{T_A(n)}{T_B(n)} = \lim_{n\to\infty} \frac{100n+30}{100n\cdot\log_2 n} = 0.$$

Aufgabe: Suche asymptotisch schnellstes Programm

Effizienz

Laufzeit/Speicherbedarf - unit cost

4 aufeinander aufbauende Abstraktionen:

- τ_P und σ_P will man nicht für jede konkrete Eingabe exakt wissen => Zusammenfassung von Eingaben zu sinnvollen Klassen gleichschwerer Probleme ✓
- Auswahl eines Repräsentanten, dessen Laufzeit maßgebend ist für die Klasse => Laufzeit im schlimmsten Fall (worst case) ✓
- Übergang zu einem genormten Rechner => Einheitskostenrechnermodell ✓
- Übergang zum qualitativen Verlauf von τ_P und σ_P => Einführung der (Größen-)Ordnung ✓

Effizienz
Laufzeit/Speicherbedarf - Ordnung

Anschaulich:

- *Größenordnung* oder kurz *Ordnung*, beschreibt den qualitativen Verlauf einer Funktion f
- Verhält sich f wie eine quadratische Funktion?
- ... wie eine kubische Funktion?
- ... wie eine Exponentialfunktion?

Effizienz
Ordnung - Definition

Definition:

Sei $f: \mathbb{N} \to \mathbb{N}$ eine Funktion. Dann ist

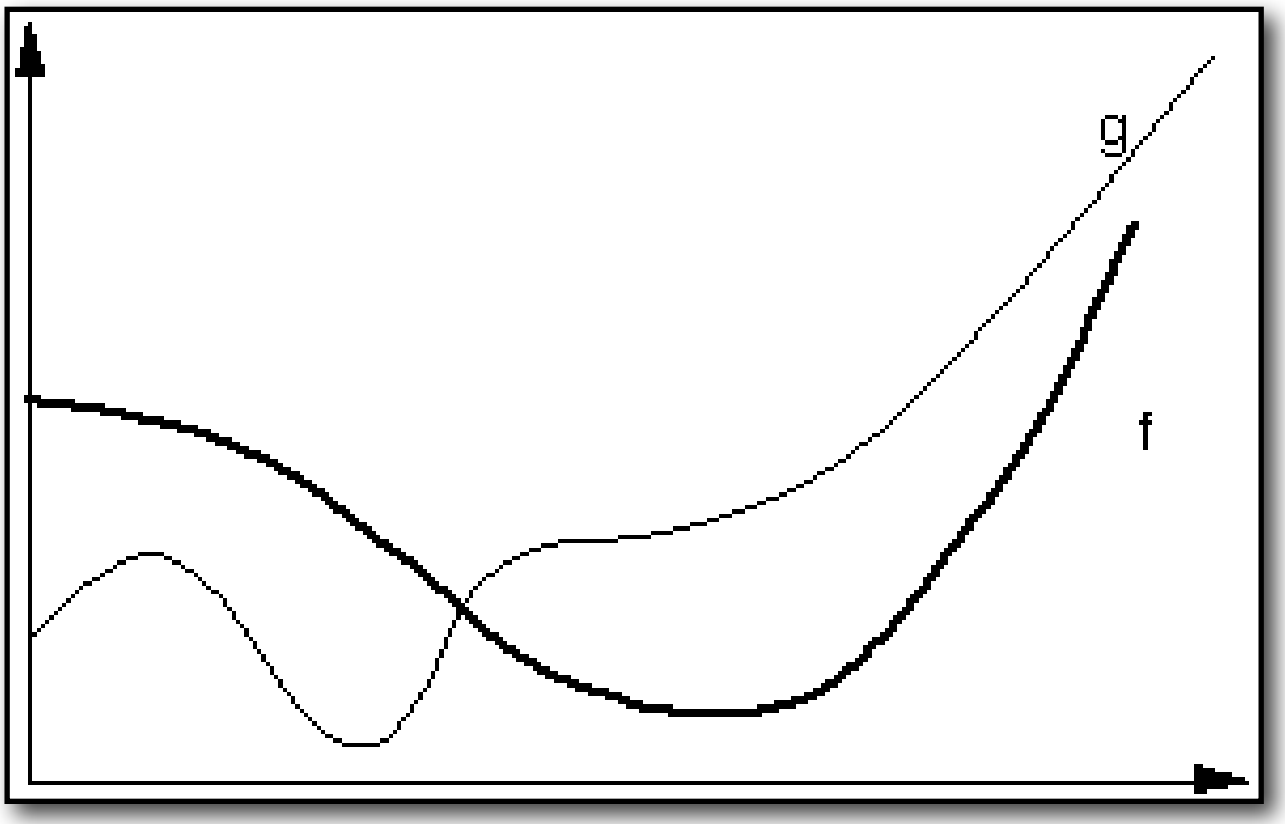
$O(f)=\{g: \mathbb{N} \to \mathbb{N} \mid$ es gibt Zahlen $c, n_0 \in \mathbb{N}$, so daß $g(n) \leq c \cdot f(n)$ für alle $n \geq n_0\}$.

Falls $g \in O(f)$ ist, so hat g die **Ordnung** f. O heißt auch **Landausches Symbol**.

Man spricht O(f) als "groß O von f" oder kurz "O von f".

Effizienz

Ordnung - anschaulich

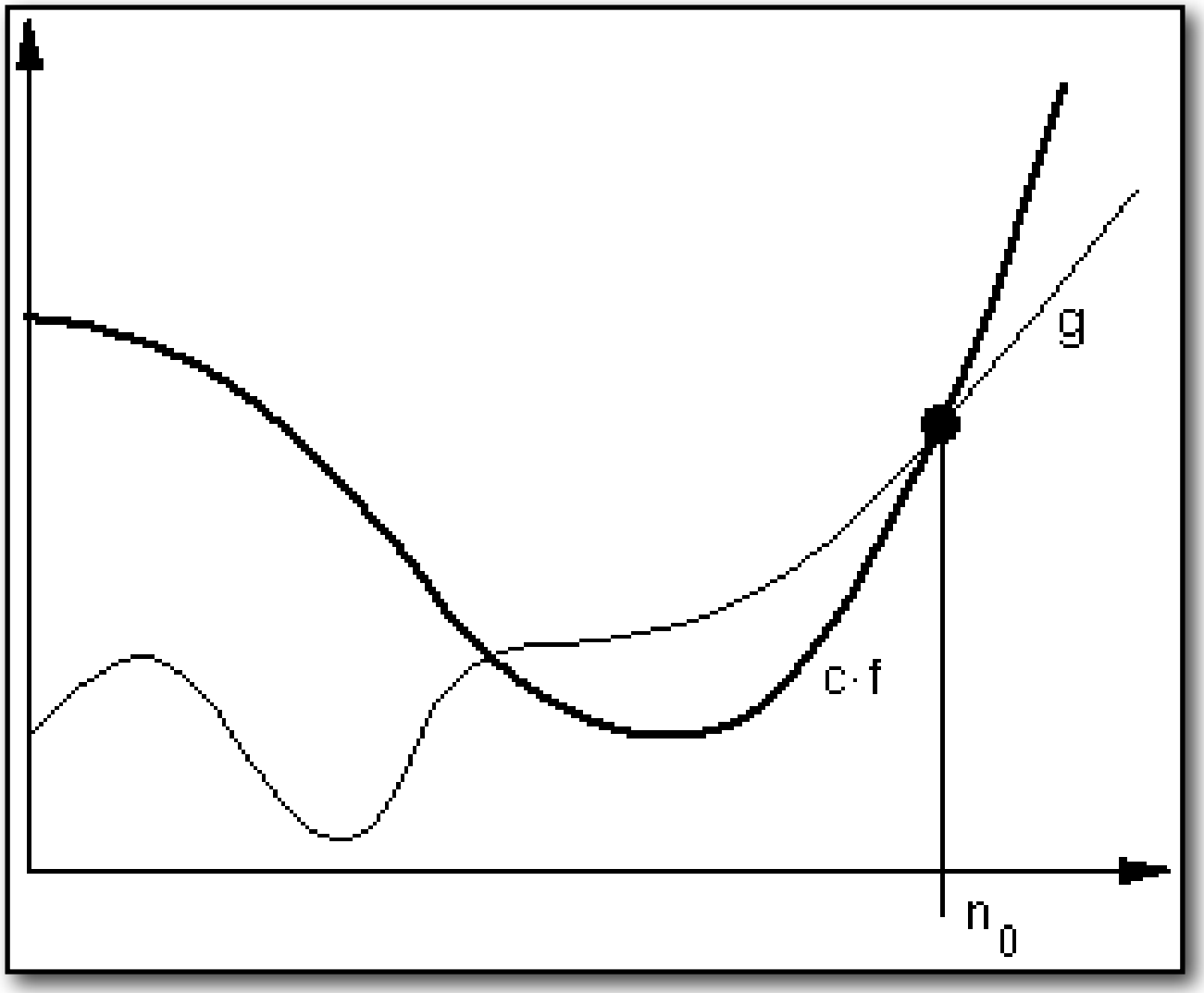


Effizienz

Ordnung - anschaulich

Effizienz

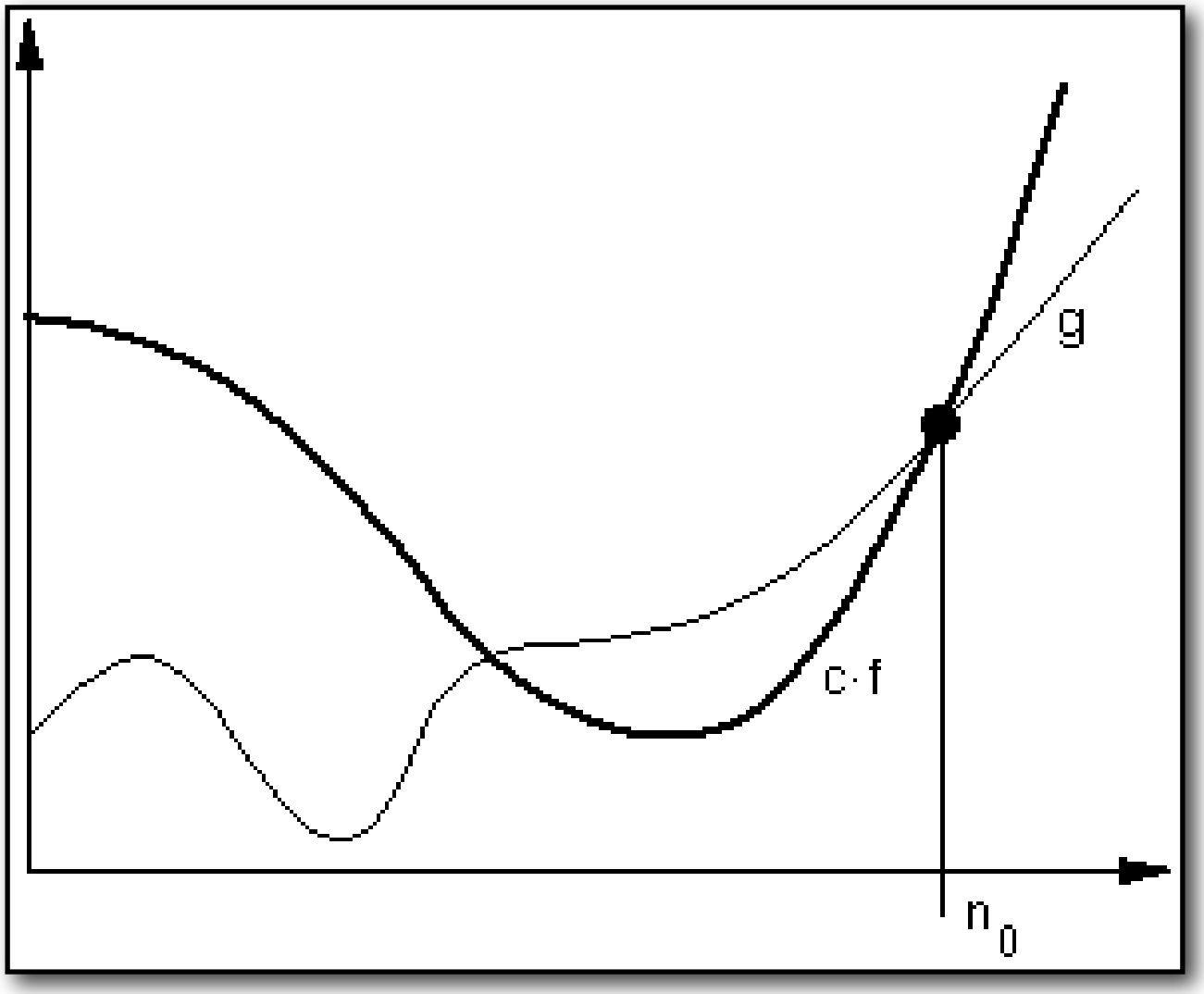
Ordnung - anschaulich



Effizienz

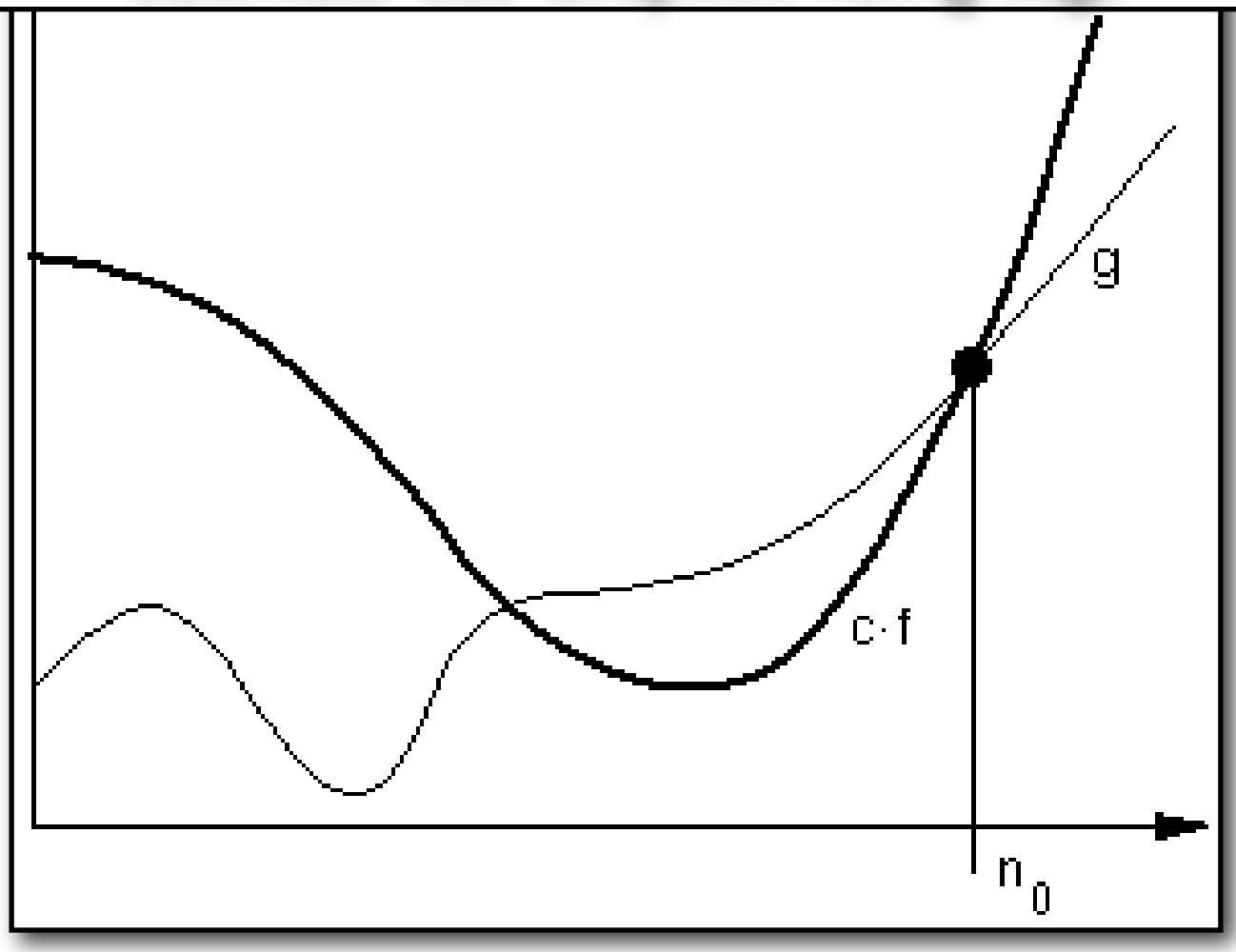
Ordnung - anschaulich

$g \in O(f)$



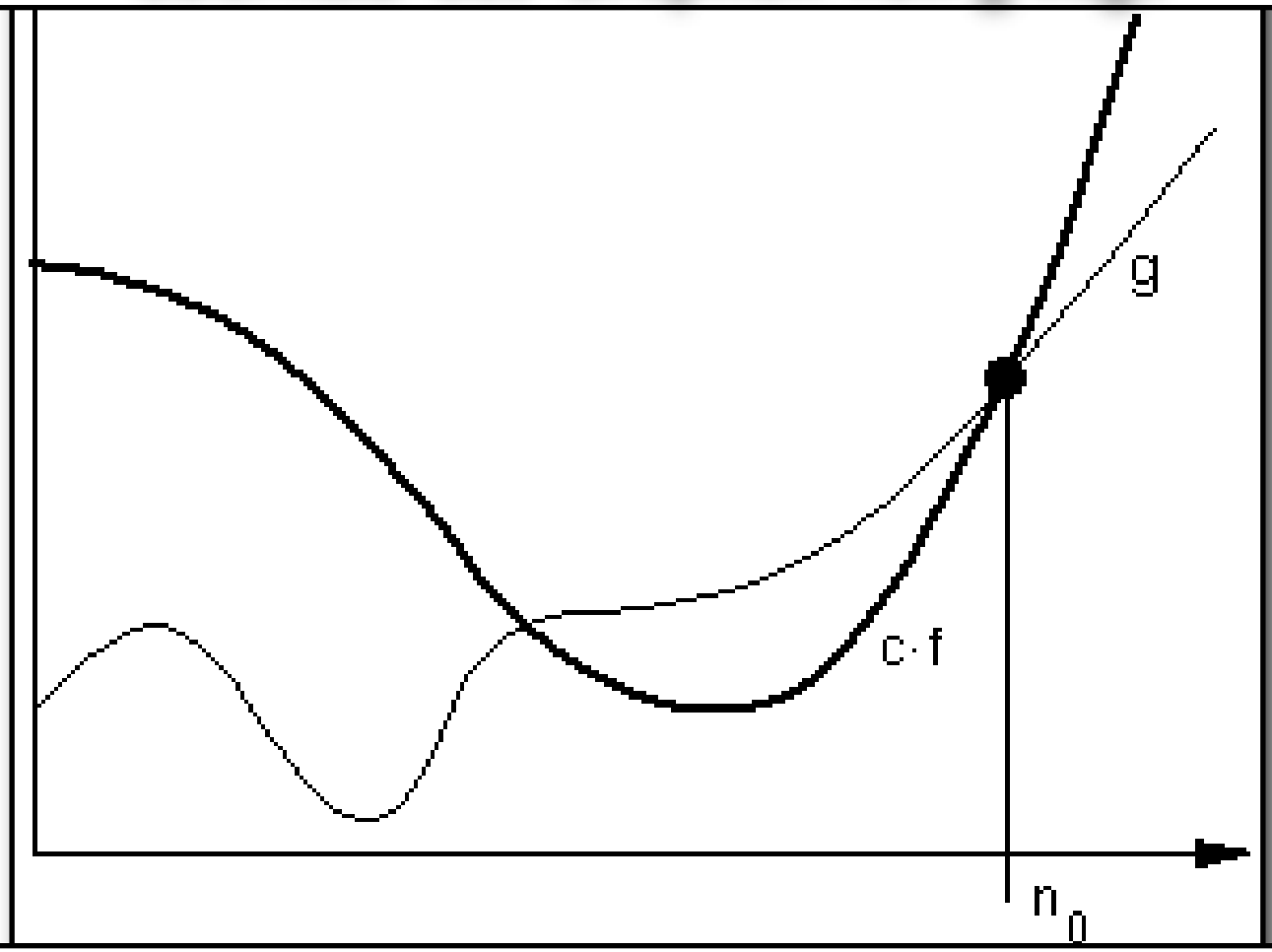
man kann f durch Multiplikation mit einer Konstanten c so nach oben verschieben, daß der Graph von f ab der Stelle n_0 stets oberhalb des Graphen von g liegt

$g \in O(f)$



man kann f durch Multiplikation mit einer Konstanten c so nach oben verschieben, daß der Graph von f ab der Stelle n_0 stets oberhalb des Graphen von g liegt

$g \in O(f)$



Nebenziele: f einfach, f nah bei g

Effizienz
Ordnung - Bezeichnungen

- keine Unterscheidung zwischen Funktion und ihrer Darstellung durch Terme:
  - O(n) statt O(f), wenn f(n)=n
  - $O(n^2)$ statt O(h), wenn $h(n)=n^2$
  - O(1) statt O(e), wenn e konstante Funktion 1 ist, also e(n)=1 für alle $n \in \mathbb{N}$

Effizienz
Ordnung - Beispiel 1

g_1 sei konstante Funktion, d.h. für festes $a \in \mathbb{N}$:

$$g_1(n)=a \text{ für alle } n \in \mathbb{N}.$$

Offensichtlich:

$$g_1 \in O(id) \text{ mit } id(n)=n \text{ für alle } n \in \mathbb{N}$$

bei $c=1$ und $n_0=a$; denn für $n \geq n_0=a$ ist
$g_1(n)=a \leq n=id(n)$.

Aber auch $g_1 \in O(1)$. Hierfür wähle $n_0=1$ und $c=a$.
Dann gilt für alle $n \geq n_0$ $g_1(n)=a \leq c \cdot 1=c$

Effizienz
Ordnung - Beispiel 2

g_2 Polynom vom Grad m, also

$$g_2(n)=c_m n^m+c_{m-1}n^{m-1}+...+c_2n^2+c_1n^1+c_0,\ \ c_m\geq 0.$$

Dann $g_2\in O(n^m)$.

Beweis: Es gibt stets einen Wert n_0, der von m und den Koeffizienten c_m, ..., c_0 abhängt, so daß für $n\geq n_0$ gilt:

$$c_m n^m \geq c_{m-1}n^{m-1}+...+c_2n^2+c_1n^1+c_0,$$

da n^m schneller wächst als die Summe der übrigen Potenzen. Dann für $n\geq n0$: $g_2(n)\leq 2c_m n^m$

also gilt Behauptung mit diesem n_0 und $c=2c_m$.

Gegeben $h(n)=3n^2+n$. Dann gilt $h\in O(n^2)$, aber auch $h\in O(n^k)$ für beliebiges $k>2$, oder auch $h\in O(n^2\cdot\log_2 n)$.

Effizienz
Ordnung - Beispiel 3

g_3 sei Fakultätsfunktion, also

$$g_3(n)=n!=1\cdot 2\cdot 3\cdot \ldots \cdot n.$$

Jeder einzelne Faktor ist höchstens gleich n, daher folgt

$$g_3(n)\leq n^n \text{ und somit } g_3\in O(n^n).$$

Effizienz

Ordnung - Schreibweisen

Gleichheitszeichen (Ungleichheitszeichen)
statt Mengensymbole $\in$ und $\subseteq$ (bzw. $\notin$): Statt

$$5n^3+2n^2 \in O(n^3+2n^2) \subseteq O(n^3) \subseteq O(n^7 \cdot \log^2 n)$$

schreibt man kurz

$$5n^3+2n^2 = O(n^3+2n^2) = O(n^3) = O(n^7 \cdot \log^2 n)$$

Effizienz
Ordnung - Schreibweisen

Gleichheitszeichen (Ungleichheitszeichen)
statt Mengensymbole $\in$ und $\subseteq$ (bzw. $\notin$): Statt

$$5n^3+2n^2 \in O(n^3+2n^2) \subseteq O(n^3) \subseteq O(n^7 \cdot \log^2 n)$$

schreibt man kurz

$$5n^3+2n^2 = O(n^3+2n^2) = O(n^3) = O(n^7 \cdot \log^2 n)$$

Gleichungen mit O-Symbolen dürfen nur von links nach rechts gelesen werden

Effizienz
Ordnung - Schreibweisen

Gleichheitszeichen (Ungleichheitszeichen)
statt Mengensymbole $\in$ und $\subseteq$ (bzw. $\notin$): Statt

$$5n^3+2n^2 \in O(n^3+2n^2) \subseteq O(n^3) \subseteq O(n^7 \cdot \log^2 n)$$

schreibt man kurz

$$5n^3+2n^2 = O(n^3+2n^2) = O(n^3) = O(n^7 \cdot \log^2 n)$$

Gleichungen mit O-Symbolen dürfen nur von links nach rechts gelesen werden

Effizienz
Ordnung - Schreibweisen

Gleichheitszeichen (Ungleichheitszeichen)
statt Mengensymbole $\in$ und $\subseteq$ (bzw. $\notin$): Statt

$$5n^3+2n^2 \in O(n^3+2n^2) \subseteq O(n^3) \subseteq O(n^7 \cdot \log^2 n)$$

schreibt man kurz

$$5n^3+2n^2 = O(n^3+2n^2) = O(n^3) \not= O(n^7 \cdot \log^2 n)$$

Gleichungen mit O-Symbolen dürfen nur von links nach rechts gelesen werden

Effizienz
Ordnung - Schreibweisen

Gleichheitszeichen (Ungleichheitszeichen)
statt Mengensymbole $\in$ und $\subseteq$ (bzw. $\notin$): Statt

$$5n^3+2n^2 \in O(n^3+2n^2) \subseteq O(n^3) \subseteq O(n^7 \cdot \log^2 n)$$

schreibt man kurz

$$5n^3+2n^2 = O(n^3+2n^2) = O(n^3) \not= O(n^7 \cdot \log^2 n)$$

Gleichungen mit O-Symbolen dürfen nur von links nach rechts gelesen werden

Effizienz
Ordnung - Komplexitätsklassen

- Einteilung von Problemen und Algorithmen bzw. Programme in Komplexitätsklassen:
  - lineare Laufzeit (Linearzeit): $T(n)=O(n)$,
  - quadratische Laufzeit: $T(n)=O(n^2)$,
  - kubische Laufzeit: $T(n)=O(n^3)$
  - polynomielle L. (Polynomialzeit): $T(n)=O(n^k)$, $k \in IN$
  - exponentielle L. (Exponentialzeit): $T(n) = O(2^{p(n)})$, p Polynom
  - superexponentielle L. $T(n) = O(2^{2^{p(n)}})$, p Polynom

Effizienz

Komplexität - leicht und schwer

- leicht (easy): polynomielle Laufzeit
- schwer (hard): übrige Probleme; praktisch nicht mehr lösbar

Effizienz

Komplexität - leicht und schwer

- leicht (easy): polynomielle Laufzeit
- schwer (hard): übrige Probleme; praktisch nicht mehr lösbar

T(n) \ n	20	30	40	50	100	
n	0.0002	0.0003	0.0004	0.0005	0.001	Sekunden
n^2	0.004	0.009	0.016	0.025	0.1	Sekunden
n^5	32	243	1024	3125	100000	Sekunden
2^n	10 Sek.	3 Stunden	4 Monate	360 Jahre	$4 \cdot 10^{17}$	Jahre

Effizienz

Komplexität - leicht und schwer

- leicht (easy): polynomielle Laufzeit
- schwer (hard): übrige Probleme; praktisch nicht mehr lösbar

T(n) \ n	20	30	40	50	100	
n	0.0002	0.0003	0.0004	0.0005	0.001	Sekunden
n^2	0.004	0.009	0.016	0.025	0.1	Sekunden
n^5	32	243	1024	3125	100000	Sekunden
2^n	10 Sek.	3 Stunden	4 Monate	360 Jahre	$4 \cdot 10^{17}$	Jahre

Schnellere Rechner helfen hier gar nichts

Effizienz
Komplexität - prakt Beobachtungen

- praktische Probleme: meist $O(n^k)$, $k \leq 3$
- große Klasse von wichtigen Problemen, für die man bisher keinen Polynomialzeitalgorithmus kennt (**NP-vollständige Probleme**)
  - Vermutung: Es gibt zu diesen Problemen überhaupt keine Polynomialzeitalgorithmen

Effizienz
Beispiele - Suchen

- Gegeben: lineares Feld a mit $n \geq 1$ Elementen vom Typ integer, ganze Zahl x;
- a ist aufsteigend sortiert
- Gesucht: Programm, das ausgibt, ob x unter den Elementen von a vorkommt oder nicht. (Problem η)

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

T(n)=

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

T(n)=

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

T(n)=

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

T(n)=

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

T(n)=

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

T(n)=

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

T(n)=

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

T(n)=

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

T(n)=

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

T(n)=

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

T(n)=

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

T(n)=

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

T(n)=

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

$T(n)=6+$

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

$T(n) = 6+$

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

$T(n)=6+$

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

$T(n)=6+$

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

$T(n)=6+$

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

$T(n)=6+$

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

$T(n)=6+$

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

$T(n) = 6+$

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

$T(n)=6+$

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

$T(n) = 6 +$

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

$T(n)=6+$

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

$T(n) = 6 + 4n$

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

$$T(n) = 6 + 4n = O(n)$$

Effizienz
Beispiele - sequentielles Suchen

- vergleiche nacheinander alle Elemente von a mit x und stoppe, falls x gefunden bzw. das Ende des Feldes erreicht ist, ohne daß x gefunden wurde
- Programm:

```
program seqsuche(input,output);
const n=...;
var  a: array [1..n] of integer;
     i,x: integer;
     gefunden: boolean;
begin
     {Das Feld a sei gegeben}
     read(x); i:=1; gefunden:=false;
     while (not gefunden) and (i≤n) do
     if a[i]=x then gefunden:=true else i:=i+1;
     write(gefunden)
end.
```

$S(n)=O(1)$

$T(n)=6+4n=O(n)$

Effizienz
Beispiele - binäres Suchen

- O.B.d.A.: n gerade
- Zuerst vergleiche x mit Mittelelement, also mit a[n/2]
  - x=a[n/2] => Suche erfolgreich beendet
  - x<a[n/2] => wende Verfahren rekursiv auf linkes Teilfeld von a[1], ..., a[n/2-1]
  - x>a[n/2] => wende Verfahren rekursiv auf rechtes Teilfeld a[n/2+1], ..., a[n] an
- bis x gefunden oder Teilfeld leer.

Effizienz

Beispiele - binäres Suchen

Effizienz

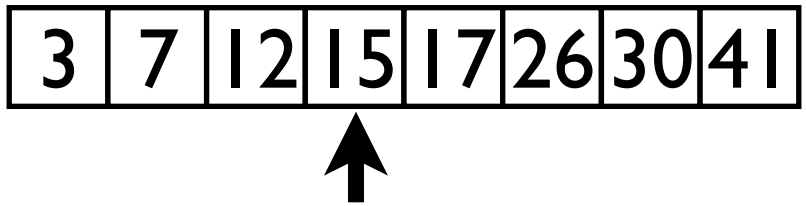
Beispiele - binäres Suchen

Suche 12

Effizienz

Beispiele - binäres Suchen

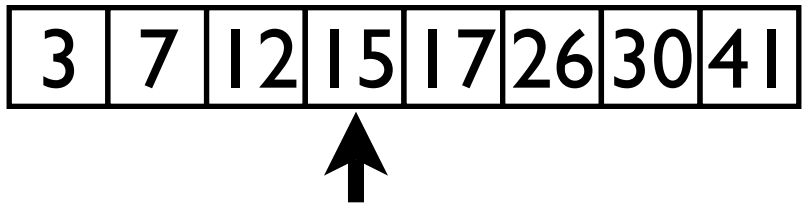
Suche 12



Effizienz

Beispiele - binäres Suchen

Suche 12

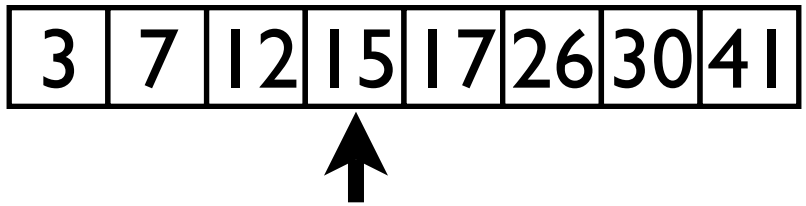


12=15? => “<”

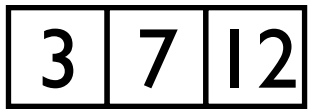
Effizienz

Beispiele - binäres Suchen

Suche 12



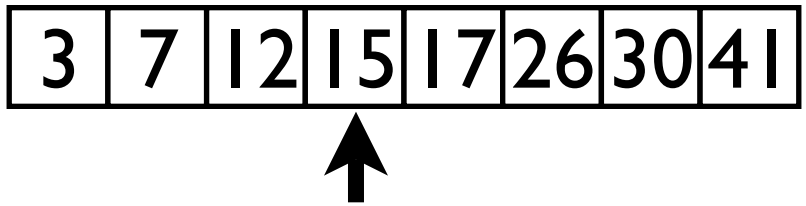
12=15? => “<”



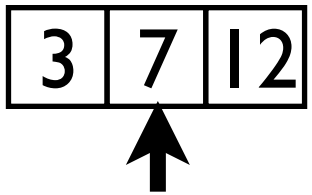
Effizienz

Beispiele - binäres Suchen

Suche 12



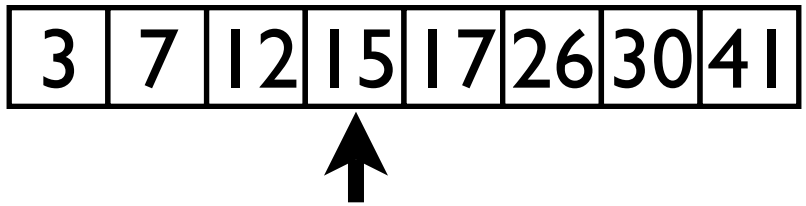
12=15? => “<”



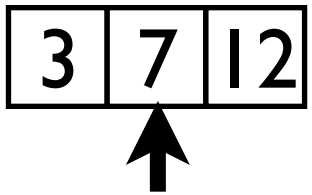
Effizienz

Beispiele - binäres Suchen

Suche 12



12=15? => “<”

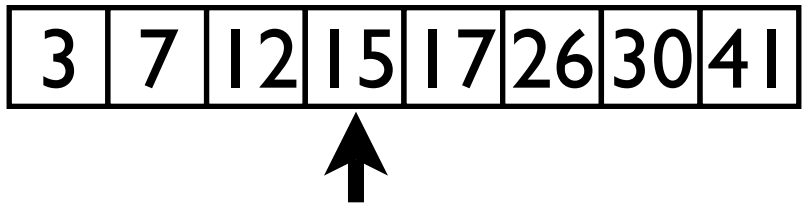


12=7? => “>”

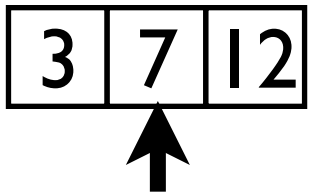
Effizienz

Beispiele - binäres Suchen

Suche 12



12=15? => “<”

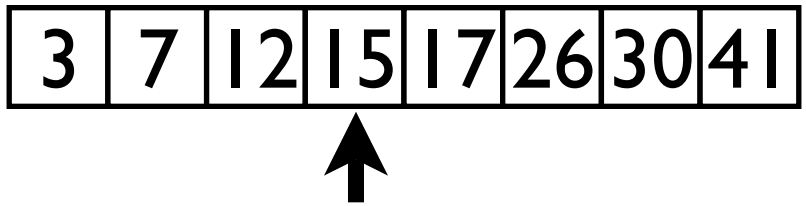


12=7? => “>”

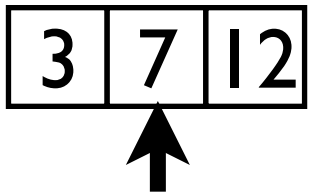
Effizienz

Beispiele - binäres Suchen

Suche 12



12=15? => “<”

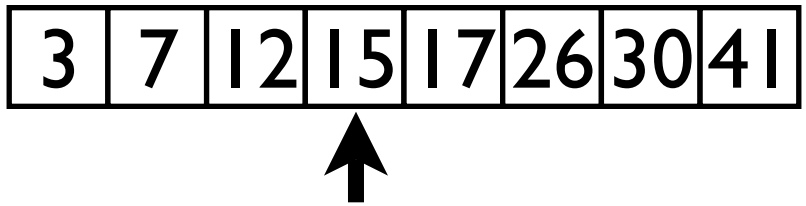


12=7? => “>”

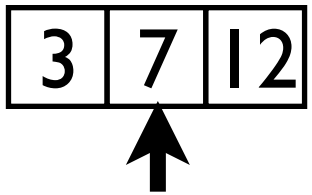
Effizienz

Beispiele - binäres Suchen

Suche 12



12=15? => “<”



12=7? => “>”

12=12? => “=”

Effizienz
binäres Suchen - Programm

```
program binaersuche(input,output);
const n=...;
var a: array [1..n] of integer;
           x: integer;
function binsuche(i,j,x: integer): boolean;
var m: integer;
begin
    if j<i then binsuche:=false else
    begin
        m:=(i+j) div 2;
        if x<a[m] then binsuche:=binsuche(i,m-1,x) else
            if x>a[m] then binsuche:=binsuche(m+1,j,x) else
                binsuche:=true
    end
end;
```

Aufruf: binsuche(1,n,x)

Effizienz
binäres Suchen - Programm

```
program binaersuche(input,output);
const n=...;
var a: array [1..n] of integer;
          x: integer;
function binsuche(i,j,x: integer): boolean;
var m: integer;
begin
    if j<i then binsuche:=false else
    begin
        m:=(i+j) div 2;
        if x<a[m] then binsuche:=binsuche(i,m-1,x) else
            if x>a[m] then binsuche:=binsuche(m+1,j,x) else
                binsuche:=true
    end
end;

Aufruf: binsuche(1,n,x)
```

Effizienz
binäres Suchen - Programm

```
program binaersuche(input,output);
const n=...;
var a: array [1..n] of integer;
           x: integer;
function binsuche(i,j,x: integer): boolean;
var m: integer;
begin
    if j<i then binsuche:=false else
    begin
        m:=(i+j) div 2;
        if x<a[m] then binsuche:=binsuche(i,m-1,x) else
            if x>a[m] then binsuche:=binsuche(m+1,j,x) else
                binsuche:=true
    end
end;
```

Aufruf: binsuche(1,n,x)

Effizienz
binäres Suchen - Programm

```
program binaersuche(input,output);
const n=...;
var a: array [1..n] of integer;
           x: integer;
function binsuche(i,j,x: integer): boolean;
var m: integer;
begin
    if j<i then binsuche:=false else
    begin
        m:=(i+j) div 2;
        if x<a[m] then binsuche:=binsuche(i,m-1,x) else
            if x>a[m] then binsuche:=binsuche(m+1,j,x) else
                binsuche:=true
    end
end;

Aufruf: binsuche(1,n,x)
```

Effizienz
binäres Suchen - Programm

```
program binaersuche(input,output);
const n=...;
var a: array [1..n] of integer;
           x: integer;
function binsuche(i,j,x: integer): boolean;
var m: integer;
begin
    if j<i then binsuche:=false else
    begin
        m:=(i+j) div 2;
        if x<a[m] then binsuche:=binsuche(i,m-1,x) else
            if x>a[m] then binsuche:=binsuche(m+1,j,x) else
                binsuche:=true
    end
end;
```

Aufruf: binsuche(1,n,x)

Effizienz
binäres Suchen - Programm

```
program binaersuche(input,output);
const n=...;
var a: array [1..n] of integer;
           x: integer;
function binsuche(i,j,x: integer): boolean;
var m: integer;
begin
    if j<i then binsuche:=false else
    begin
        m:=(i+j) div 2;
        if x<a[m] then binsuche:=binsuche(i,m-1,x) else
            if x>a[m] then binsuche:=binsuche(m+1,j,x) else
                binsuche:=true
    end
end;

Aufruf: binsuche(1,n,x)
```

Effizienz
binäres Suchen - Programm

```
program binaersuche(input,output);
const n=...;
var a: array [1..n] of integer;
           x: integer;
function binsuche(i,j,x: integer): boolean;
var m: integer;
begin
    if j<i then binsuche:=false else
    begin
        m:=(i+j) div 2;
        if x<a[m] then binsuche:=binsuche(i,m-1,x) else
            if x>a[m] then binsuche:=binsuche(m+1,j,x) else
                binsuche:=true
    end
end;
```

Aufruf: binsuche(1,n,x)

Effizienz
binäres Suchen - Programm

```
program binaersuche(input,output);
const n=...;
var a: array [1..n] of integer;
           x: integer;
function binsuche(i,j,x: integer): boolean;
var m: integer;
begin
    if j<i then binsuche:=false else
    begin
        m:=(i+j) div 2;
        if x<a[m] then binsuche:=binsuche(i,m-1,x) else
            if x>a[m] then binsuche:=binsuche(m+1,j,x) else
                binsuche:=true
    end
end;

Aufruf: binsuche(1,n,x)
```

Effizienz
binäres Suchen - Programm

```
program binaersuche(input,output);
const n=...;
var a: array [1..n] of integer;
           x: integer;
function binsuche(i,j,x: integer): boolean;
var m: integer;
begin
    if j<i then binsuche:=false else
    begin
        m:=(i+j) div 2;
        if x<a[m] then binsuche:=binsuche(i,m-1,x) else
            if x>a[m] then binsuche:=binsuche(m+1,j,x) else
                binsuche:=true
    end
end;
```

Aufruf: binsuche(1,n,x)

Effizienz
binäres Suchen - Programm

```
program binaersuche(input,output);
const n=...;
var a: array [1..n] of integer;
           x: integer;
function binsuche(i,j,x: integer): boolean;
var m: integer;
begin
    if j<i then binsuche:=false else
    begin
        m:=(i+j) div 2;
        if x<a[m] then binsuche:=binsuche(i,m-1,x) else
            if x>a[m] then binsuche:=binsuche(m+1,j,x) else
                binsuche:=true
    end
end;

Aufruf: binsuche(1,n,x)
```

Effizienz
binäres Suchen - Programm

```
program binaersuche(input,output);
const n=...;
var a: array [1..n] of integer;
           x: integer;
function binsuche(i,j,x: integer): boolean;
var m: integer;
begin
    if j<i then binsuche:=false else
    begin
        m:=(i+j) div 2;
        if x<a[m] then binsuche:=binsuche(i,m-1,x) else
            if x>a[m] then binsuche:=binsuche(m+1,j,x) else
                binsuche:=true
    end
end;
```

Aufruf: binsuche(1,n,x)

Effizienz
binäres Suchen - Programm

```
program binaersuche(input,output);
const n=...;
var a: array [1..n] of integer;
           x: integer;
function binsuche(i,j,x: integer): boolean;
var m: integer;
begin
    if j<i then binsuche:=false else
    begin
        m:=(i+j) div 2;
        if x<a[m] then binsuche:=binsuche(i,m-1,x) else
            if x>a[m] then binsuche:=binsuche(m+1,j,x) else
                binsuche:=true
    end
end;

Aufruf: binsuche(1,n,x)
```

Effizienz
binäres Suchen - Programm

```
program binaersuche(input,output);
const n=...;
var a: array [1..n] of integer;
           x: integer;
function binsuche(i,j,x: integer): boolean;
var m: integer;
begin
    if j<i then binsuche:=false else
    begin
        m:=(i+j) div 2;
        if x<a[m] then binsuche:=binsuche(i,m-1,x) else
            if x>a[m] then binsuche:=binsuche(m+1,j,x) else
                binsuche:=true
    end
end;
```

Aufruf: binsuche(1,n,x)

Effizienz
binäres Suchen - Programm

```
program binaersuche(input,output);
const n=...;
var a: array [1..n] of integer;
           x: integer;
function binsuche(i,j,x: integer): boolean;
var m: integer;
begin
    if j<i then binsuche:=false else
    begin
        m:=(i+j) div 2;
        if x<a[m] then binsuche:=binsuche(i,m-1,x) else
            if x>a[m] then binsuche:=binsuche(m+1,j,x) else
                binsuche:=true
    end
end;

Aufruf: binsuche(1,n,x)
```

Effizienz
binäres Suchen - Programm

```
program binaersuche(input,output);
const n=...;
var a: array [1..n] of integer;
           x: integer;
function binsuche(i,j,x: integer): boolean;
var m: integer;
begin
    if j<i then binsuche:=false else
    begin
        m:=(i+j) div 2;
        if x<a[m] then binsuche:=binsuche(i,m-1,x) else
            if x>a[m] then binsuche:=binsuche(m+1,j,x) else
                binsuche:=true
    end
end;
```

Aufruf: binsuche(1,n,x)

Effizienz
binäres Suchen - Programm

```
program binaersuche(input,output);
const n=...;
var a: array [1..n] of integer;
           x: integer;
function binsuche(i,j,x: integer): boolean;
var m: integer;
begin
    if j<i then binsuche:=false else
    begin
        m:=(i+j) div 2;
        if x<a[m] then binsuche:=binsuche(i,m-1,x) else
            if x>a[m] then binsuche:=binsuche(m+1,j,x) else
                binsuche:=true
    end
end;
```

Aufruf: `binsuche(1,n,x)`

Effizienz
binäres Suchen - Programm

```
program binaersuche(input,output);
const n=...;
var a: array [1..n] of integer;
           x: integer;
function binsuche(i,j,x: integer): boolean;
var m: integer;
begin
    if j<i then binsuche:=false else
    begin
        m:=(i+j) div 2;
        if x<a[m] then binsuche:=binsuche(i,m-1,x) else
            if x>a[m] then binsuche:=binsuche(m+1,j,x) else
                binsuche:=true
    end
end;
```

Aufruf: binsuche(1,n,x)

Effizienz
binäres Suchen - Komplexitätsanalyse

- schlimmster Fall? x kommt in a nicht vor.
- binsuche teilt in jedem Durchlauf den Teil des Feldes von a[i] bis a[j], in dem x vermutet wird, in zwei etwa gleichgroße Hälften. Halbierung stoppt, wenn $j<i$.
- Je Halbierung konstant viele Rechenschritte C
- rekursive Anwendung von binsuche auf Feld der Länge n/2 nach Annahme T(n/2) Schritte
- Ergebnis: $T(n)=C+T(n/2)$, $n \geq 2$
- n=1: T(1)=C'

Effizienz

binäres Suchen - Komplexitätsanalyse

- O.B.d.A.: C>C'.
- Löse: $T(n)=C+T(n/2),\ n\geq 2; T(1)=C$
- Setze $n=2^k$: $T(2^k)=C+T(2^{k-1}); T(2^0)=C$
- Fortlfd. Ausrechnen: $T(2^k)=C(k+1)$
- Also:

$$T(n)=C(\log_2 n+1)=O(\log_2 n)$$

Effizienz

binäres Suchen - Speic

x, a rechnen nicht mit (Teil der Eingabe)

```
const n=...;
var a: array [1..n] of integer;
    x: integer;
function binsuche(i,j,x: integer): boolean;
var m: integer;
```

- erster Eindruck: Handvoll Variablen
- aber: Rekursionsstack mitrechnen
- Rekursionstiefe: $\log_2 n$
- je Stackeintrag konstante Zahl von Variablen: i, j, x, Rücksprungadresse etc.
- Also:

$$S(n)=O(\log_2 n)$$

Effizienz

binäres Suchen - Speic

x, a rechnen nicht mit
(Teil der Eingabe)

```
const n=...;
var a: array [1..n] of integer;
    x: integer;
function binsuche(i,j,x: integer): boolean;
var m: integer;
```

- erster Eindruck: Handvoll Variablen
- aber: Rekursionsstack mitrechnen
- Rekursionstiefe: $\log_2 n$
- je Stackeintrag konstante Zah
  x, Rücksprungadresse etc.
- Also:

nicht-rekursive Lösung:
S(n)=O(1)

$$S(n)=O(\log_2 n)$$

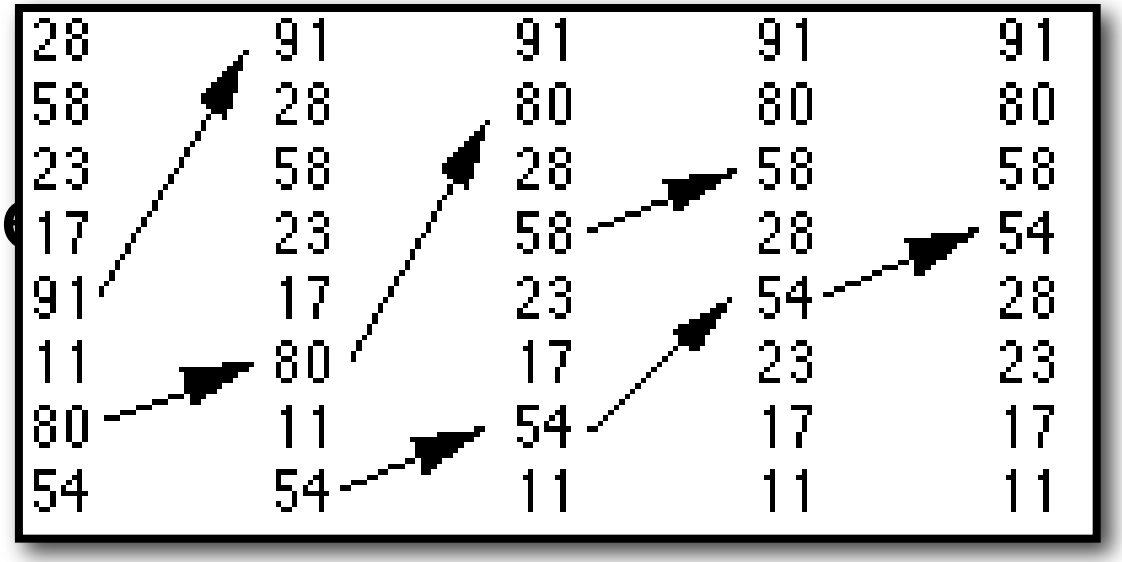
Effizienz
Sortieren - Bubblesort

- Geg.: Array a mit n Elementen vom Typ integer
- Aufgabe: Array aufsteigend sortieren
- Vorgehen: vergleiche fortwährend benachbarte Elemente, wenn sie in der falschen Reihenfolge stehen

Effizienz
Bubblesort - Programm

```
program sort;
const n=...;
vara: array [1..n] of integer;
        i,j,t: integer;
begin
   for i:=n-1 downto 1 do
      for j:=1 to i do
         if a[j]>a[j+1] then
         begin
            t:=a[j]; a[j]:=a[j+1]; a[j+1]:=t
         end
end.
```

Bubble




```
program sort;
const n=...;
vara: array [1..n] of integer;
        i,j,t: integer;
begin
   for i:=n-1 downto 1 do
      for j:=1 to i do
         if a[j]>a[j+1] then
         begin
            t:=a[j]; a[j]:=a[j+1]; a[j+1]:=t
         end
end.
```

Effizienz
Bubblesort - Komplexitätsanalyse

- Annahme: innerer Schleifenrumpf kostet C Schritte
- i=1: innere Schleife einmal durchlaufen
- i=2: innere Schleife zweimal durchlaufen
- ...
- i=n-1: innere Schleife (n-1)-mal durchlaufen
- Insgesamt: 1+2+3+...+(n-2)+(n-1) Durchläufe
- $T(n)=C(1+2+3+...+(n-2)+(n-1))=C/2 \cdot n(n-1)=O(n^2)$

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f: 7 19 2 13

h:

g: 1 24 3 4

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f: 7 19 2 13

h:

g: 1 24 3 4

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f: 7 19 2 13

h:

g: 1 24 3 4

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f: 7 19 2 13

h:

g: 24 3 4

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f: 7 19 2 13

h: 1

g: 24 3 4

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f: 7 19 2 13

h: 1

g: 24 3 4

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f: 7 19 2 13

h: 1

g: 24 3 4

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f: 19 2 13

h: 1

g: 24 3 4

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f: 19 2 13

h: 1 7

g: 24 3 4

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f: 19 2 13

h: 1 7

g: 24 3 4

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f: 19 2 13

h: 1 7

g: 24 3 4

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f: 2 13

h: 1 7

g: 24 3 4

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f: 2 13

h: 1 7 19

g: 24 3 4

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f: 2 13

h: 1 7 19

g: 24 3 4

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f: 2 13

h: 1 7 19

g: 24 3 4

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f: 13

h: 1 7 19

g: 24 3 4

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f: 13

h: 1 7 19 2

g: 24 3 4

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f:

h: 1 7 19 2

g:

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f:

h: 1 7 19 2 13 24 3 4

g:

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f:

h: 1 7 19 2 13 24 3 4

g:

Bez.: **Lauf** (**run**)=maximale aufsteigend sortierte Teilfolge

Effizienz
Sortieren durch Mischen

- Idee: Wiederholtes Mischen von Teilfolgen zu immer längeren sortierten Folgen
- Erinnerung:

f:

h: 1 7 19 2 13 24 3 4

g:

Bez.: **Lauf** (**run**)=maximale aufsteigend sortierte Teilfolge

Zahl der Läufe hat sich vermindert

Effizienz
Sortieren durch Mischen - Methode

- Vorgehen:
  - 1. Schritt: Mische Nachbarn=bilde Läufe der Länge 2
  - 2. Schritt: Mische benachbarte Paare=bilde Läufe der Länge 4
  - 3. Schritt: Mische benachbarte Quadrupel=bilde Läufe der Länge 8
  - ...

Effizienz

Sortieren durch Mischen - Beispiel

f: 7 19 2 13

h:

g: 1 24 3 4

f:

h:

g:

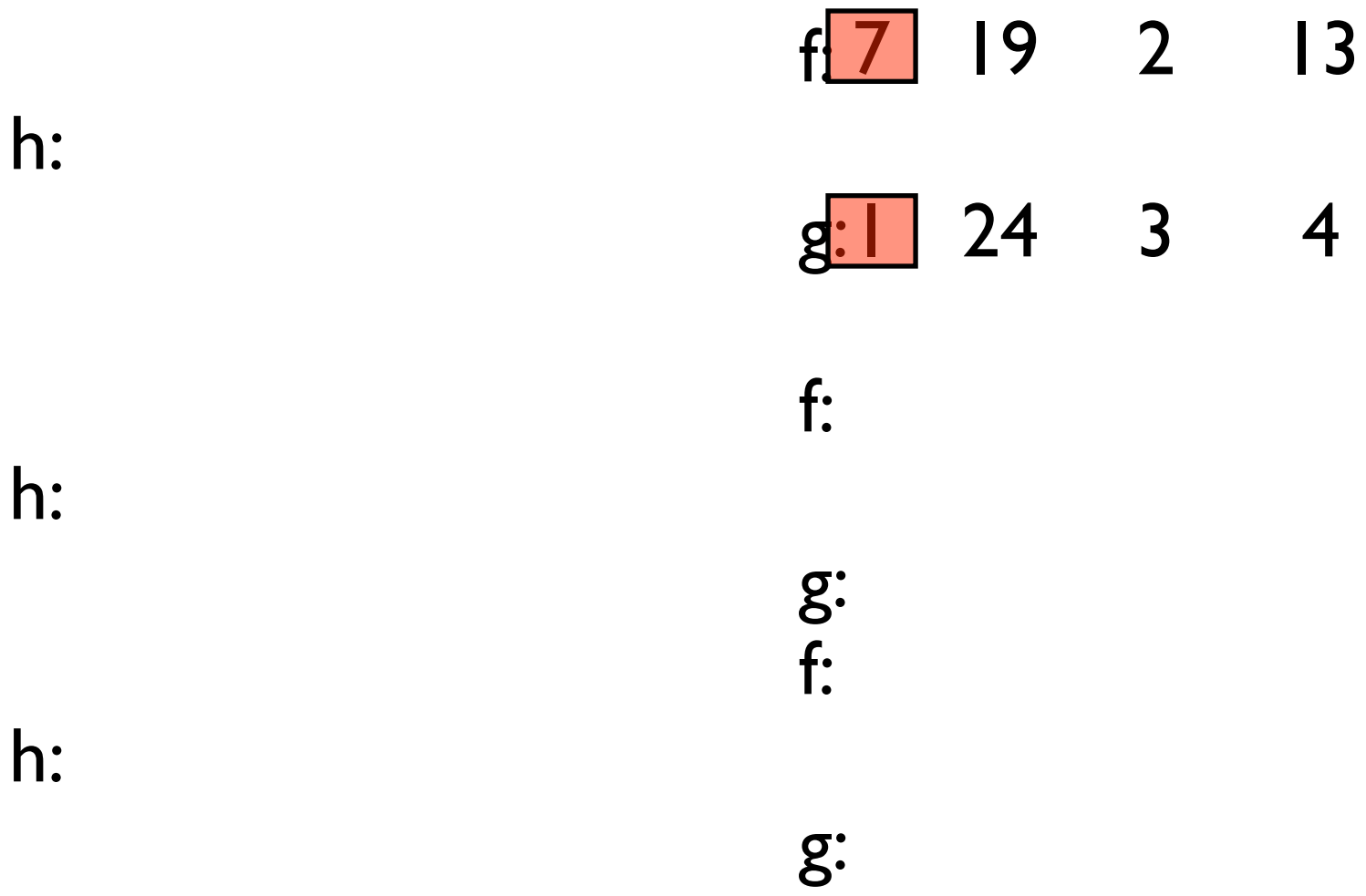
f:

h:

g:

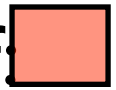
Effizienz

Sortieren durch Mischen - Beispiel



Effizienz

Sortieren durch Mischen - Beispiel

f:  19 2 13

h:

g: 24 3 4

f:

h:

g:

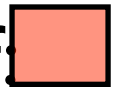
f:

h:

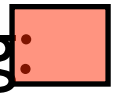
g:

Effizienz

Sortieren durch Mischen - Beispiel

f:  19 2 13

h: 1 7

g:  24 3 4

f:

h:

g:

f:

h:

g:

Effizienz

Sortieren durch Mischen - Beispiel

f: 19 2 13

h: 1 7

g: 24 3 4

f:

h:

g:

f:

h:

g:

Effizienz

Sortieren durch Mischen - Beispiel

f: 19 2 13

h: 1 7

g: 24 3 4

f:

h:

g:

f:

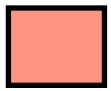
h:

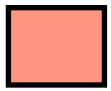
g:

Effizienz

Sortieren durch Mischen - Beispiel

h: 1 7

f:  2 13

g:  3 4

f:

h:

g:

f:

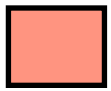
h:

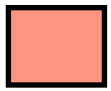
g:

Effizienz

Sortieren durch Mischen - Beispiel

h: 1 7 19 24

f:  2 13

g:  3 4

f:

h:

g:

f:

h:

g:

Effizienz

Sortieren durch Mischen - Beispiel

h: 1 7 19 24

f: 2 13

g: 3 4

f:

h:

g:

f:

h:

g:

Effizienz

Sortieren durch Mischen - Beispiel

h: 1 7 19 24

f: 2 13

g: 3 4

f:

h:

g:

f:

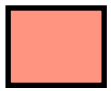
h:

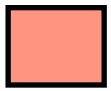
g:

Effizienz

Sortieren durch Mischen - Beispiel

h: 1 7 19 24

f:  13

g:  4

f:

h:

g:

f:

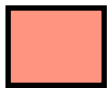
h:

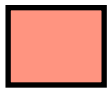
g:

Effizienz

Sortieren durch Mischen - Beispiel

h: 1 7 19 24 2 3

f:  13

g:  4

f:

h:

g:

f:

h:

g:

Effizienz

Sortieren durch Mischen - Beispiel

h: 1 7 19 24 2 3

f: 13

g: 4

f:

h:

g:

f:

h:

g:

Effizienz

Sortieren durch Mischen - Beispiel

h: 1 7 19 24 2 3

f: 13

g: 4

f:

h:

g:

f:

h:

g:

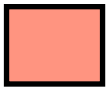
Effizienz

Sortieren durch Mischen - Beispiel

h: 1 7 19 24 2 3

f:

g:



f:

h:

g:

f:

h:

g:

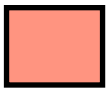
Effizienz

Sortieren durch Mischen - Beispiel

f:

h: 1 7 19 24 2 3 4 13

g:



f:

h:

g:

f:

h:

g:

Effizienz

Sortieren durch Mischen - Beispiel

f:

h: 1 7 19 24 2 3 4 13

g:

f:

h:

g:

f:

h:

g:

Effizienz

Sortieren durch Mischen - Beispiel

f:

h: 1 7 19 24 2 3 4 13

g:

f: 1 7

h:

g:

f:

h:

g:

Effizienz

Sortieren durch Mischen - Beispiel

f:

h: 1 7 19 24 2 3 4 13

g:

f: 1 7 19 24

h:

g:

f:

h:

g:

Effizienz

Sortieren durch Mischen - Beispiel

f:

h: 1 7 19 24 2 3 4 13

g:

f: 1 7 19 24

h:

g: 2 3

f:

h:

g:

Effizienz

Sortieren durch Mischen - Beispiel

f:

h: 1 7 19 24 2 3 4 13

g:

f: 1 7 19 24

h:

g: 2 3 4 13

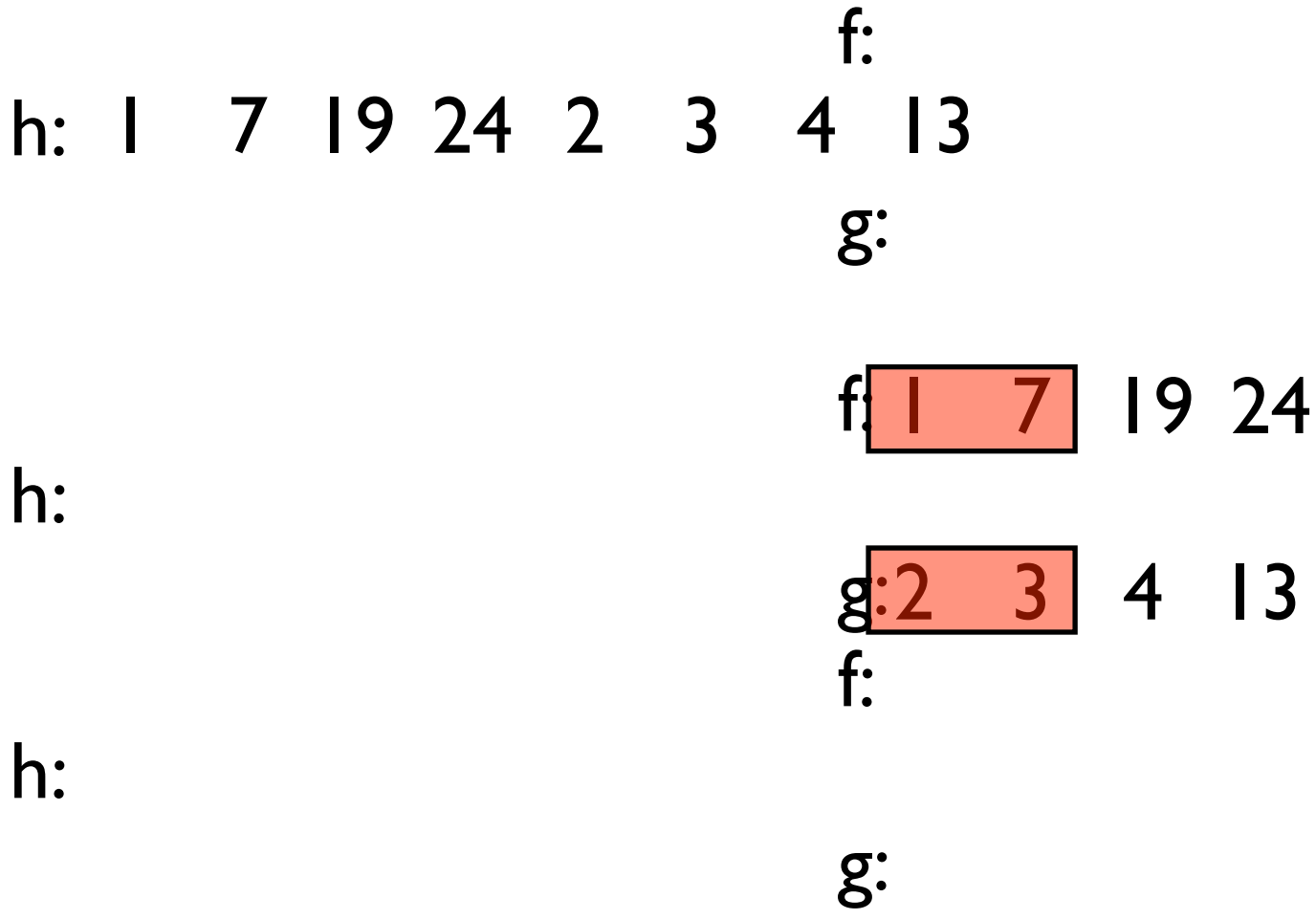
f:

h:

g:

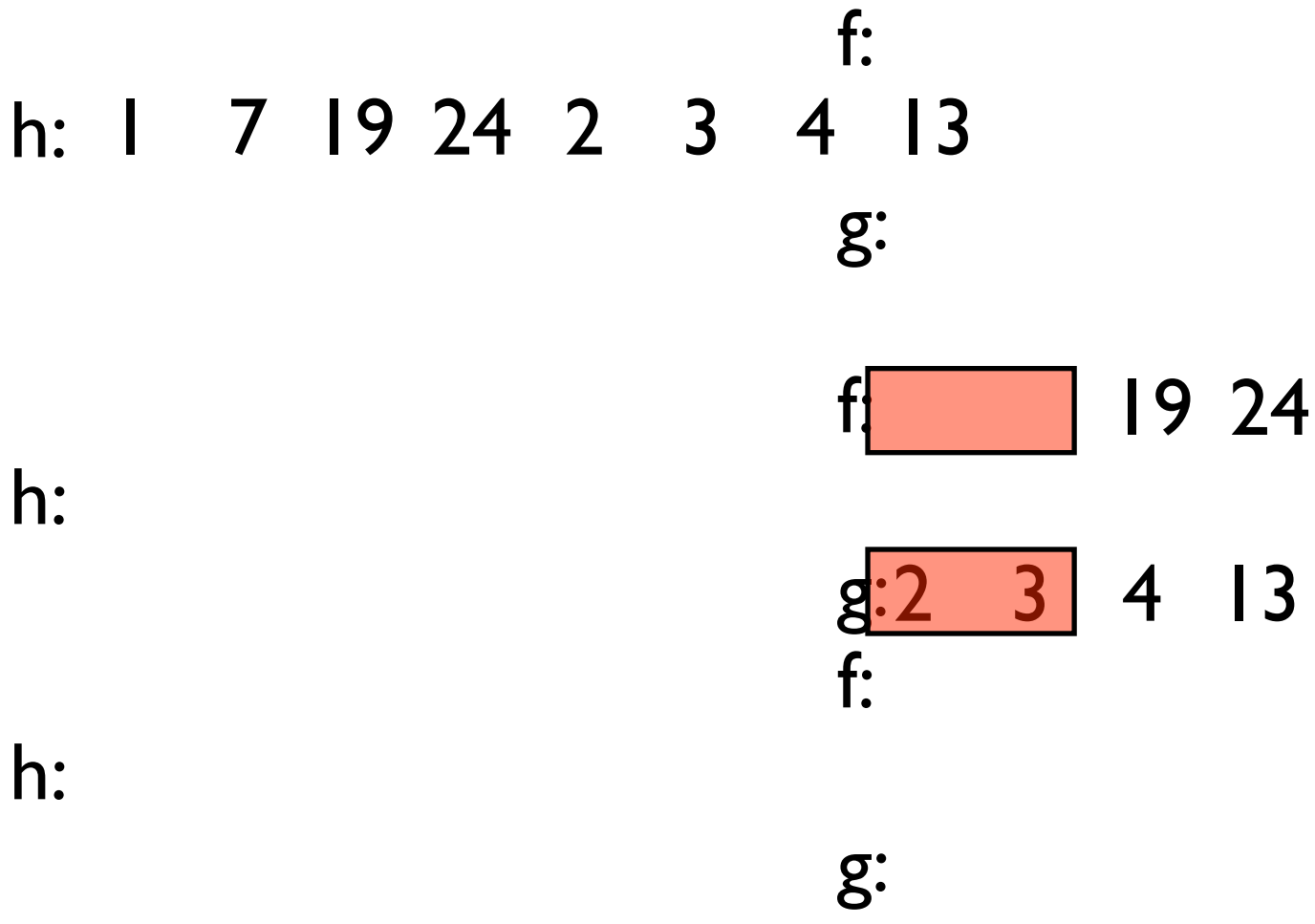
Effizienz

Sortieren durch Mischen - Beispiel



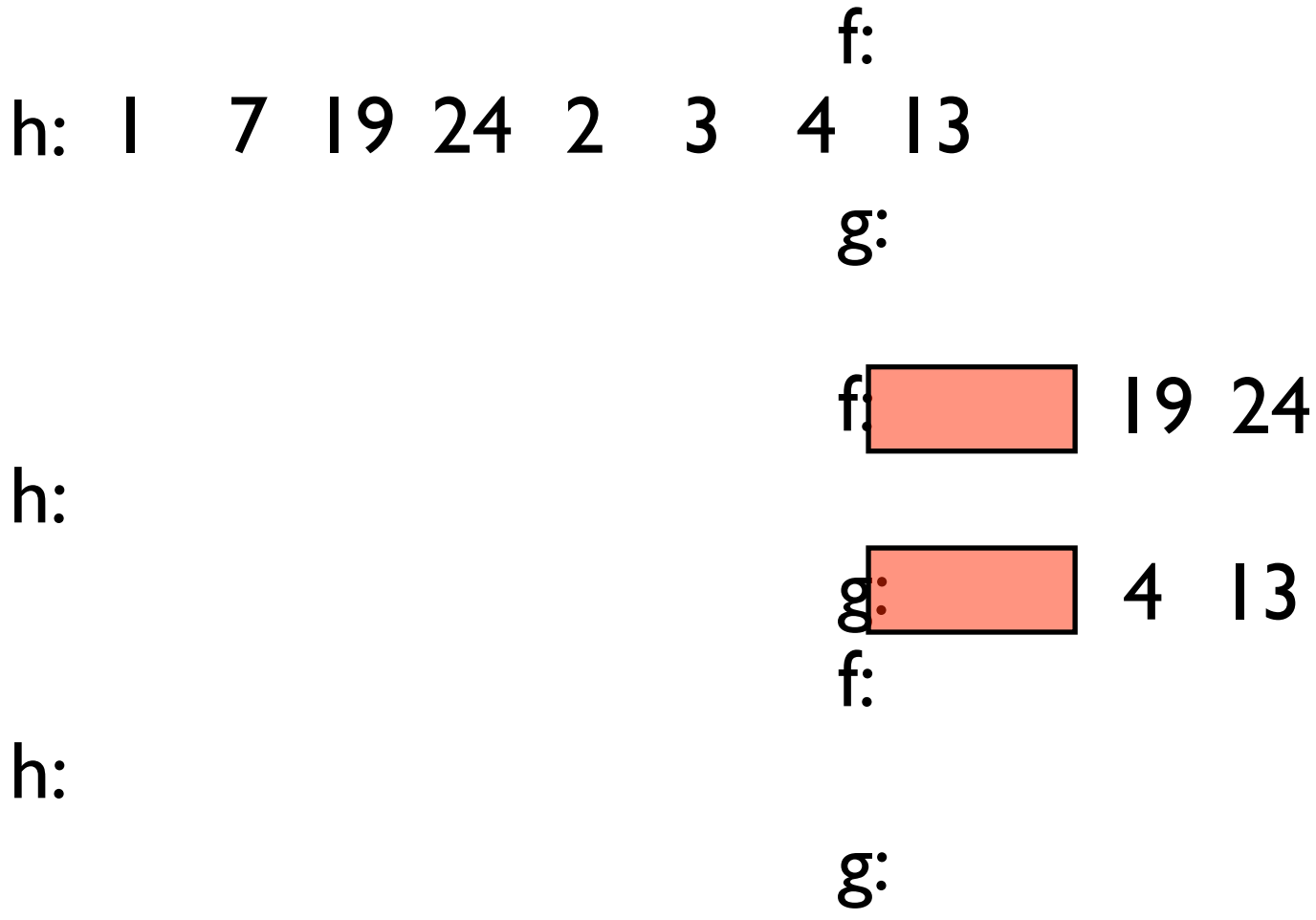
Effizienz

Sortieren durch Mischen - Beispiel



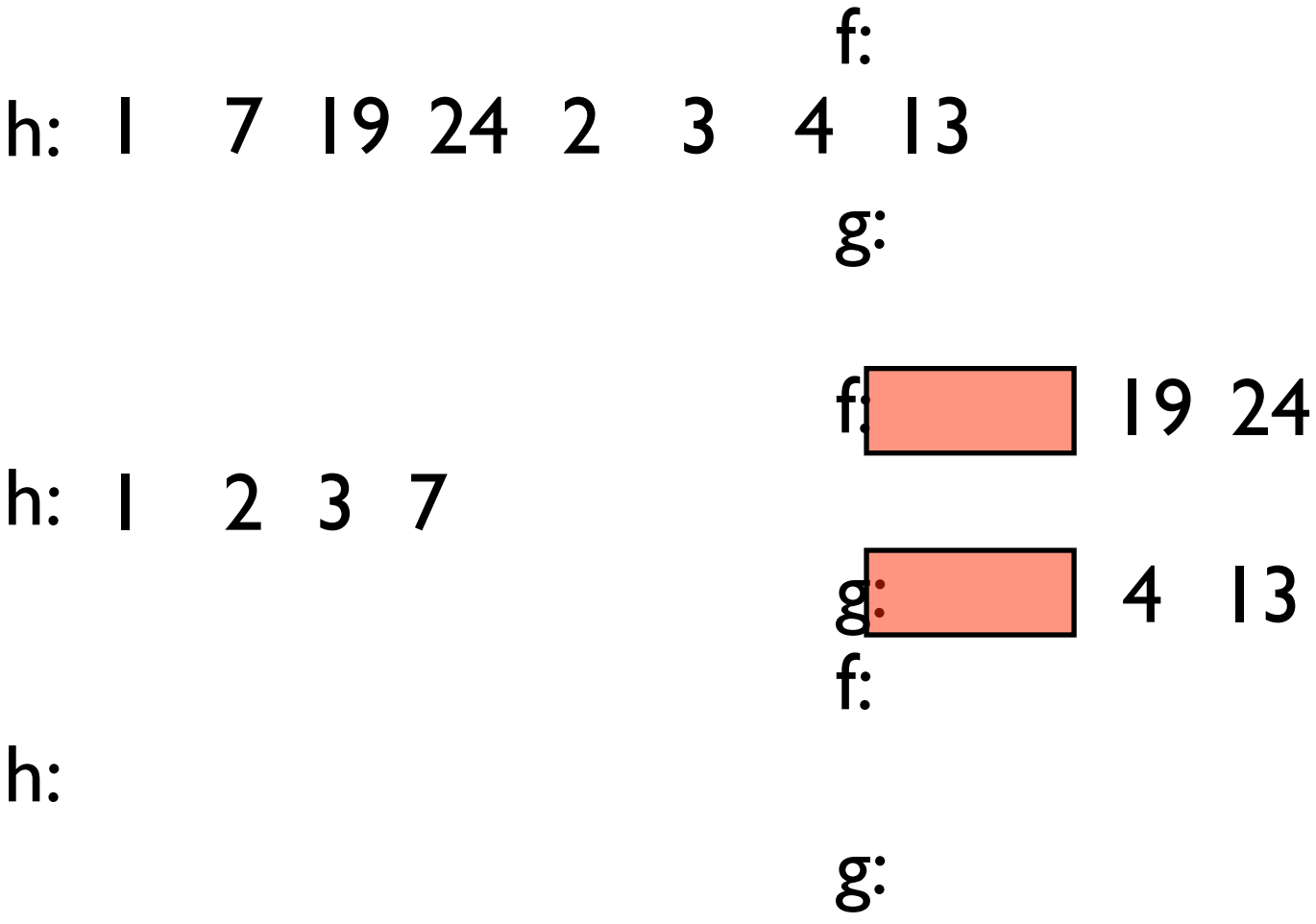
Effizienz

Sortieren durch Mischen - Beispiel



Effizienz

Sortieren durch Mischen - Beispiel



Effizienz

Sortieren durch Mischen - Beispiel

f:

h: 1 7 19 24 2 3 4 13

g:

f: 19 24

h: 1 2 3 7

g: 4 13

f:

h:

g:

Effizienz

Sortieren durch Mischen - Beispiel

f:

h: 1 7 19 24 2 3 4 13

g:

f: 19 24

h: 1 2 3 7

g: 4 13

f:

h:

g:

Effizienz

Sortieren durch Mischen - Beispiel

f:

h: 1 7 19 24 2 3 4 13

g:

f:

h: 1 2 3 7

g: 4 13

f:

h:

g:

Effizienz

Sortieren durch Mischen - Beispiel

f:

h: 1 7 19 24 2 3 4 13

g:

f:

h: 1 2 3 7

g:

f:

h:

g:

Effizienz

Sortieren durch Mischen - Beispiel

f:

h: 1 7 19 24 2 3 4 13

g:

f:

h: 1 2 3 7 4 13 19 24

g:

f:

h:

g:

Effizienz

Sortieren durch Mischen - Beispiel

f:

h: 1 7 19 24 2 3 4 13

g:

f:

h: 1 2 3 7 4 13 19 24

g:

f:

h:

g:

Effizienz

Sortieren durch Mischen - Beispiel

f:

h: 1 7 19 24 2 3 4 13

g:

f:

h: 1 2 3 7 4 13 19 24

g:

f: 1 2 3 7

h:

g:

Effizienz

Sortieren durch Mischen - Beispiel

f:

h: 1 7 19 24 2 3 4 13

g:

f:

h: 1 2 3 7 4 13 19 24

g:

f: 1 2 3 7

h:

g: 4 13 19 24

Effizienz

Sortieren durch Mischen - Beispiel

f:

h: 1 7 19 24 2 3 4 13

g:

f:

h: 1 2 3 7 4 13 19 24

g:

f:

h:

g: 4 13 19 24

Effizienz

Sortieren durch Mischen - Beispiel

f:

h: 1 7 19 24 2 3 4 13

g:

f:

h: 1 2 3 7 4 13 19 24

g:

f:

h:

g:

Effizienz

Sortieren durch Mischen - Beispiel

f:

h: 1 7 19 24 2 3 4 13

g:

f:

h: 1 2 3 7 4 13 19 24

g:

f:

h: 1 2 3 4 7 13 19 24

g:

Effizienz

Sortieren durch Mischen - Analyse

- 1. Phase: bilde n/2 Läufe der Länge 2
- 2. Phase: bilde n/4 Läufe der Länge 4
- 3. Phase: bilde n/8 Läufe der Länge 8
- ...
- k-te Phase: bilde $n/2^k$ Läufe der Länge 2^k
- nach log n Phasen liegt nur noch ein Lauf vor: Folge ist sortiert
- je Mischphase offenbar O(n) Operationen
- Laufzeit des Sortierens durch Mischen also:

$$T(n)=O(n \cdot \log_2 n)$$

Effizienz
Sortieren - Allgemeines

- Sortierverfahren schlechter als $O(n \cdot \log_2 n)$ sind unbrauchbar
- ... oder schlechter als $O(n)$ ... ?
- Wie schnell kann man überhaupt sortieren?

Effizienz
Sortieren - untere Schranken

- Grundfragen:
  - obere Schranke: wieviel Zeit benötigt man zur Lösung eines Problems höchstens?
    - Vorgehen: finde **einen** Algorithmus, bestimme seine Laufzeit => obere Schranke
  - untere Schranke: wieviel Zeit benötigt man zur Lösung eines Problems mindestens?
    - Vorgehen: überprüfe **alle** Algorithmen, bestimme ihre Laufzeit; "keiner zu schnell" => untere Schranke

Effizienz
Sortieren - untere Schranken

- Grundfragen:
  - obere Schranke: wieviel Zeit benötigt man zur Lösung eines Problems höchstens?
    - Vorgehen: finde **einen** Algorithmus, bestimme seine Laufzeit => obere Schranke
  - untere Schranke: wieviel Zeit benötigt man zur Lösung eines Problems mindestens?
    - Vorgehen: überprüfe **alle** Algorithmen, bestimme ihre Laufzeit; "keiner zu schnell" => untere Schranke

Effizienz
Sortieren - untere Schranken

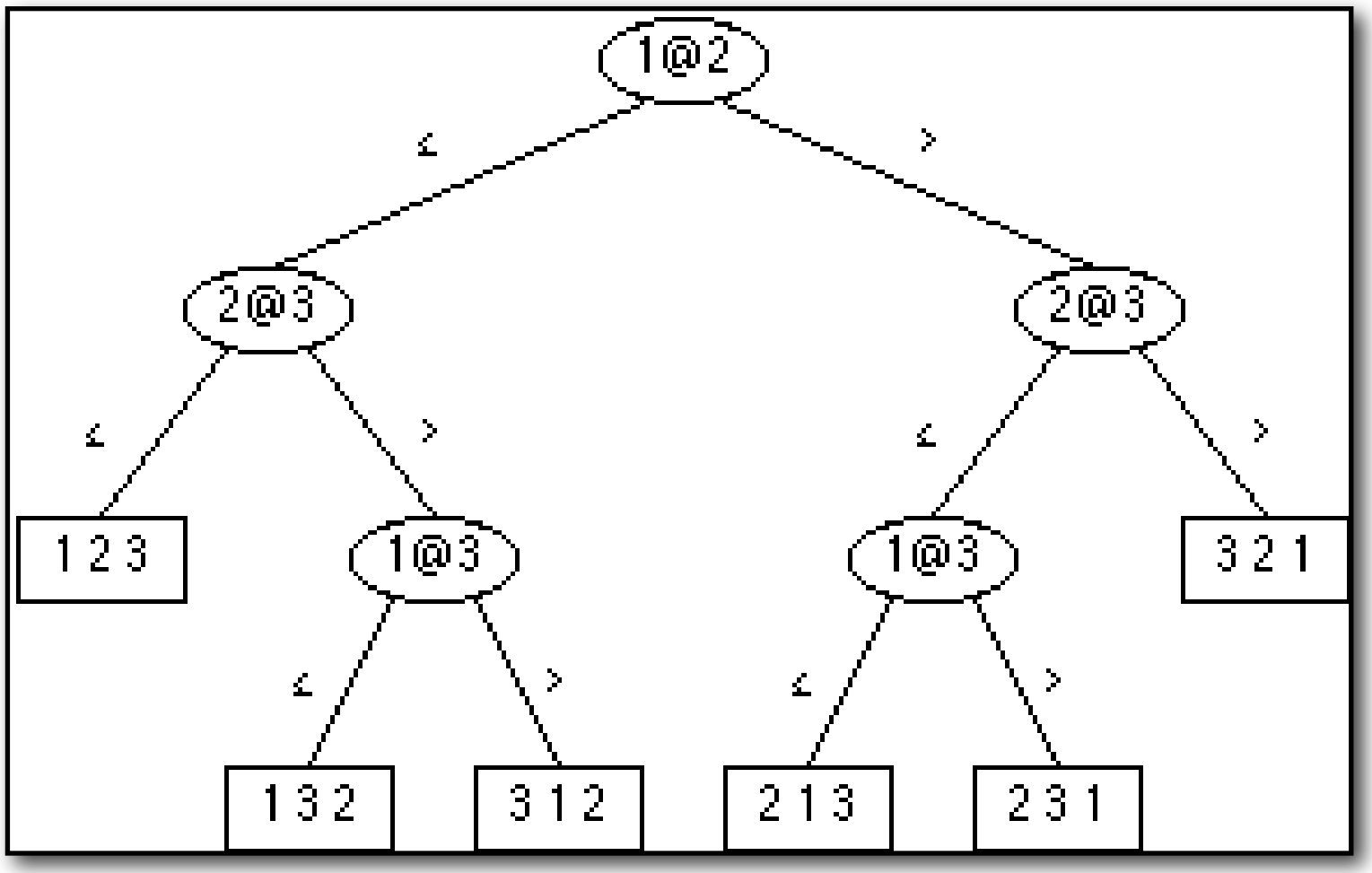
- Annahme: nur Sortieralgorithmen, die auf Vergleichen basieren
- Gesucht: Anzahl der Vergleiche, die im schlimmsten Fall notwendig ist, um die sortierte Reihenfolge herzustellen
- z.B. Bubblesort: $O(n^2)$ Vergleiche

Effizienz
Sortieren - n=3

- Sortierung von drei Elementen a_1,a_2,a_3 (paarweise verschieden
  - vergleiche a_1 mit a_2
    - anschließend a_2 mit a_3; falls $a_1<a_2$ und $a_2<a_3$, so war die Folge schon sortiert
    - falls $a_1<a_2$ und $a_2>a_3$, so
      - vergleiche noch a_1 mit a_3; falls z.B. $a_1<a_2$ und $a_1>a_3$, so ist a_3,a_1,a_2 die sortierte Folge.
- Diese aufeinanderfolgenden Vergleiche kann man als binären Baum (**Entscheidungsbaum**) darstellen

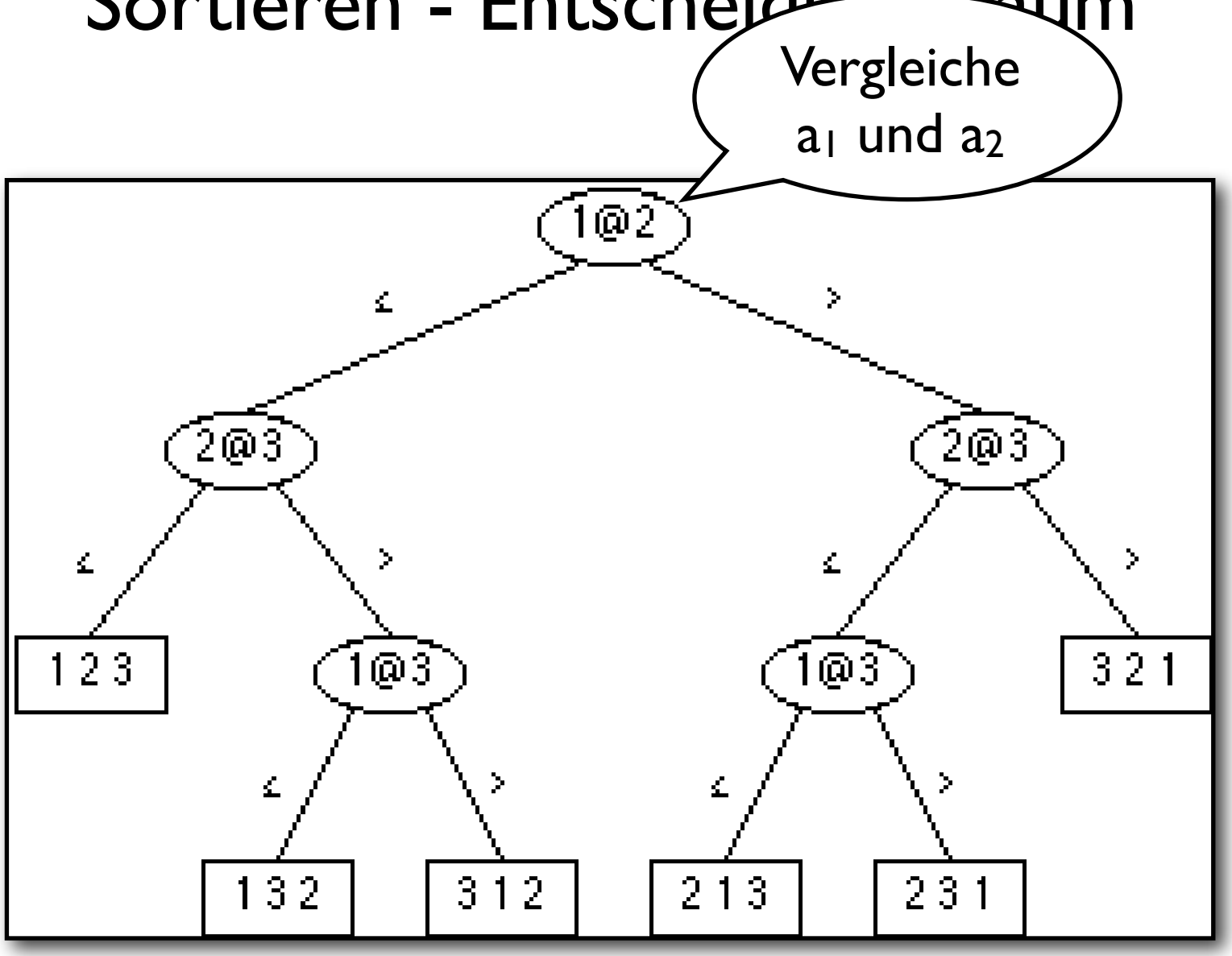
Effizienz

Sortieren - Entscheidungsbaum



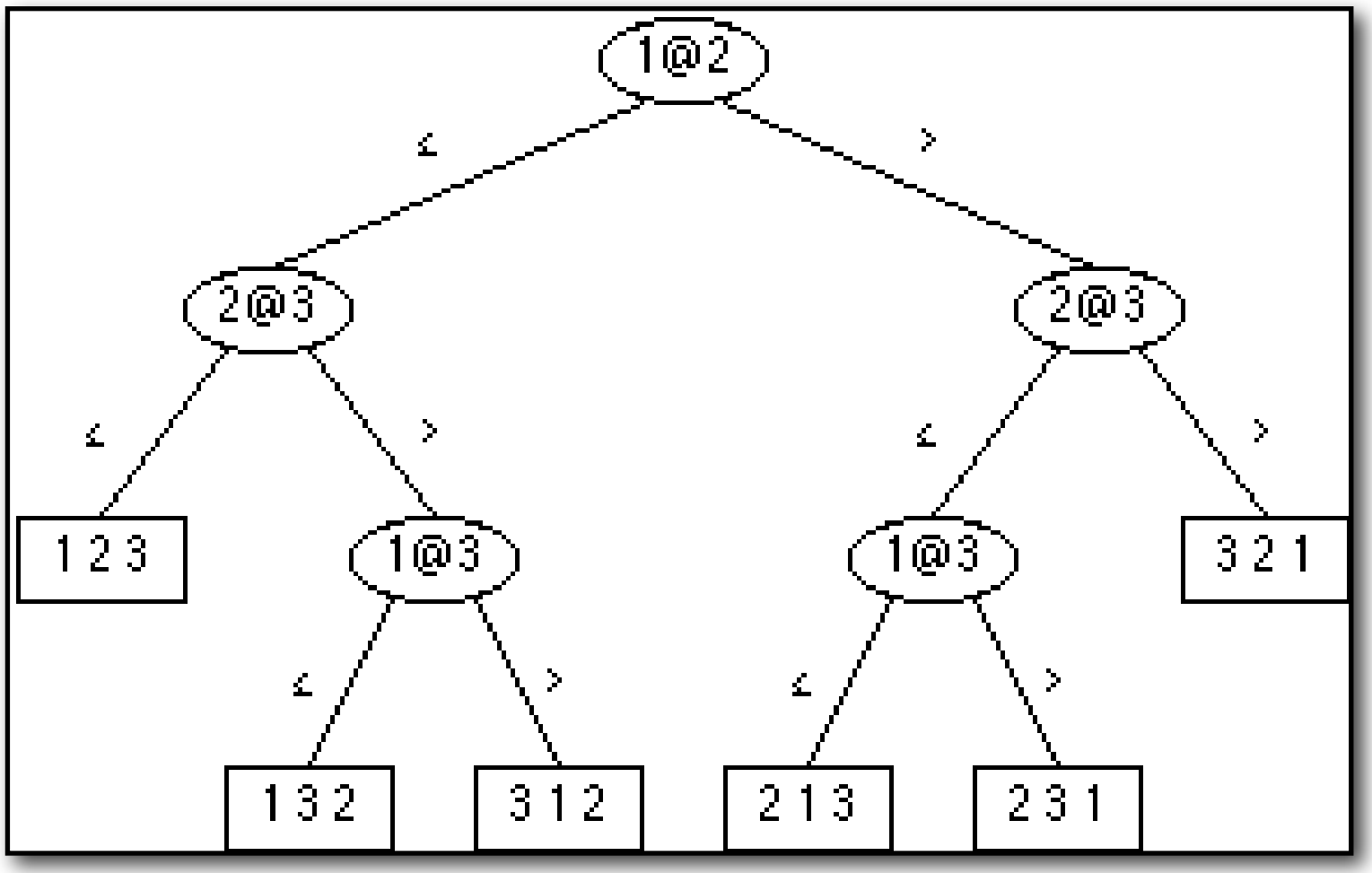
Effizienz

Sortieren - Entscheidungsbaum



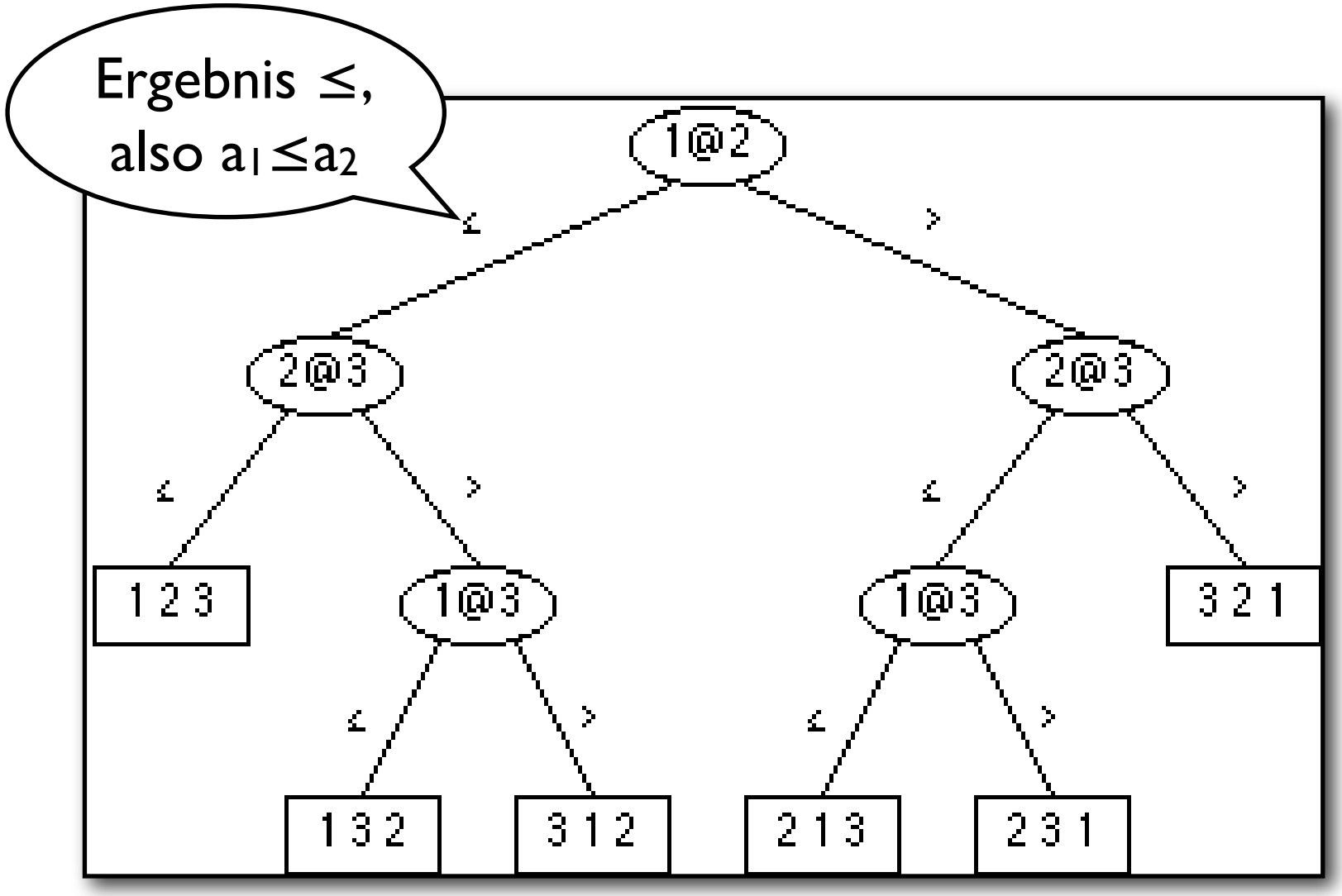
Effizienz

Sortieren - Entscheidungsbaum



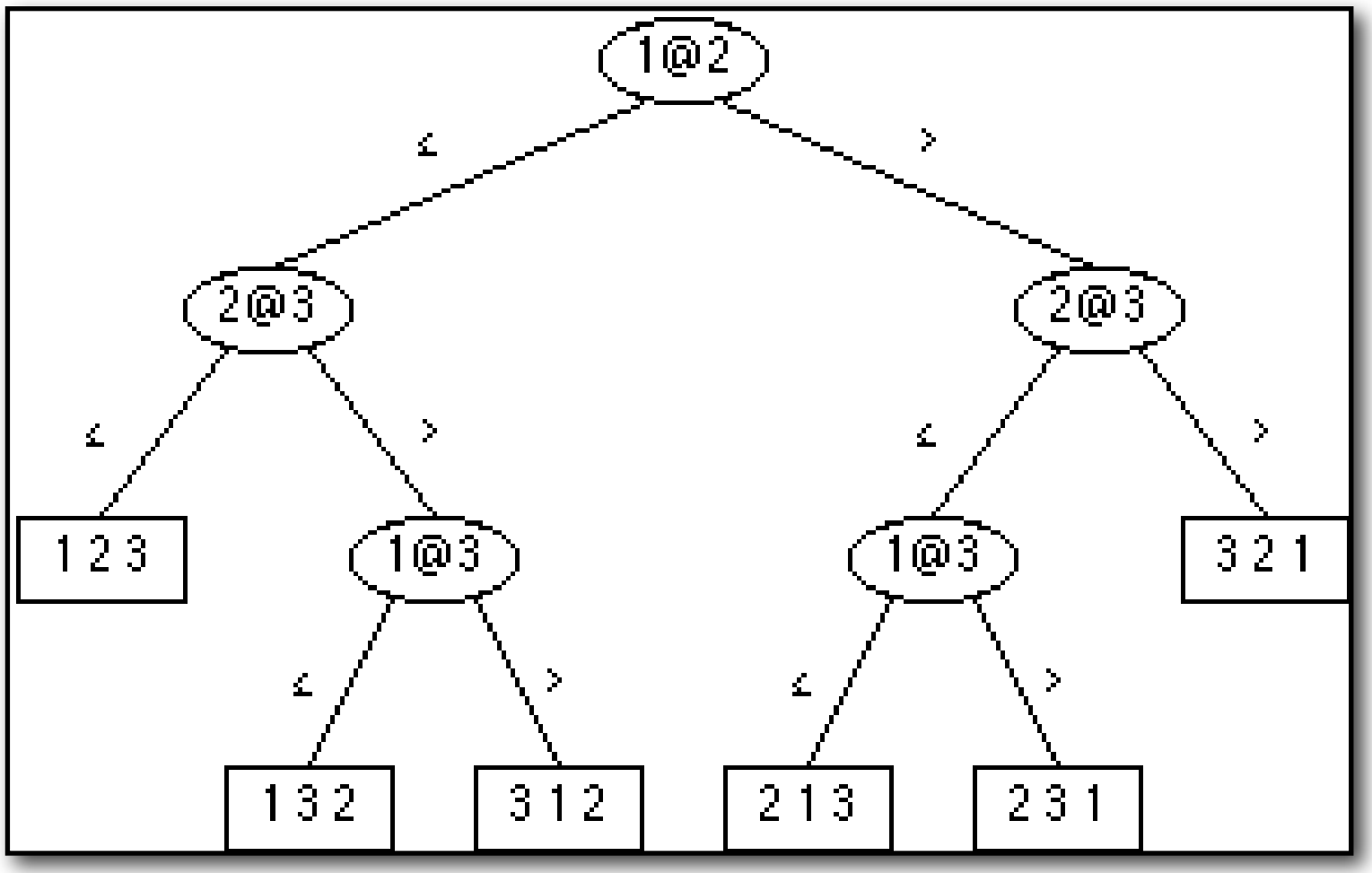
Effizienz

Sortieren - Entscheidungsbaum



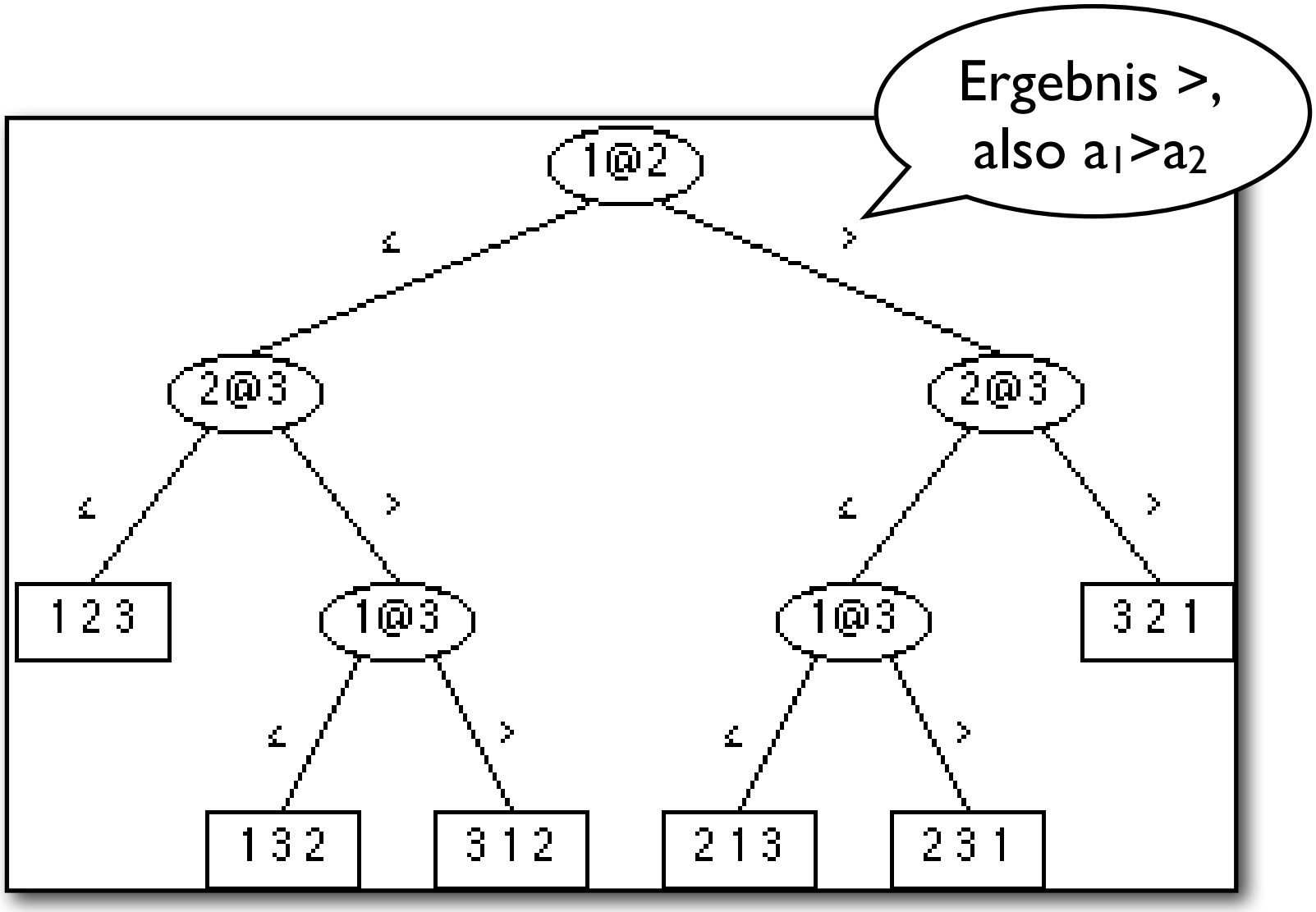
Effizienz

Sortieren - Entscheidungsbaum



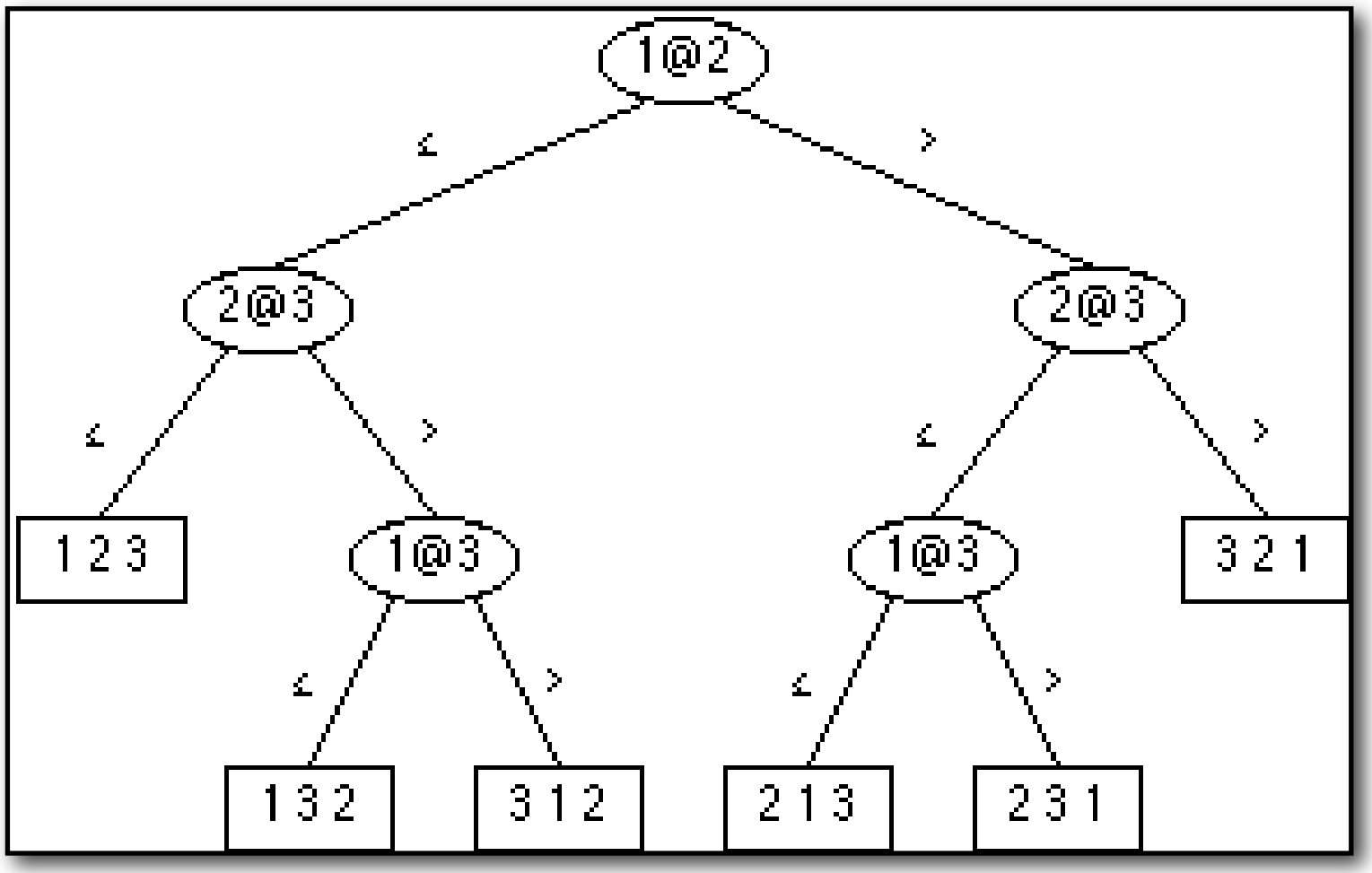
Effizienz

Sortieren - Entscheidungsbaum



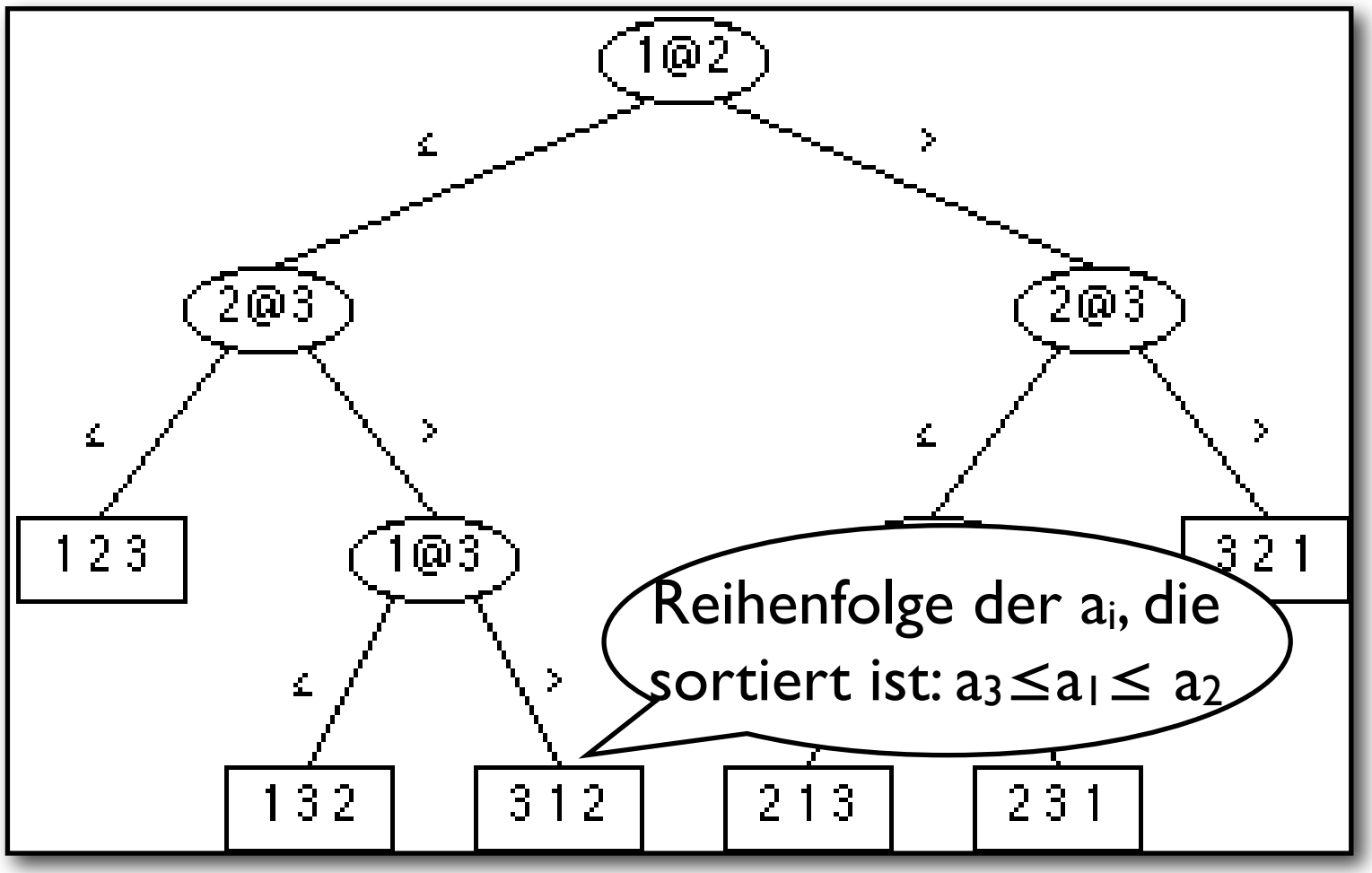
Effizienz

Sortieren - Entscheidungsbaum



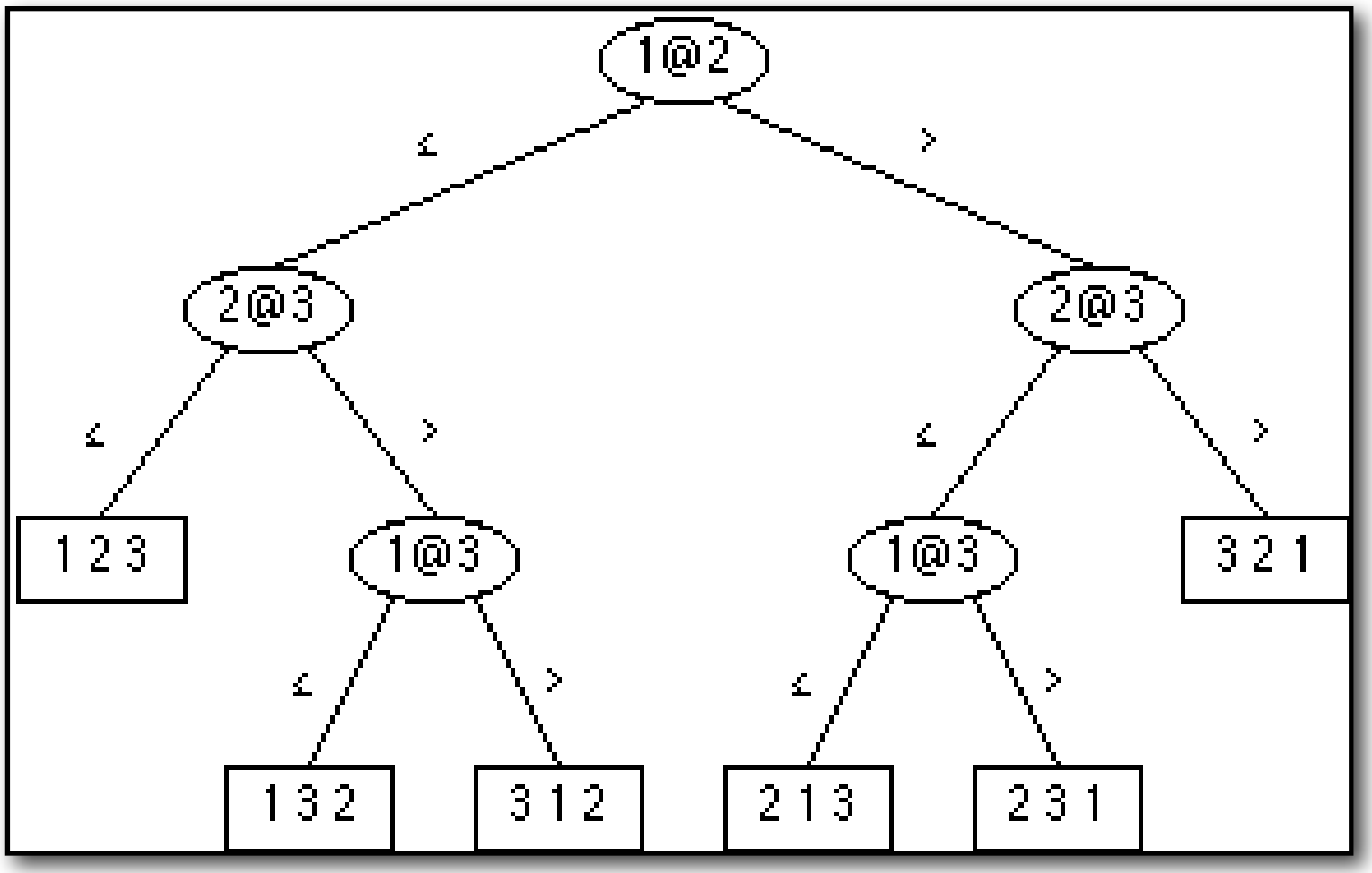
Effizienz

Sortieren - Entscheidungsbaum



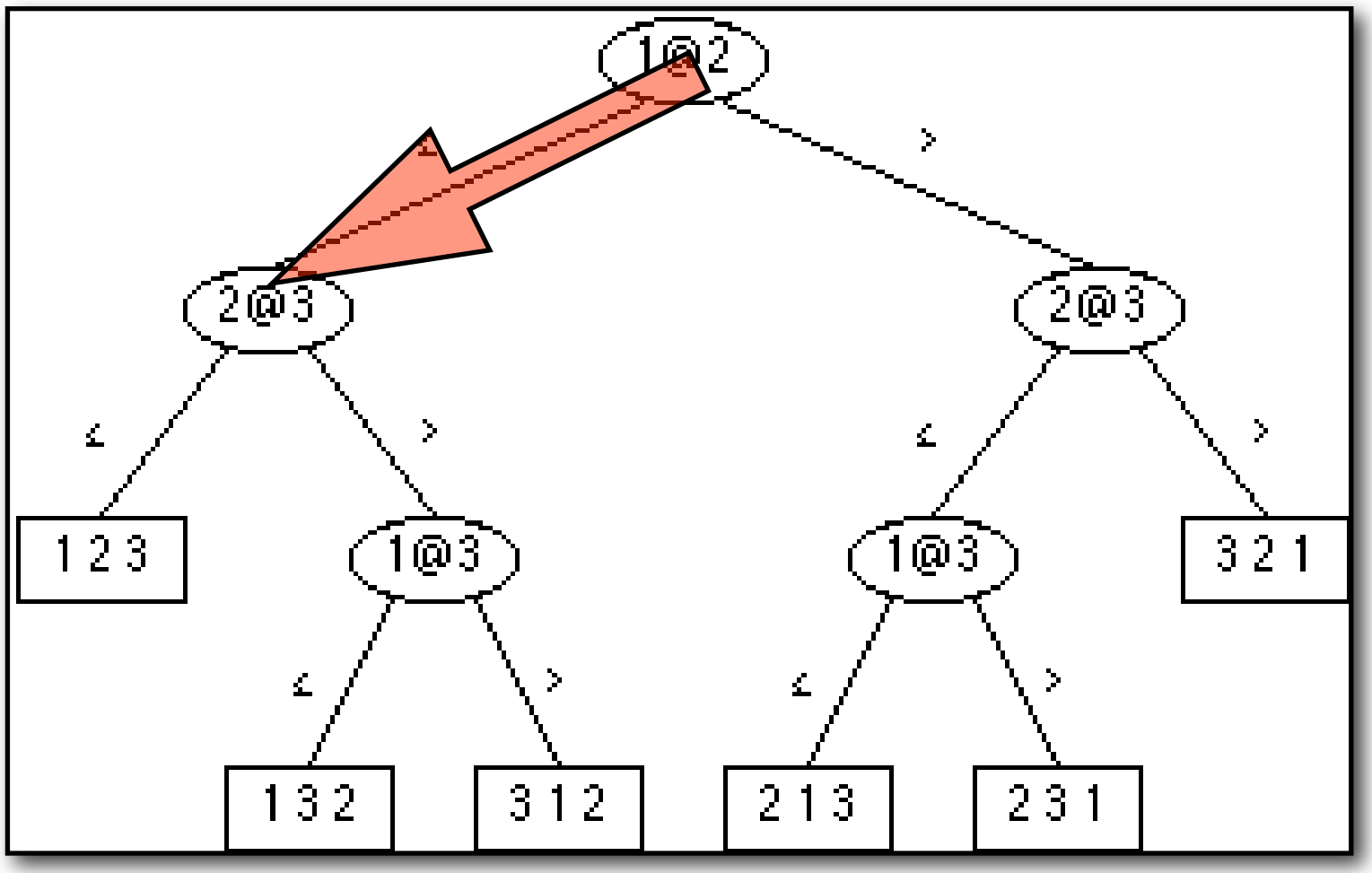
Effizienz

Sortieren - Entscheidungsbaum



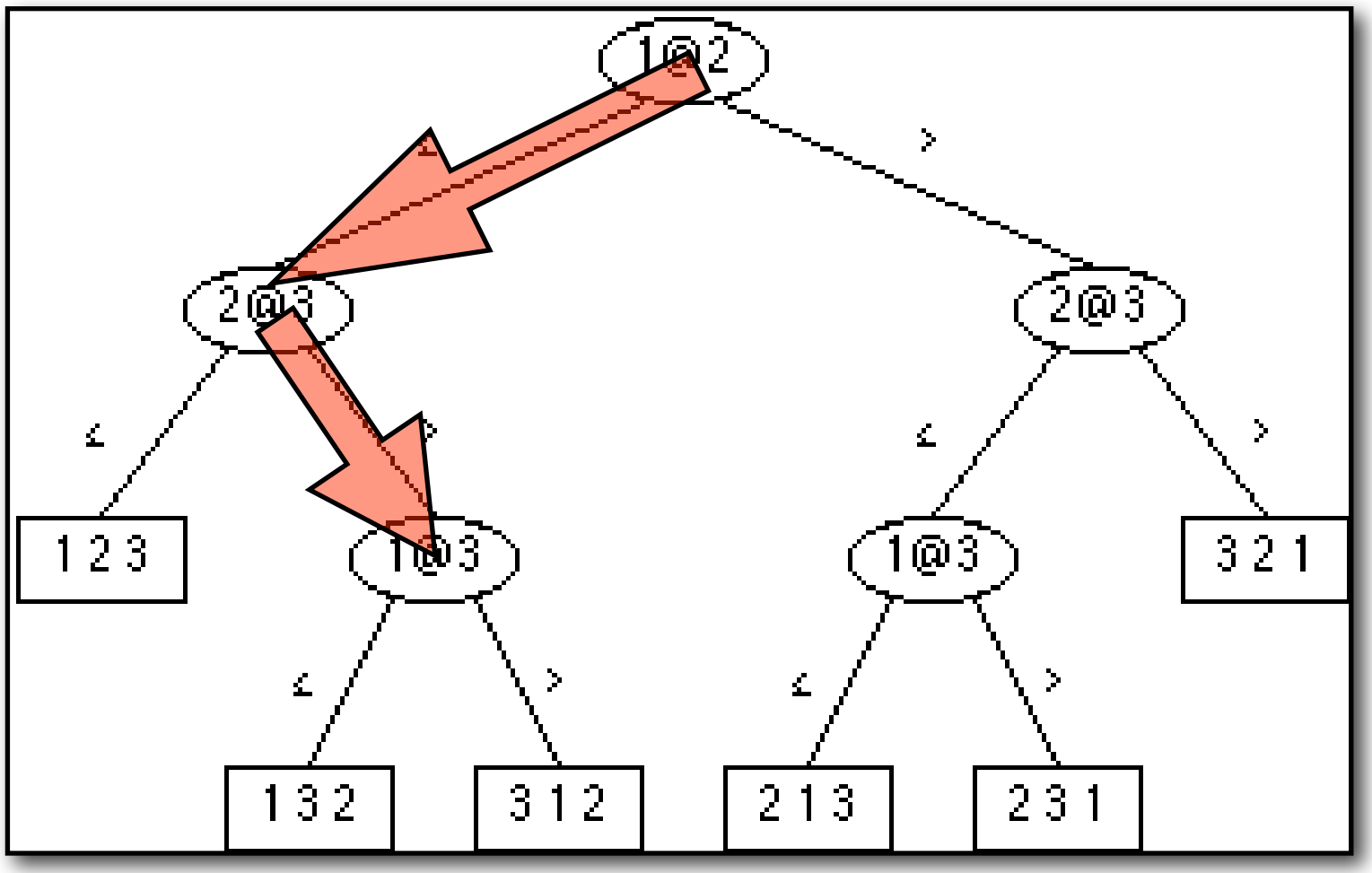
Effizienz

Sortieren - Entscheidungsbaum



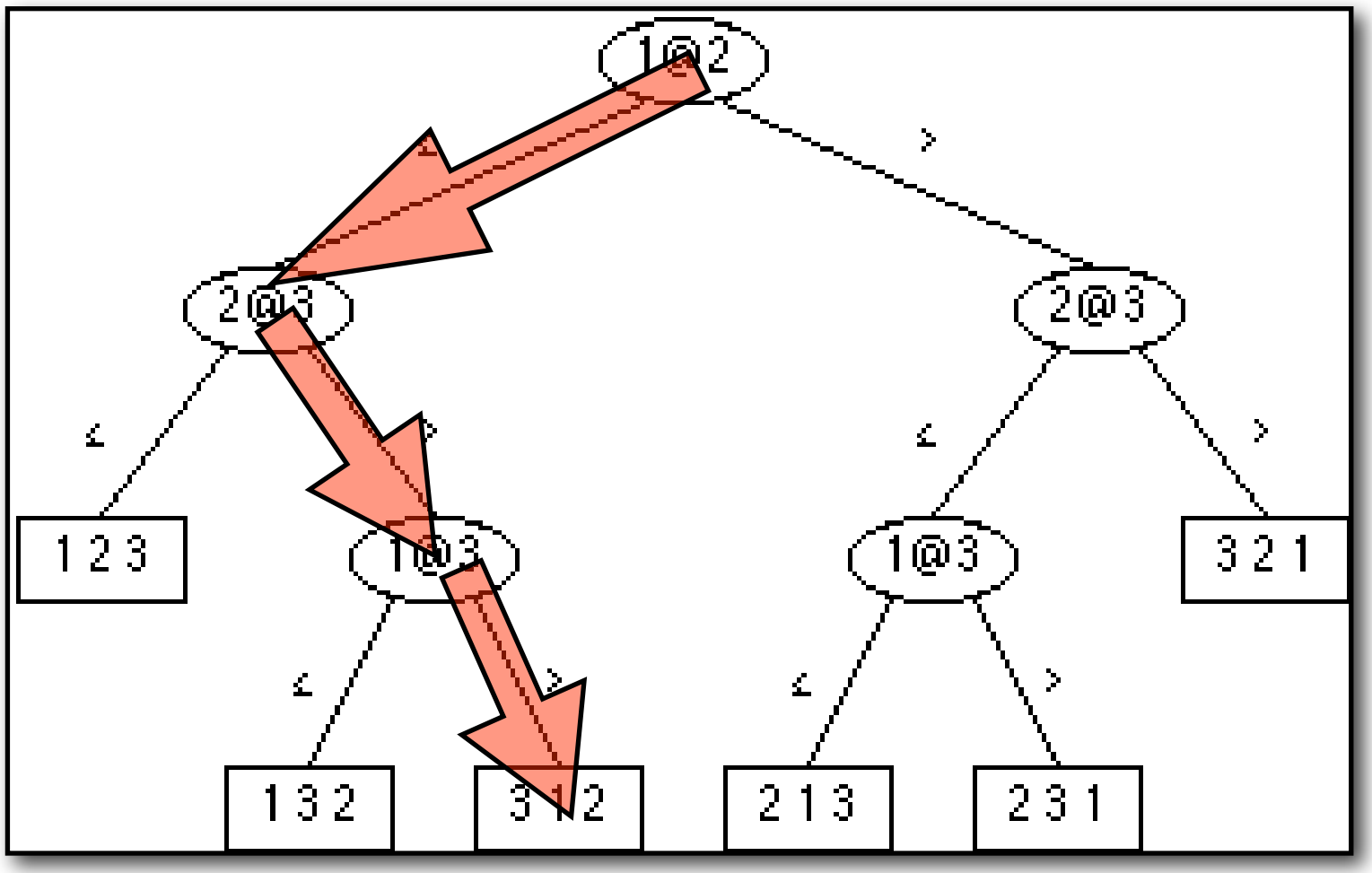
Effizienz

Sortieren - Entscheidungsbaum



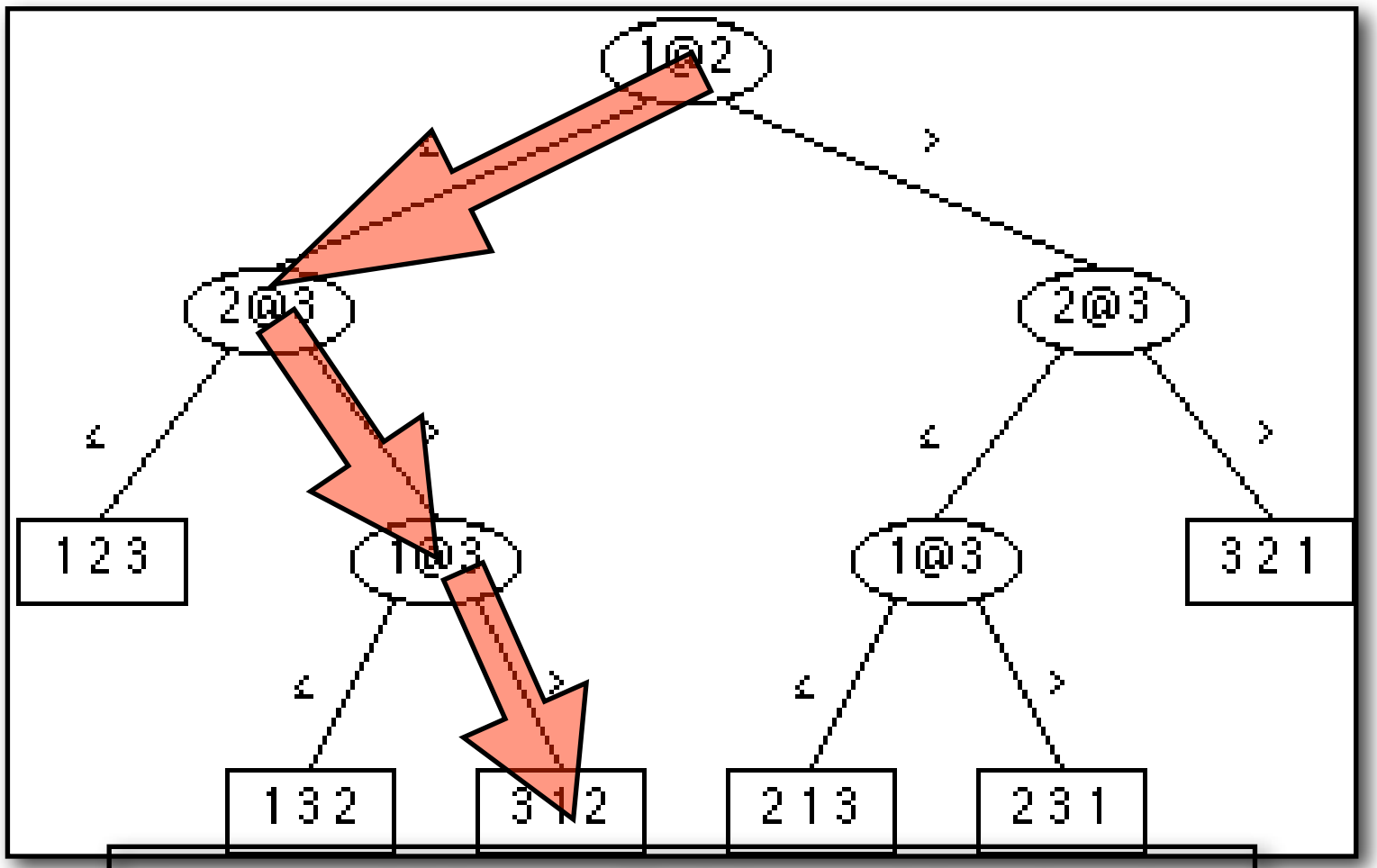
Effizienz

Sortieren - Entscheidungsbaum



Effizienz

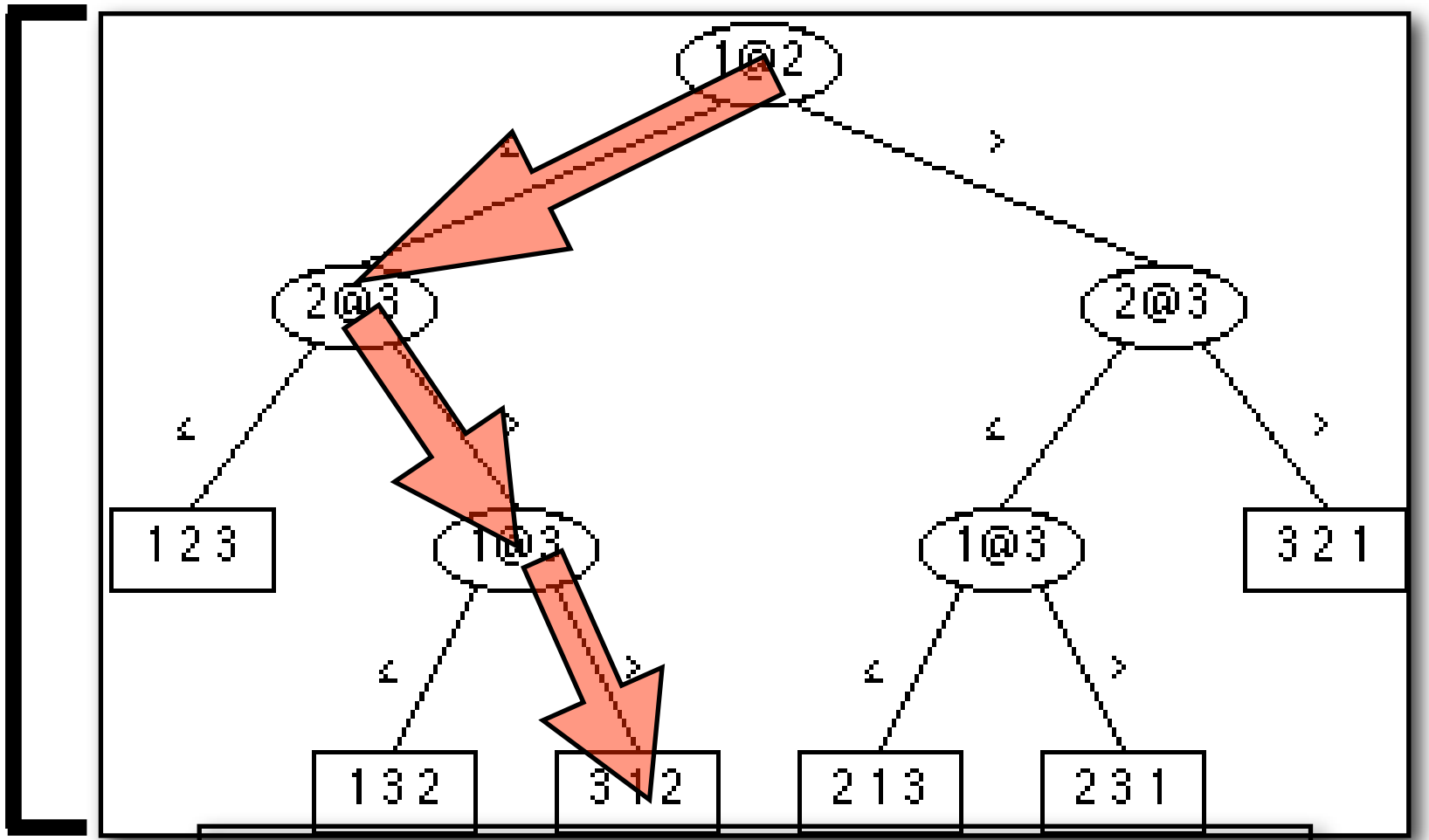
Sortieren - Entscheidungsbaum



längster Weg = Zahl von Vergleichen

Effizienz

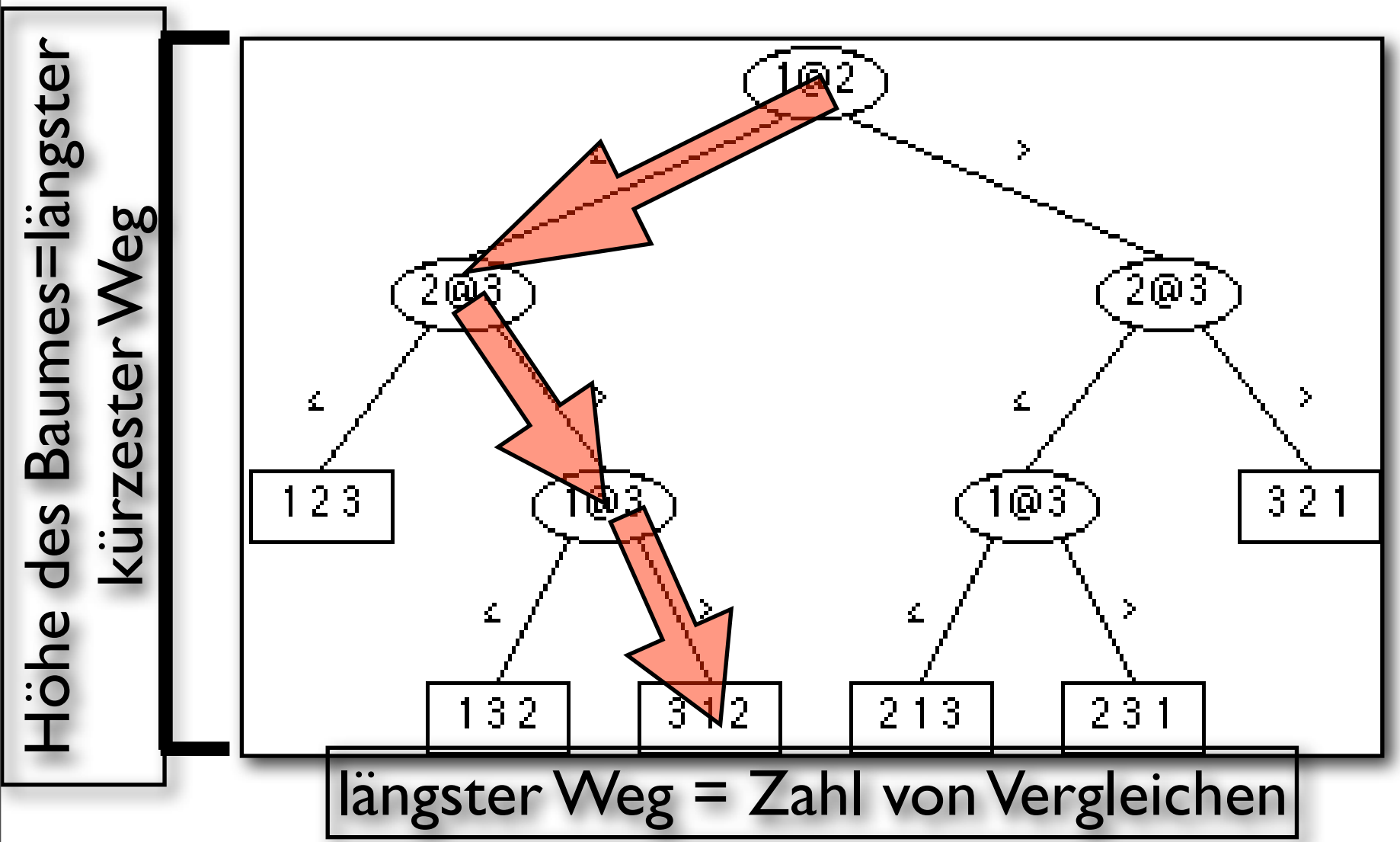
Sortieren - Entscheidungsbaum



längster Weg = Zahl von Vergleichen

Effizienz

Sortieren - Entscheidungsbaum



Effizienz
Sortieren - Entscheidungsbaum

- Baum geringster Höhe für festes n: Sortierverfahren mit minimaler Laufzeit im schlimmsten Fall

Definition:

Ein **Entscheidungsbaum** für eine Folge $a_1,a_2,...,a_n$ ist ein binärer Baum, dessen Knoten mit Ausnahme der Blätter mit Markierungen der Form i@j versehen sind. Die beiden Kanten zu den Söhnen jedes Knotens sind mit $\leq$ bzw. $>$ markiert. Jedes Blatt ist mit der Umordnung der Folge $a_1,...,a_n$ markiert, die alle Vergleiche erfüllt, die auf dem Weg von der Wurzel zu diesem Blatt auftreten.

Effizienz
Sortieren - Entscheidungsbaum

Zusammenfassung:

- Jeder Algorithmus, der auf Vergleichen beruht, entspricht einem Entscheidungsbaum
- Laufzeit eines Algorithmus im schlimmsten Fall ist mindestens gleich Maximalzahl von Vergleichen, um die Folge zu sortieren
- Diese Zahl ist gleich Länge des längsten Weges von der Wurzel zu einem Blatt (minus Eins)

Effizienz
Sortieren - Entscheidungsbaum

- Sei $V_B(n)$: Maximalzahl von Vergleichen, die zur Sortierung von n Objekten mit Entscheidungsbaum B nötig ist
- Gesucht:

  $\min\{V_B(n) |$ B ist Entscheidungsbaum für n Elemente$\}$
- Anschaulich:
  - Suche den flachsten Entscheidungsbaum zur Sortierung von n Objekten
  - Seine Höhe ist Mindestzahl von Vergleichen, die man im schlimmsten Fall aufwenden muß, um n Objekte zu sortieren

Effizienz
Sortieren - Entscheidungsbaum

- Wie flach kann denn ein Entscheidungsbaum überhaupt sein?
  - er soll n Objekte sortieren
  - er muß genügend groß sein, um alle Blätter aufzunehmen
  - als Blätter können alle Sortierreihenfolgen der n Objekte auftreten
  - der Baum muß Platz für n! Blätter haben

Effizienz

Sortieren - Entscheidungsbaum

- Ein Baum B der Höhe $V_B(n)$ besitzt $\leq 2^{V_B(n)}$ Blätter (selbst nachweisen)
- Folglich:

$$2^{V_B(n)} \geq n!$$

$$V_B(n) \geq \log_2(n!)$$

- Jeder Sortieralgorithmus benötigt im schlimmsten Fall mindestens

$$\log_2(n!)$$

Vergleiche, um n Elemente zu sortieren.

Effizienz

Sortieren - Entscheidungsbaum

Weiterrechnen liefert:

$$\log_2(n!) = \sum_{i=1}^{n} \log_2 i \geq \sum_{i=n/2}^{n} \log_2 i \geq n/2 \cdot \log_2(n/2) = n/2 \cdot (\log_2 n - 1)$$

Effizienz

Sortieren - Entscheidungsbaum

Weiterrechnen liefert:

$$\log_2(n!) = \sum_{i=1}^{n} \log_2 i \geq \sum_{i=n/2}^{n} \log_2 i \geq n/2 \cdot \log_2(n/2) = n/2 \cdot (\log_2 n - 1)$$

Effizienz

Sortieren - Entscheidungsbaum

Weiterrechnen liefert:

$$\log_2(n!) = \sum_{i=1}^{n} \log_2 i \geq \sum_{i=n/2}^{n} \log_2 i \geq n/2 \cdot \log_2(n/2) = n/2 \cdot (\log_2 n - 1)$$

Effizienz

Sortieren - Entscheidungsbaum

Weiterrechnen liefert:

$$\log_2(n!) = \sum_{i=1}^{n} \log_2 i \geq \sum_{i=n/2}^{n} \log_2 i \geq n/2 \cdot \log_2(n/2) = n/2 \cdot (\log_2 n - 1)$$

Effizienz

Sortieren - Entscheidungsbaum

Weiterrechnen liefert:

$$\log_2(n!) = \sum_{i=1}^{n} \log_2 i \geq \sum_{i=n/2}^{n} \log_2 i \geq n/2 \cdot \log_2(n/2) = n/2 \cdot (\log_2 n - 1)$$

Effizienz

Sortieren - Entscheidungsbaum

Weiterrechnen liefert:

$$\log_2(n!) = \sum_{i=1}^{n} \log_2 i \geq \sum_{i=n/2}^{n} \log_2 i \geq n/2 \cdot \log_2(n/2) = n/2 \cdot (\log_2 n - 1)$$

Effizienz

Sortieren - Entscheidungsbaum

Weiterrechnen liefert:

$$\log_2(n!) = \sum_{i=1}^{n} \log_2 i \geq \sum_{i=n/2}^{n} \log_2 i \geq n/2 \cdot \log_2(n/2) = n/2 \cdot (\log_2 n - 1)$$

Effizienz

Sortieren - Entscheidungsbaum

Weiterrechnen liefert:

$$\log_2(n!) = \sum_{i=1}^{n} \log_2 i \geq \sum_{i=n/2}^{n} \log_2 i \geq n/2 \cdot \log_2(n/2) = n/2 \cdot (\log_2 n - 1)$$

Effizienz

Sortieren - Entscheidungsbaum

Weiterrechnen liefert:

$$\log_2(n!) = \sum_{i=1}^{n} \log_2 i \geq \sum_{i=n/2}^{n} \log_2 i \geq n/2 \cdot \log_2(n/2) = n/2 \cdot (\log_2 n - 1)$$

Effizienz

Sortieren - Entscheidungsbaum

Weiterrechnen liefert:

$$\log_2(n!) = \sum_{i=1}^{n} \log_2 i \geq \sum_{i=n/2}^{n} \log_2 i \geq n/2 \cdot \log_2(n/2) = n/2 \cdot (\log_2 n - 1)$$

Satz:
Jedes allgemeine auf Vergleichen beruhende Sortierverfahren benötigt im schlimmsten Fall mindestens $O(n \cdot \log_2 n)$ Vergleiche bzw. Rechenschritte.

Effizienz

Sortieren - Entscheidungsbaum

Weiterrechnen liefert:

$$\log_2(n!) = \sum_{i=1}^{n} \log_2 i \geq \sum_{i=n/2}^{n} \log_2 i \geq n/2 \cdot \log_2(n/2) = n/2 \cdot (\log_2 n - 1)$$

Satz:
Jedes allgemeine auf Vergleichen beruhende Sortierverfahren benötigt im schlimmsten Fall mindestens $O(n \cdot \log_2 n)$ Vergleiche bzw. Rechenschritte.

“Sortieren durch Mischen" ist optimal.