Grundlagen der Informatik

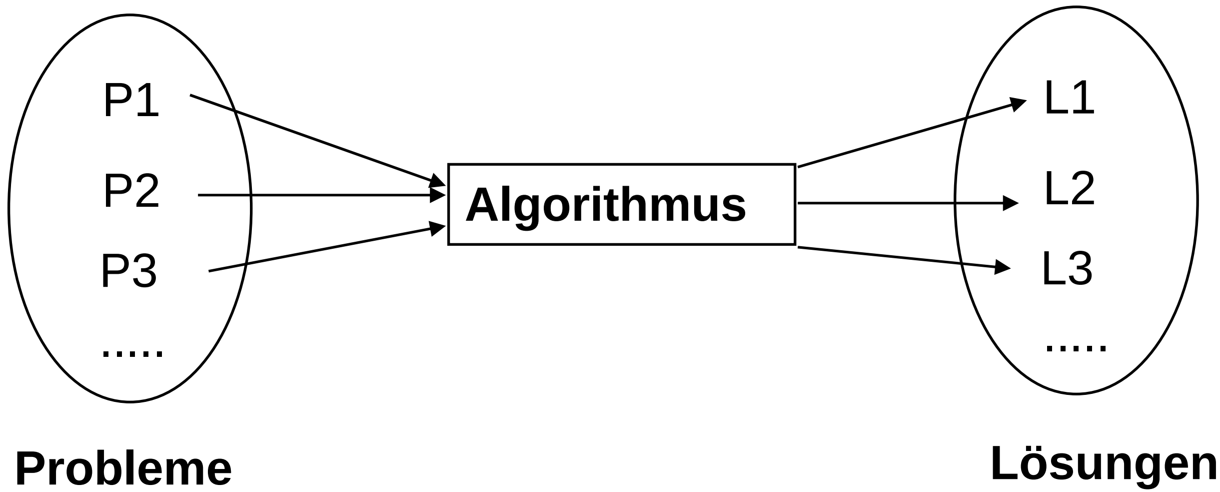
1. Semester, 1999

Kapitel 2: Algorithmen

2.1 Eigenschaften von Algorithmen

Definition:

Ein Algorithmus ist eine Vorschrift zur Lösung einer Klasse gleichartiger Probleme



Beispiele: (Problemklassen)

- Sortieren von Namenslisten
- Bestimmen kürzester Wege in Graphen
- Lineare Gleichungssysteme

Aufbau eines Algorithmus

Ein Algorithmus setzt sich aus 4 Teilen zusammen:

Name des Algorithmus
Beschreibung der Eingabedaten
Beschreibung der Ausgabedaten
Vorschrift

Beispiel:

Algorithmus: FLOYD

Eingabedaten: Graph mit positven Entfernungsangaben zwischen den Knoten

Ausgabedaten: Tabelle, die für je 2 Knoten die kürzeste Entfernung angibt

Vorschrift: ...... siehe später ....

2.1.1 Forderungen an Vorschrift

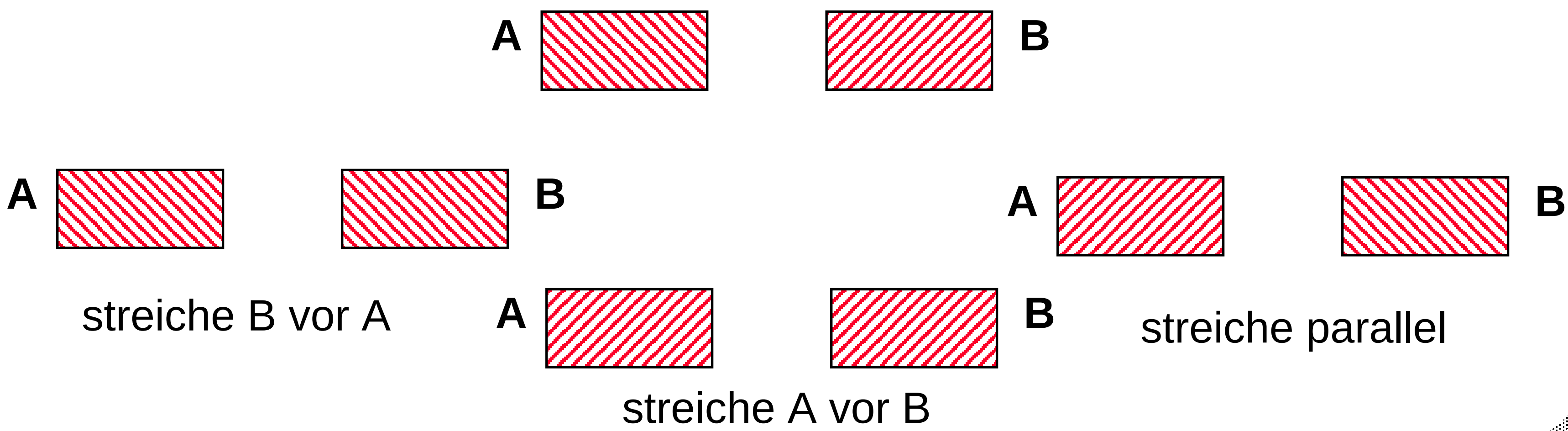
- Die Vorschrift muß durch einen **Text endlicher Länge** beschrieben werden
- Die Vorschrift ist aus **Einzelschritten** zusammengesetzt
- Einzelschritte müssen **ausführbar** sein und eine **eindeutige Wirkung** besitzen

    nicht: "Finde Yeti und Nessie."

    "Rühre die Mischung bis sie sämig ist"

- Die **Reihenfolge** der Einzelschritte muß **eindeutig** festgelegt sein (Kontrollfluß)

    nicht: "Streiche A im gleichen Muster wie B und B im gleichen Muster wie A"



Forderungen an Vorschrift

- Der **Datenfluß** muß **eindeutig bestimmt** sein ("welche Operation mit welchen Daten?")

  nicht: "Verbinde zwei Punkte des Rechtecks durch eine Gerade"

- Die Vorschrift muß nach **endlich vielen Schritten** zum Abbruch kommen, egal welche Eingabedaten verwendet wurden

  nicht: "Spiele so lange Lotto bis Du 1 Million DM gewonnen hast"

2.1.2 Spezifikation von Algorithmen

Um die Einsetzbarkeit eines Algorithmus zu beurteilen genügen oft

- sein Name

- Eigenschaften seiner Eingabedaten

- Eigenschaften seiner Ausgabedaten

Diese Information wird durch eine Spezifikation vermittelt, welche die Form hat: {
... **Eigenschaften der Eingabedaten** ... }
... **Name des Algorithmus** { ... **Eigenschaften der Ausgabedaten** ... }

Beispiele:

- { N ist eine natürliche Zahl }
PERMUTATION
{ P ist eine Permutation der Zahlen 1, 2, ..., N }

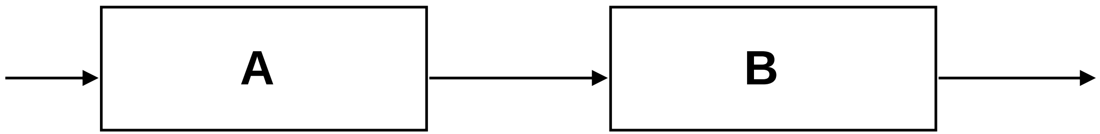
- { L ist eine aufsteigend sortierte Liste von Namen, N ist ein Name }
BINSEARCH {
Position von N in L (falls N in L enthalten ist); sonst 0 }

2.1.3 Zusammensetzung von Algorithmen

Prinzipiell gibt es 3 Arten mit denen man aus Algorithmen komplexere

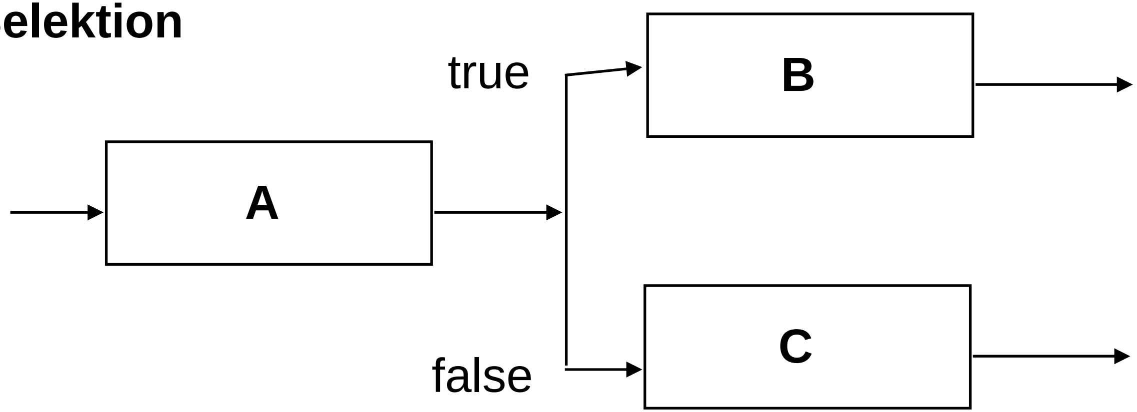
Algorithmen zusammensetzen kann:

- Komposition



Notation: A ; B

- Selektion



Bedingung:

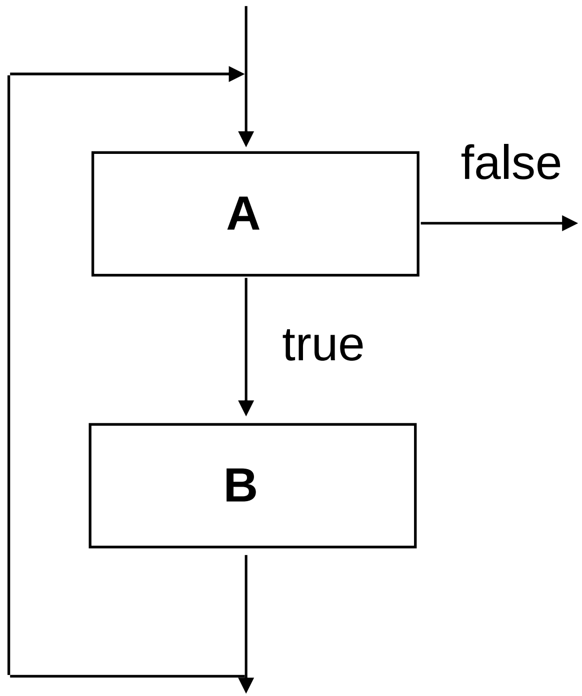
Das Ergebnis von A

muß true oder false sein

Notation: if A then B else C

Zusammensetzung von Algorithmen

- Iteration



Bedingung:

- Das Ergebnis von A muß true oder false sein
- A muß garantiert nach endlich vielen Ausführungen das Ergebnis false liefern

Notation: while A do B

Zusammensetzung von Algorithmen

Frage: Was sind die Voraussetzungen um Algorithmen A, B un C zusammenzusetzen durch

- A ; B
- if A then B else C
- while A do B ?

Komposition:

Gegeben A mit der Spezifikation { VA } A { NA }

B mit der Spezifikation { VB } B { NB }

Falls aus der Gültigkeit von { NA } die Gültigkeit von { VB } folgt, gilt die Specifikation { VA } A ; B { NB }

Notation:

$$\frac{\{ VA \}\ A\ \{ NA \},\ \{ VB \}\ B\ \{ NB \},\ \{ NA \} \Rightarrow \{ VB \}}{\{ VA \}\ A ; B\ \{ NB \}}$$

Häufig sind { NA } und { VB } identisch!

Zusammensetzung von Algorithmen

Beispiel (Komposition)

{ L ist eine Liste von maximal 1000 Namen }
HEAPSORT
{ L ist eine aufsteigend sortierte Liste von maximal 1000 Namen }

{ L ist eine aufsteigend sortierte Liste von maximal 1000 Namen, N ist ein Name }
BINSEARCH
{ P ist die Position von N in L, falls N in L enthalten ist; sonst ist P gleich 0 }

$\Downarrow$

{ L ist eine Liste von maximal 1000 Namen, N ist ein Name }
HEAPSORT ; BINSEARCH
{ P ist die Position von N in L, falls N in L enthalten ist; sonst ist P gleich 0 }

Zusammensetzung von Algorithmen

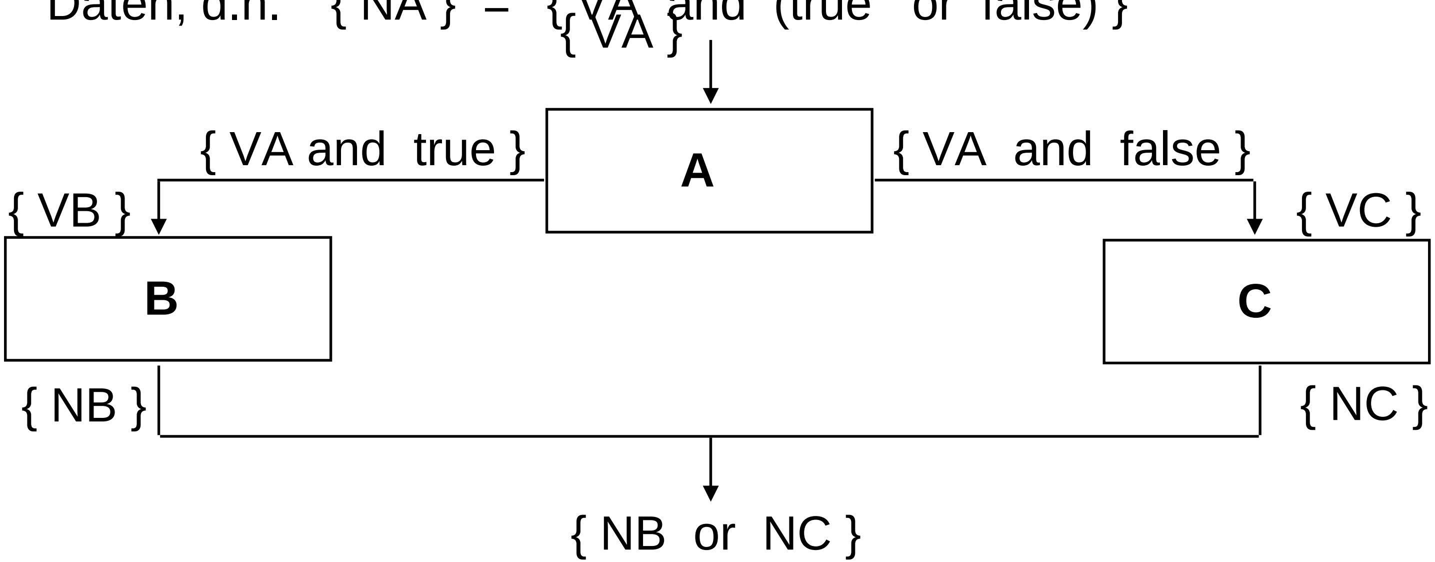
Selektion:

Gegeben A mit der Spezifikation { VA } A { NA }

B mit der Spezifikation { VB } B { NB }

C mit der Spezifikation { VC } C { NC }

Forderung: Das Ergebnis von A ist true oder false und A verändert sonst keine Daten, d.h. { NA } = { VA and (true or false) }



Zusammensetzung von Algorithmen

Falls { VB } aus { VA and true } folgt und falls { VC } aus { VA and false } folgt,

dann gilt die Spezifikation { VA } if A then B else C { NB or NC }

Notation: { VA } A { VA and (true or false) } ,

{ VA and true } => { VB } , { VA and false } => { VC }

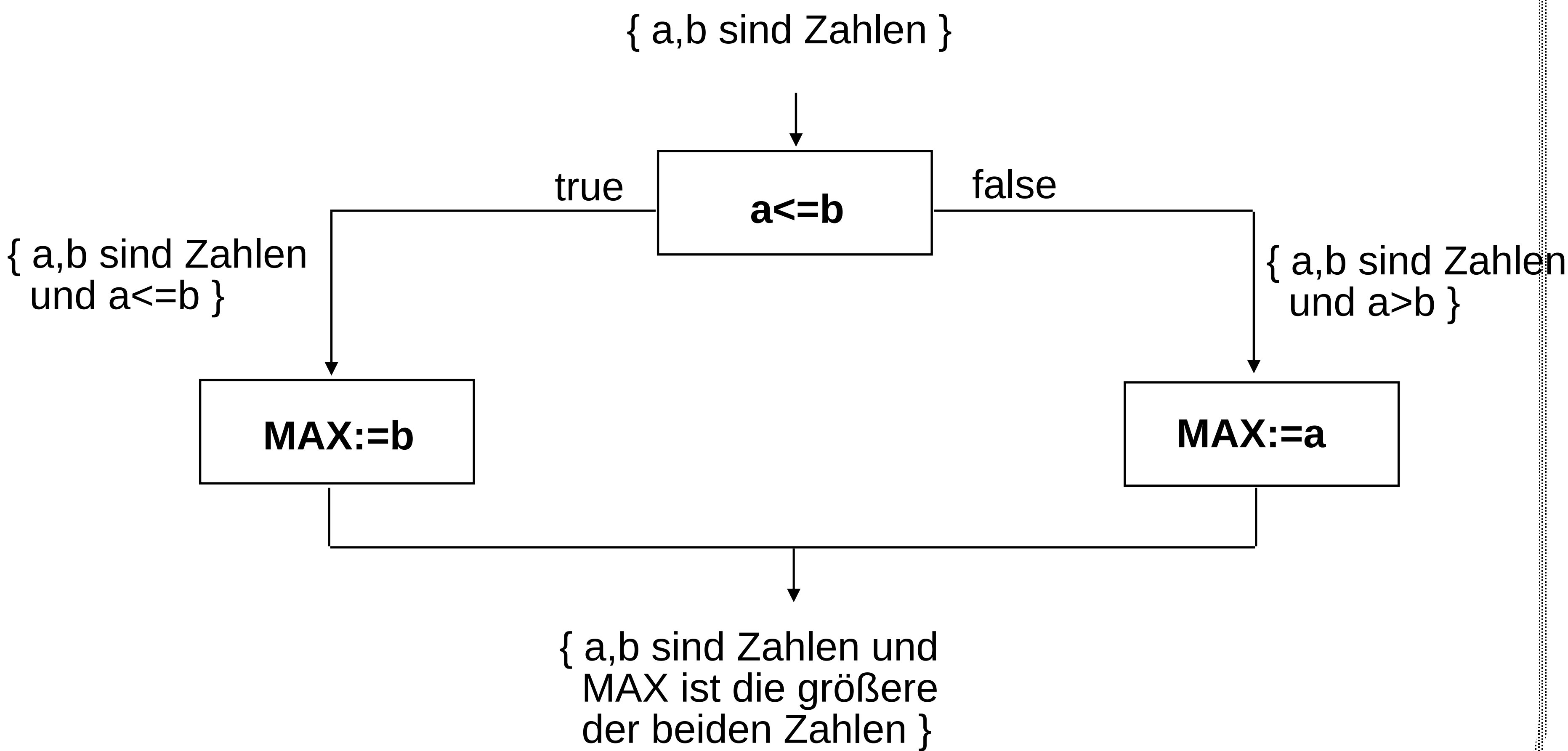
{ VB } B { NB } , { VC } C { NC }

{ VA } if A then B else C { NB or NC }

Häufig sind { NB } und { NC } identisch!

Zusammensetzung von Algorithmen

Beispiel: Maximum zweier Zahlen



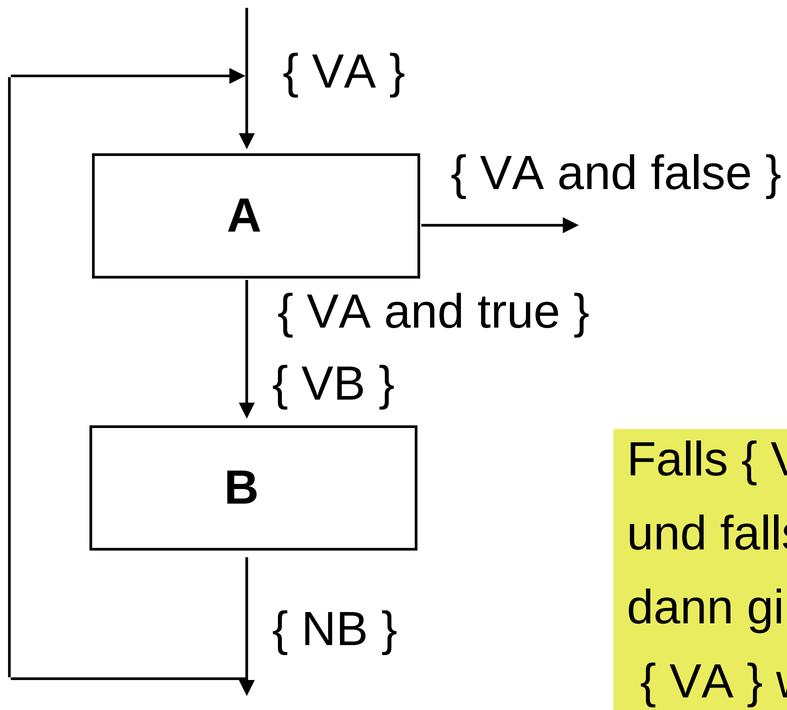
Zusammensetzung von Algorithmen

Iteration

Gegeben A mit der Spezifikation { VA } A { NA }

wobei { NA } wie bei der Selektion die Form { VA and (true or false) } hat

B mit der Spezifikation { VB } A { NB }



Falls { VB } aus { VA and true } folgt
und falls { VA } aus { NB } folgt
dann gilt die Spezifikation
{ VA } while A do B { VA and false }

Zusammensetzung von Algorithmen

Notation: { VA } A { VA and (true or false) } ,

{ VA and true } => { VB } , { VB } B { NB } ,

{ NB } => { VA }

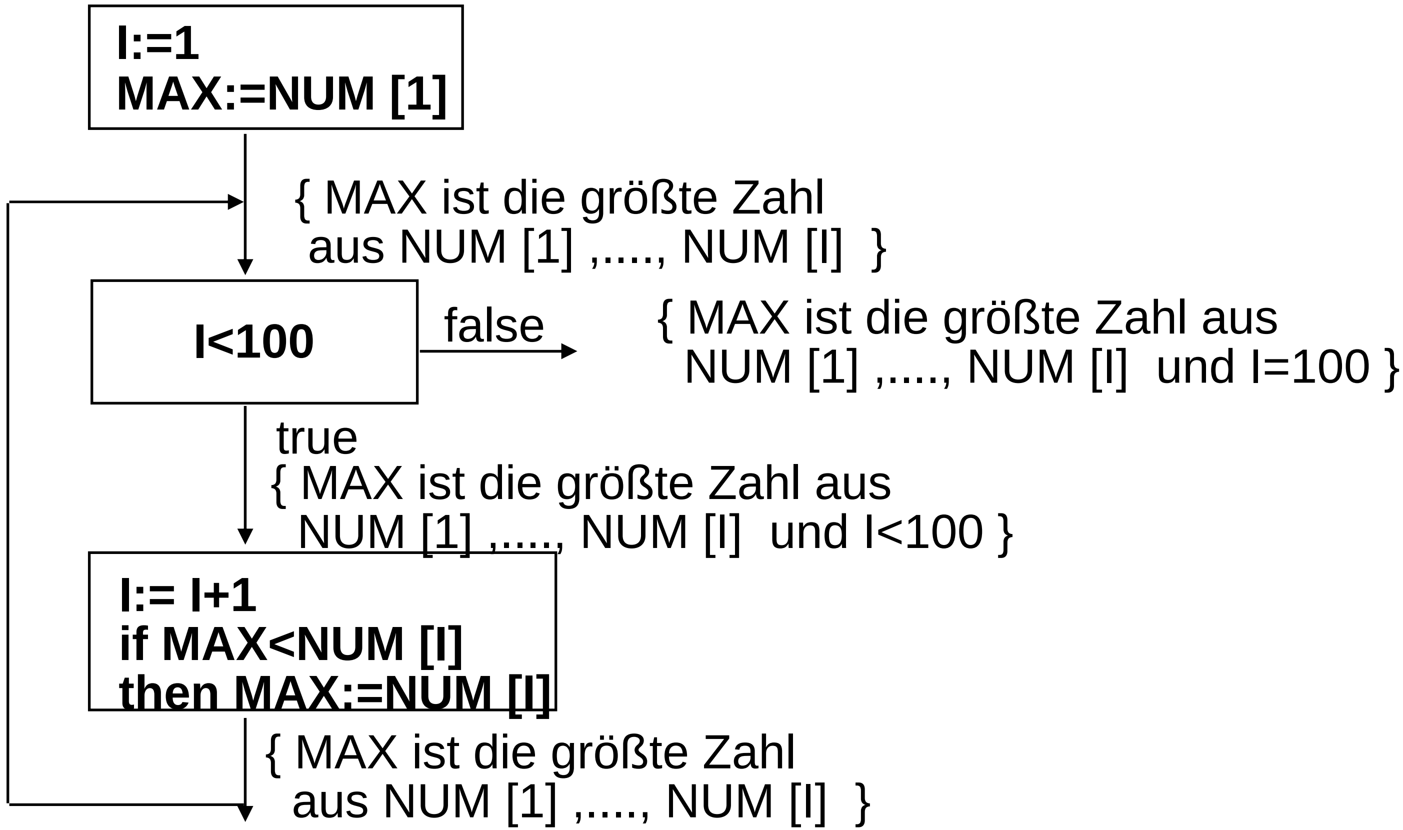
{ VA } while A do B { VA and false }

Häufig sind { VA } und { NB } identisch. Dann gilt für den Rumpf B die Spezifikation { VA and true } B { VA },
d.h. die Gültigkeit von { VA } ändert sich nicht, wenn B ausgeführt wird

Deshalb wird { VA } auch Schleifeninvariante genannt.

Zusammensetzung von Algorithmen

Beispiel: Suche nach dem Maximum von 100 Zahlen



Die Schleifeninvariante ist { MAX ist die größte Zahl von NUM [1] ,...., NUM [I] }

2.1.4 Analyse von Algorithmen

Die Ausführung eines Algorithmus verursacht Kosten bzgl. - Zeit

- Speicherplatz

Die benötigte Zeit hat dabei eine größere Bedeutung als der Speicherplatz.

Der Zeitaufwand bei der Ausführung von zusammengesetzten Algorithmen

A ; B

if A then B else C

while A do B

läßt sich aus dem Zeitaufwand t(A), t(B) und t(C) der Teilalgorithmen ermitteln :

t(A;B) = t(A) + t(B)

t(if A then B else C) = t(A) + max(t(A) , t(B))

t(while A do B) = n * (t(A) + t(B)) + t(A)

wobei n die Anzahl der Iterationen darstellt

Analyse von Algorithmen

Die Komplexität eines Algorithmus hängt stark von den Eingabedaten ab. Deshalb werden

- der günstigste Fall
- der schlechteste Fall
- der durchschnittliche Fall

betrachtet.

Beispiel: Sequentielle Suche

Algorithmus: SEQ_SEARCH

Eingabedaten: Liste L mit 1000 Namen und ein Name N

Ausgabedaten: Position von N in L, falls N in L enthalten ist; 0 sonst

Vorschrift:
```
POS:=0;
FOUND:=false;
while (POS<1000) and not FOUND do POS:=POS+1;
                                  if L POS =N then FOUND:=true;
if not FOUND then POS:=0
```

Analyse von Algorithmen

Um die Laufzeit des Algorithmus zu ermitteln, bestimmen wir die Anzahl der Vergleiche zwischen Namen (aus der Liste) und dem gesuchten Namen.

Günstigster Fall: N ist das erste Element in L

=> **1 Vergleich**

Schlechtester Fall: N muß mit allen Namen in L verglichen werden

=> **1000 Vergleiche**

Durchschnittlicher Fall: Insgesamt kann man 1001 mögliche Positionen von N unterscheiden : N ist an der 1. Position

N ist an der 2. Position

....

N ist an der 1000. Position

N ist nicht in L enthalten

Für jede Möglichkeit nehmen wir (willkürlich!) die Wahrscheinlichkeit 1/1001 an.

Analyse von Algorithmen

Man benötigt für den

1. Fall	1 Vergleich
2. Fall	2 Vergleiche
....	
1000. Fall	1000 Vergleiche
1001. Fall	1000 Vergleiche

Im Durchschnitt benötigt man

$$1*1/1001 + 2*1/1001 + ...+ 1000*1/1001 + 1000*1/1001$$

$$= 1/1001 * \left(\sum_{i=1}^{1000} i + 1000 \right)$$

$$= 1/1001 * (1001 * 500 + 1000)$$

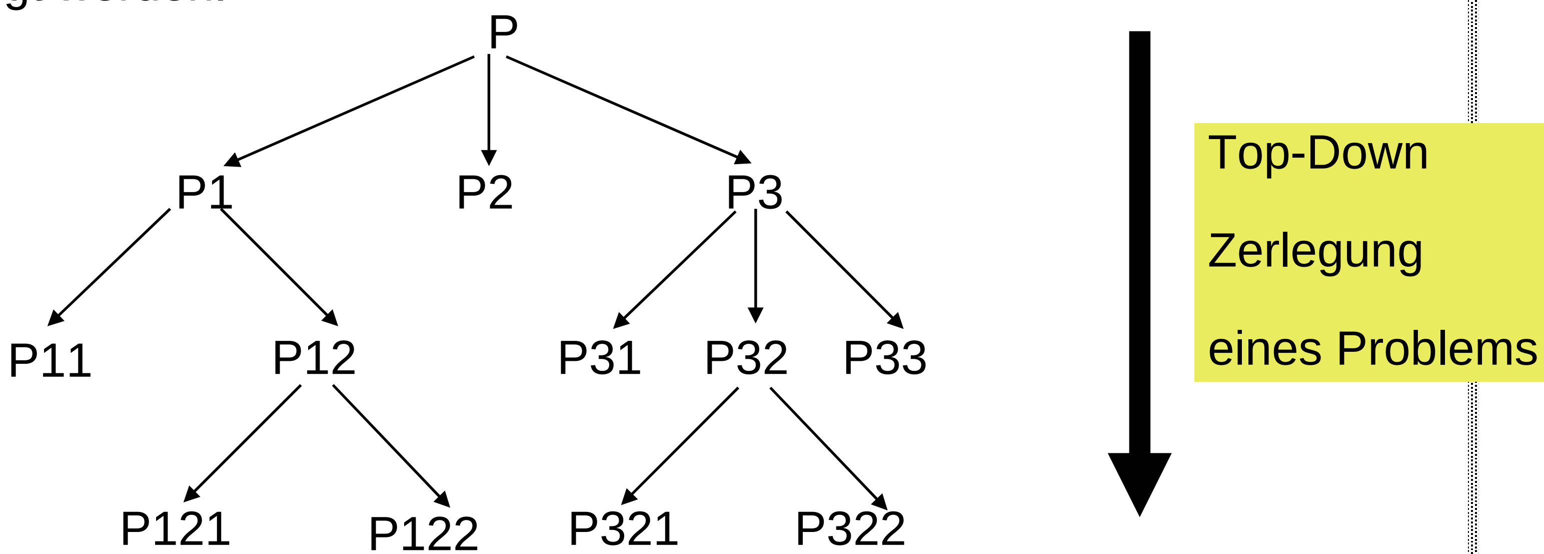
$$= 500 + 1000/1001$$

d.h etwa **501 Vergleiche**

Top-Down Zerlegung von Problemen

Wie findet man einen Algorithmus zur Lösung komplexer Probleme?

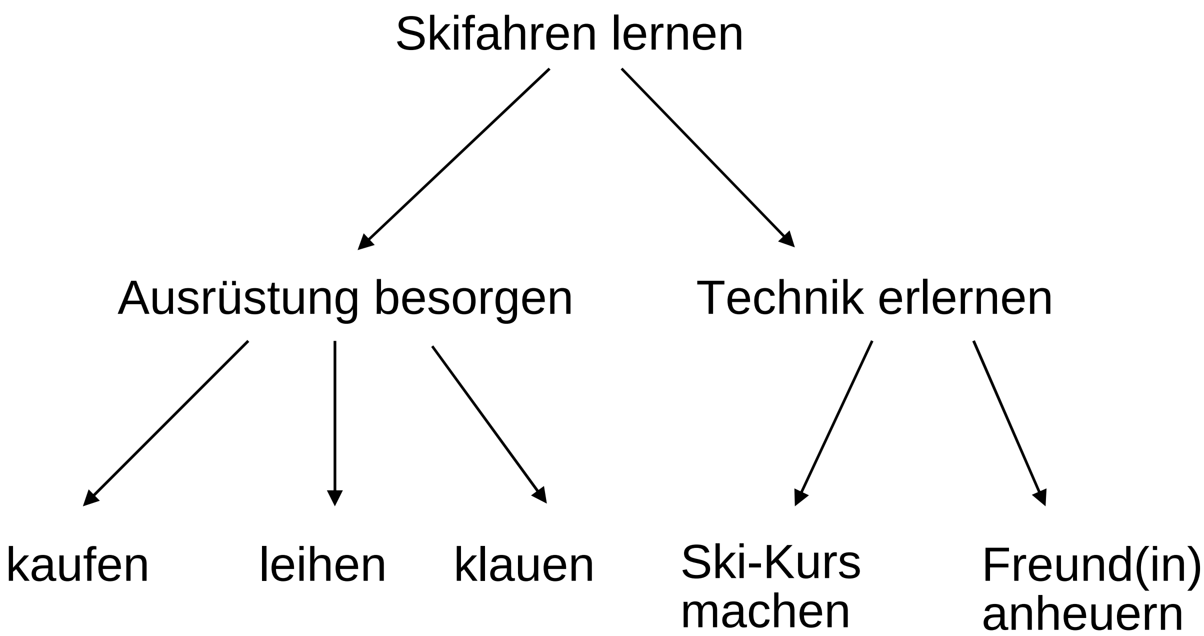
Ein komplexes Problem P läßt sich in einfachere Teilprobleme P1, P2, ... zerlegen. Jedes Teilproblem Pi kann nach Bedarf weiter in Teilprobleme Pi1, Pi2, ... zerlegt werden.



Die Zerlegung endet beim Erreichen von Problemen, deren Lösung unmittelbar erkennbar ist.

Top-Down Zerlegung von Problemen

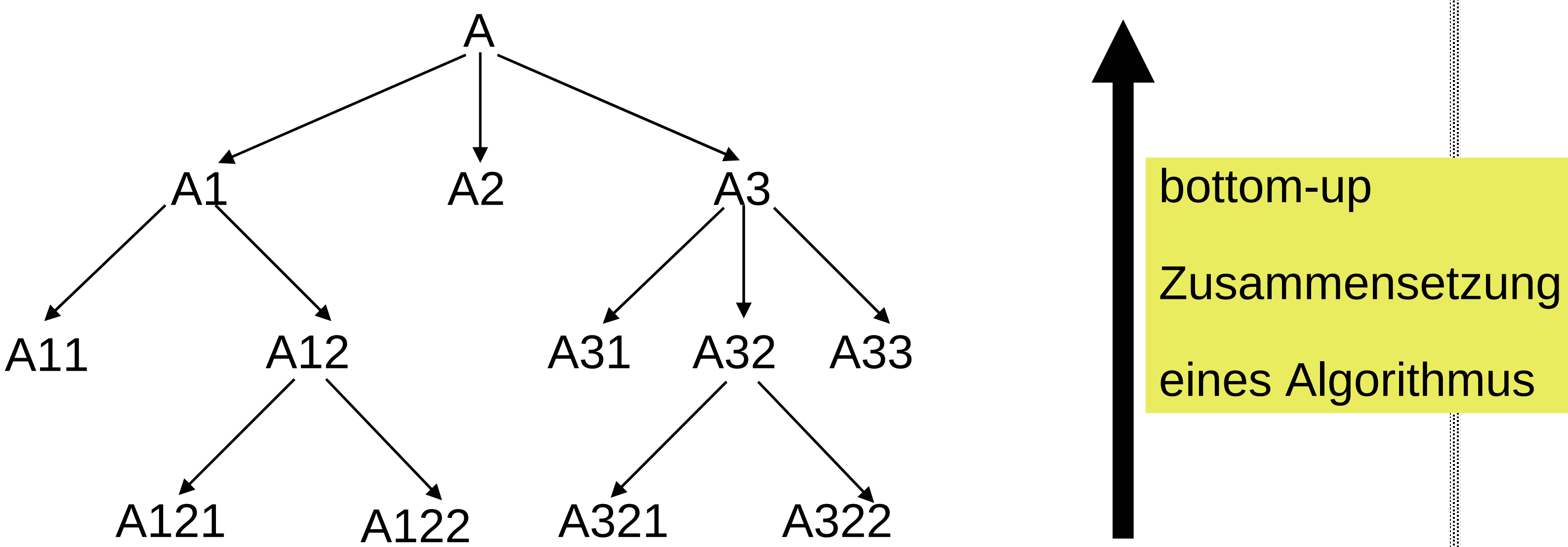
Beispiel:



Bottom-Up Zusammensetzung

Die Lösung eines top-down zerlegten Problems ergibt sichaus der Zusammensetzung von Algorithmen A1, A2, ... zur Lösung der Teilprobleme P1, P2, ...

Der Algorithmus Ai zur Lösung des Teilproblems Pi ergibt sich aus dem Zusammen-bau von Algorithmen Ai1, Ai2,....... zur Lösung der Teilprobleme Pi1, Pi2, ...



Verwende zum Zusammensetzen: - Komposition
- Selektion
- Iteration

Rekursion

Bei der top-down Zerlegung von Problemen taucht häufig die folgende Situation auf:

Problem P

Problem P ...... Problem P

d.h. **ein Problem P läßt sich in Teilprobleme zerlegen, unter denen das Problem P wieder vorkommt**

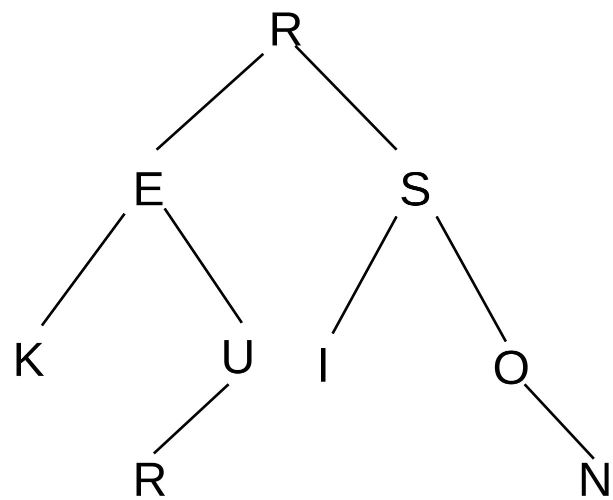
Beispiele: - Sortieren einer Namensliste

Sortiere Liste

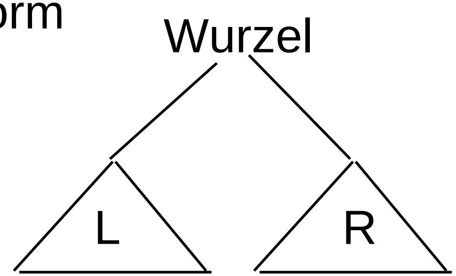
Sortiere linke Hälfte | Sortiere rechte Hälfte | Mische die beiden sortierten Hälften

Rekursion

- Durchlaufen eines binären Baumes



Jeder binäre Baum hat die Form



wobei L und R auch binäre Bäume sind

Rekursion

Durchlaufstrategie **VORORDNUNG**

- bearbeite die Wurzel
- wende **VORORDNUNG** auf den Teilbaum L an
- wende **VORORDNUNG** auf den Teilbaum R an

Durchlaufstrategie **ZWISCHENORDNUNG**

- wende **ZWISCHENORDNUNG** auf den Teilbaum L an
- bearbeite die Wurzel
- wende **ZWISCHENORDNUNG** auf den Teilbaum R an

Durchlaufstrategie **NACHORDNUNG**

- wende **NACHORDNUNG** auf den Teilbaum L an
- wende **NACHORDNUNG** auf den Teilbaum R an
- bearbeite die Wurzel

Anwendung dieser Strategien auf den Baum (vorige Seite) ergibt

VORORDNUNG: R E K U R S I O N

ZWISCHENORDNUNG: K E R U R I S O N

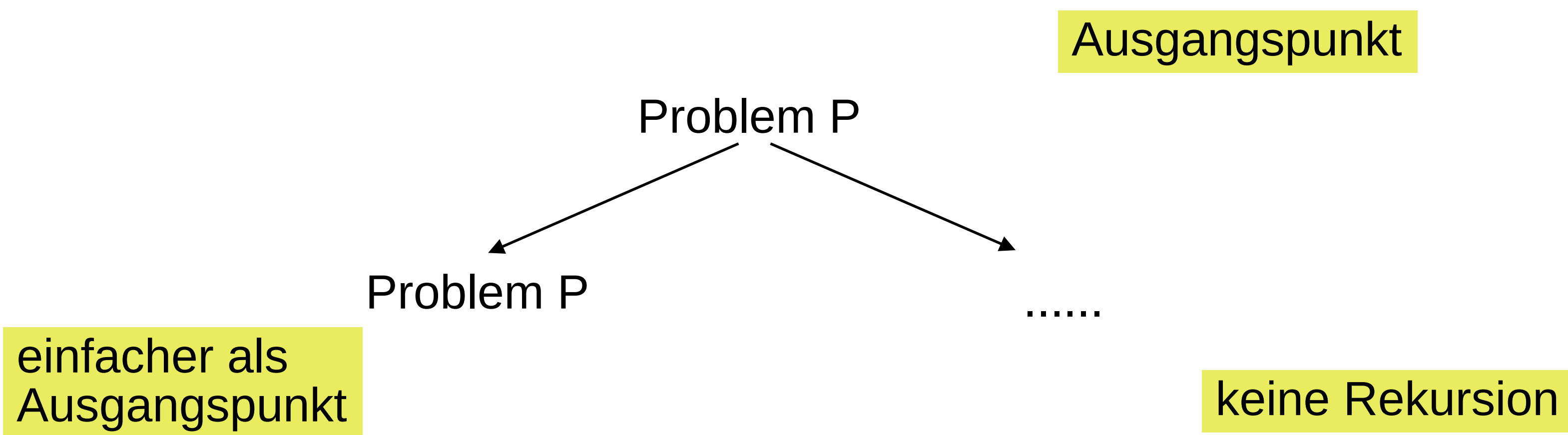
NACHORDNUNG: K R U E I N O S R

Rekursion

Termination ist ein wichtiger Aspekt

- in manchen Fällen kann das Problem direkt gelöst werden

- rekursive Zerlegung des Problems führt zu (echt) einfacheren Fassungen des Problems



Rekursion

Beispiele: - Sortieren durch Zerlegen und Mischen

Algorithmus: MERGESORT

Eingabedaten: Liste L von Namen

Ausgabedaten: Names von L in (aufsteigend) sortierter Reihenfolge

Vorschrift: **if** L enthält nur 1 Namen

then tue nichts

else zerlege L in 2 Hälften LEFT und RIGHT;

wende MERGESORT auf LEFT an;

wende MERGESORT auf RIGHT an;

mische LEFT und RIGHT zusammen

MERGESORT terminiert, weil:

- für Listen mit nur 1 Element erfolgt keine rekursive Anwendung von MERGESORT
- für die Hälften LEFT und RIGHT ist die Problemstellung einfacher

Rekursion

- Durchlaufen von binären Bäumen

Algorithmus: VORORDNUNG

Eingabedaten: Binärer Baum B mit Wurzel W, linkem Teilbaum L und rechtem Teilbaum R

Ausgabedaten: Folge der Knoten von Baum B

Vorschrift: KNOTENFOLGE := leer;
if B enthält keinen Knoten
then tue nichts
else verlängere KNOTENFOLGE um W;
wende VORORDNUNG auf L an;
wende VORORDNUNG auf R an;

Algorithmus VORORDNUNG terminiert, weil:

- für leere binäre Bäume kein rekursiver Aufruf von VORORDNUNG erfolgt
- die Teilbäume L und R echt kleiner als B sind

Analyse rekursiver Algorithmen

Die Kosten von rekursiven Algorithmen lassen sich mit Hilfe von Rekursionsgleichungen beschreiben.

Form: K(a) = b

K(N) = ...K(N/c)

wobei K(a) die Kosten der nicht-rekursiven Problemstellung der Größe a angibt

K(N) die Kosten der rekursiven Problemstellung der Größe N beschreibt

Beispiel: MERGESORT mit den folgenden Modifikationen

if L nur 2 Namen enthält

then vergleiche diese und vertausche, falls nötig

else ....

Analyse von MERGESORT:

K(N) ist die Anzahl der benötigten Vergleiche für eine Liste mit N Namen

Analyse rekursiver Algorithmen

$K(2) = 1$ 1 Vergleich für 2 Namen

$K(N) = K(N/2) + K(N/2) + N$

wobei $K(N/2)$ die Kosten des Sortierens einer Liste mit $N/2$ Namen darstellt

i.e. $K(N) = 2 * K(N/2) + N$

$K(N/2) = 2 * K(N/4) + N/2$

=> $K(N) = 4 * K(N/4) + N + N$

$K(N/4) = 2 * K(N/8) + N/4$

=> $K(N) = 8 * K(N/8) + N + N + N$

....

=> $K(N) = 2^{\log N - 1} * K(2) + N * (\log N - 1)$

$= N/2 + N * (\log N - 1)$

$= N * \log N - N/2$

=> die Anzahl der benötigten Vergleiche liegt im Bereich $O(N * \log N)$