Informatik II

Übungsstunde 10

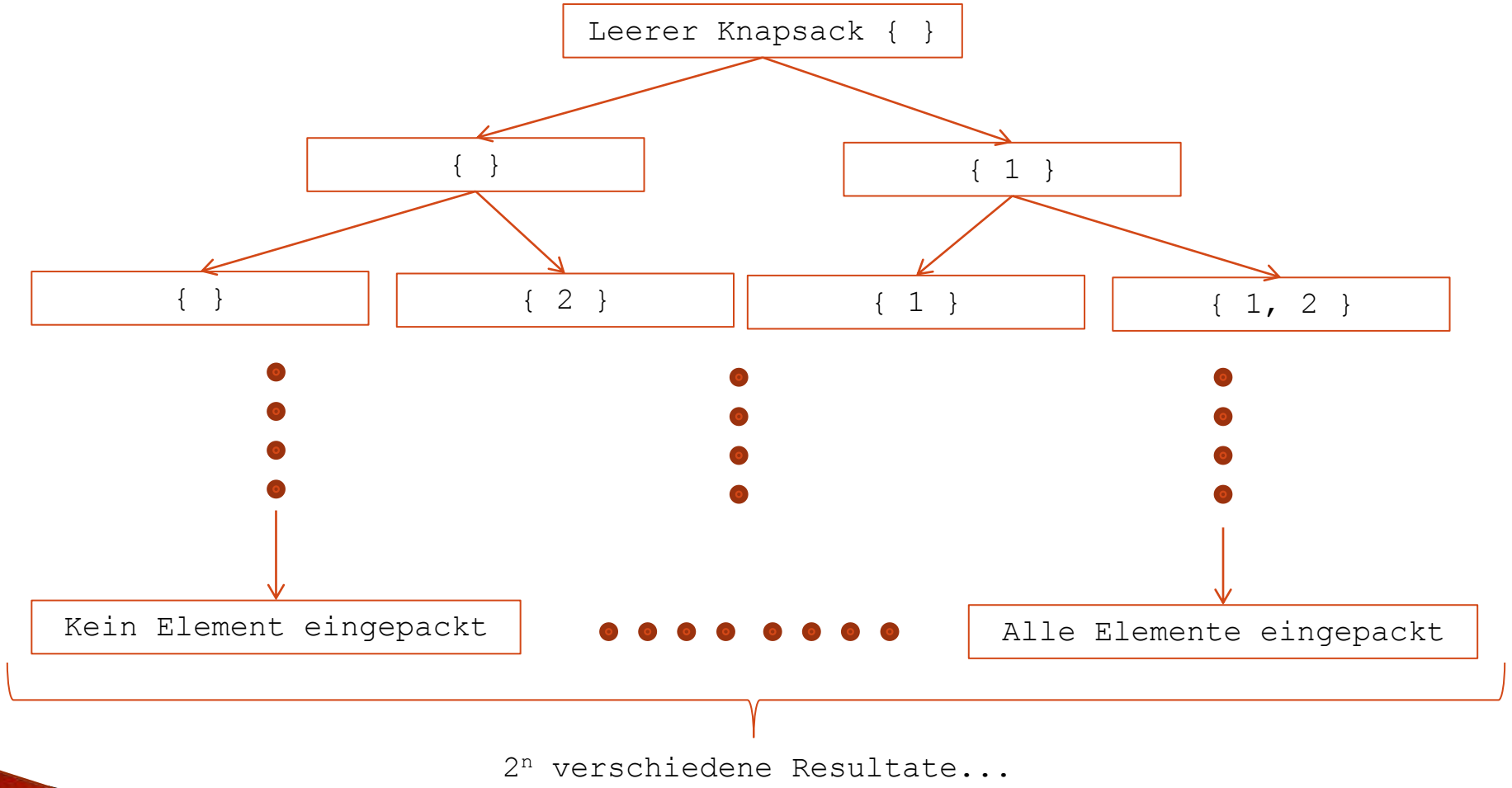
simon.mayer@inf.ethz.ch
Distributed Systems Group, ETH Zürich

Das hier ist von Gábor abgeschrieben!

Der Knapsack aus Serie 9

- Knapsack-Problem klar?
- Brute Force
  - «Idee»: Alles durchprobieren
  - Frage hierzu: Warum ist ein Problem ueberhaupt «schwer»?
- Backtracking
  - Nichtnaiver Ansatz: Konstruktion eines binaeren Entscheidungsbaumes ueber dem Knapsack

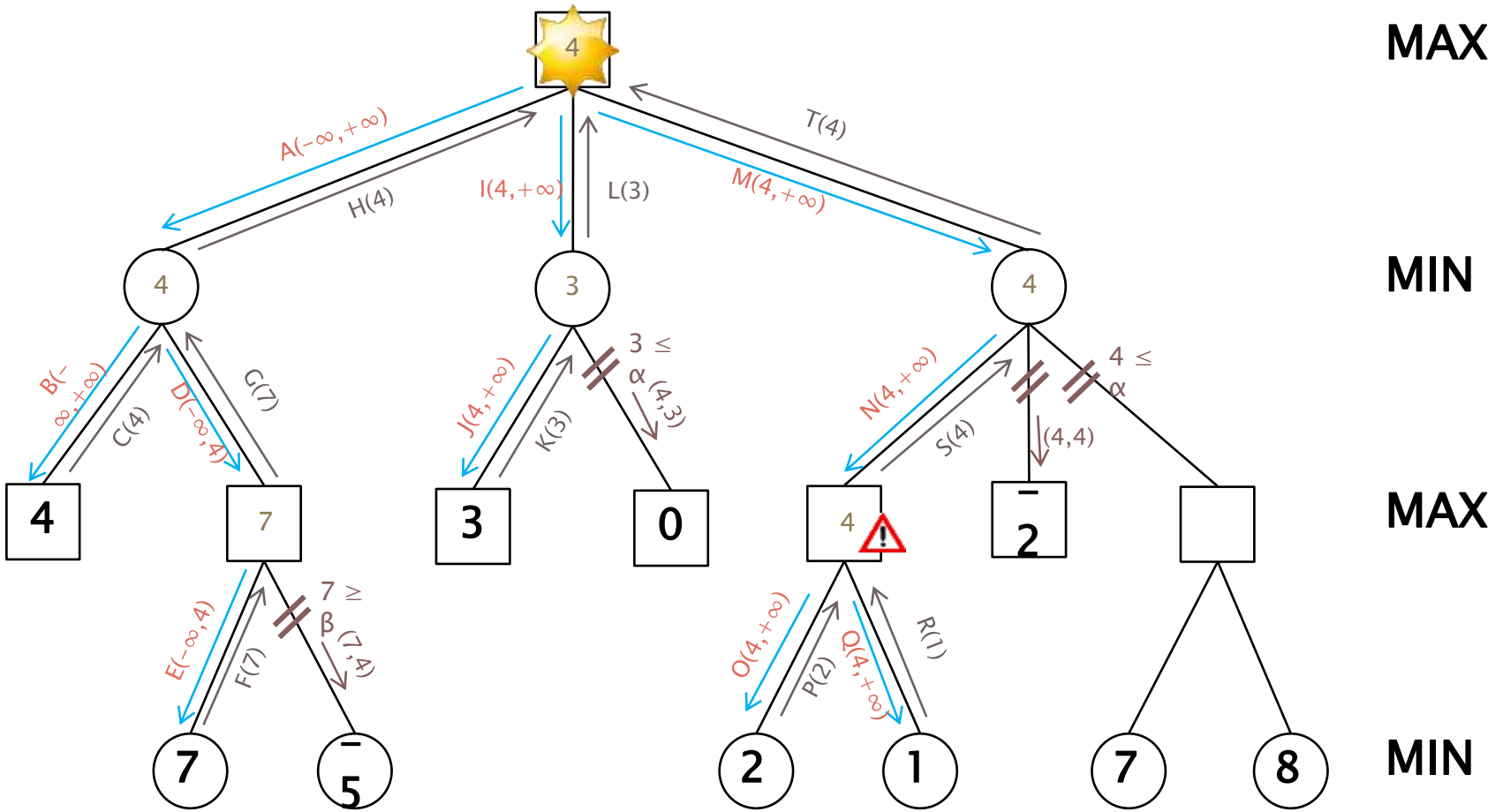
Der Knapsack aus Serie 9: Backtracking



Warum sparen wir aber trotzdem Zeit?

Das hier ist von Gábor abgeschrieben!

L9.A2d – Der Baum



Das hier ist von Gábor abgeschrieben!

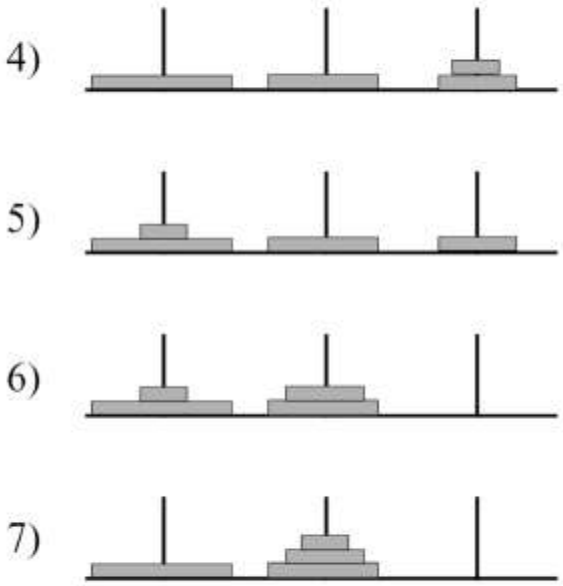
Reversi – MinMaxPlayer

- … ?

Ablauf

- Besprechung der Vorlesung
- Uebungsbezogene Themen:
  Sortieren, Rekursion, Reversi
- Zeit zum Programmieren...
  und fuer noch mehr Fragen

Die Programmausgabe von „hanoi(4,1,3)“



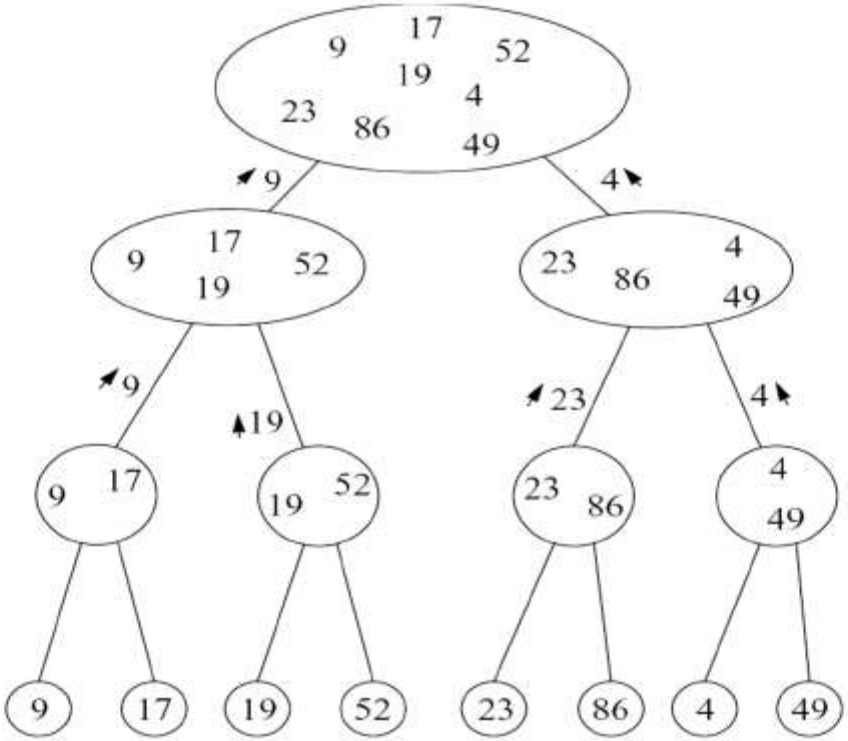

```
void hanoi(int groesse, int von, int nach)
{
  if (groesse == 1)
    bewege(von, nach)
  else {
    hanoi(groesse-1, von, 6-von-nach);
    bewege(von, nach);
    hanoi(groesse-1, 6-von-nach, nach);
  }
}
```

0)
1) Eine Scheibe von Turm 1 nach Turm 2
2) Eine Scheibe von Turm 1 nach Turm 3
3) Eine Scheibe von Turm 2 nach Turm 3
4) Eine Scheibe von Turm 1 nach Turm 2
5) Eine Scheibe von Turm 3 nach Turm 1
6) Eine Scheibe von Turm 3 nach Turm 2
7) Eine Scheibe von Turm 1 nach Turm 2

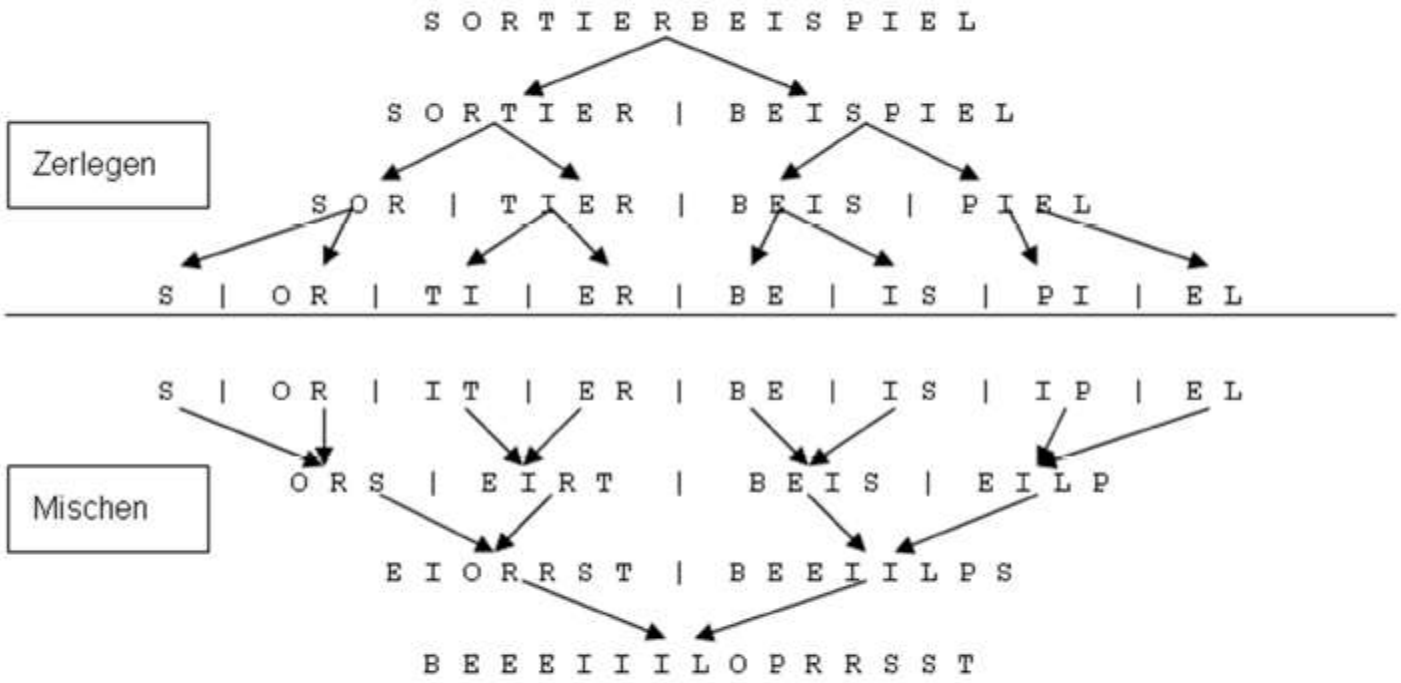
Dìvide et ìmpera – Ein interessantes Paradigma

- Grundprinzip zur Problemlösung in der Informatik
- Ursprung der Redewendung aus der Politik- und Sozialwissenschaften
  - "Feinde" (anhand von Religion, Ethnie etc.) in Untergruppen aufspalten
  - Einzelne Untergruppen "besiegen" oder "unter Kontrolle halten"
  - Ev. Untergruppen gegeneinander anstiften
- Von den alten Römer bis zu Kolonialzeit und darüber hinaus angewendet
  - Oft mit langwierige Konsequenzen

Eine andere Darstellung des Rekursionsbaums



Mergesort – noch ein Beispiel



Bottom-up-Mergesort

- Nicht rekursiv!

1) Durchlaufe die Liste der Elemente und erzeuge geordnete Paare durch Mergen benachbarter sortierter Teillisten der Länge 1
   - Es werden sortierte Teillisten der Länge 2 erzeugt

2) Durchlaufe die Liste der Elemente und erzeuge geordnete Quadrupel durch Mergen benachbarter sortierter Teillisten der Länge 2
   - Es werden sortierte Teillisten der Länge 4 erzeugt

- → Bis eine sortierte Liste der Länge n erzeugt wird

Bottom-up-Mergesort (2)

- Nach k Durchläufen hat man sortierte Teillisten der Länge 2^k
- Nach log n Durchläufen hat man die Länge n (→ fertig!)
- Ein Durchlauf erfordert jeweils n „Schritte"
  - Pro Schritt: Vergleichen zweier Werte, Erhöhen eines Index...
- Mergesort benötigt also (grössenordnungsmässig) n log n „Schritte"
- Top-down- und Bottom-up-Mergesort führen i.w. die gleichen Merge-Operationen aus, allerdings in unterschiedlicher Reihenfolge!

- Beim rekursiven Top-down-Mergesort gibt es folgenden Zwischenzustand: | sortiert | unsortiert |
  - Dieser Zustand kann beim Bottom-up-Mergesort nicht auftreten, dort sind alle Teile stets etwa „gleich weit"

Multiply and Surrender?

- Andere Problemloesungsstrategie (...aber rein didaktischer Natur ☺)

> *Consider the following problem: we are given a table of n integer keys* `A1,A2, ...,An` *and a query integer* `X`*. We want to locate X in the table, but we are in no particular hurry to succeed; in fact, we would like to delay success as much as possible.*
>
> *(Broder and Stolfi, 1986)*

http://www.cs.uiuc.edu/class/fa05/cs473ug/resources/pessimal-algorithms.pdf

Multiply and Surrender?

- Slowsort
  - Wähle das letzte Element der (rekursiv) sortierten ersten Hälfte. Dies ist das Maximum der ersten Haelfte.
  - Wähle das letzte Element der (rekursiv) sortierten zweiten Hälfte. Dies ist das Maximum der zweiten Haelfte.
  - Bestimme das Maximum der beiden Teilmaxima und platziere es am Ende des Feldes

-----> Komplexitaet von Algorithmen

Multiply and Surrender?

- Komplexitaet von Mergesort?
  - Ist also ein optimaler Algorithmus fuer das Sortierproblem...
- Komplexitaet von Slowsort?
  - Rechnen wir das mal durch...

Begriffe bei der Aufwandsanalyse

- Problemumfang bzw. -grösse wird meist mit n bezeichnet
  - Oft: Anzahl der Eingabewerte
  - Manchmal aber auch: Anzahl der Bits der Eingabe
- Der Aufwand (Bedarf an Zeit bzw. Speicherplatz in sinnvollen Einheiten) wird dann als Funktion f(n) angegeben
- Beim Zeitaufwand wird i.Allg. von konstanten Faktoren (und additiven Termen) abstrahiert
  - Also z.B.: $f(n) = n \log(n)$ statt genauer $3 + 7\, n \log(n)$
  - Beim Speicheraufwand (bzgl. der Implementierung eines Algorithmus) abstrahiert man aber meist nicht von konstanten Grössen

Begriffe bei der Aufwandsanalyse (2)

- Oft ist der Aufwand eines Algorithmus nicht nur von der Problemgrösse n, sondern von den konkreten Eingabewerten (bzw. deren Reihenfolge) abhängig.
  Dann unterscheidet man:
  - günstigster Aufwand („best case")
  - mittlerer Aufwand („average case")
  - ungünstigster Aufwand („worst case")

- Oft ist man nur am (i.Allg. leichter zu bestimmenden) asymptotischen Aufwand (für $n \to \infty$) als Funktion von n interessiert
  - Achtung: für „kleine" n kann ein Algorithmus mit schlechterem asymptotischen Aufwand besser sein (→ break even points)!

- Komplexität eines *Problems* = geringstmöglicher Aufwand, der mit dem dafür besten Lösungsalgorithmus erreicht werden kann

Die O-Notation

- Man sagt, die Komplexität eines Algorithmus ist von der Grössenordnung O(f(n)), wenn für die „wirkliche" Komplexität g(n) des Algorithmus gilt:

$$\exists n_0, c>0 : \forall n \geq n_0 : g(n) \leq c\, f(n)$$

- Beispiel: Für $7 + 1.5\, n + 3n^2$ schreibe $O(n^2)$

  $O(n^3)$ oder $O(2^n)$ wäre auch richtig, aber „übertrieben"

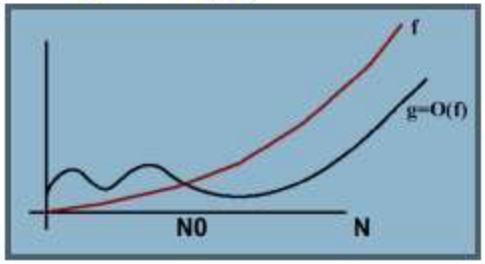
- Informelle Interpretation: Die Laufzeit wird „im wesentlichen" durch O(...) beschränkt (oder oft auch unpräziser: „ist etwa proportional zu")

  Interessant ist i.Allg. die „kleinste" Funktion f, die g majorisiert

- Oft kann man die Grössenordnung in O-Notation angeben, ohne die wirkliche Komplexität zu kennen!

Komplexitätsklassen O(f)

- Mit O(f) wird jeweils eine ganze Funktionsklasse bezeichnet
  - Genauer: $O(f(n)) = \{ g(n) \mid \exists n_0, c > 0: \forall n \geq n_0 : g(n) \leq c\, f(n) \}$
- Statt $g \in O(f)$ schreibt man oft salopper: $g = O(f)$
  - Interpretation: g wächst höchstens so schnell wie f



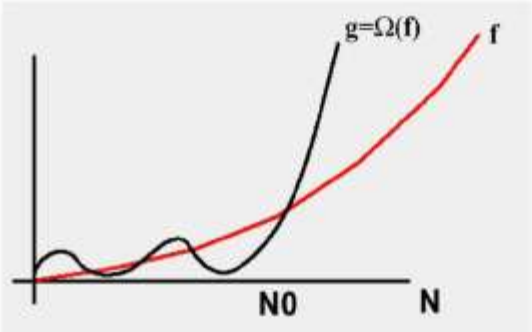
- $g = O(1)$ heisst: g ist beschränkt
  - Überschreitet bestimmten Wert nicht
- Wichtig ist auch die Klasse der polynomiellen Funktionen:

  $POLY(n) = \bigcup_{p>0} O(n^p)$
  - Falls $f \in POLY(n)$, dann spricht man von polynomiellem Aufwand (wesentlicher Unterschied dazu: *exponentieller* Aufwand $O(a^n)$)

Die Ω-Notation

- Für untere Schranken verwendet man die Ω-Notation:
  - $\Omega(f(n)) = \{ g(n) \mid \exists n_0, c > 0: \forall n \geq n_0 : c\, f(n) \leq g(n) \}$

- Interpretation von $g = \Omega(f)$:
  - g wächst mindestens so schnell wie f



Inhalt der Vorlesung

1. Ein Algorithmus und seine Implementierung in Java
2. Java: Elementare Aspekte
3. Klassen und Referenzen
4. Syntaxanalyse und Compiler
5. Pakete in Java
6. Objektorientierung
7. Exceptions
8. Binärbäume als Zeigergeflechte
9. Binärsuche
10. Backtracking
11. Spielbäume
12. Rekursives Problemlösen
13. Komplexität von Algorithmen
14. Simulation
15. Heaps
16. Parallele Prozesse und Threads

Ablauf

- Besprechung der Vorlesung
- Uebungsbezogene Themen:
  Sortieren, Rekursion, Reversi
- Zeit zum Programmieren...
  und fuer noch mehr Fragen

Übung 10

1. Mergesort
   a) Manuell
   b) Implementieren
   c) Zeitmessung
   d) Graph: Ueberlagern mit theoretischer Komplexitaet

Übung 10

2. Rekursion/Hanoi

a) Bisschen Theorie

b) Einfacher Algorithmus (nicht-rekursiv) fuer Hoehe 4?

c) Auch moeglich fuer Hoehe 5?

Übung 10

3. Reversi

- Umbauen des MinMax-Spielers zu einem alpha/beta-Spieler

Ablauf

- Besprechung der Vorlesung
- Uebungsbezogene Themen:
  Sortieren, Rekursion, Reversi
- Zeit zum Programmieren...
  und fuer noch mehr Fragen

Zeit fuer Fragen? Vielleicht fuer Reversi-Tipps?

- Bewertungsfunktion
- Cutoff
- Zeitmessung
- Heuristik

Informatik II

Übungsstunde 10

simon.mayer@inf.ethz.ch
Distributed Systems Group, ETH Zürich