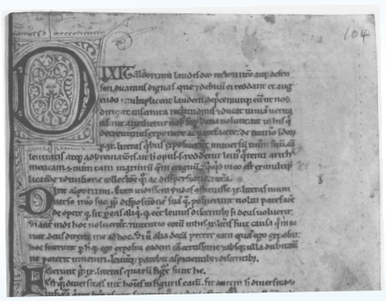




Schnelldurchlauf!

Algorithmen und Datenstrukturen
WS 2022/23

Prof. Dr. Sándor Fekete

Algorithmen

Datenstrukturen

Ordentlich,
hält Regeln ein

Locker drauf,
zielorientiert

Algorithmen

Datenstrukturen

I get the job done. What the hell do you want?

I don‘t make things difficult. That‘s the way they get, all by themselves.

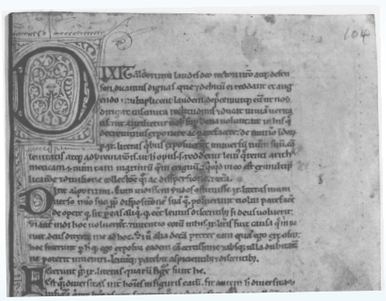
Then don‘t.

Can you make it without killing yourself?

I don‘t want to work with you.

Ain‘t got no choice.

I‘m getting too old for this…


1 Einführung: Algorithmen

Algorithmen und Datenstrukturen
WS 2022/23

Prof. Dr. Sándor Fekete

1.1 Was ist ein Algorithmus?

WIKIPEDIA
Die freie Enzyklopädie

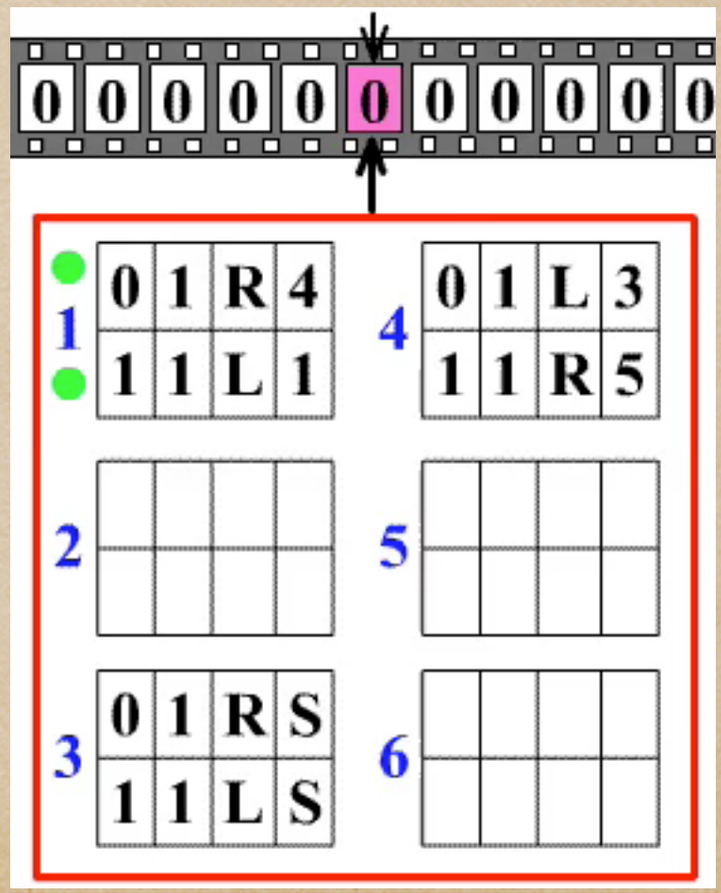
"Ein **Algorithmus** ist eine aus endlich vielen Schritten bestehende eindeutige Handlungsvorschrift zur Lösung eines Problems oder einer Klasse von Problemen."

Beispiele:
- Kochrezept
- Bedienungsanleitung
- Notenblatt
- Programmablaufplan

Kein Computer ohne Algorithmen!
Keine Informatik ohne Algorithmen!
Kein Informatikstudium ohne Algorithmen!

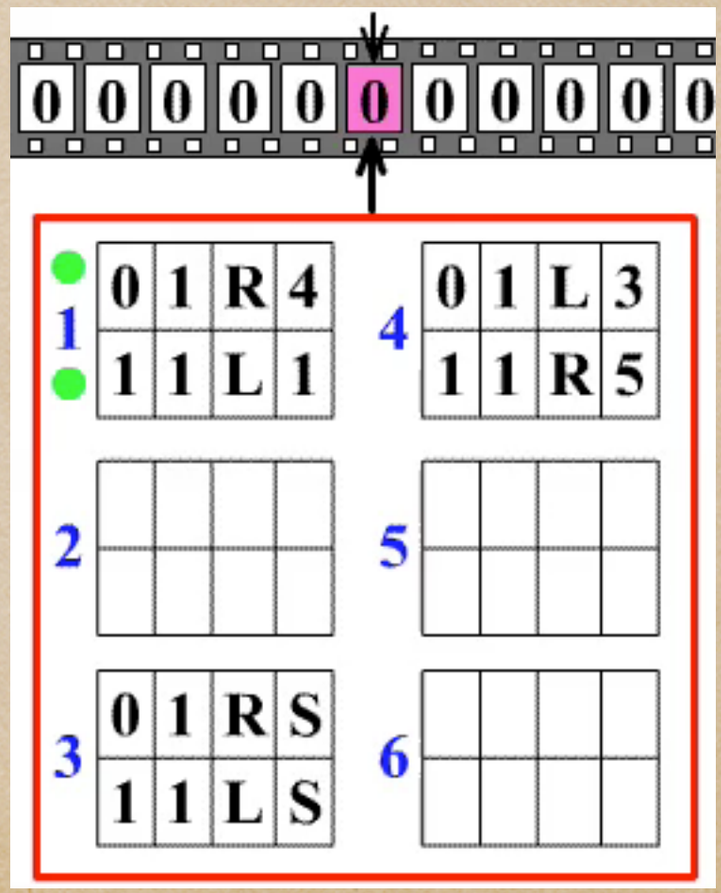
1.2 Wie formalisiert man einen Algorithmus?

Idee: Abstrakte Formalisierung beliebiger Berechnungen!



1.2 Wie formalisiert man einen Algorithmus?

Idee: Abstrakte Formalisierung beliebiger Berechnungen!



1.3 Eigenschaften von Algorithmen

1. Das Verfahren muss in einem endlichen Text eindeutig beschreibbar sein (Finitheit).
2. Jeder Schritt des Verfahrens muss tatsächlich ausführbar sein (Ausführbarkeit).
3. Das Verfahren darf zu jedem Zeitpunkt nur endlich viel Speicherplatz benötigen (Dynamische Finitheit, siehe Platzkomplexität).
4. Das Verfahren darf nur endlich viele Schritte benötigen (Terminierung, siehe auch Zeitkomplexität).

1.4 Datenstrukturen

Eine Datenstruktur erlaubt es, die für eine Aufgabe notwendigen Informationen

- geeignet zu repräsentieren
- den Zugriff und die Verwaltung während der Bearbeitung in effizienter Weise zu ermöglichen.

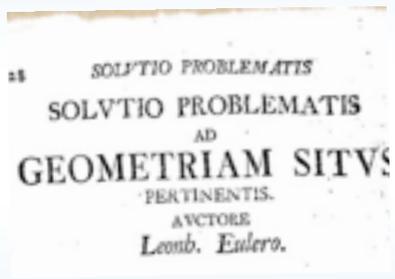
Mehr Einzelheiten werden uns ab dem dritten Kapitel begegnen!

XLII + XCVIII = CXL

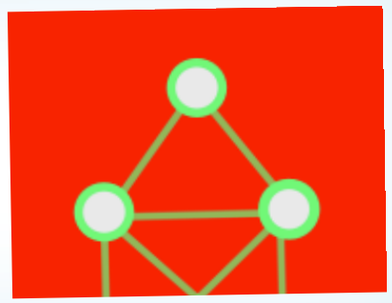
42 + 98 = 140!

1.5 Ausblick

Wir werden uns in dieser Vorlesung mit verschiedenen Aspekten von Algorithmen beschäftigen. Dazu gehören oft auch Analyse und Verständnis der zugrundeliegenden mathematischen Strukture. Gerade letzteres macht oft den eigentlichen Witz aus!


SOLVTIO PROBLEMATIS

SOLVTIO PROBLEMATIS
AD
GEOMETRIAM SITVS
PERTINENTIS.
AVCTORE
Leonh. Eulero.

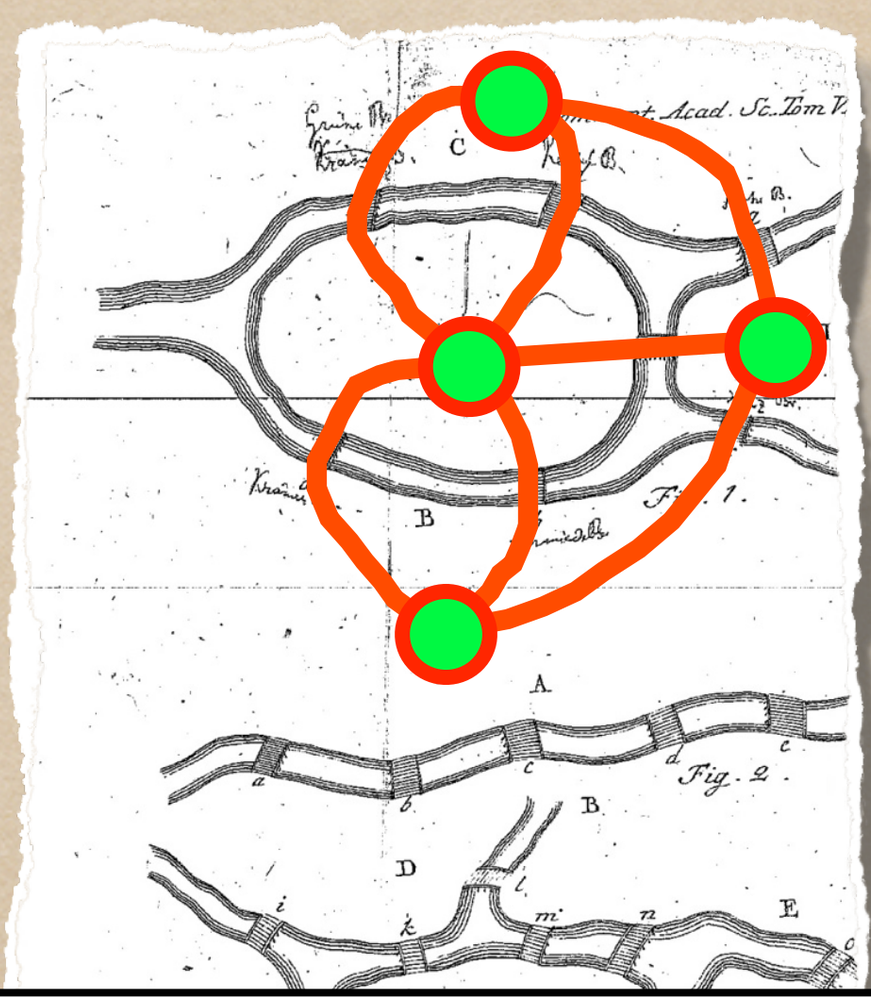


Kapitel 2: Graphen

Algorithmen und Datenstrukturen
WS 2022/23

Prof. Dr. Sándor Fekete

2.1 Historie

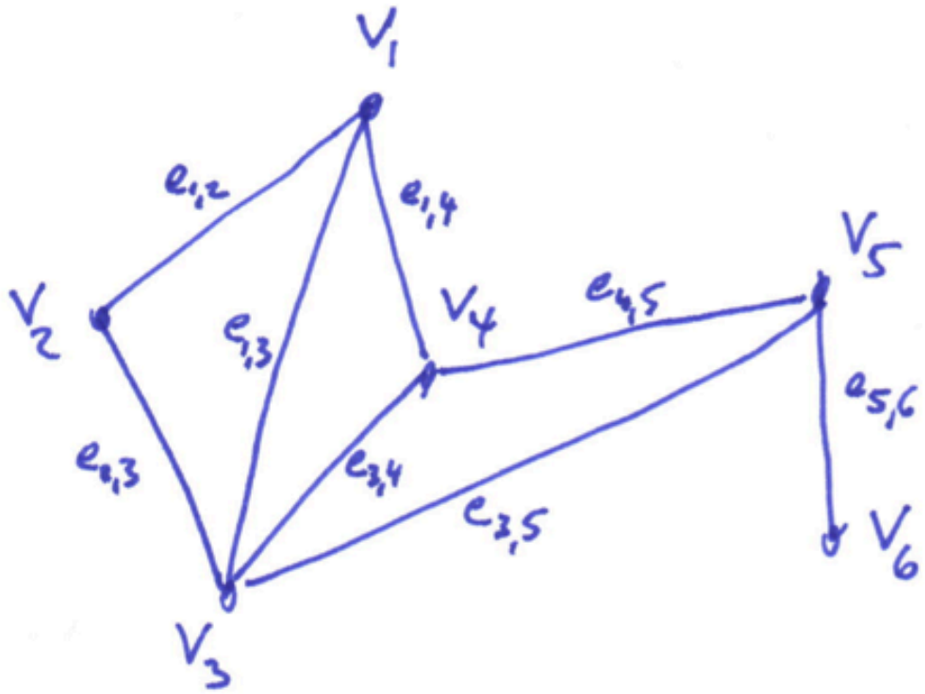


- Alle Knoten sind ungerade?!
- Man müsste an allen anfangen oder aufhören!
- Das geht nicht an einem Stück!

Euler: (1) Das gilt für jede beliebige Instanz: Mit mehr als zwei ungeraden Knoten gibt es keinen solchen Weg.

(2) Man kann auch charakterisieren, unter welchen Bedingungen es einen Weg tatsächlich gibt.

2.2 Formale Graphenbegriffe



Knoten: $v_1, v_2, v_3, v_4, v_5, v_6$

Kanten: $e_{1,2}, e_{1,3}, e_{1,4}$
$e_{2,3}, e_{3,4}, e_{3,5}$
$e_{4,5}, e_{5,6}$

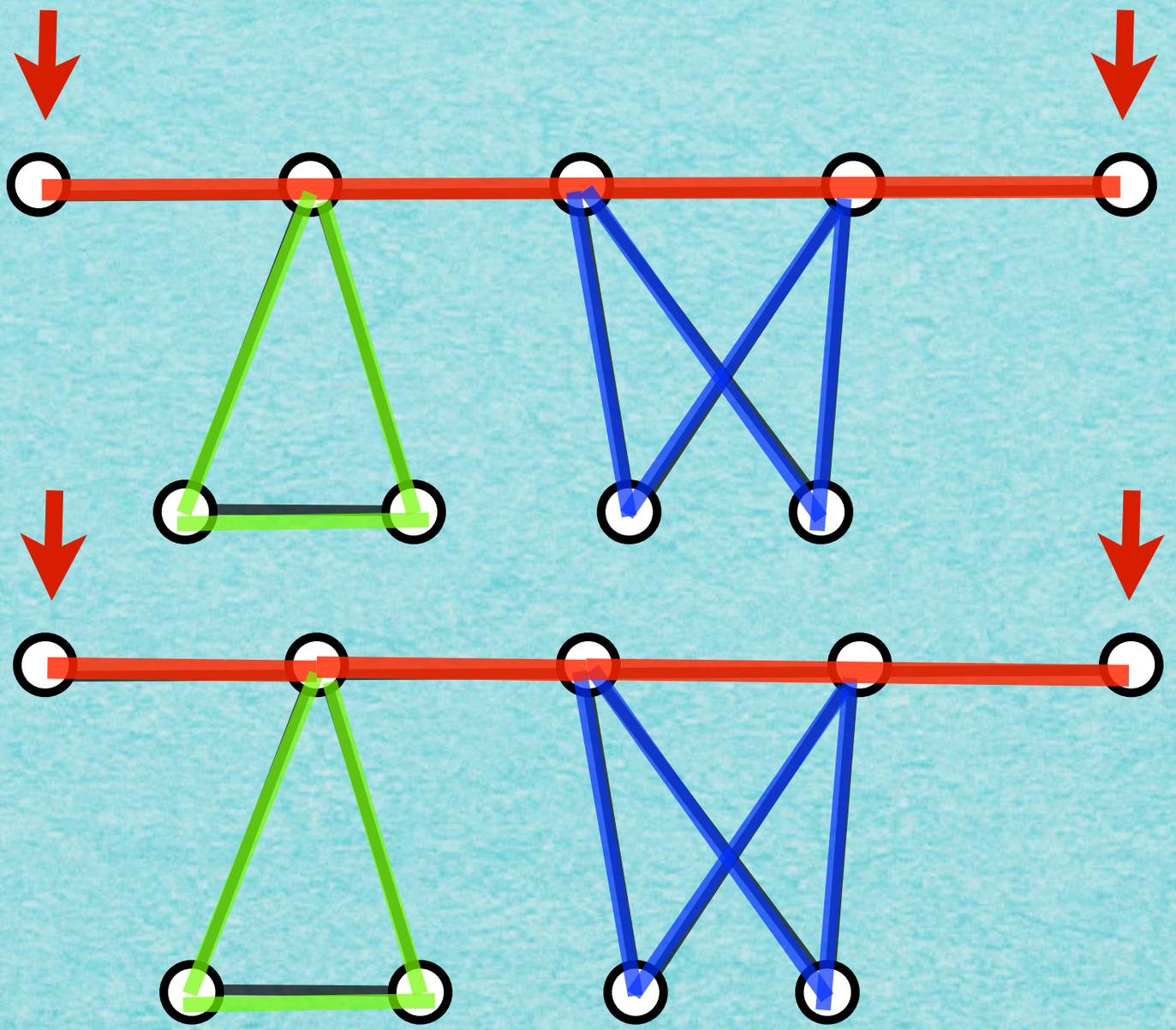
Bezeichnungen:

Knotenmenge V ("vertices")

Kantenmenge E ("edges")

Schreibweise: $G = (V, E)$

Wegebau



Algorithmus 2.8

Algorithmus von Hierholzer

INPUT:	Ein zusammenhängender Graph G mit höchstens zwei ungeraden Knoten
OUTPUT:	Ein Eulerweg bzw. eine Eulertour in G

A. Wähle einen Startknoten v (ungerade falls vorhanden);

B. Verwende Algorithmus 2.7, um einen Weg W von v aus zu bestimmen;

C. Solange es noch unbenutzte Kanten gibt:

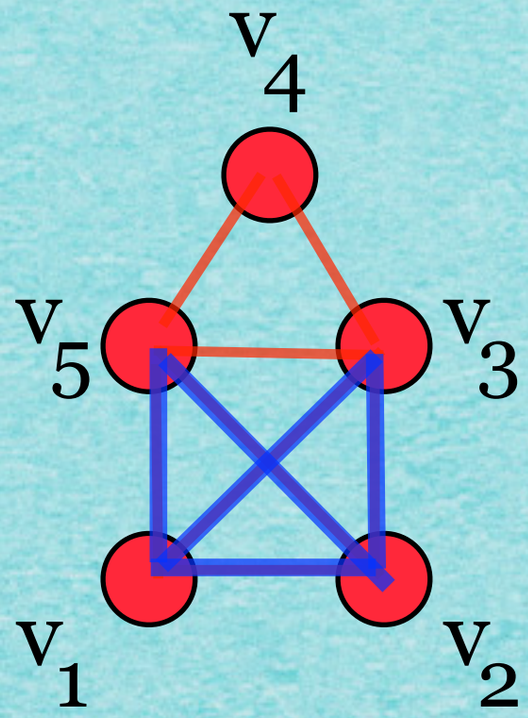
    C.1. Wähle einen von W besuchten Knoten w mit positivem Grad im Restgraphen;

    C.2.Verwende Algorithmus 2.7, um einen Weg W' von w aus zu bestimmen;

    C.3.Verschmelze W und W'

D. STOP

Es geht auch anders!



Wir hinterlassen Kanten?!

Algorithmus 2.13

Algorithmus von Fleury

INPUT: Graph G mit höchstens zwei ungeraden Knoten

OUTPUT: Ein Weg in G.

1. Starte in einem Knoten v_0 (ungerade, sonst beliebig);

2. Solange es eine zum gegenwärtigen Knoten v_i inzidente unbenutzte Kante $\{v_i, v_j\}$ gibt:

   2.1. Wähle eine dieser Kanten aus, $e_i = \{v_i, v_j\}$ der den Restgraphen zshgd. lässt

   2.2. Laufe zum Nachbarknoten v_j

   2.3. Lösche die Kante aus der Liste der unbenutzten Kanten.

   2.4. Setze $v_{i+1} := v_j$

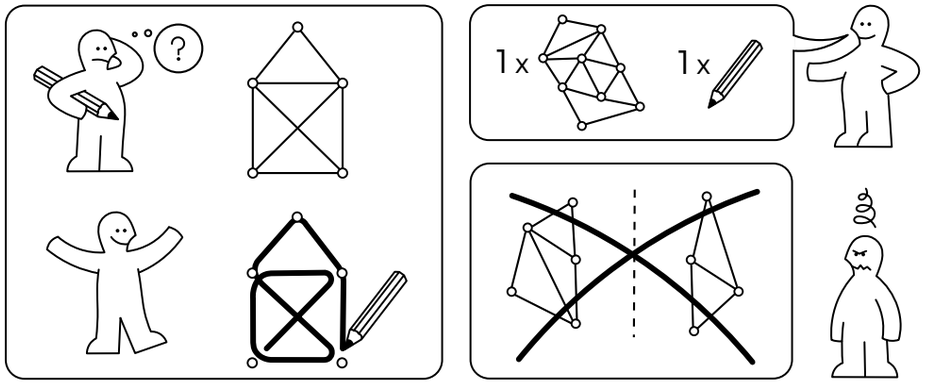
   2.5. Setze i := i+1

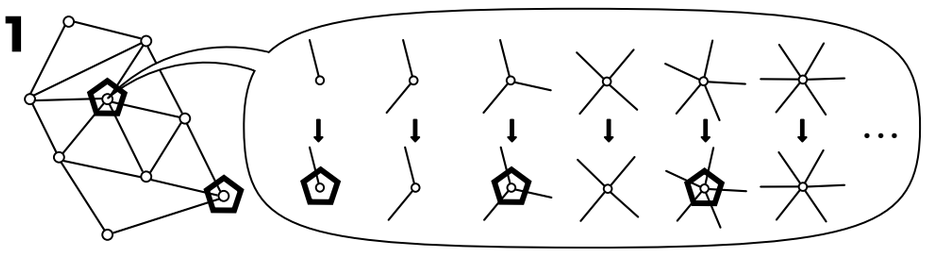
3. STOP

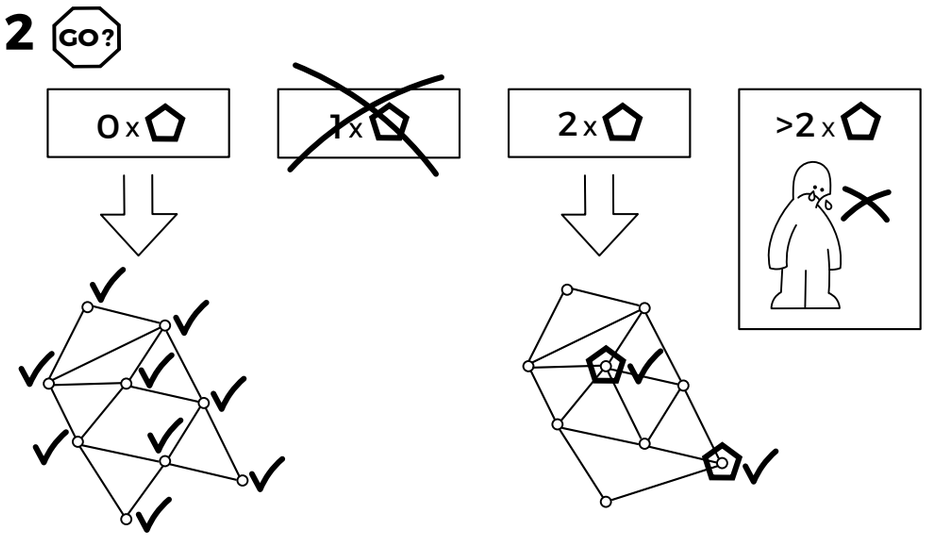
ONE STRÖKE DRÅW

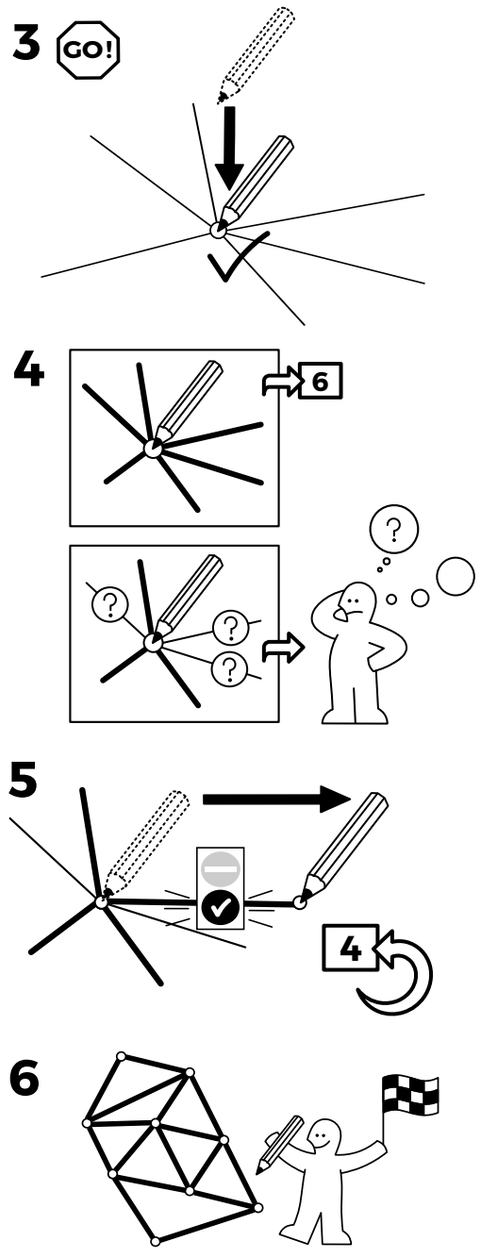
idea-instructions.com/euler-path/
v1.0, CC by-nc-sa 4.0

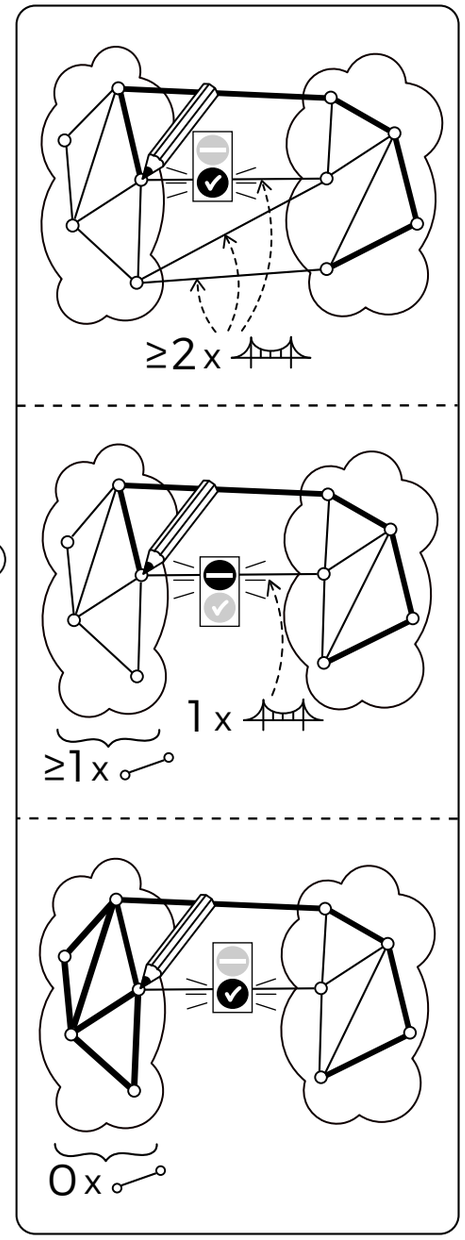
IDEA



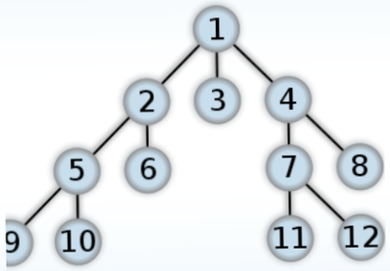








Zusammenfassung Kapitel 2!



Kapitel 3: Suche in Graphen

Algorithmen und Datenstrukturen
WS 2022/23

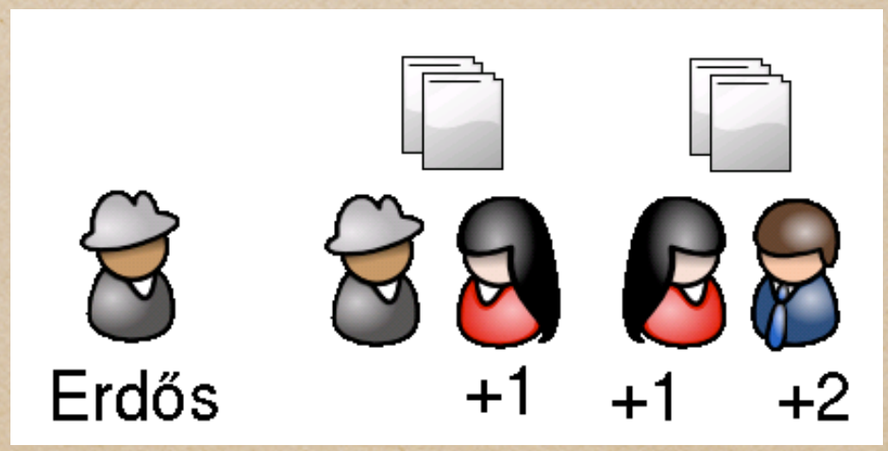
Prof. Dr. Sándor Fekete

3.1 Vorspann

Paul Erdös, 1913-1996

- *Produktivster Mathematiker aller Zeiten (~1500 Artikel)*
- *"Zweitbedeutendster Mathematiker nach Euler"*

Erdös-Zahl



3.2 Problemdefinitionen:

PROBLEM 3.1 (s-t-Weg)

Gegeben: Graph $G=(V,E)$, Startknoten s, Zielknoten t
Gesucht: Weg von s nach t, falls einer existiert

Allgemeiner:

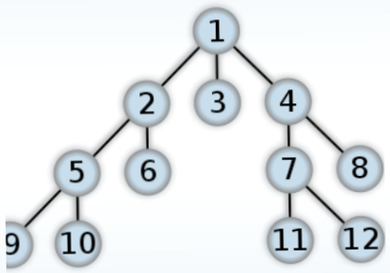
PROBLEM 3.2 (Zusammenhangskomponente)

Gegeben: Graph $G=(V,E)$, Startknoten s
Gesucht: Menge aller von s erreichbaren Knoten (→ „Zusammenhangskomponente von s")
Wege, die die Erreichbarkeit sichern

Beobachtung:

SATZ 3.3

Wenn ein Weg zwischen zwei Knoten s und t in einem Graphen existiert, dann existiert auch ein Pfad.

Kapitel 3.3: Zusammenhangskomponenten

Algorithmen und Datenstrukturen
WS 2022/23

Prof. Dr. Sándor Fekete

Algorithmus 3.7

INPUT: Graph G = (V,E), Knoten s

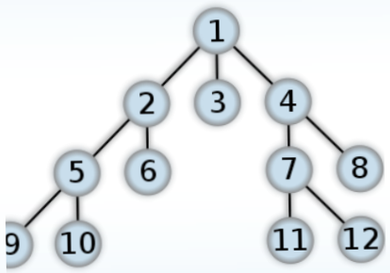
OUTPUT: Knotenmenge $Y \subseteq V$, die von s aus erreichbar ist,

Kantenmenge $T \subseteq E$, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. wähle Element $v \in R$
   - 2.2. IF (es gibt kein $w \in V \setminus Y$ mit $e=\{v,w\} \in E$) THEN
     - 2.2.1. $R := R \setminus \{v\}$
   - 2.3. ELSE {
     - 2.3.1. wähle ein $w \in V \setminus R$ mit $e=\{v,w\} \in E$;
     - 2.3.2. setze $R := R \cup \{w\}$, $Y := Y \cup \{w\}$, $T := T \cup \{e\}$;

     }

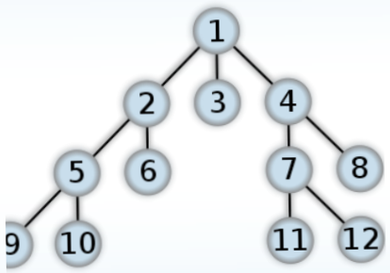
}



Kapitel 3.4: Wartenschlange und Stapel

Algorithmen und Datenstrukturen
WS 2022/23

Prof. Dr. Sándor Fekete



Kapitel 3.5: Tiefensuche und Breitensuche

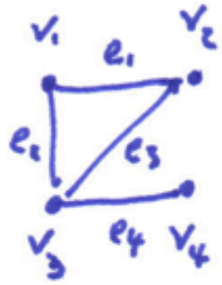
Algorithmen und Datenstrukturen
WS 2022/23

Prof. Dr. Sándor Fekete

3.6 Datenstrukturen für Graphen

Wie beschreibt man einen Graphen?

(1) Inzidenzmatrix



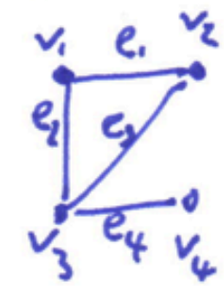
	e_1	e_2	e_3	e_4
v_1	1	1	0	0
v_2	1	0	1	0
v_3	0	1	1	1
v_4	0	0	0	1

(„inzident": zusammentreffend)

Also: $A \in \{0,1\}^{n \times m}$ mit $a_{v,e} := \begin{cases} 1 & \text{für } v \in e \\ 0 & \text{sonst} \end{cases}$

Größe: nm für einen Graphen mit n Knoten, m Kanten. (Viele Nullen!)

(2) Adjazenzmatrix



	v_1	v_2	v_3	v_4
v_1	0	1	1	0
v_2	1	0	1	0
v_3	1	1	0	1
v_4	0	0	1	0

(„adjazent": verbunden)

3.7 Wachstum von Funktionen

Im letzten Abschnitt haben wir Funktionen abgeschätzt und auf die „wesentlichen" Bestandteile reduziert, um Größenordnungen und Wachstumsverhalten zu beschreiben. Ein bisschen formaler:

Definition 3.9 (Θ-Notation)

Seien $f, g : \mathbb{N} \to \mathbb{R}$ Funktionen.

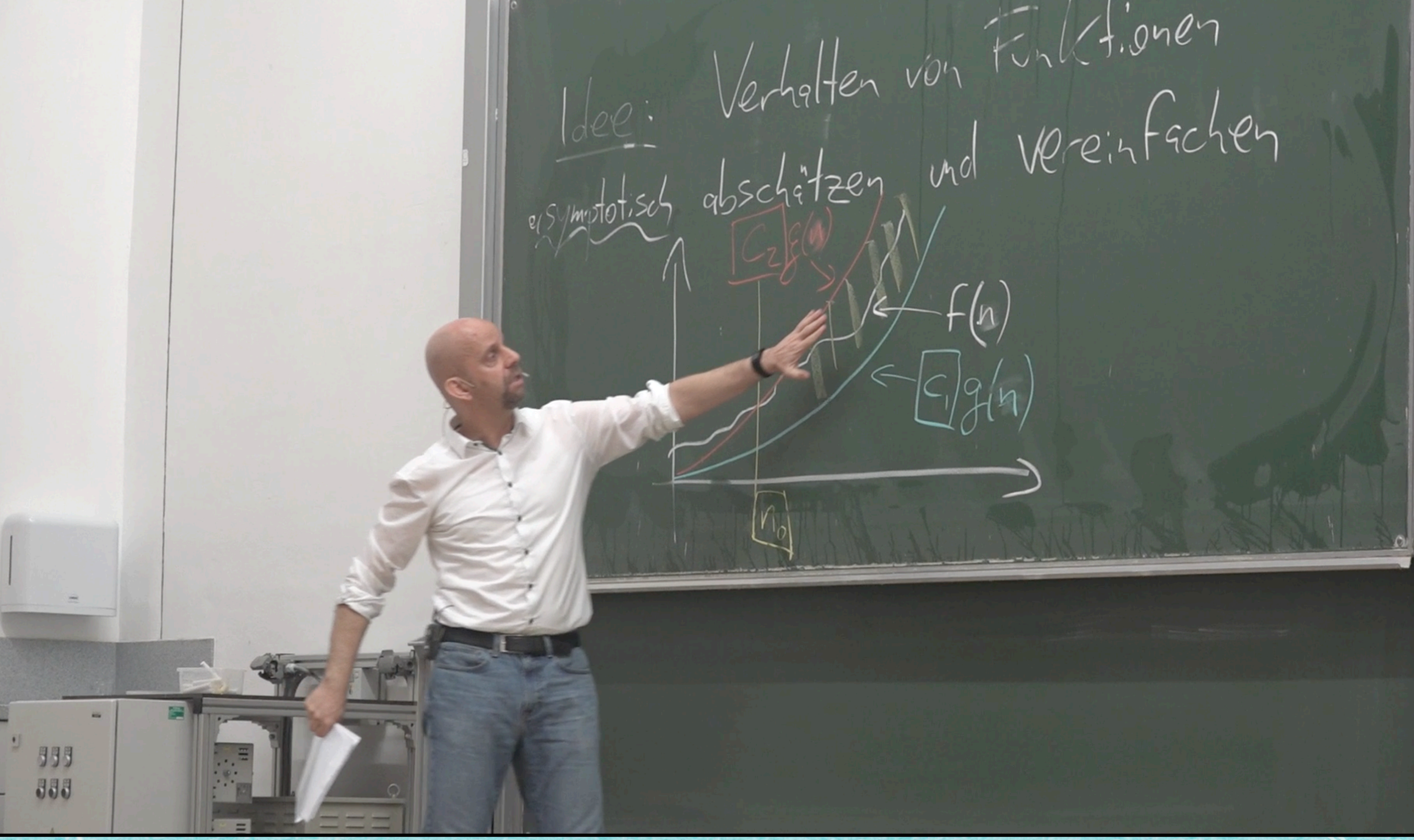
Dann gilt

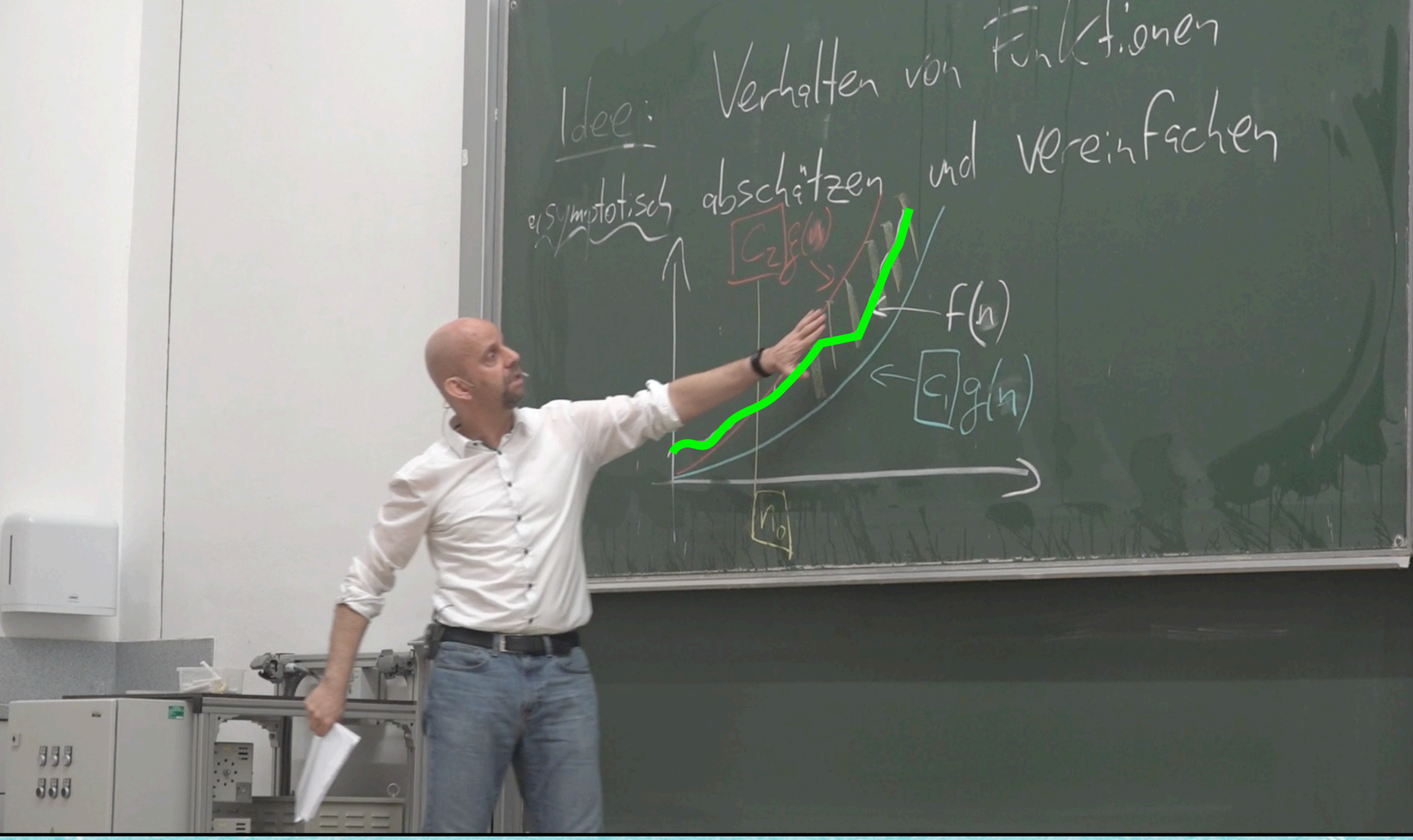
$f \in \Theta(g) :\Leftrightarrow$ Es gibt positive Konstanten c_1, c_2, n_0 mit
$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ für alle $n \geq n_0$.

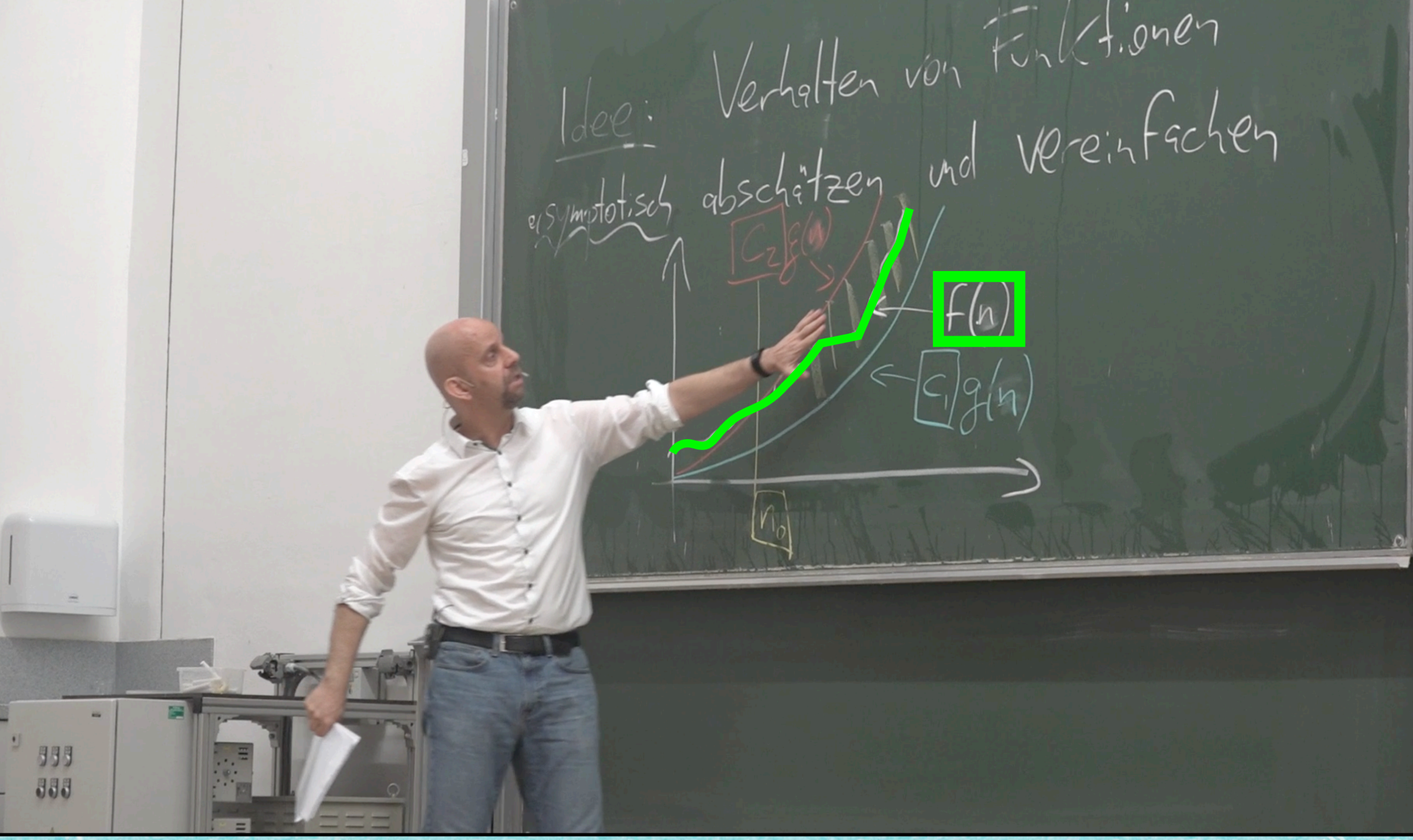
Man sagt: f wächst asymptotisch in derselben Größenordnung wie g.

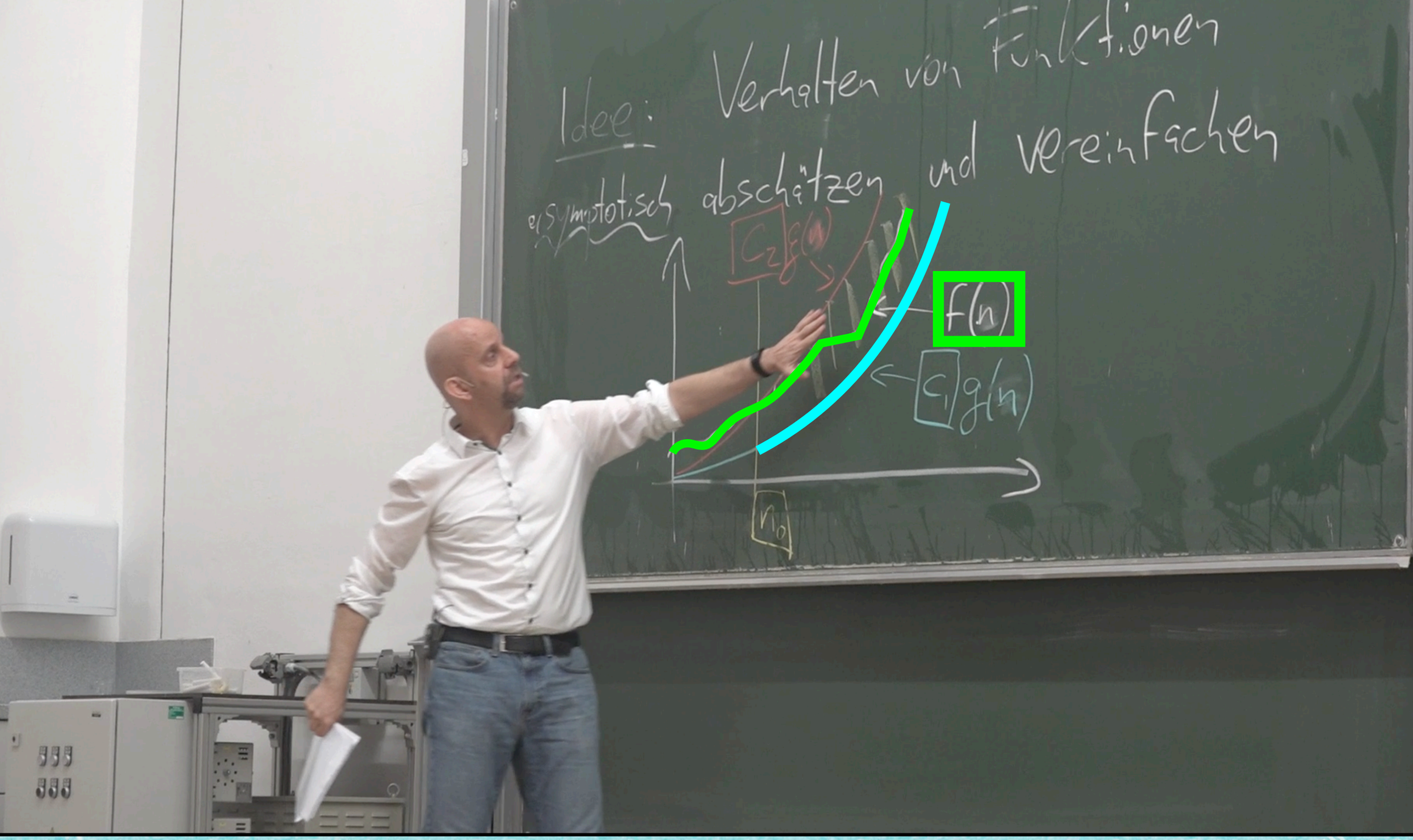
Beispiele: $2n^2 - 1 \in \Theta(n^2)$

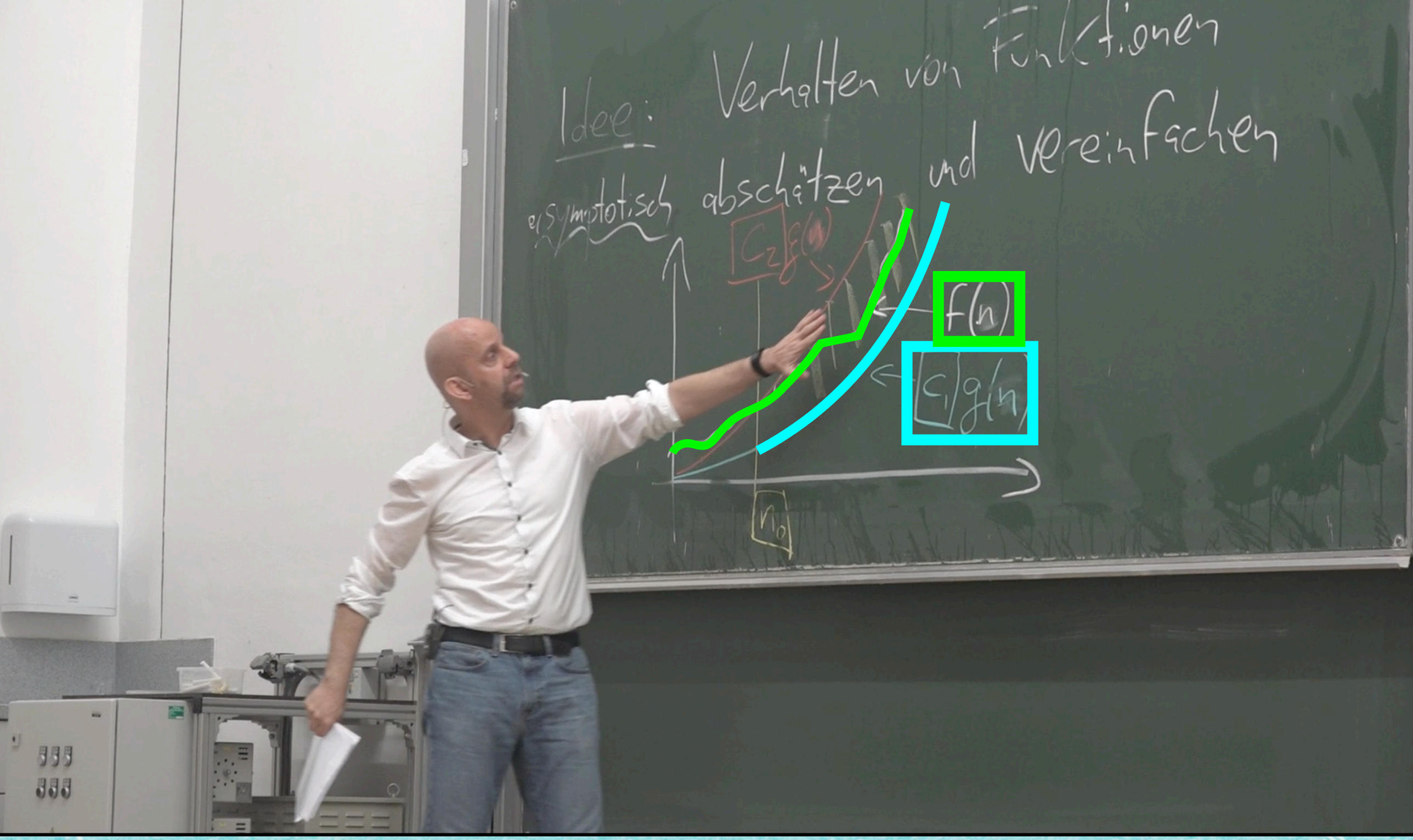
$\frac{n^3}{1000} + n^2 + n \log n \in \Theta(n^3)$

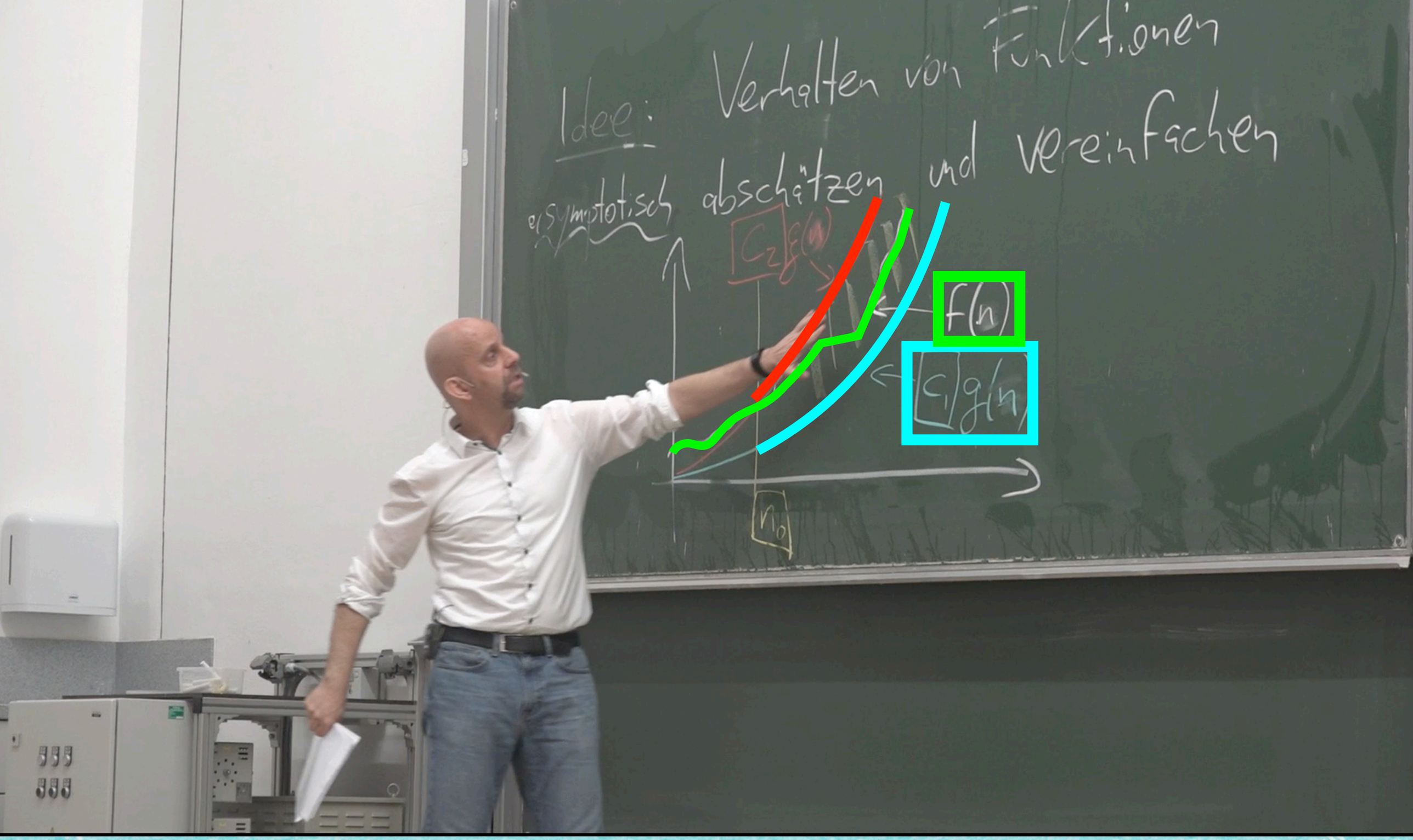

Idee:
Verhalten von Funktionen
asymptotisch abschätzen und vereinfachen
$c_2 g(n)$
$f(n)$
$c_1 g(n)$
n_0

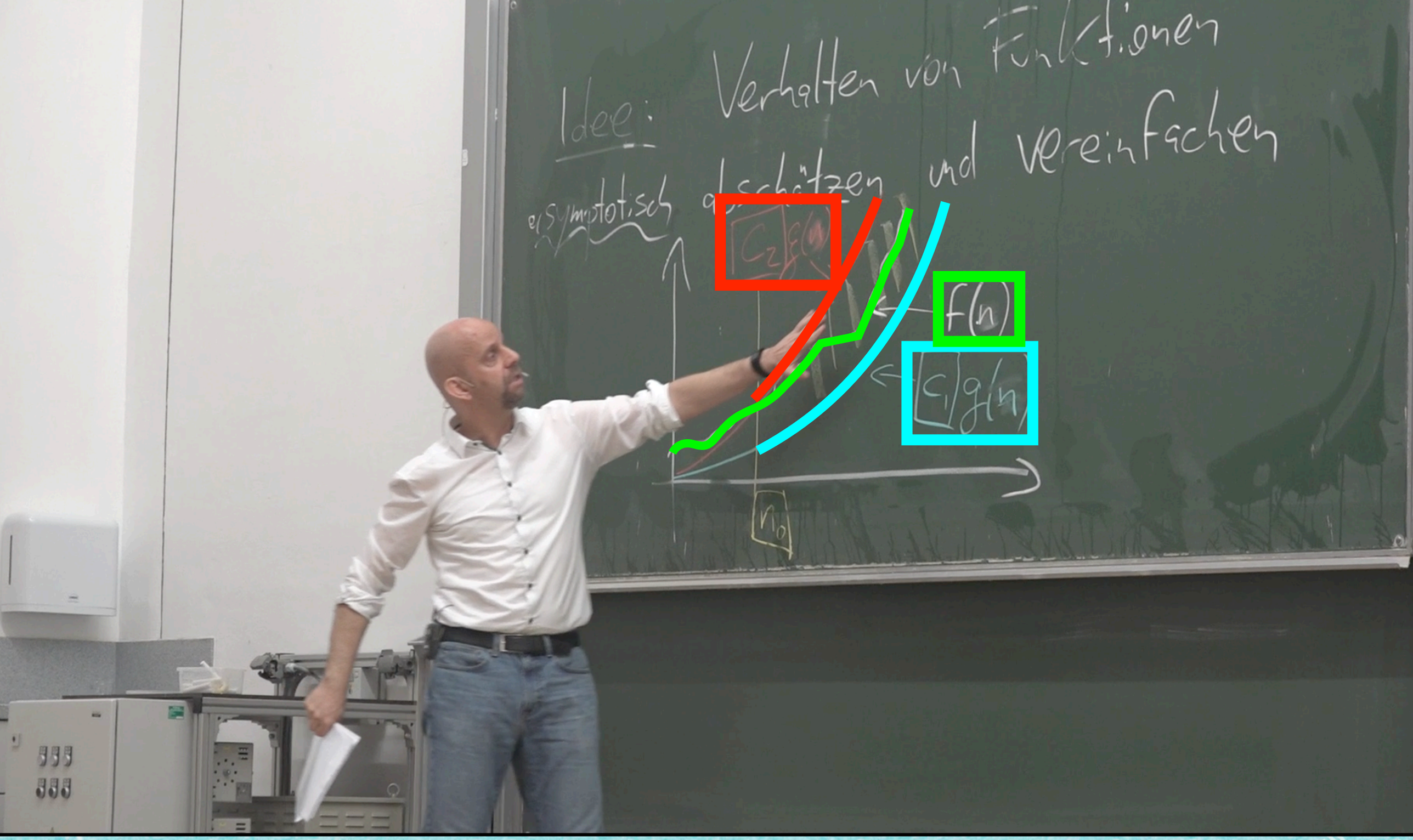

Idee:
Verhalten von Funktionen
asymptotisch abschätzen und vereinfachen
$c_2 g(n)$
$f(n)$
$c_1 g(n)$
n_0

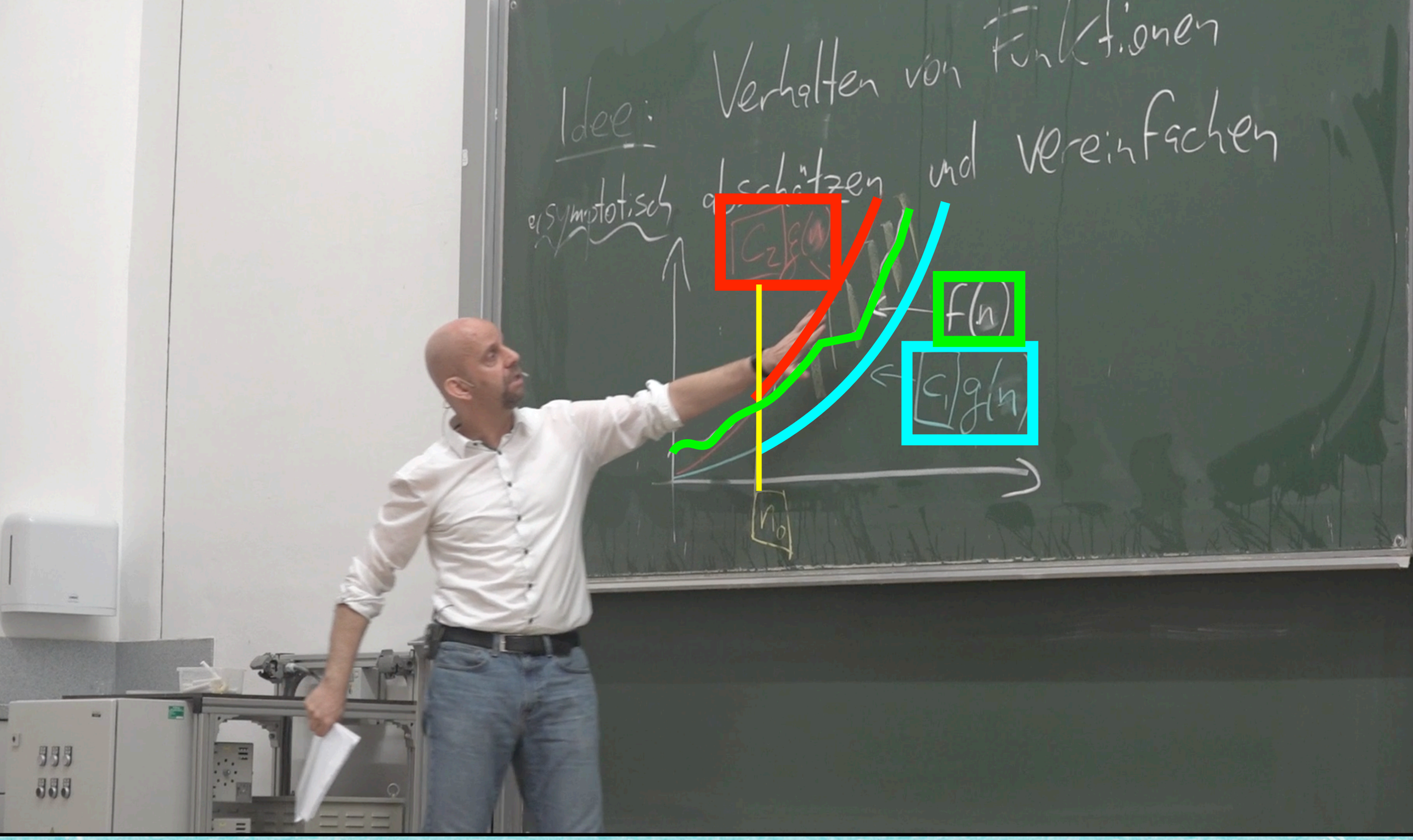

Idee:
Verhalten von Funktionen
asymptotisch abschätzen und vereinfachen
$c_2 g(n)$
$f(n)$
$c_1 g(n)$
n_0

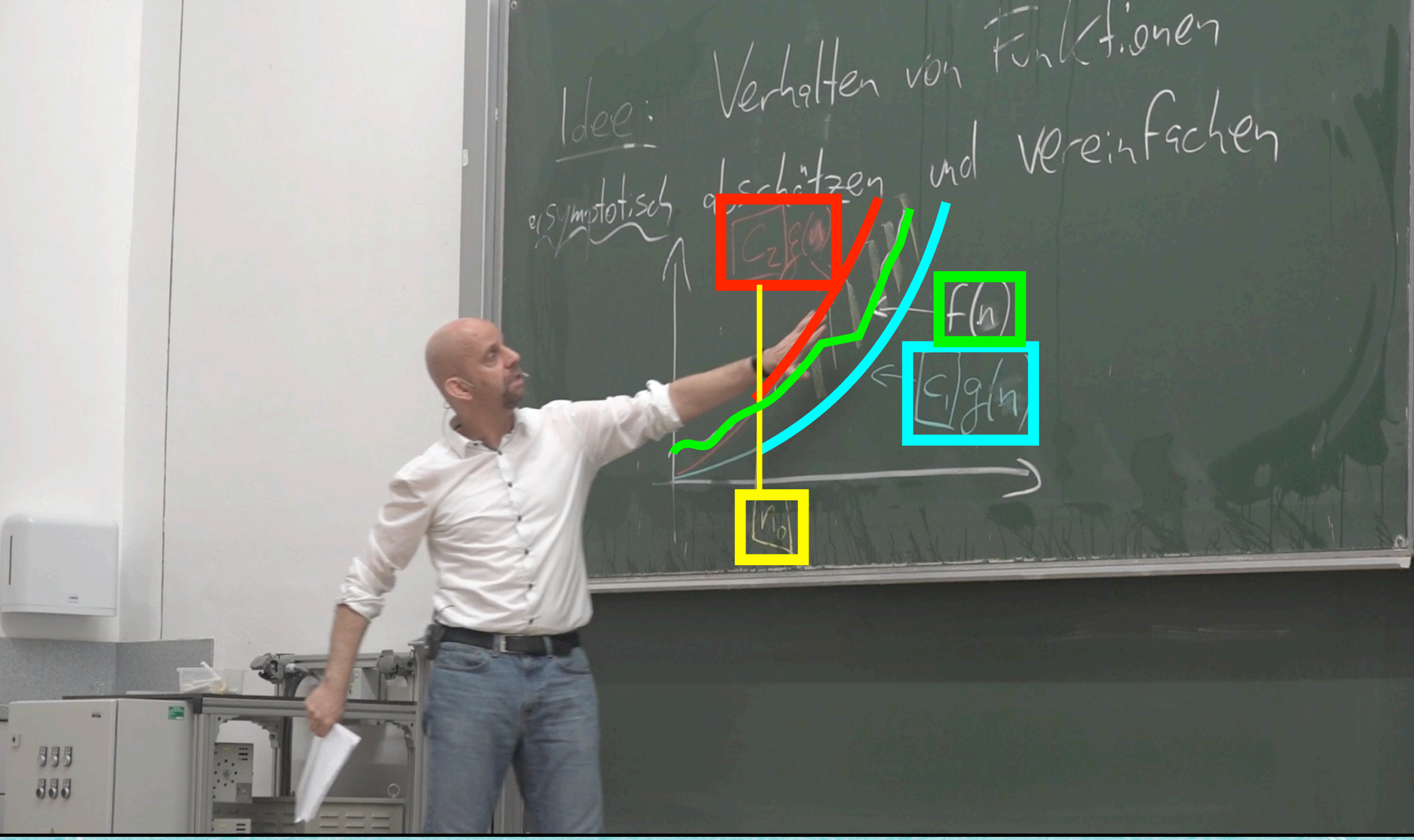

Idee:
Verhalten von Funktionen
asymptotisch abschätzen und vereinfachen
$c_2 g(n)$
$f(n)$
$c_1 g(n)$
n_0

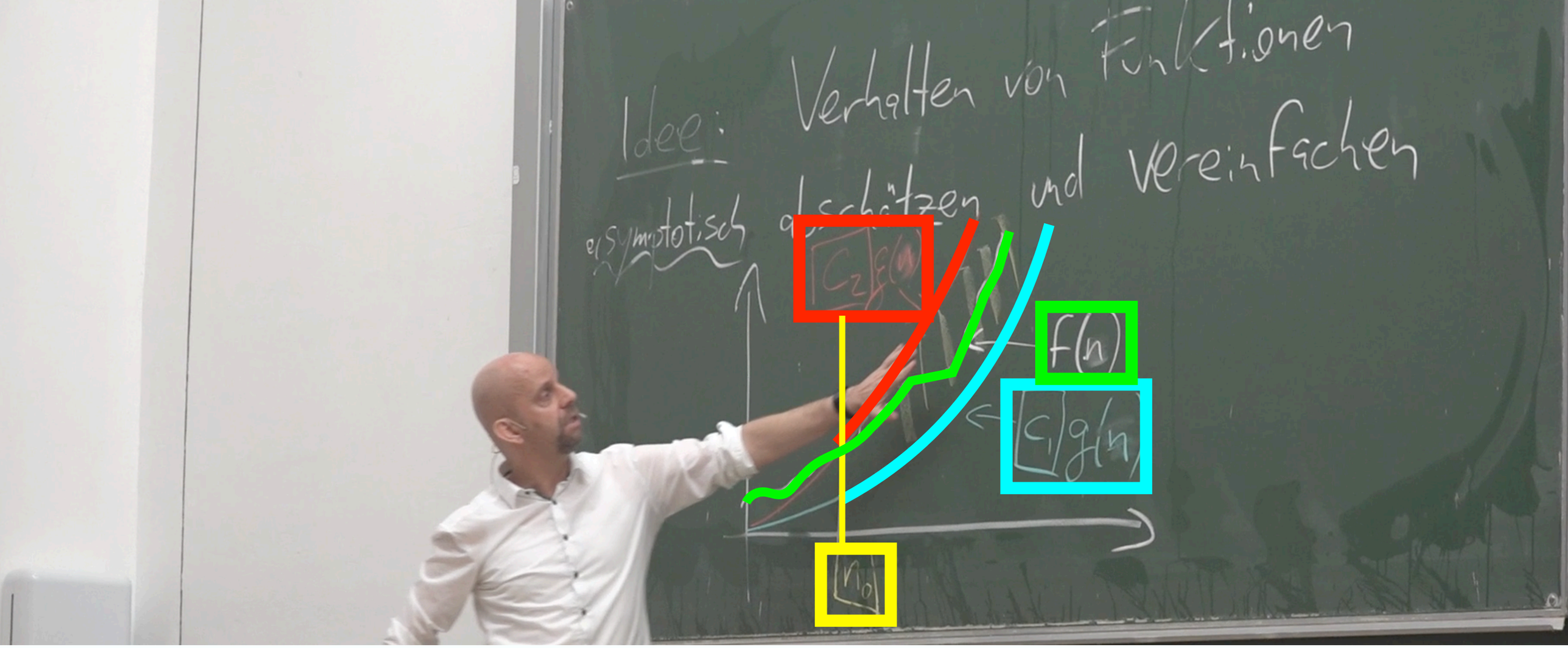

Idee:
Verhalten von Funktionen
asymptotisch abschätzen und vereinfachen
$c_2 f(n)$
$f(n)$
$c_1 g(n)$
n_0


Idee:
Verhalten von Funktionen
asymptotisch abschätzen und vereinfachen
$c_2 f(n)$
$f(n)$
$c_1 g(n)$
n_0


Idee:
Verhalten von Funktionen
asymptotisch
und vereinfachen
$c_2 f(n)$
$f(n)$
$c_1 g(n)$
n_0


Idee:
Verhalten von Funktionen
asymptotisch
und vereinfachen
$c_2 f(n)$
$f(n)$
$c_1 g(n)$
n_0


Idee:
Verhalten von Funktionen
asymptotisch abschätzen und vereinfachen
$c_2 f(n)$
$f(n)$
$c_1 g(n)$
n_0



Definition 3.9 (Θ-Notation)

Seien $f, g : \mathbb{N} \to \mathbb{R}$ Funktionen.

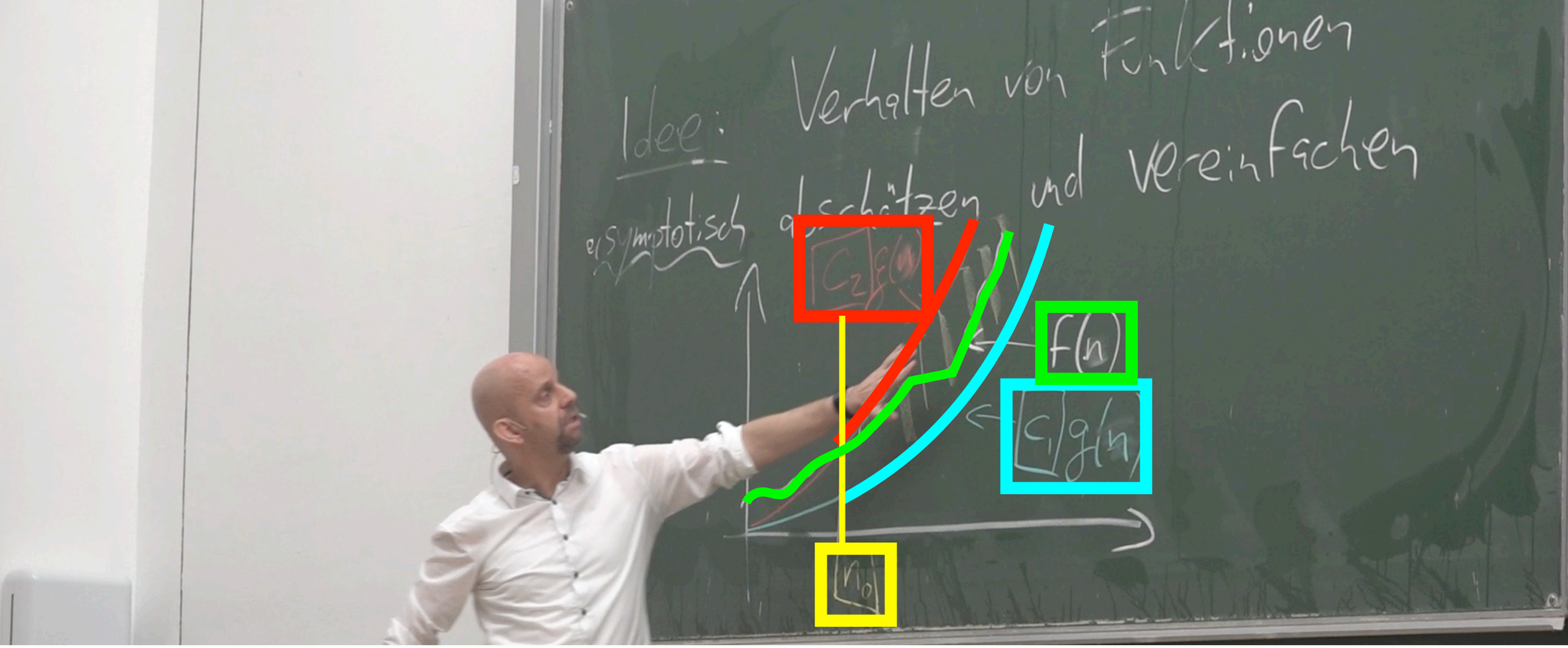
Dann gilt

$f \in \Theta(g) :\Leftrightarrow$ Es gibt positive Konstanten c_1, c_2, n_0 mit

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ für alle } n \geq n_0.$$

Man sagt: f wächst asymptotisch in derselben Größenordnung wie g.

Idee: Verhalten von Funktionen asymptotisch abschätzen und vereinfachen



Definition 3.9 (Θ-Notation)

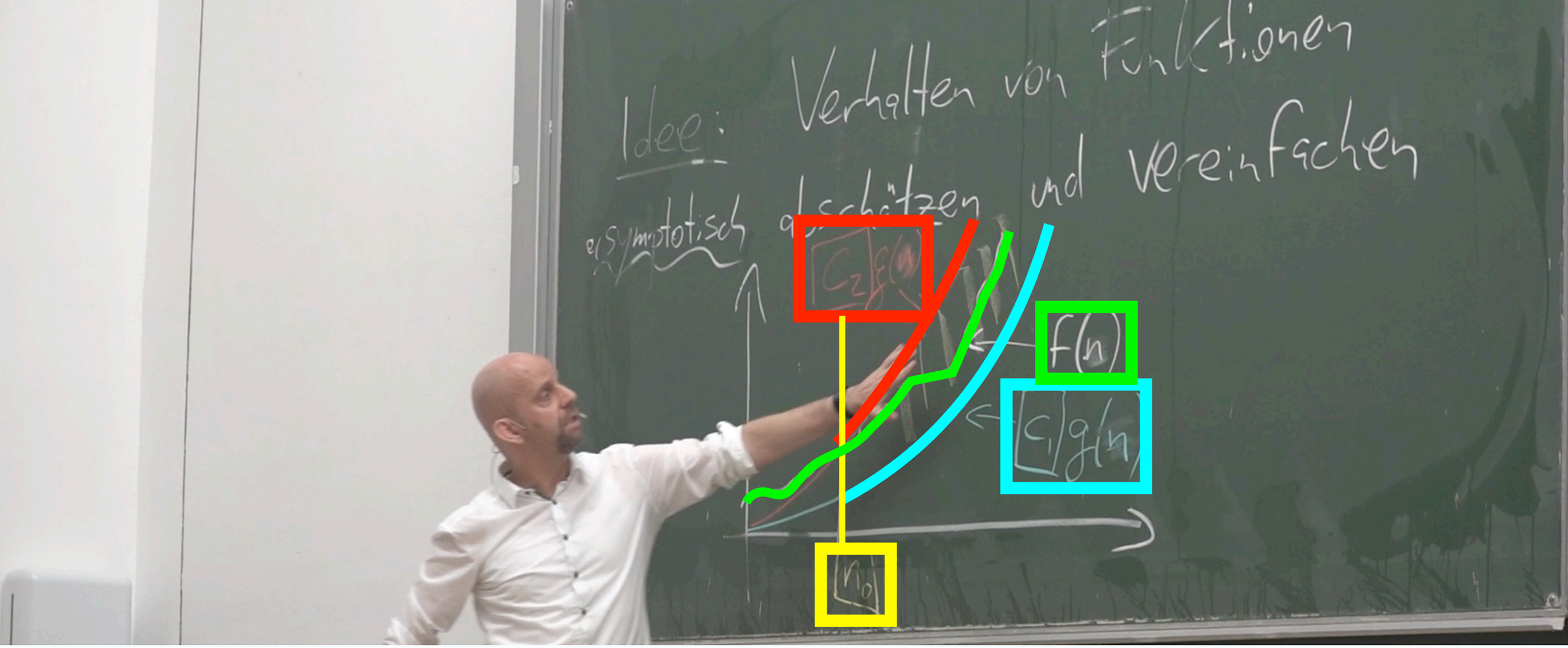
Seien $f, g : \mathbb{N} \to \mathbb{R}$ Funktionen.

Dann gilt

$f \in \Theta(g) :\Leftrightarrow$ Es gibt positive Konstanten c_1, c_2, n_0 mit

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ für alle } n \geq n_0.$$

Man sagt: f wächst asymptotisch in derselben Größenordnung wie g.



Definition 3.9 (Θ-Notation)

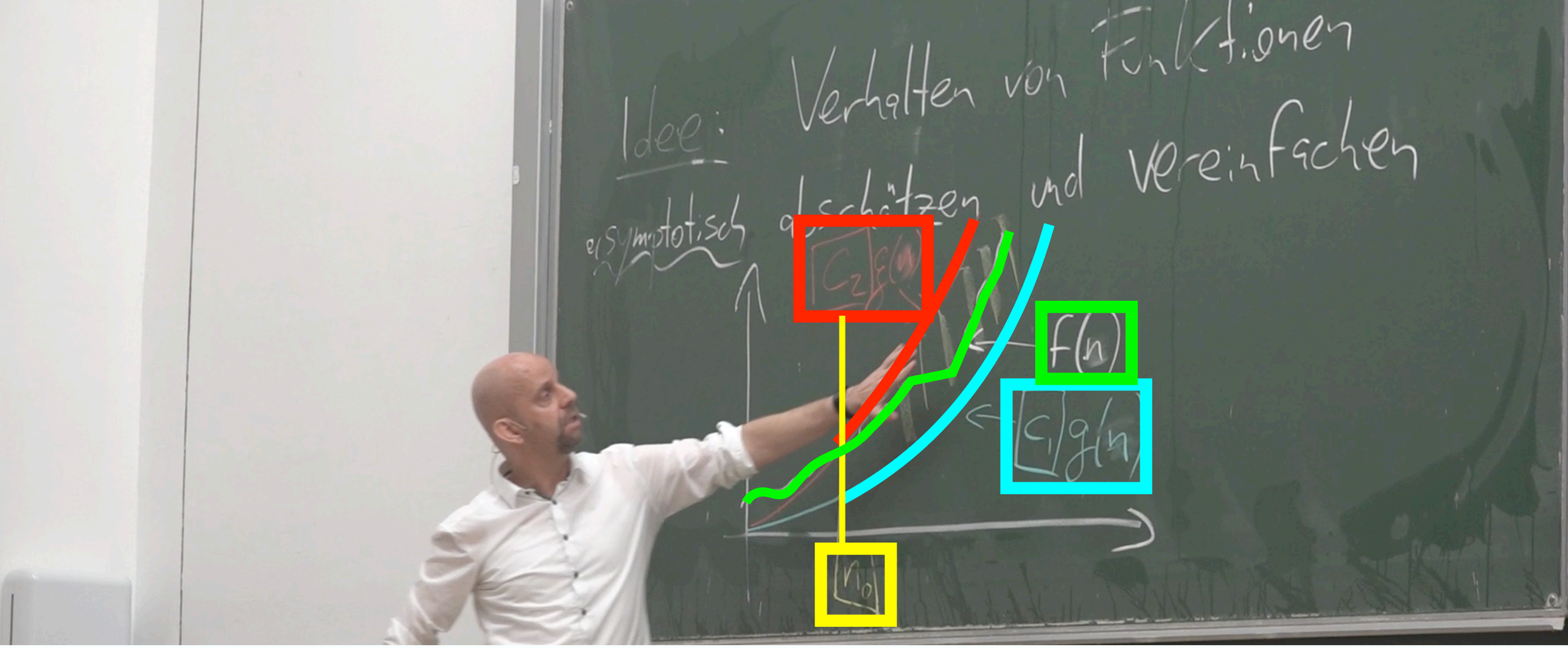
Seien $f, g : \mathbb{N} \to \mathbb{R}$ Funktionen.

Dann gilt

$f \in \Theta(g) :\Leftrightarrow$ Es gibt positive Konstanten c_1, c_2, n_0 mit

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ für alle } n \geq n_0.$$

Man sagt: f wächst asymptotisch in derselben Größenordnung wie g.



Definition 3.9 (Θ-Notation)

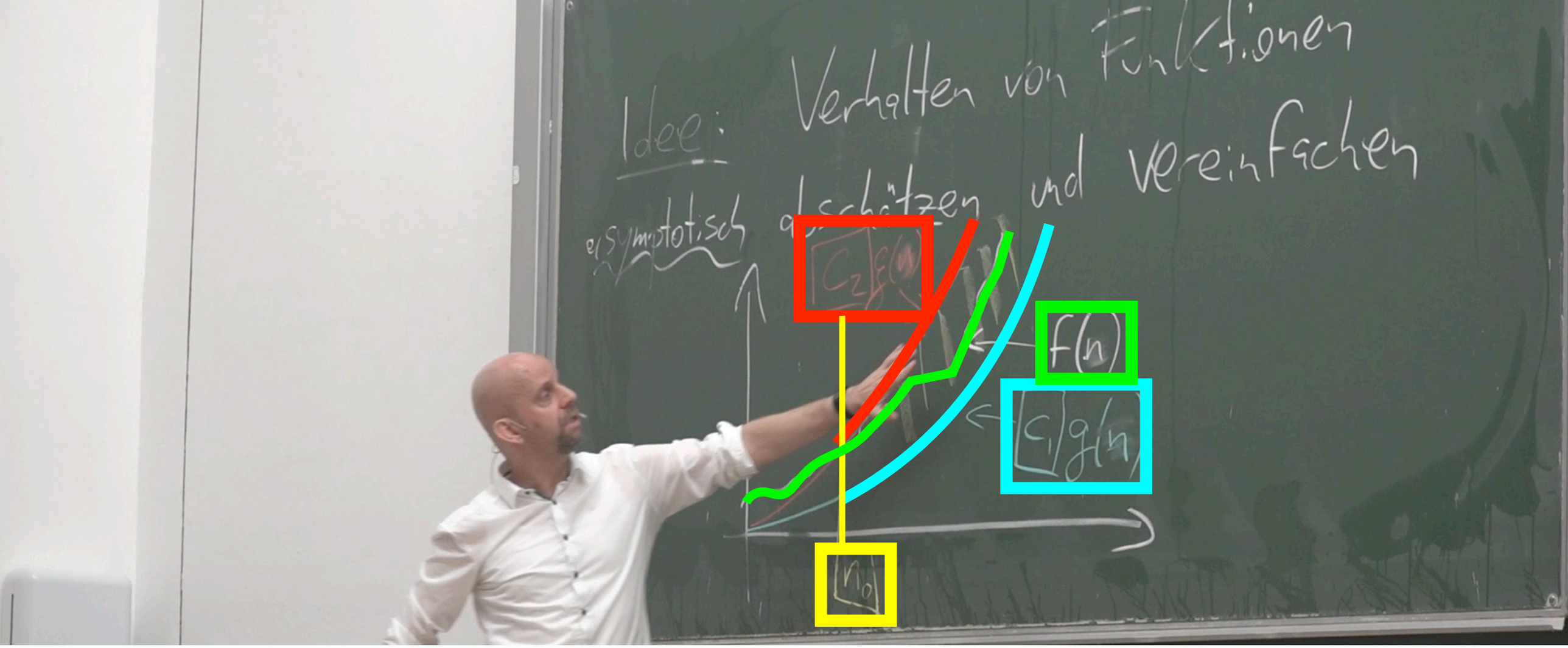
Seien $f, g : \mathbb{N} \to \mathbb{R}$ Funktionen.

Dann gilt

$f \in \Theta(g) :\Leftrightarrow$ Es gibt positive Konstanten c_1, c_2, n_0 mit

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \text{für alle } n \geq n_0.$$

Man sagt: f wächst asymptotisch in derselben Größenordnung wie g.



Definition 3.9 (Θ-Notation)

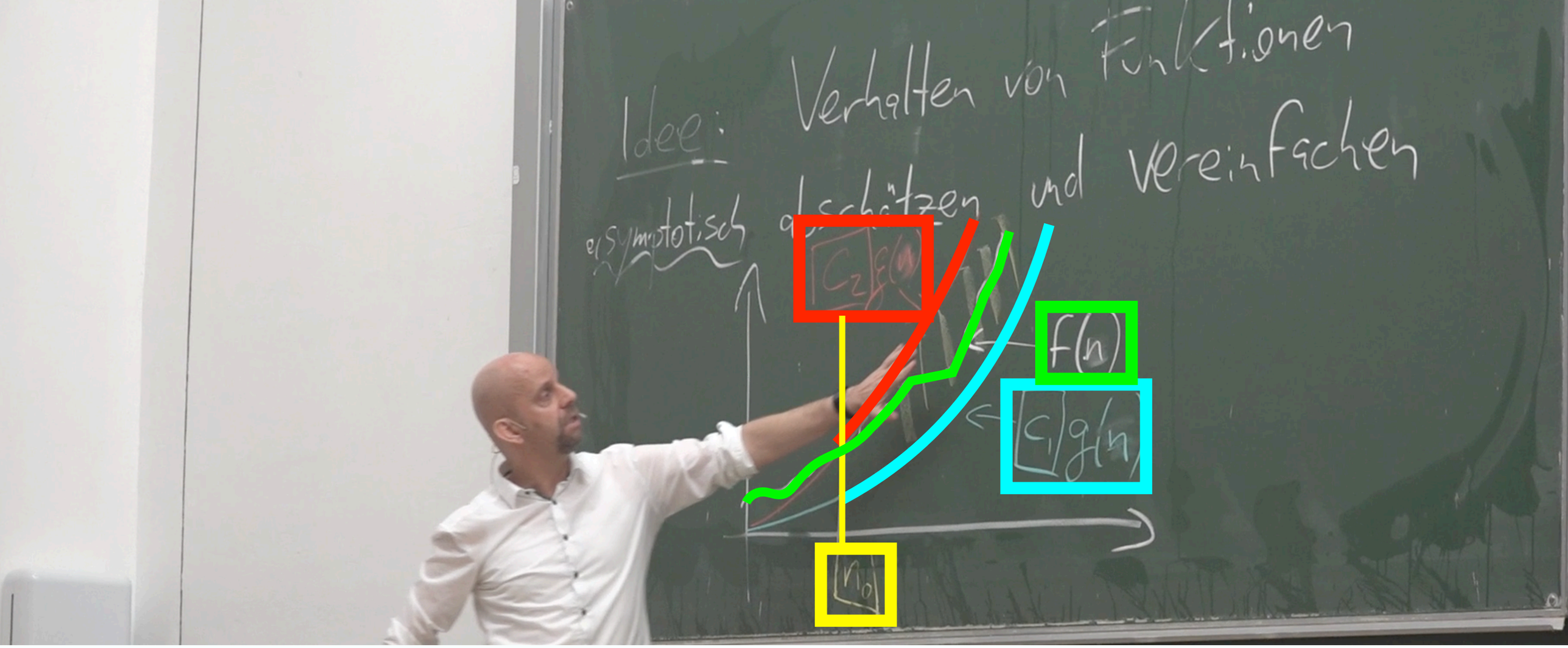
Seien $f, g : \mathbb{N} \to \mathbb{R}$ Funktionen.

Dann gilt

$f \in \Theta(g) :\Leftrightarrow$ Es gibt positive Konstanten c_1, c_2, n_0 mit

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \text{für alle } n \geq n_0.$$

Man sagt: f wächst asymptotisch in derselben Größenordnung wie g.



Definition 3.9 (Θ-Notation)

Seien $f, g : \mathbb{N} \to \mathbb{R}$ Funktionen.

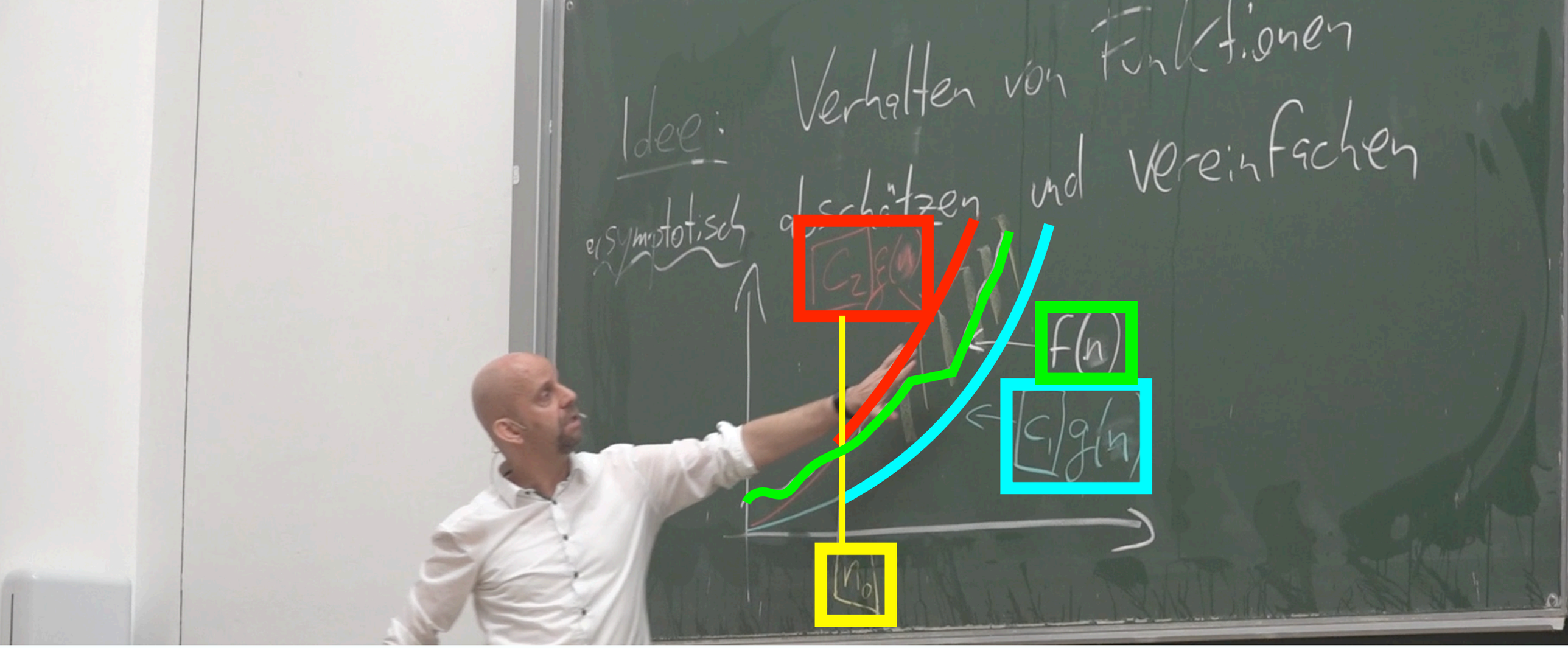
Dann gilt

$f \in \Theta(g) :\Leftrightarrow$ Es gibt positive Konstanten c_1 c_2 n_0 mit

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \text{für alle } n \geq n_0.$$

Man sagt: f wächst asymptotisch in derselben Größenordnung wie g.

Idee: Verhalten von Funktionen asymptotisch abschätzen und vereinfachen



Definition 3.9 (Θ-Notation)

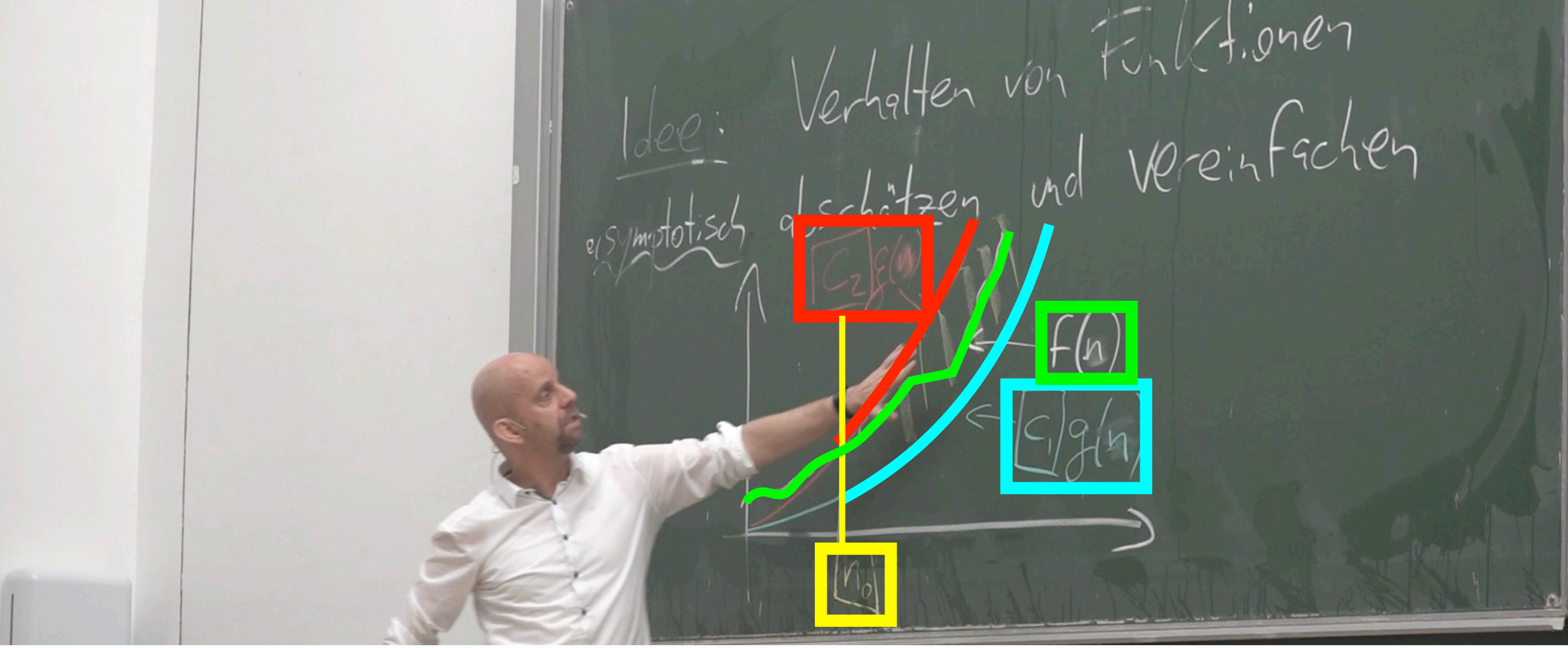
Seien $f, g : \mathbb{N} \to \mathbb{R}$ Funktionen.

Dann gilt

$f \in \Theta(g) :\Leftrightarrow$ Es gibt positive Konstanten c_1 c_2 n_0 mit

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \text{für alle } n \geq n_0.$$

Man sagt: f wächst asymptotisch in derselben Größenordnung wie g.



Definition 3.9 (Θ-Notation)

Seien $f, g : \mathbb{N} \to \mathbb{R}$ Funktionen.

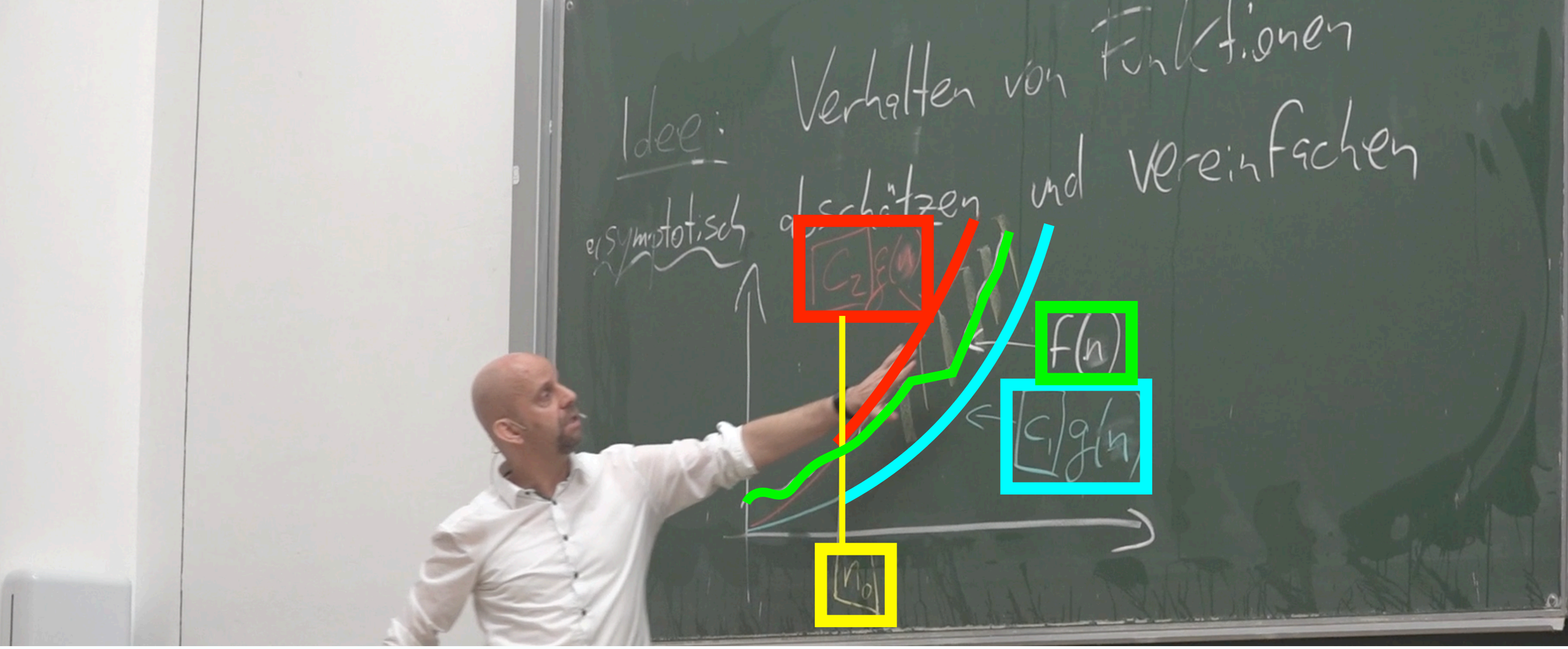
Dann gilt

$f \in \Theta(g) :\Leftrightarrow$ Es gibt positive Konstanten c_1 c_2 n_0 mit

$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ für alle $n \geq n_0$

Man sagt: f wächst asymptotisch in derselben Größenordnung wie g.

Idee: Verhalten von Funktionen asymptotisch abschätzen und vereinfachen



Definition 3.9 (Θ-Notation)

Seien $f, g : \mathbb{N} \to \mathbb{R}$ Funktionen.

Dann gilt

$f \in \Theta(g) \Leftrightarrow$ Es gibt positive Konstanten c_1 c_2 n_0 mit

$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ für alle $n \geq n_0$

Man sagt: f wächst asymptotisch in derselben Größenordnung wie g.

Idee: Verhalten von Funktionen asymptotisch abschätzen und vereinfachen

$c_2 f(n)$ $f(n)$ $c_1 g(n)$ n_0

$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$f \in \Theta(g) :\Leftrightarrow$ Es gibt positive Konstanten c_1 c_2 n_0 mit

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \text{für alle} \quad n \geq n_0$$

Man sagt: f wächst asymptotisch in derselben Größenordnung wie g.

Idee: Verhalten von Funktionen asymptotisch abschätzen und vereinfachen

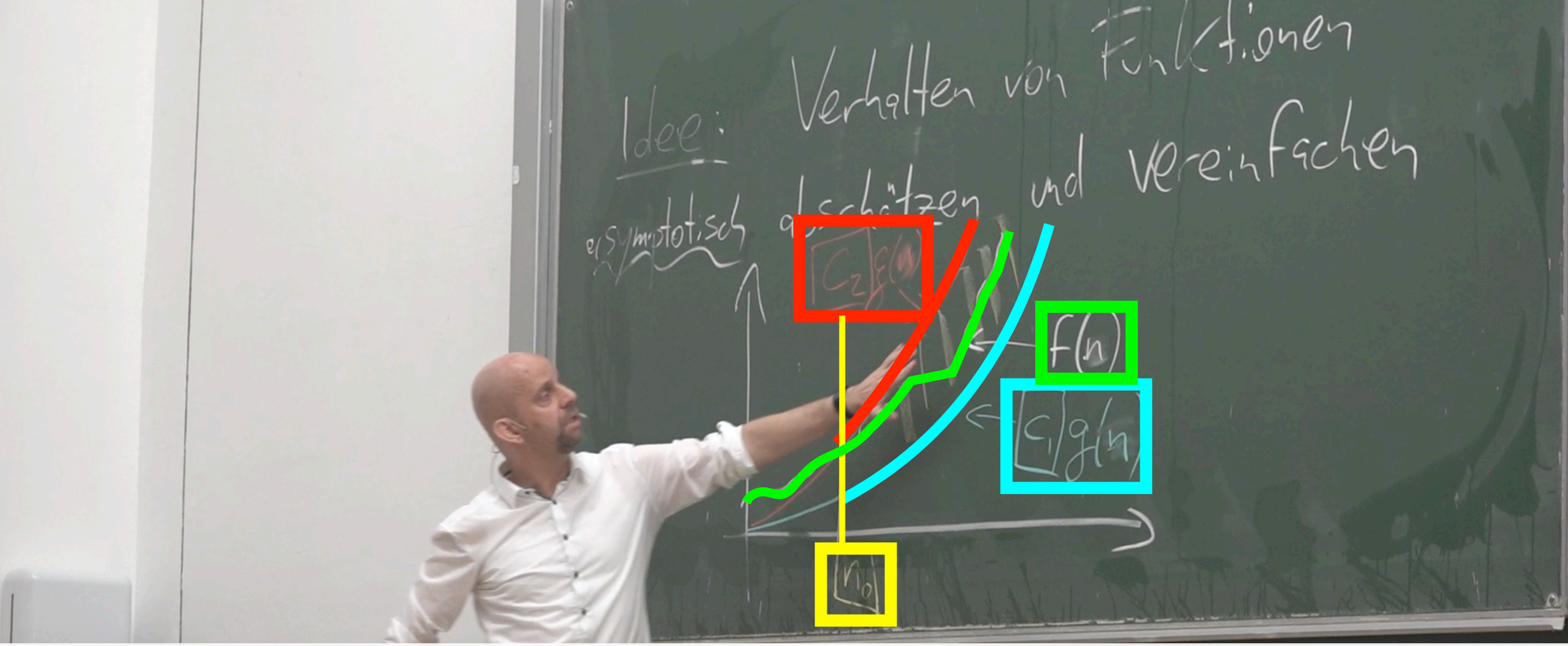
$c_2 f(n)$

$f(n)$

$c_1 g(n)$

n_0

$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$



$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$$c_1 \leq \frac{f(n)}{g(n)} \leq c_2$$

Idee: Verhalten von Funktionen asymptotisch abschätzen und vereinfachen

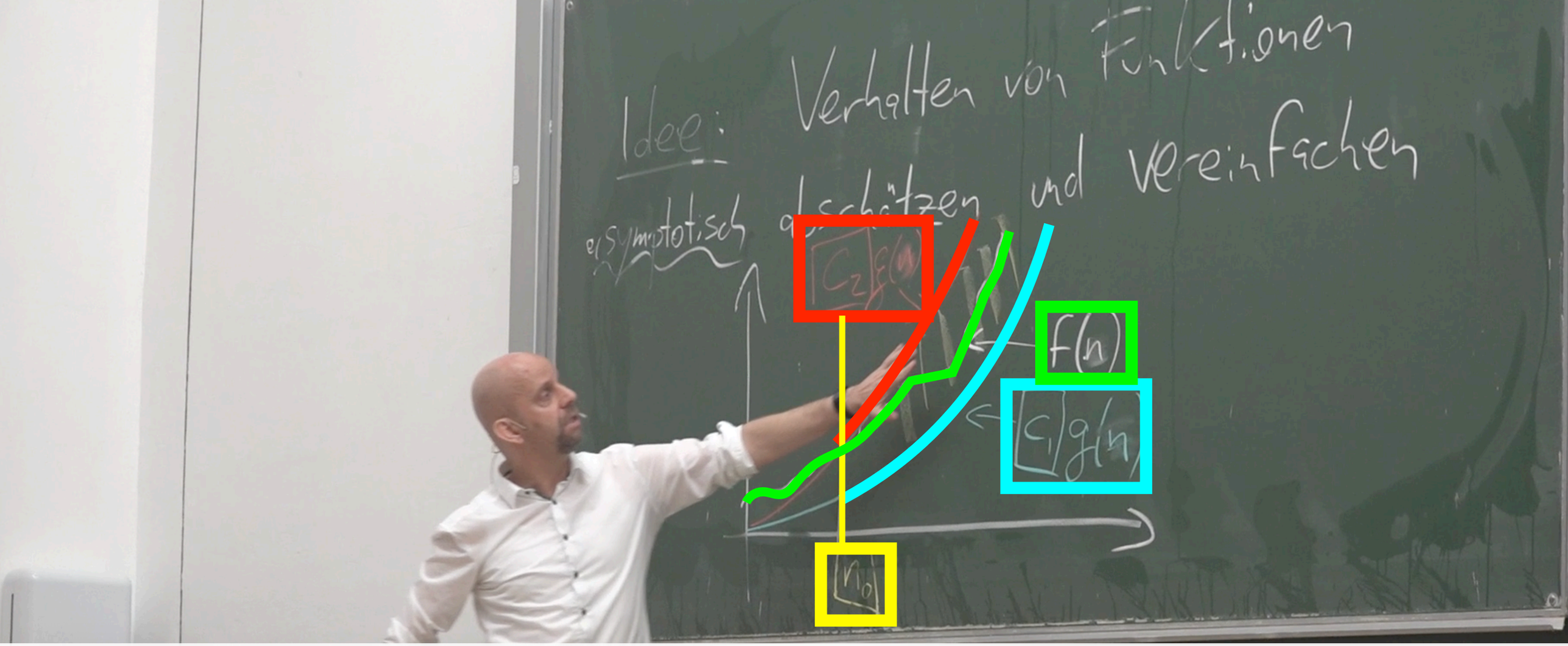
$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$$c_1 \leq \frac{f(n)}{g(n)} \leq c_2$$

Es gibt positive Konstanten c_1 c_2 n_0 mit

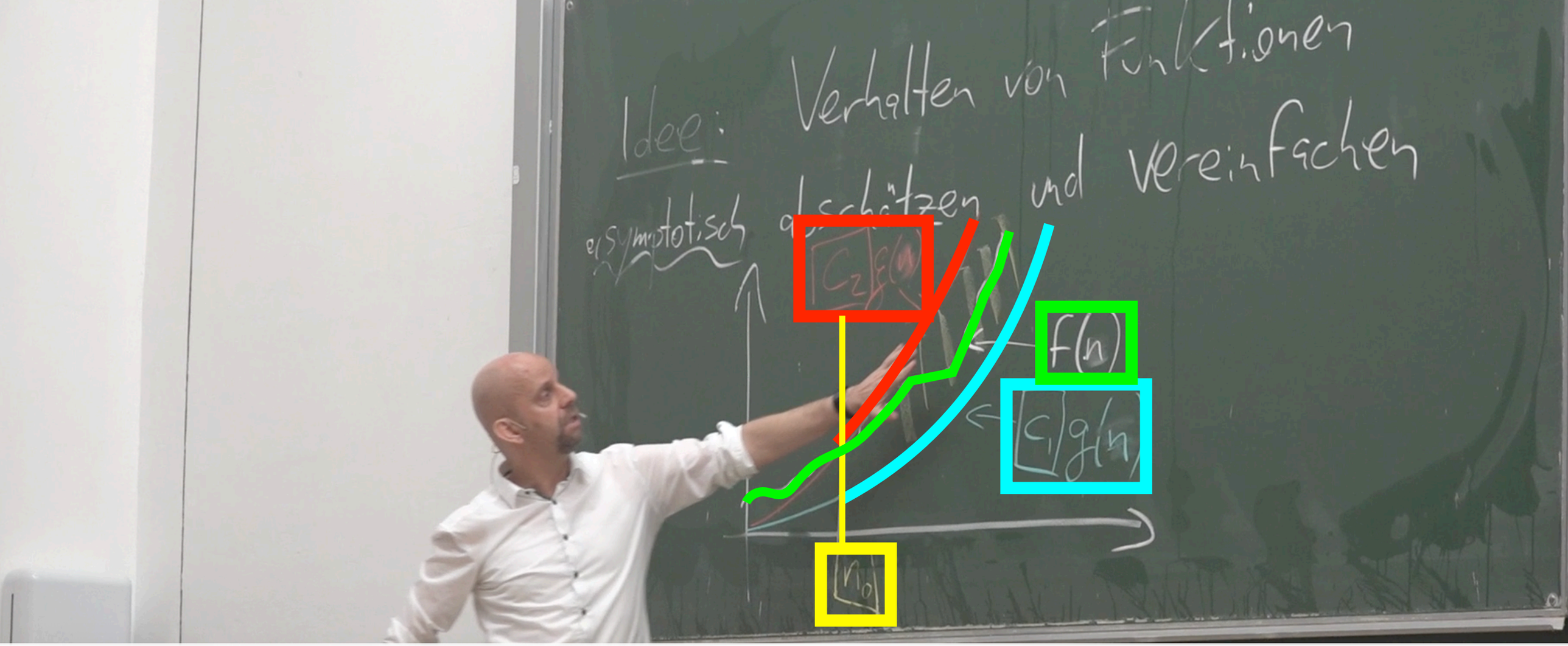
$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ für alle $n \geq n_0$

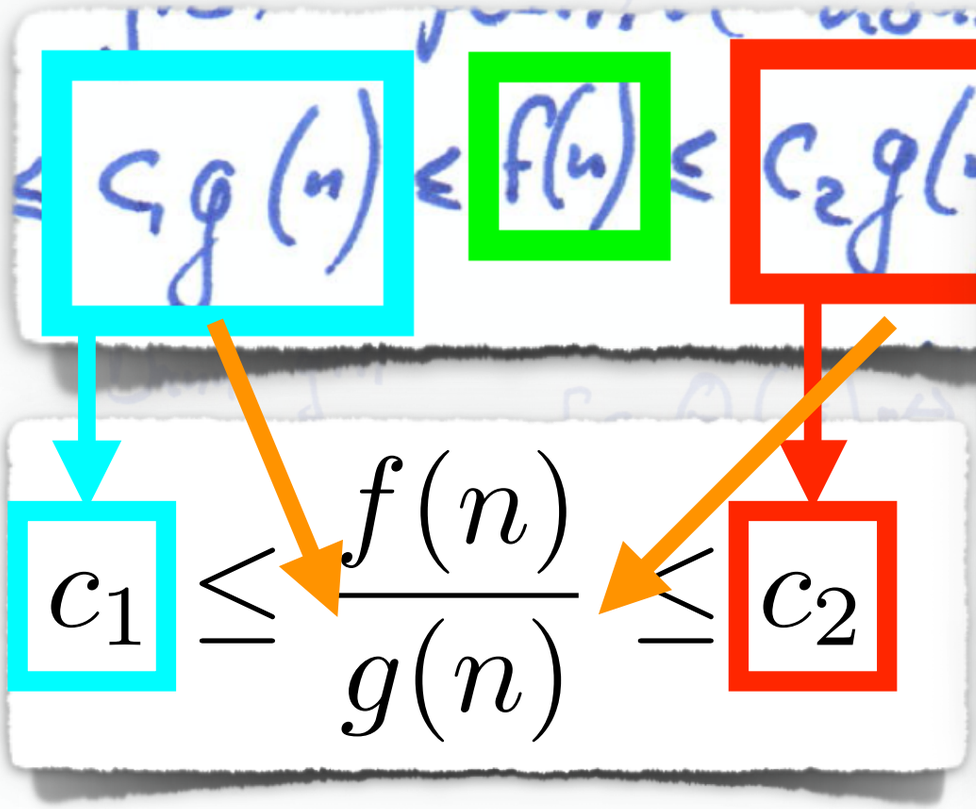
asymptotisch in derselben Größenordnung wie g



$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$$c_1 \leq \frac{f(n)}{g(n)} \leq c_2$$

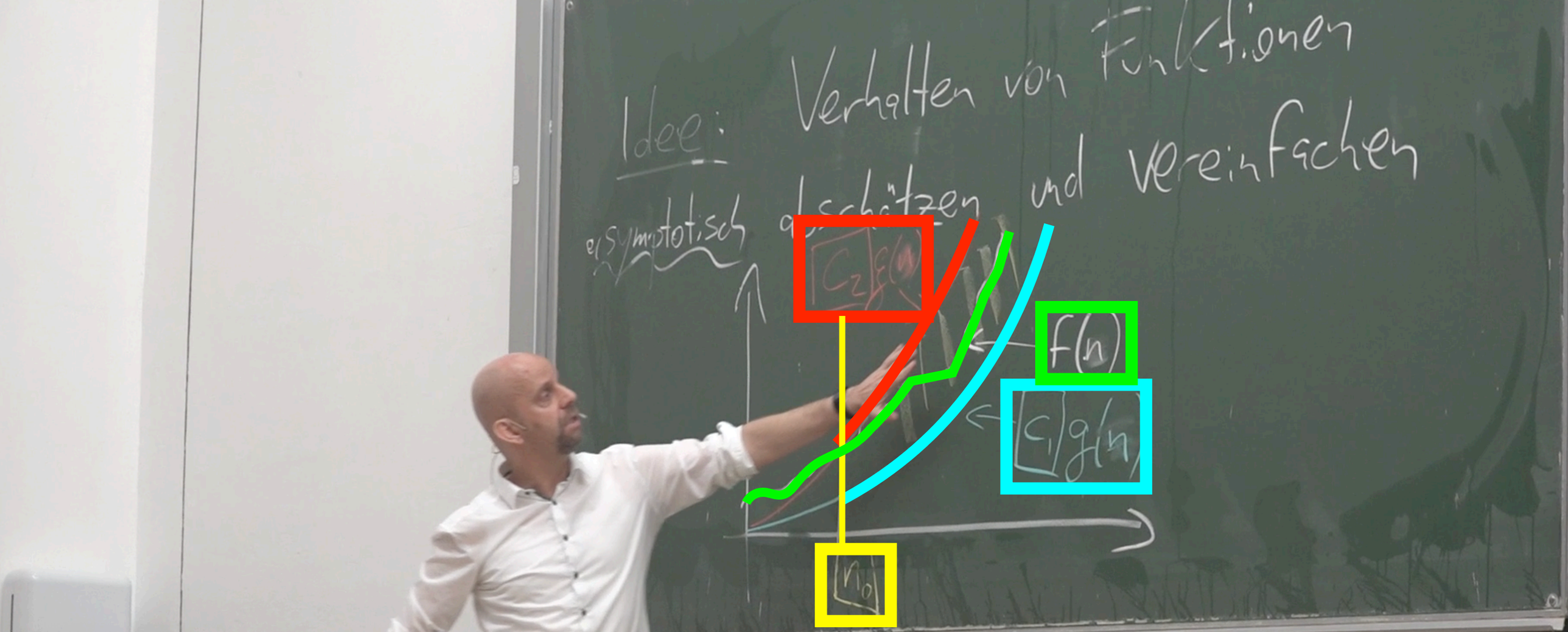




Frage 7:

Die Funktion $f(n) := 5n^6 + 4n^4 + 7n^3 + 27n - 9$ liegt...

- ... nur in $O(n^6)$.
- ... nur in $\Omega(n^6)$.
- ... in $\Theta(n^6)$.



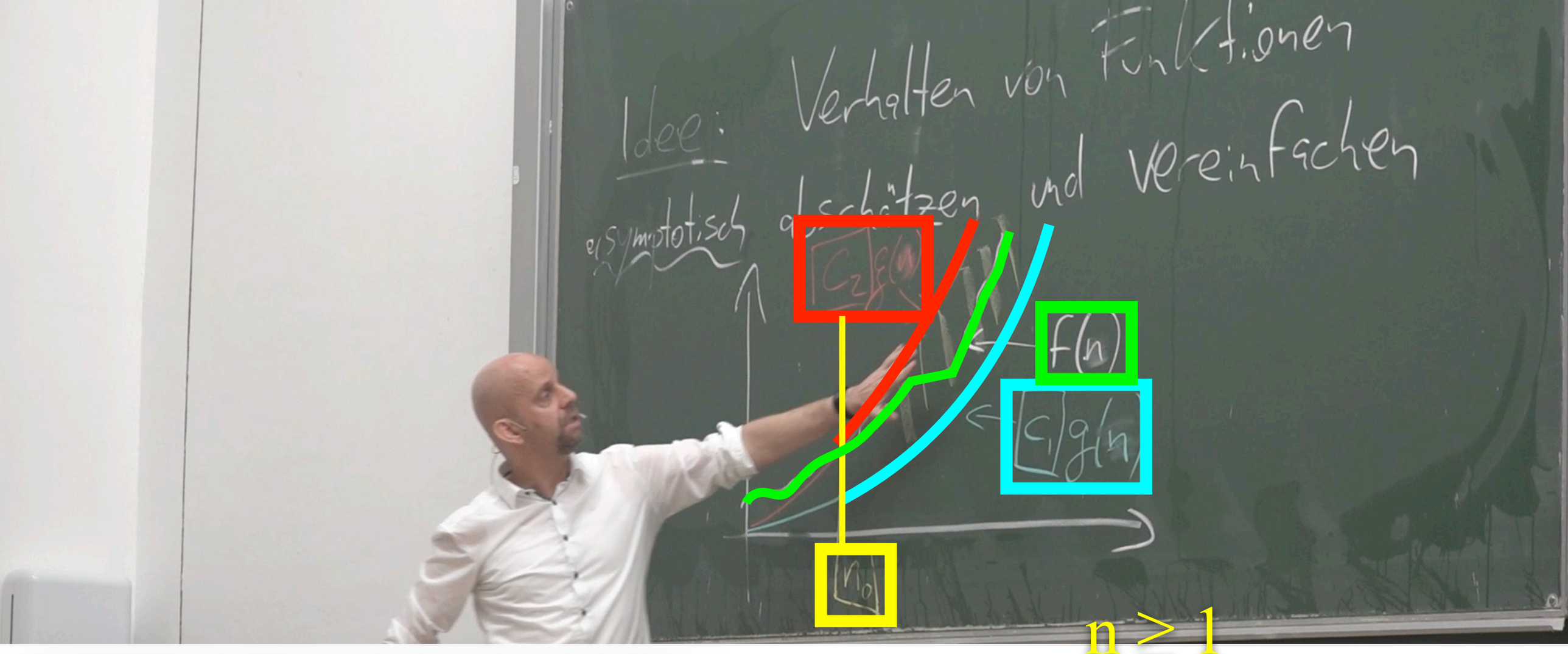
$$c_1 \leq \frac{f(n)}{g(n)} \leq c_2$$

$$\frac{f(n)}{g(n)} = 5 + \frac{4}{n^2} + \frac{7}{n^3} + \frac{27}{n^5} - \frac{9}{n^6}$$

Frage 7:

Die Funktion $f(n) := 5n^6 + 4n^4 + 7n^3 + 27n - 9$ liegt...

- ... nur in $O(n^6)$.
- ... nur in $\Omega(n^6)$.
- ... in $\Theta(n^6)$.



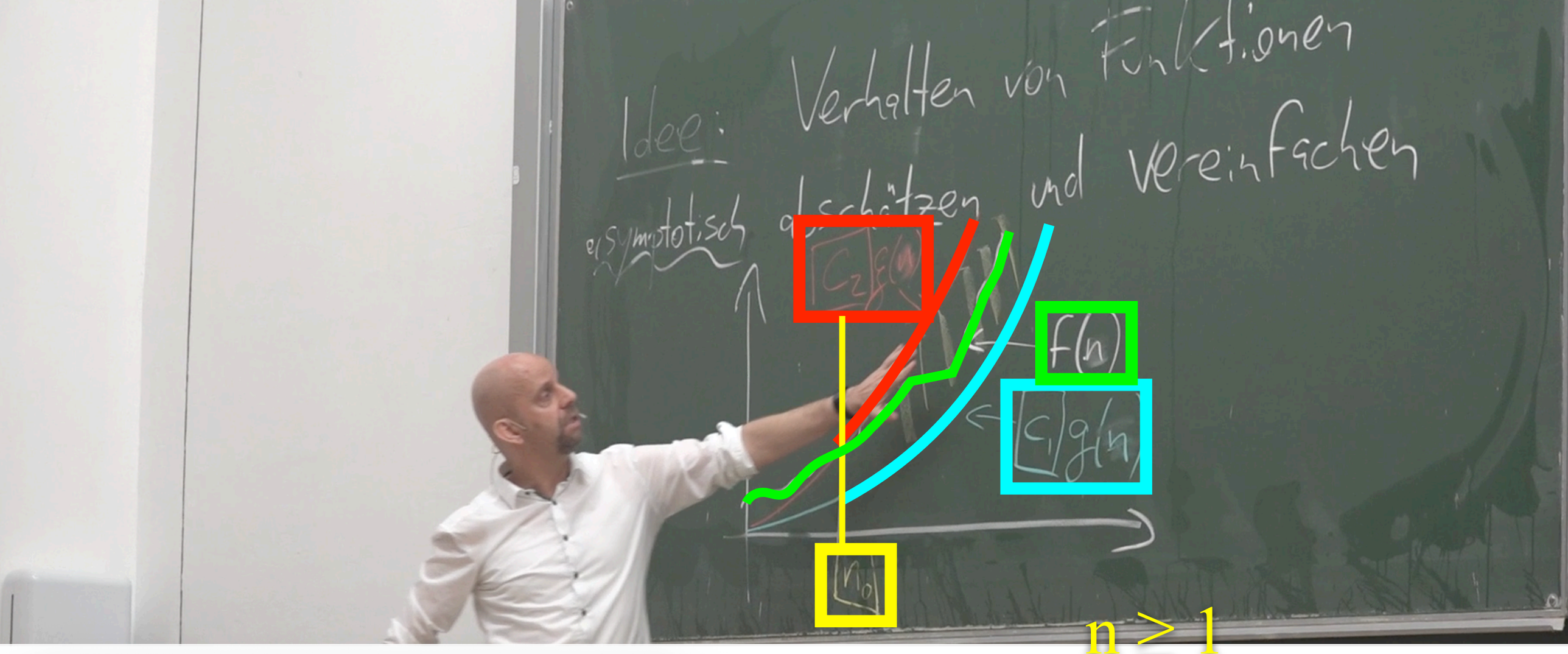
$$c_1 \leq \frac{f(n)}{g(n)} \leq c_2$$

Frage 7:

$$\frac{f(n)}{g(n)} = 5 + \frac{4}{n^2} + \frac{7}{n^3} + \frac{27}{n^5} - \frac{9}{n^6}$$

Die Funktion $f(n) := 5n^6 + 4n^4 + 7n^3 + 27n - 9$ liegt...

- ... nur in $O(n^6)$.
- ... nur in $\Omega(n^6)$.
- ... in $\Theta(n^6)$.



$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

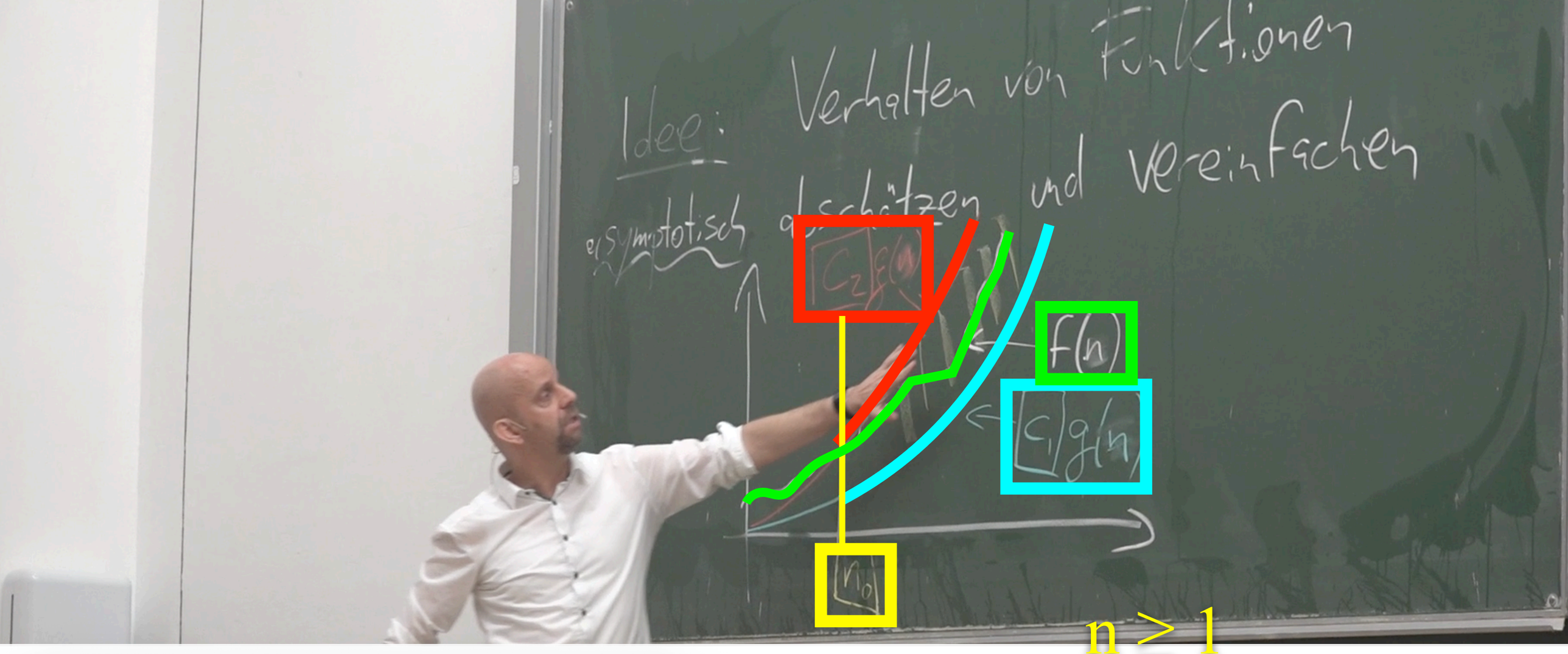
$$c_1 \leq \frac{f(n)}{g(n)} \leq c_2$$

$$5 \leq \frac{f(n)}{g(n)} = 5 + \frac{4}{n^2} + \frac{7}{n^3} + \frac{27}{n^5} - \frac{9}{n^6}$$

Frage 7:

Die Funktion $f(n) := 5n^6 + 4n^4 + 7n^3 + 27n - 9$ liegt...

- ... nur in $O(n^6)$.
- ... nur in $\Omega(n^6)$.
- ... in $\Theta(n^6)$.



$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

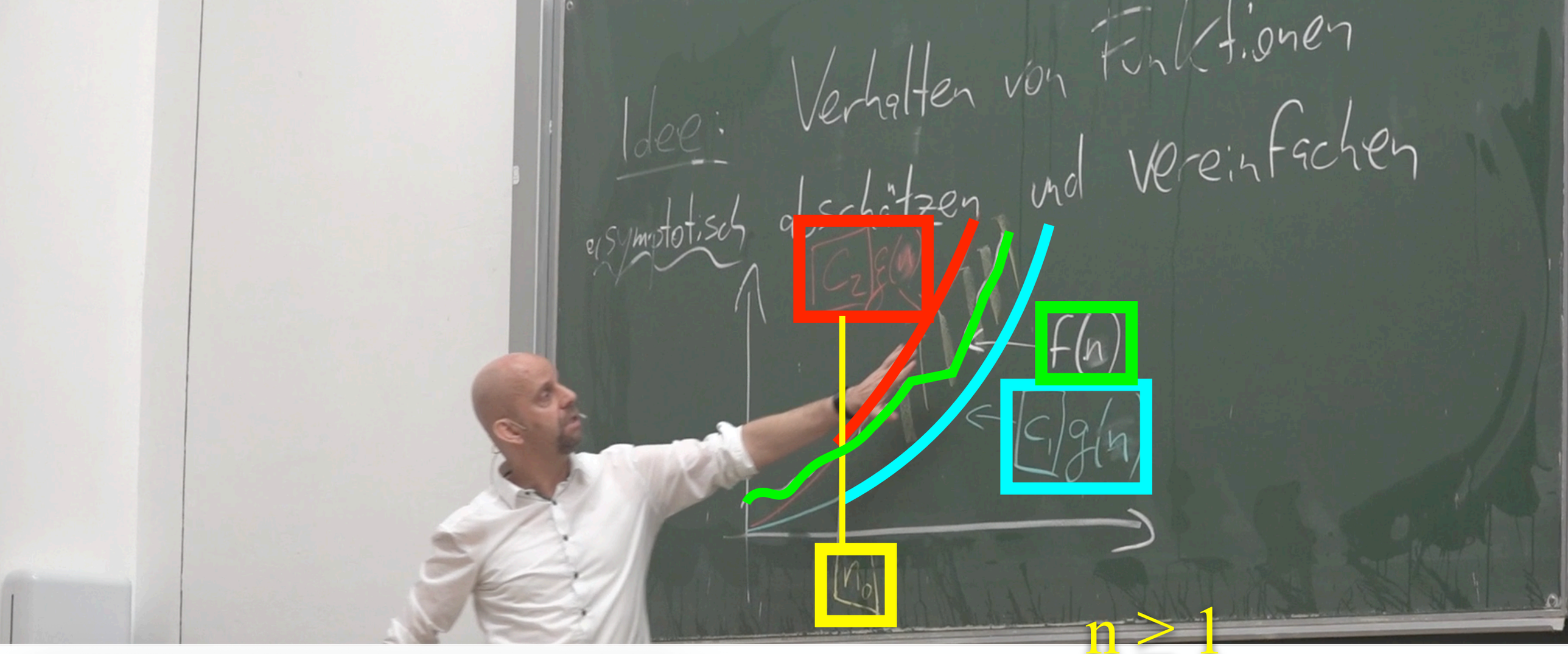
$$c_1 \leq \frac{f(n)}{g(n)} \leq c_2$$

$$5 \leq \frac{f(n)}{g(n)} = 5 + \frac{4}{n^2} + \frac{7}{n^3} + \frac{27}{n^5} - \frac{9}{n^6} \leq 43$$

Frage 7:

Die Funktion $f(n) := 5n^6 + 4n^4 + 7n^3 + 27n - 9$ liegt...

- ... nur in $O(n^6)$.
- ... nur in $\Omega(n^6)$.
- ... in $\Theta(n^6)$.



$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

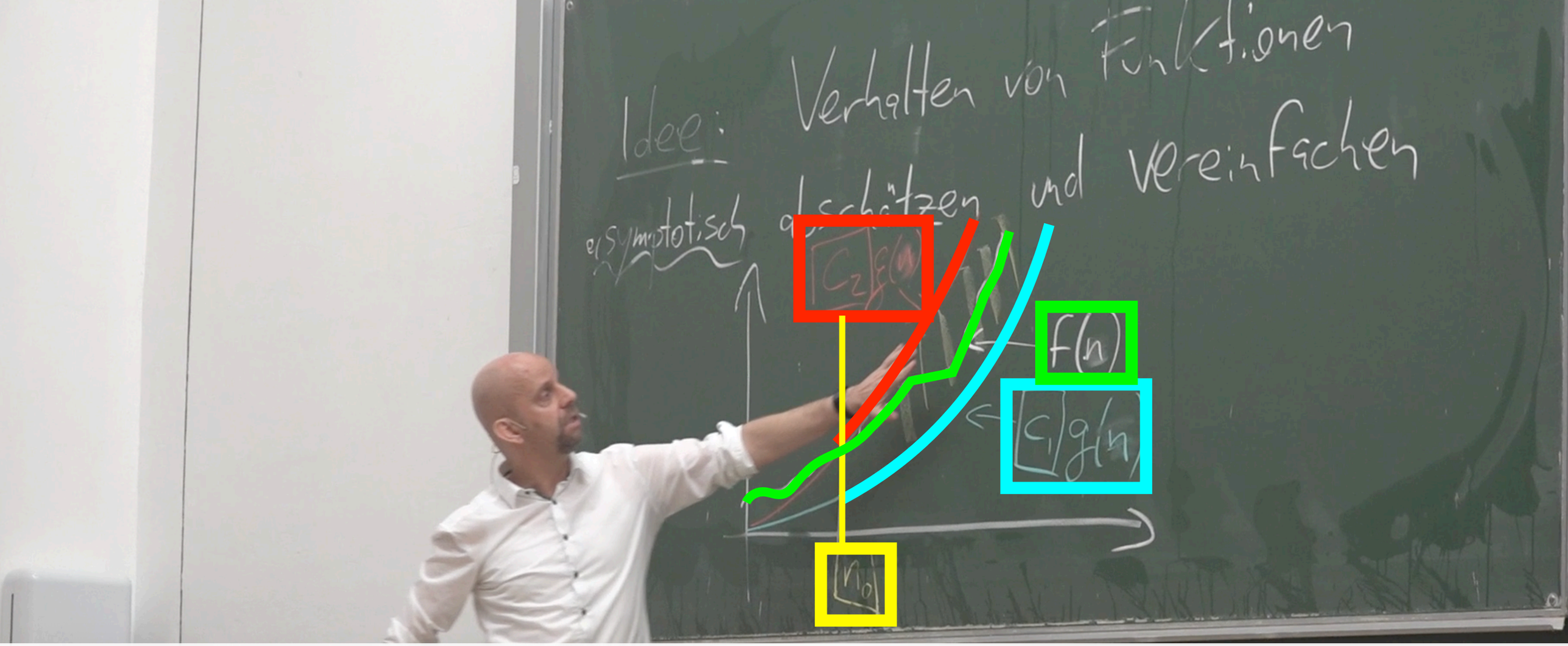
$$c_1 \leq \frac{f(n)}{g(n)} \leq c_2$$

$$5 \leq \frac{f(n)}{g(n)} = 5 + \frac{4}{n^2} + \frac{7}{n^3} + \frac{27}{n^5} - \frac{9}{n^6} \leq 43$$

Frage 7:

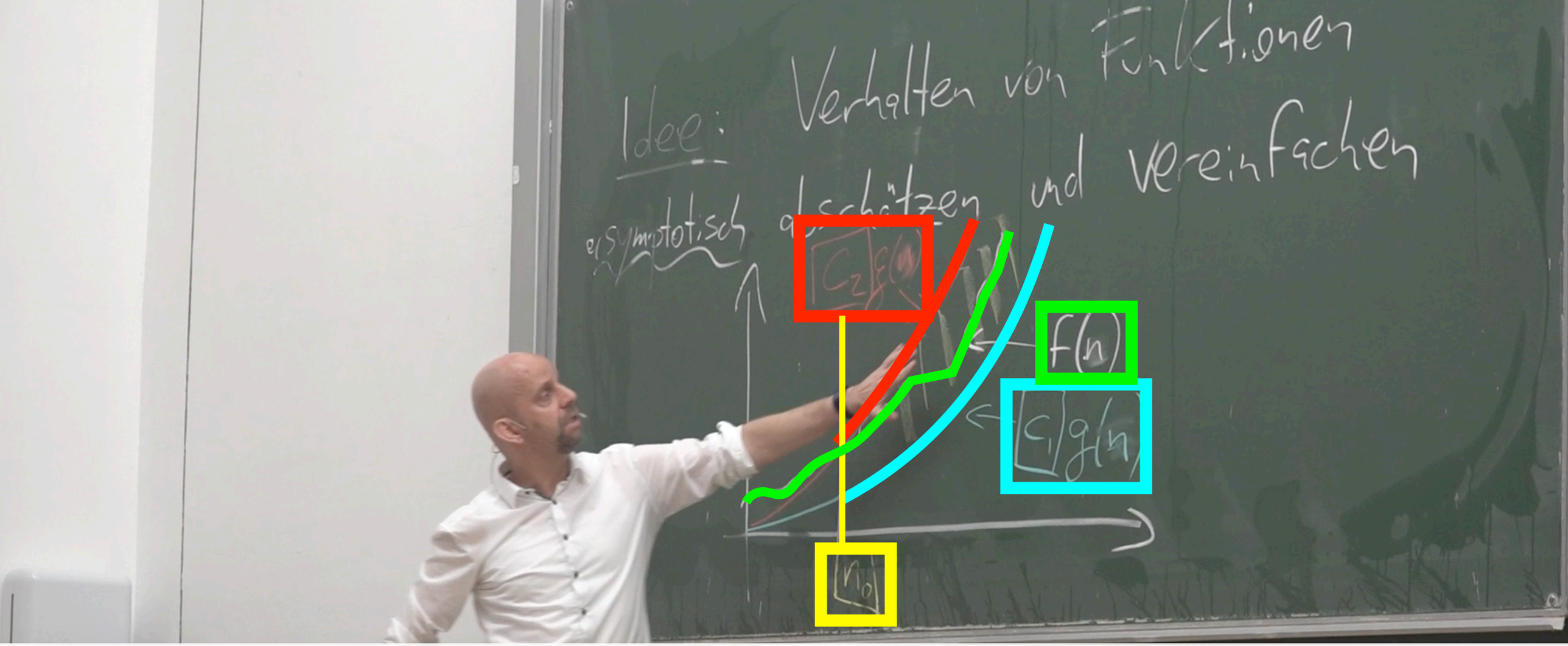
Die Funktion $f(n) := 5n^6 + 4n^4 + 7n^3 + 27n - 9$ liegt...

- ... nur in $O(n^6)$.
- ... nur in $\Omega(n^6)$.
- ... in $\Theta(n^6)$. ✓



$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$$c_1 \leq \frac{f(n)}{g(n)} \leq c_2$$

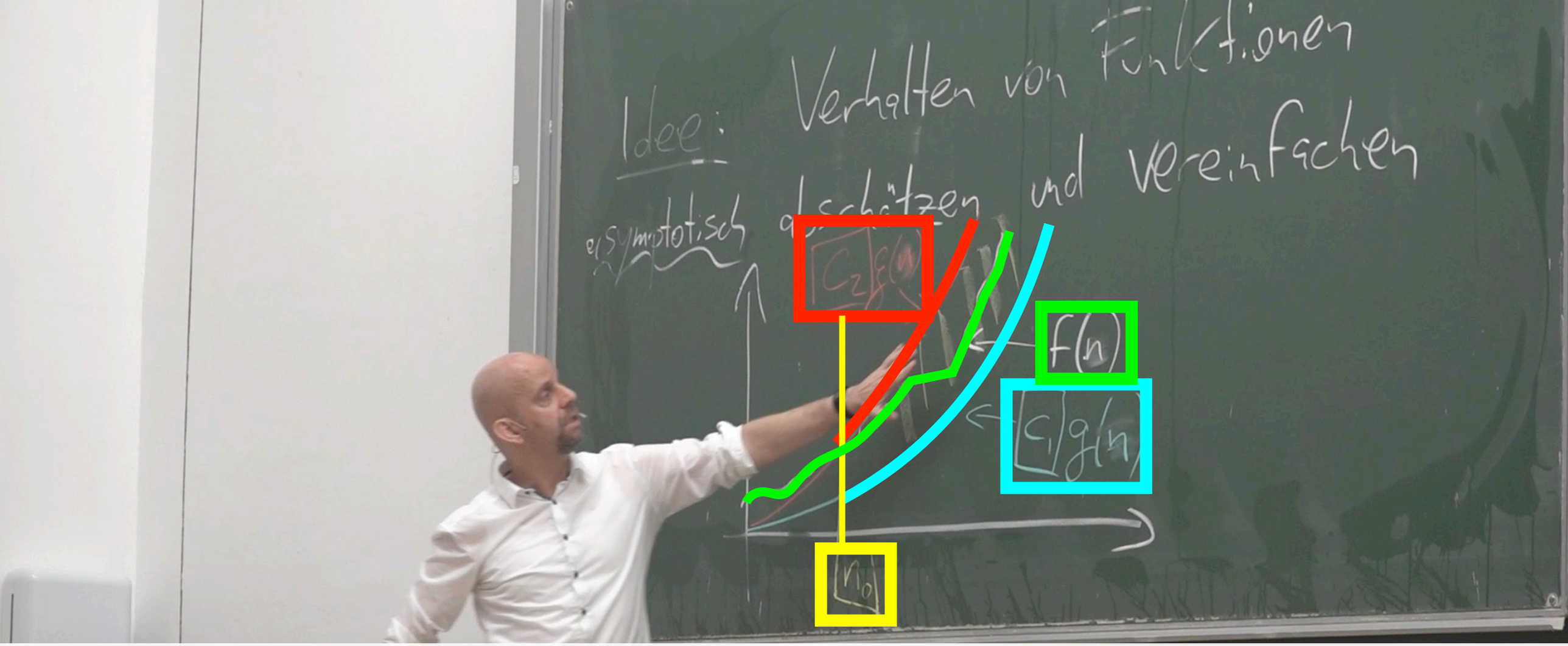


$$c_1 \leq \frac{f(n)}{g(n)} \leq c_2$$

Frage 8:

Die Funktion $f(n) := 2^n$ liegt...

- ... nur in $O(3^n)$.
- ... nur in $\Omega(3^n)$.
- ... in $\Theta(3^n)$.



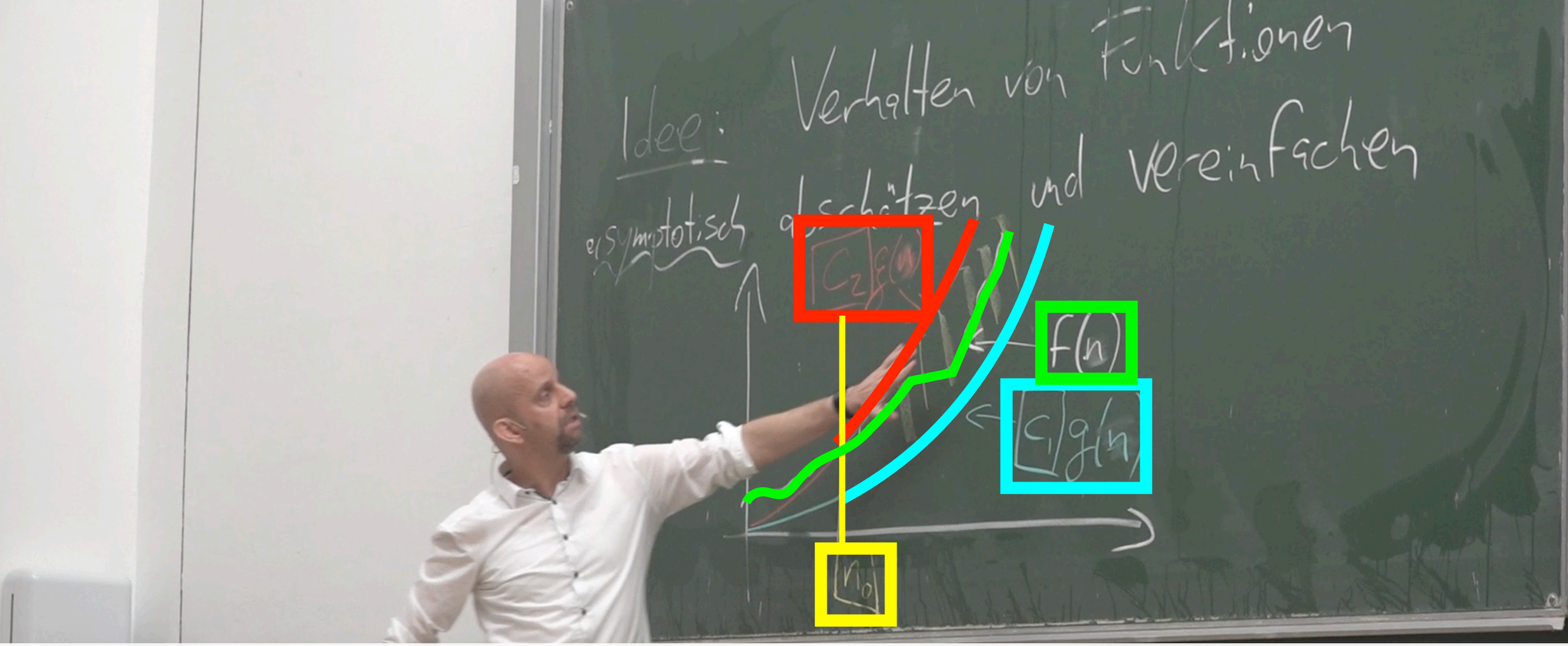
$$c_1 \leq \frac{f(n)}{g(n)} \leq c_2$$

Frage 8:

Die Funktion $f(n) := 2^n$ liegt...

- ... nur in $O(3^n)$.
- ... nur in $\Omega(3^n)$.
- ... in $\Theta(3^n)$.

$$\frac{f(n)}{g(n)} = \frac{2^n}{3^n} = \left(\frac{2}{3}\right)^n$$



$$c_1 \leq \frac{f(n)}{g(n)} \leq c_2$$

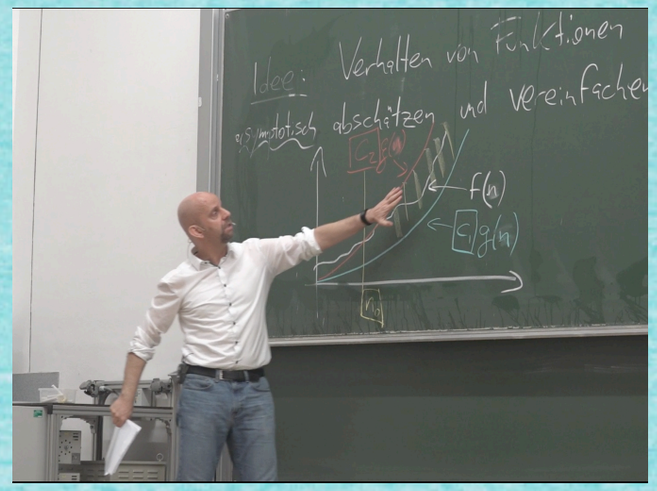
Frage 8:

Die Funktion $f(n) := 2^n$ liegt...

- ... nur in $O(3^n)$. ✓
- ... nur in $\Omega(3^n)$.
- ... in $\Theta(3^n)$.

$$\frac{f(n)}{g(n)} = \frac{2^n}{3^n} = \left(\frac{2}{3}\right)^n$$

Wofür ist das gut?

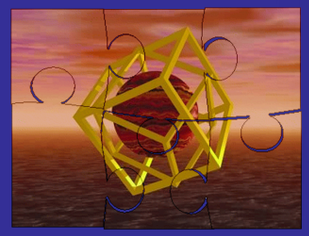


$$\log_2\left\lfloor\left(2n + 4m + n\left(\lfloor\log_2 n\rfloor + 1\right) + 2m\left(\lfloor\log_2 n\rfloor + 1\right)\right)\right\rfloor + 1$$

$$\Theta(m \log n)$$

Ein algorithmisches Problem

Gegeben: n Puzzleteile

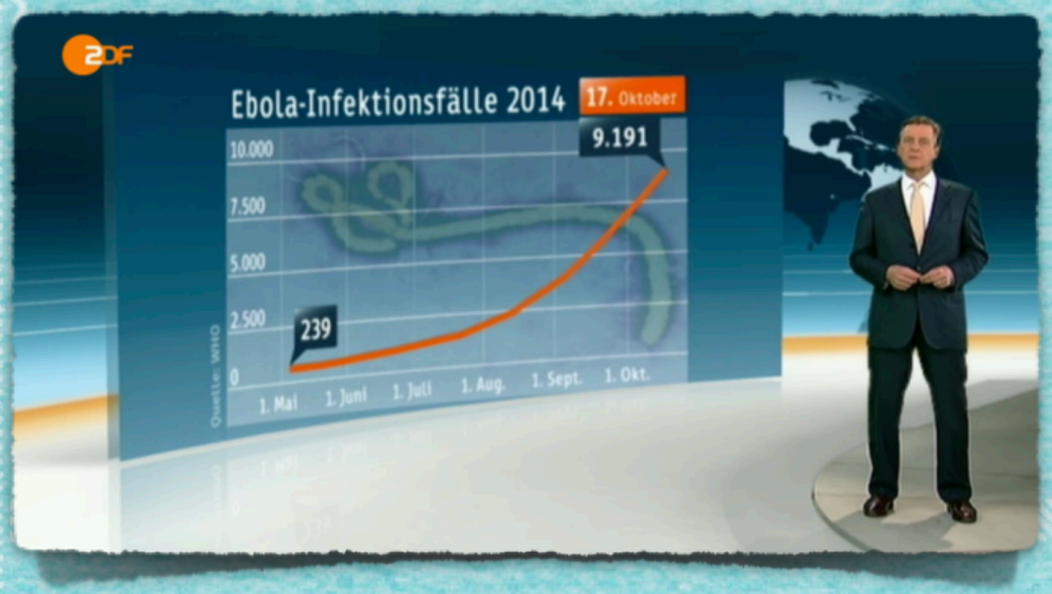


Einfach so: $O(n^2)$

Für n=6:	21
Für n=100:	5.050
Für n=5000:	12.502.500

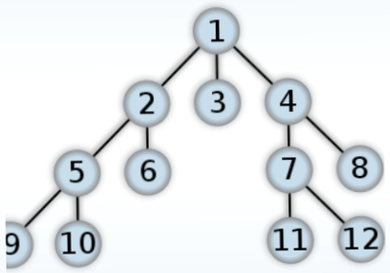
Raffiniert sortiert: $O(n \log n)$

Gesucht: Eine systematische Methode zum Puzzeln



$$\Theta(n^2) \qquad (2n)^2 = 4 \cdot n^2$$

$$\Theta(2^n) \qquad 2^{2n} = 2^n \cdot 2^n$$

Kapitel 3.8: Laufzeit von DFS und BFS

Algorithmen und Datenstrukturen
WS 2022/23

Prof. Dr. Sándor Fekete

Algorithmus 3.7

INPUT: Graph G = (V,E), Knoten s

OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

Sei R:={s}, Y:={s}, T:=∅

2. WHILE (R≠∅) DO {

    2.1. Wähle v∈ R

    2.2. IF (es gibt kein w∈ V\Y mit e={v,w}∈E) THEN

        2.2.1. R:=R\{v}

    2.3. ELSE {

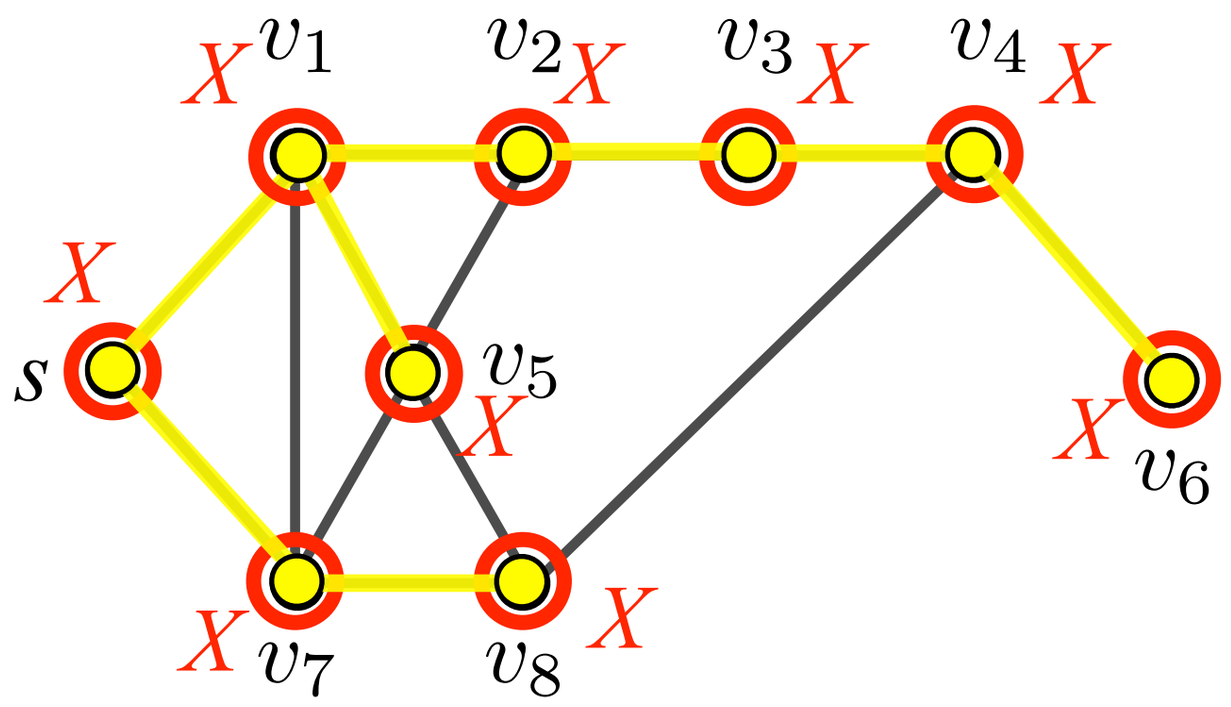
        2.3.1.Wähle ein w∈V\Y mit e={v,w} ∈ E

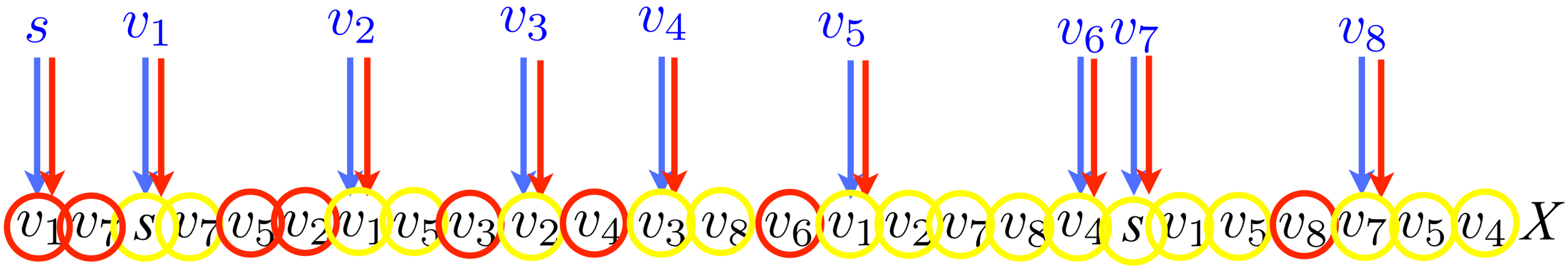
        2.3.2. Setze R := R u {w}, Y := Y u {w}, T := T u {e}

        }

}

3. STOP





Satz 3.13

Der Graphen-Scan-Algorithmus 3.7 lässt sich so implementieren, dass die Laufzeit $O(n+m)$ ist.

Algorithmus 3.7

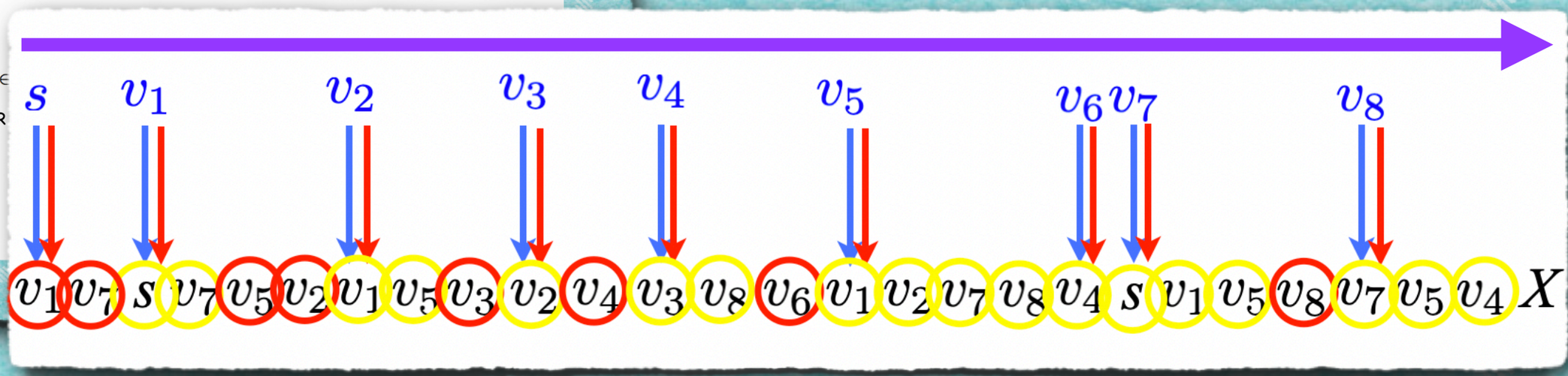
INPUT: Graph G = (V,E), Knoten s

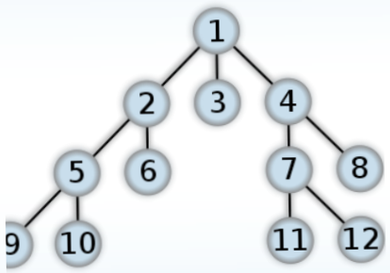
OUTPUT: Knotenmenge $Y \subseteq V$, die von s aus erreichbar ist,

Kantenmenge $T \subseteq E$, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈ V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈
     - 2.3.2. Setze R := R
     - }
   - }
3. STOP

Adjazenzliste!





Kapitel 3.9: Eigenschaften von DFS und BFS

Algorithmen und Datenstrukturen
WS 2022/23

Prof. Dr. Sándor Fekete

3.9 DFS vs. BFS

Einfach gesagt:

- ***DFS ist eine bestmögliche individuelle Suchstrategie mit lokaler Information.***
- ***BFS ist eine bestmögliche kooperative Suchstrategie mit globaler Information.***

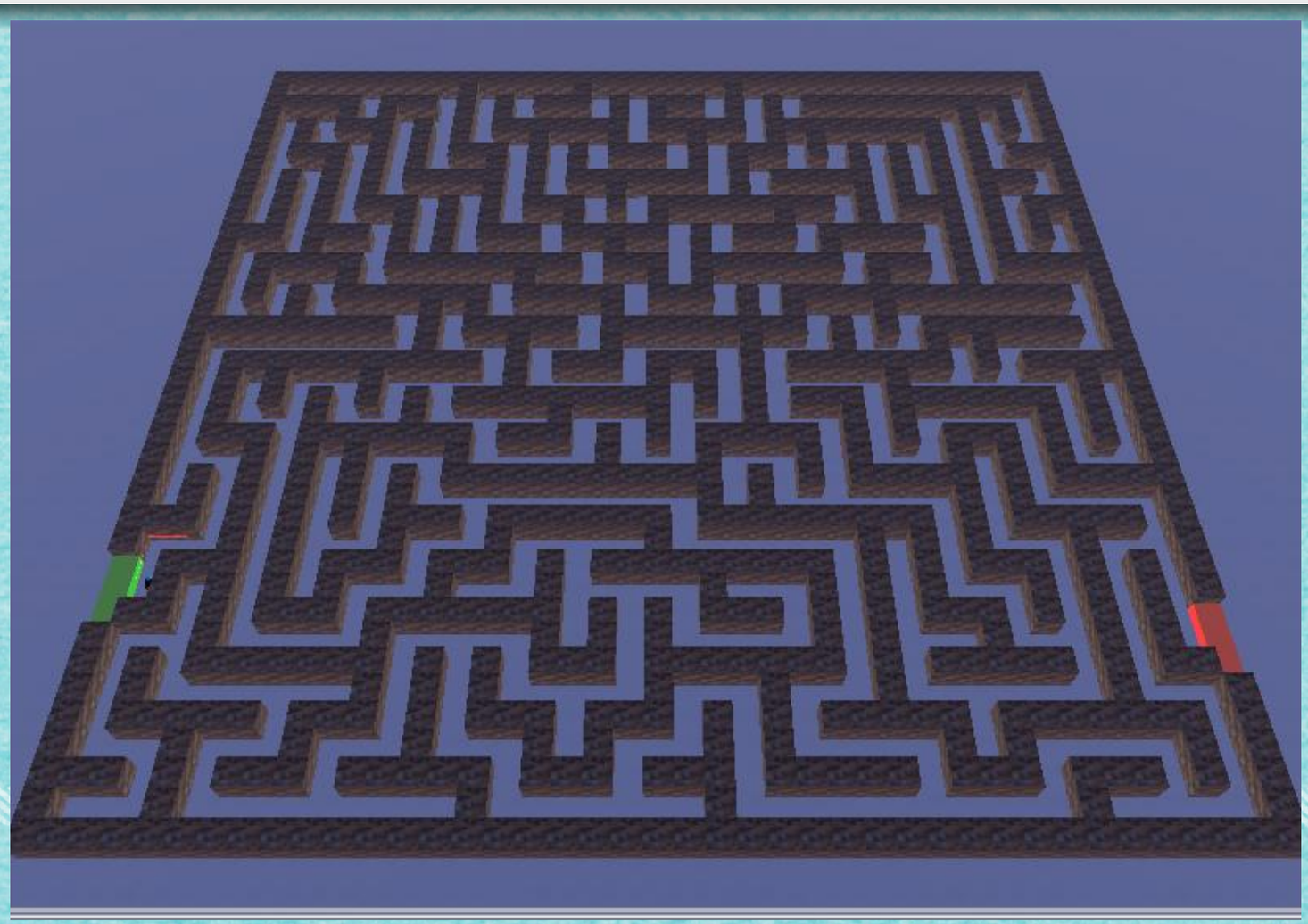
Konkret:

- ***DFS ist gut geeignet für die Suche nach einem Ausweg aus einem Labyrinth.***
- ***BFS ist gut geeignet für die Suche nach kürzesten Wegen in einem Graphen.***

Satz 3.16 (Lokale Suche mit DFS)

(1) DFS findet in jedem zusammenhängenden Graphen mit n Knoten einen Weg der Länge höchstens 2n-1, der alle Knoten besucht.

(2) Für jede lokale Suchstrategie gibt es einen Graphen mit n Knoten, so dass der letzte Knoten erst nach einer Weglänge von 2n-1 besucht wird.



Auf die Schnelle mit der Welle

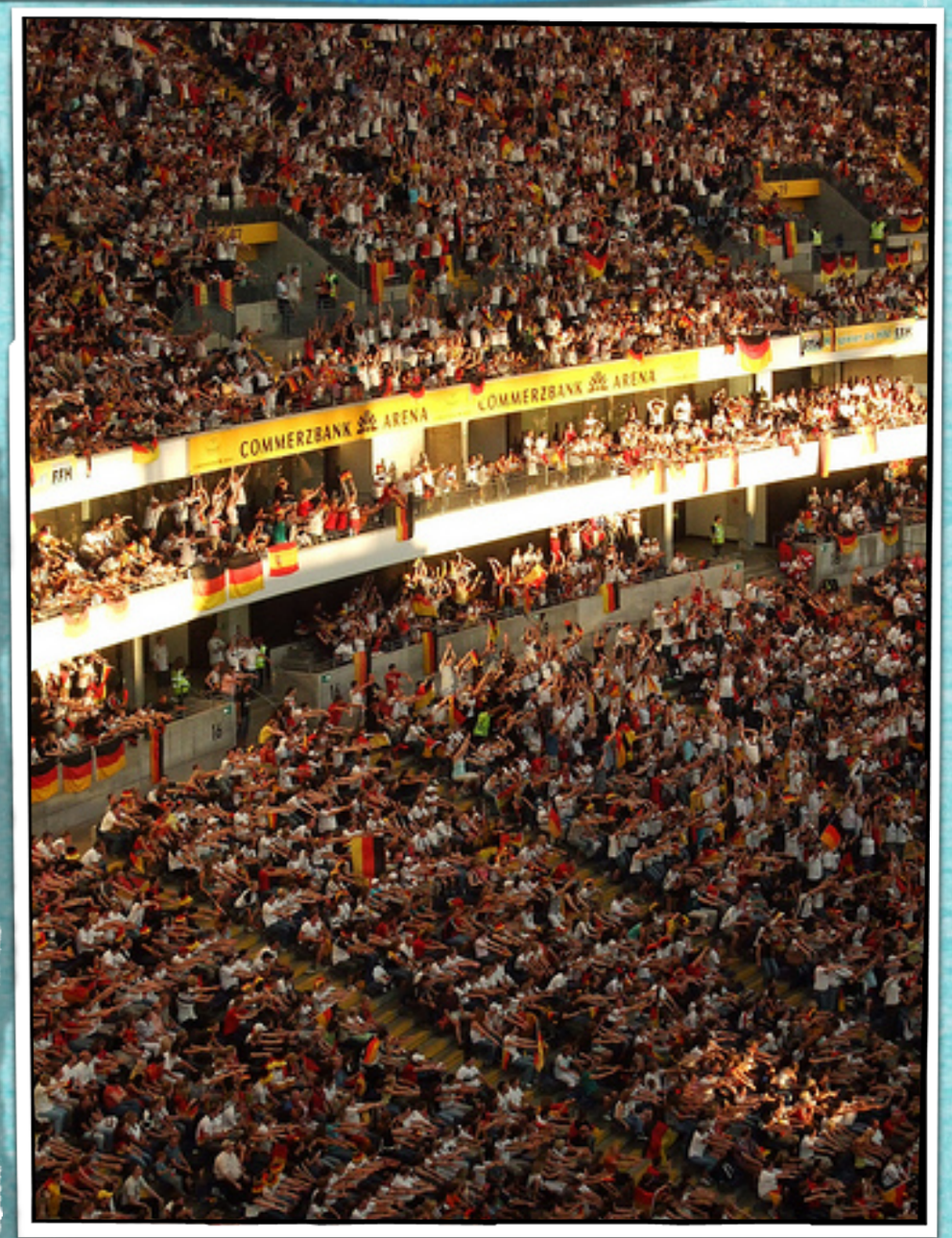
A. LOS bei „NULL“

B. Bis „ANGEKOMMEN!“:

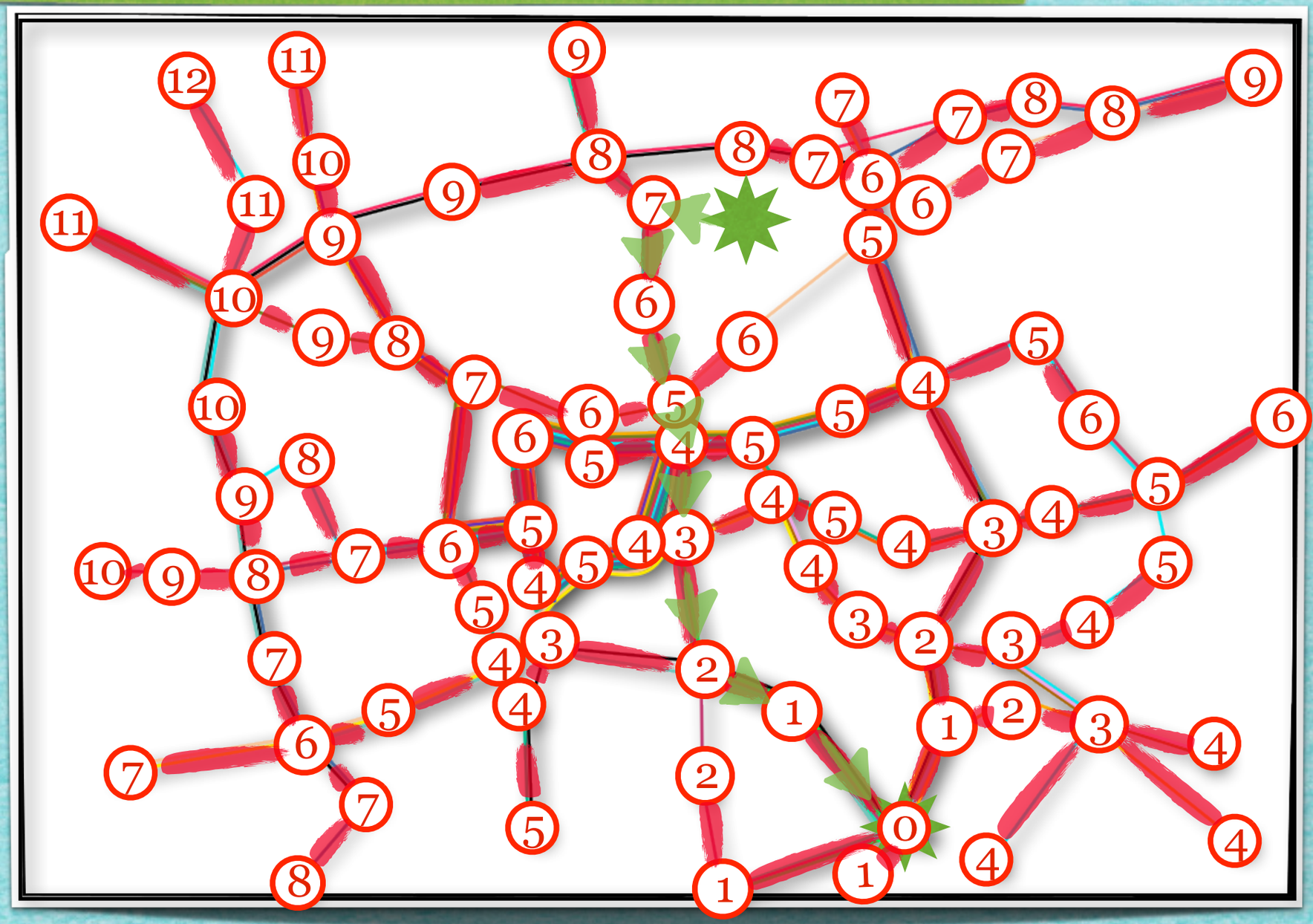
- Solange du noch nicht aufgestanden warst:
  - Wenn ein oder mehrere direkte Nachbarn aufstehen:
    1. Einen dieser Nachbarn merken
    2. In der nächsten Runde:
       2.1. aufstehen
       2.2. Zahl merken
    3. In der übernächsten Runde hinsetzen

C. Nach „ANGEKOMMEN!“:

- Auf gemerkten Nachbarn zeigen



Wellenreiten in Graphen

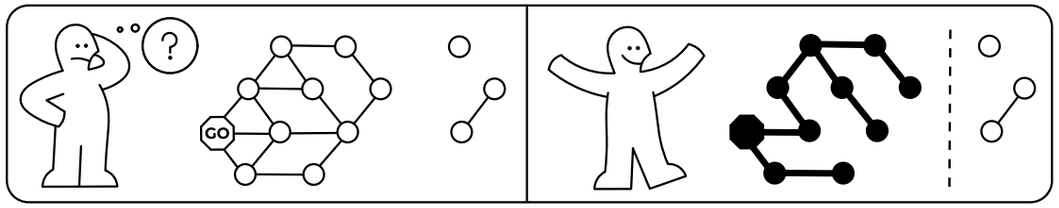


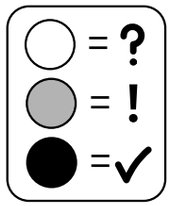
Breitensuche

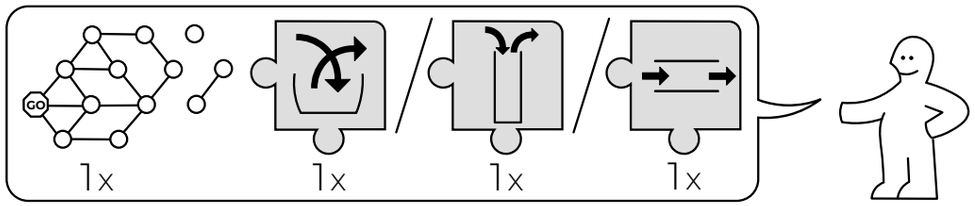
GRÅPH SKÄN

idea-instructions.com/graph-scan/
v1.3, CC by-nc-sa 4.0

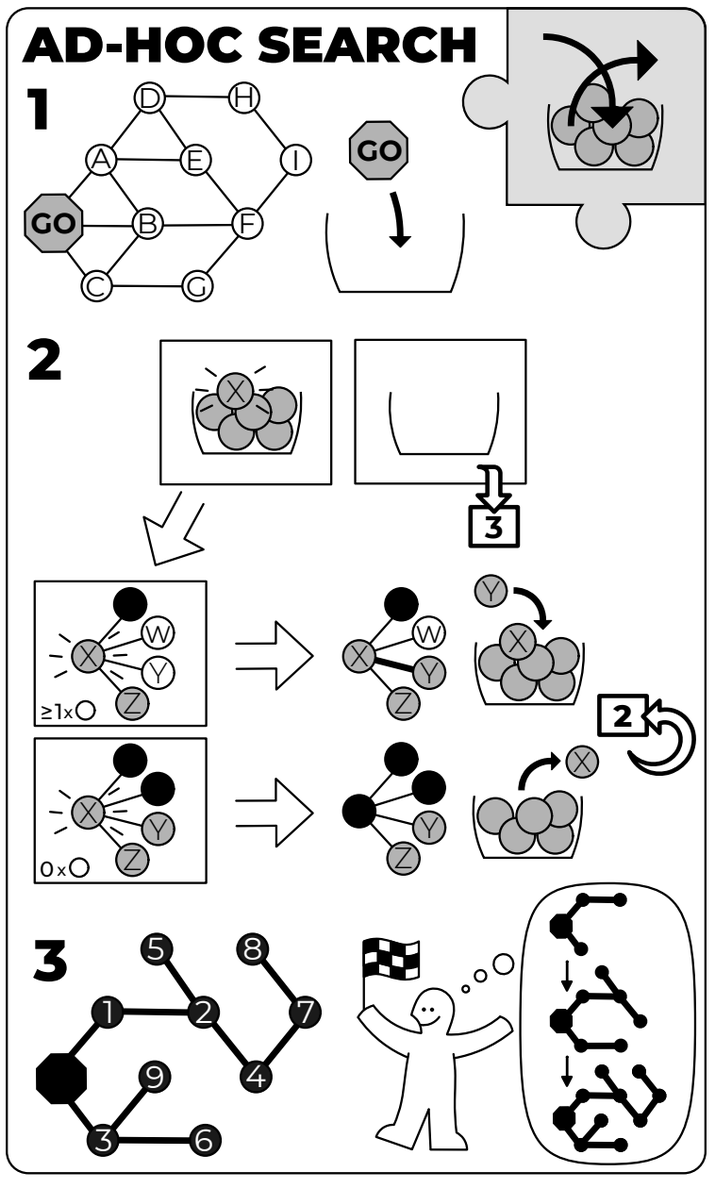
IDEA



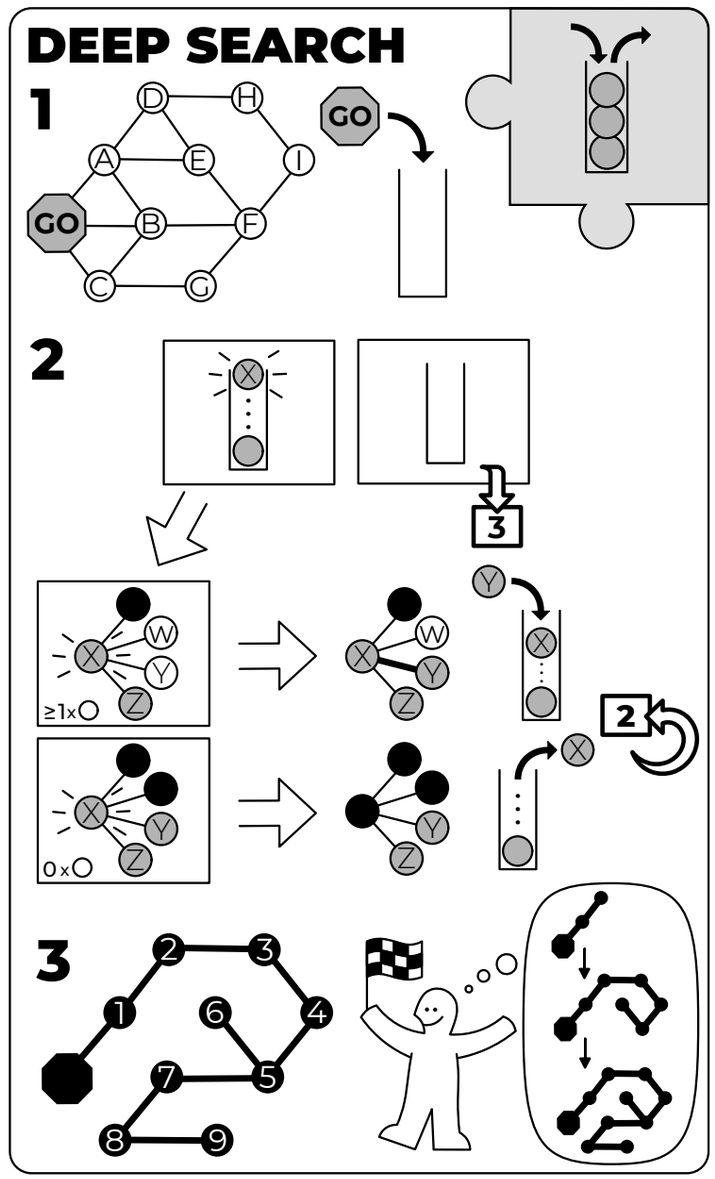




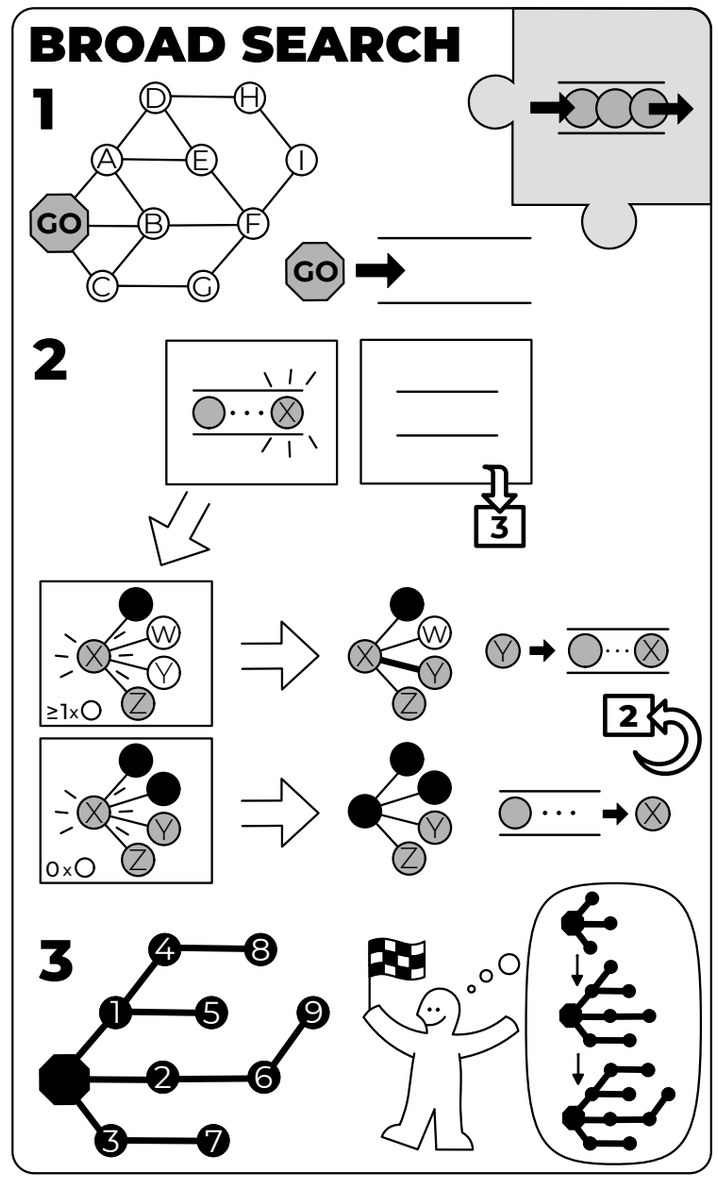
AD-HOC SEARCH



DEEP SEARCH



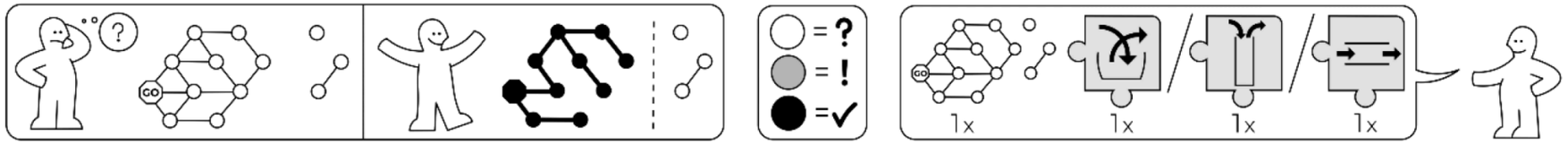
BROAD SEARCH

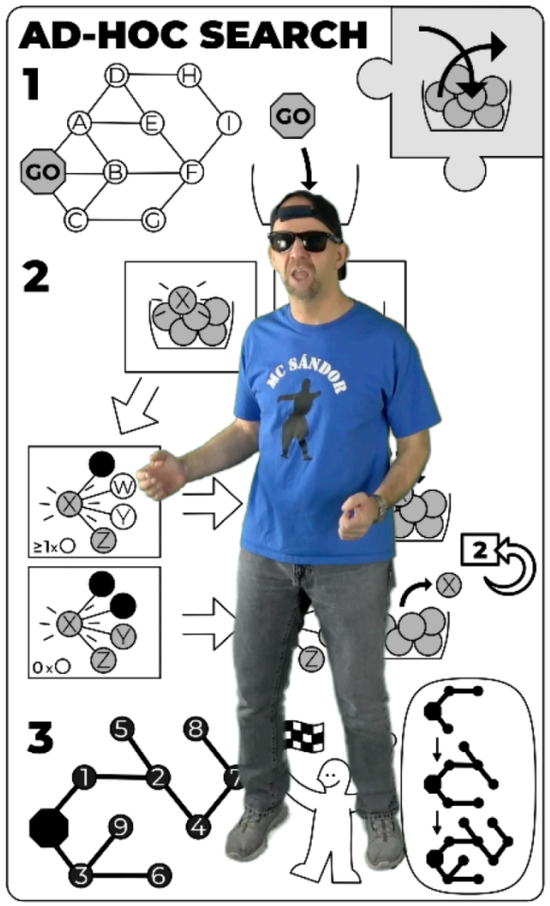


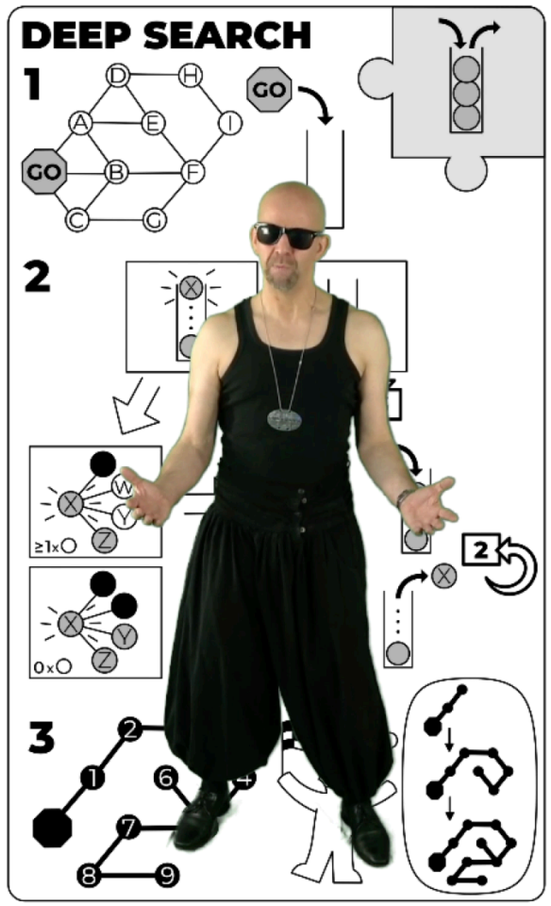
Zusammenfassung Kapitel 3!

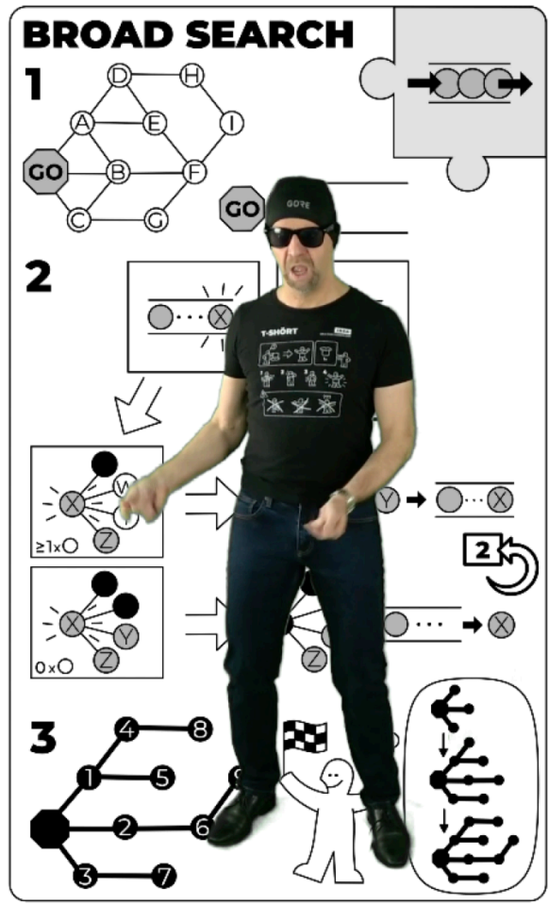
GRÅPH SKÄN

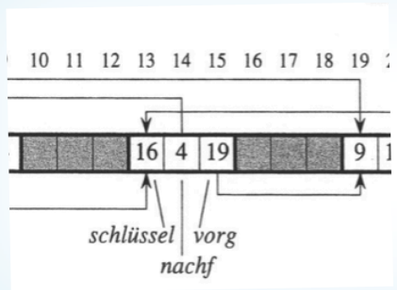
idea-instructions.com/graph-scan/
v1.3, CC by-nc-sa 4.0 IDEA

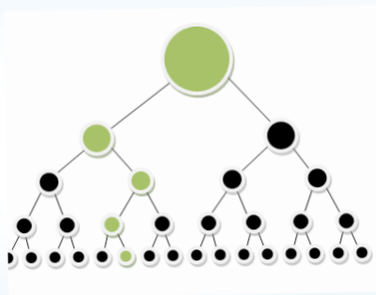













Kapitel 4: Dynamische Datenstrukturen

Algorithmen und Datenstrukturen
WS 2022/23

Prof. Dr. Sándor Fekete

4.1 Grundoperationen

Langsam:

- ***O(n):*** ***lineare Zeit***

  Alle Objekte anschauen

Sehr schnell:

- ***O(1):*** ***konstante Zeit***

  Immer gleich schnell, egal wie groß S ist.

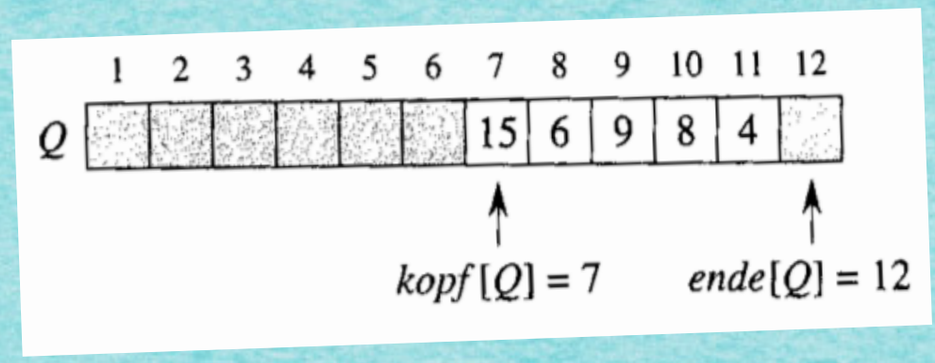
Schnell:

- ***O(log n): logarithmische Zeit***

  Wiederholtes Halbieren

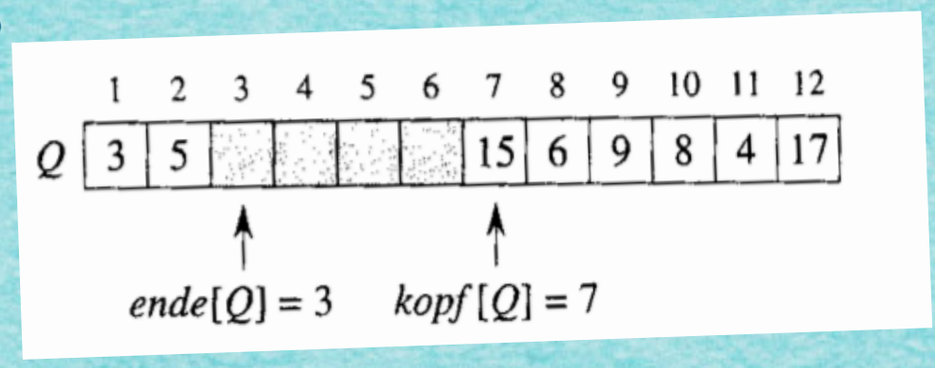
4.2 Stapel und Warteschlange

OUT

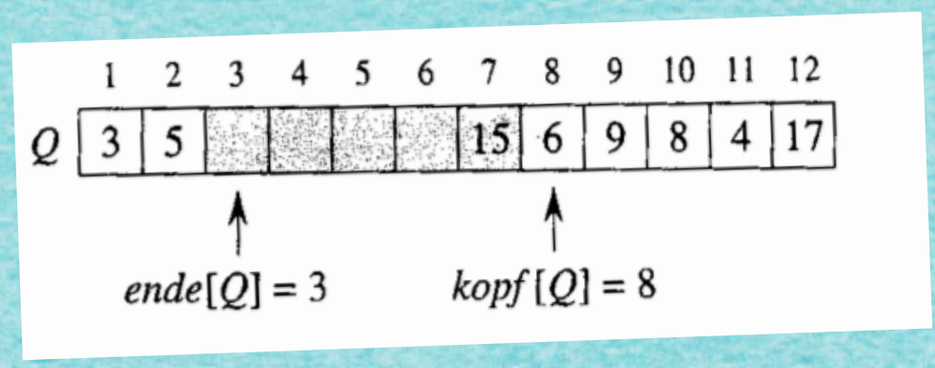


IN

Enqueue: 17, 3, 5



Dequeue:



4.3 Verkettete Listen

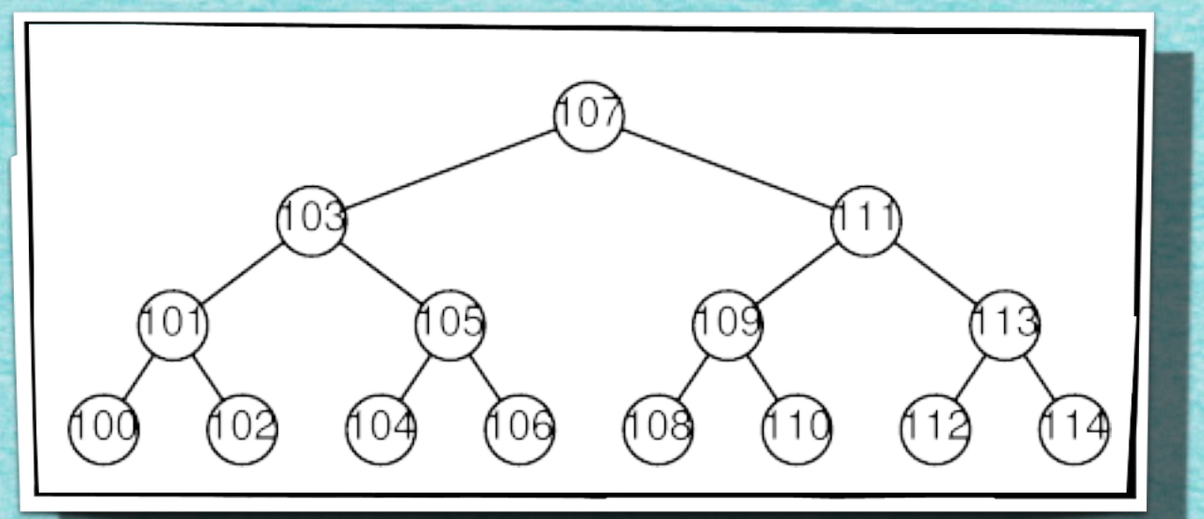
Idee:

Ordne Objekte nicht explizit in aufeinanderfolgenden Speicherzellen an, sondern gib jeweils Vorgänger und Nachfolger an.

4.4 Binäre Suche

<u>Aufgabenstellung:</u>

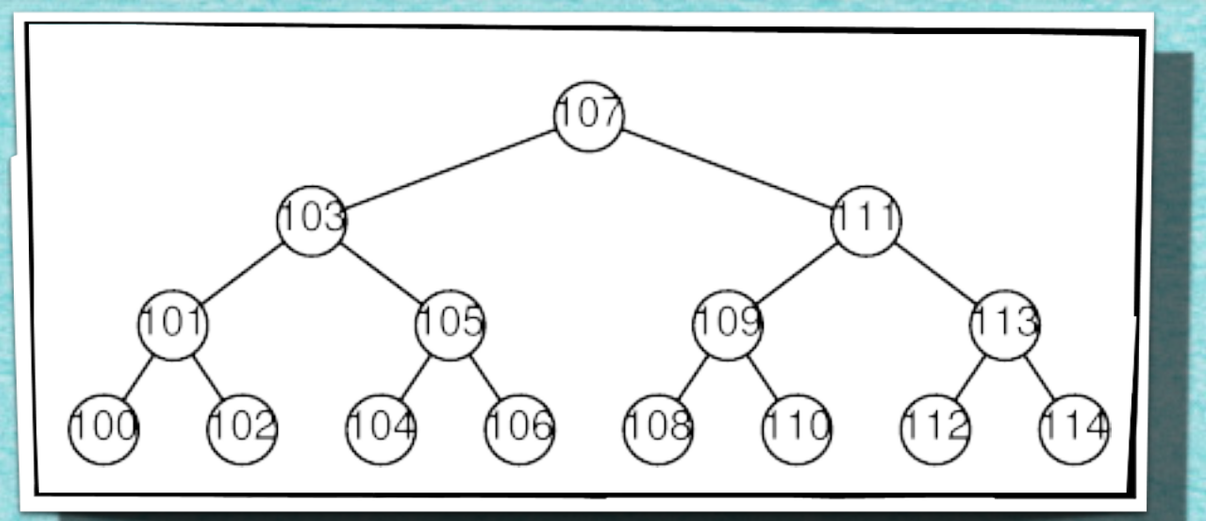
- ***Rate eine Zahl zwischen 100 und 114!***



4.5 Binäre Suchbäume

Ideen:

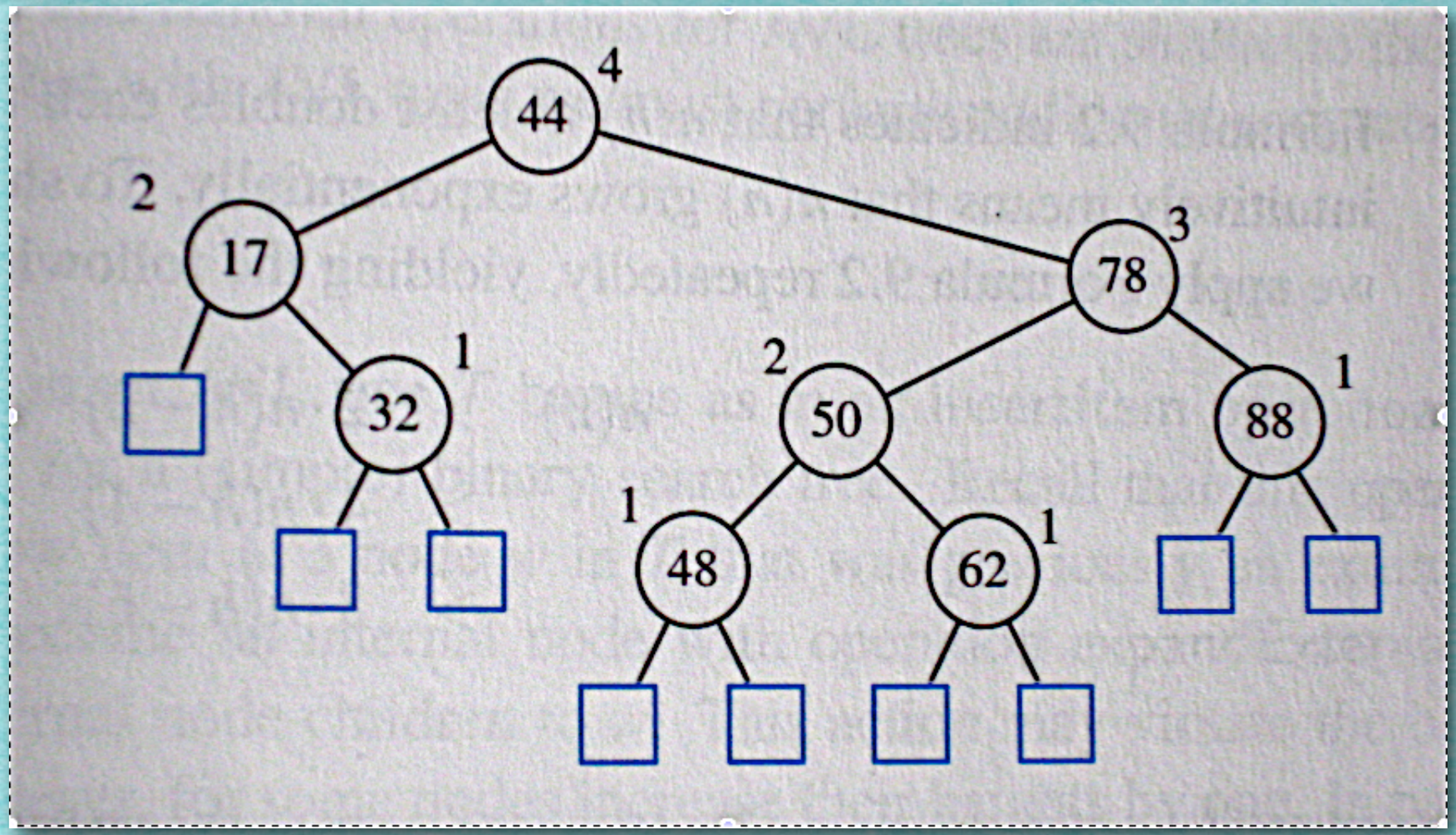
- ***Strukturiere Daten wie im möglichen Ablauf einer binären Suche!***
- ***Erziele logarithmische Zeiten!***

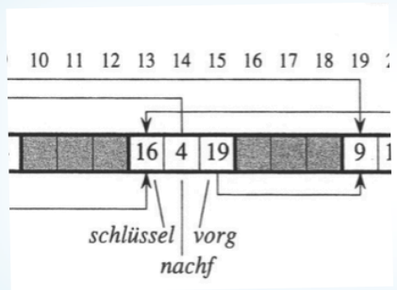


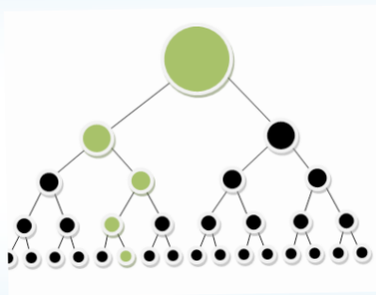
Definition 4.7

(1) Ein binärer Suchbaum ist höhenbalanciert, wenn sich für jeden inneren Konten v die Höhe der beiden Kinder von v um höchstens 1 unterscheidet.

(2) Ein höhenbalancierter Suchbaum heißt auch AVL-Baum.





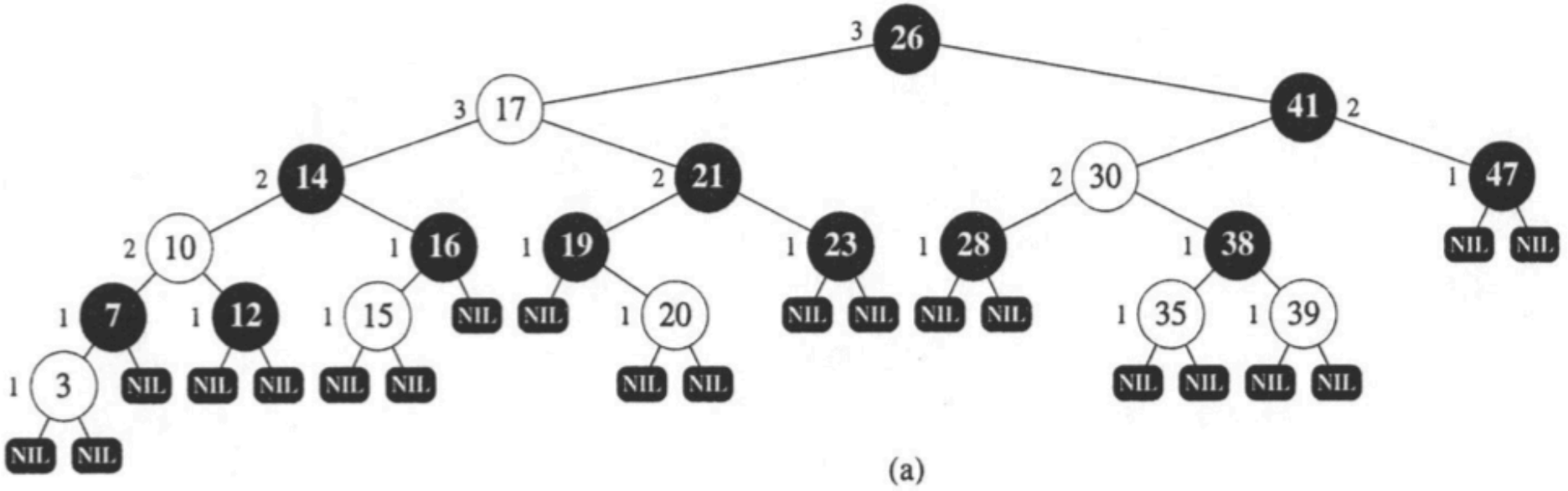



Kapitel 4.8-4.11: Andere dynamische Datenstrukturen

Algorithmen und Datenstrukturen
WS 2021/22

Prof. Dr. Sándor Fekete

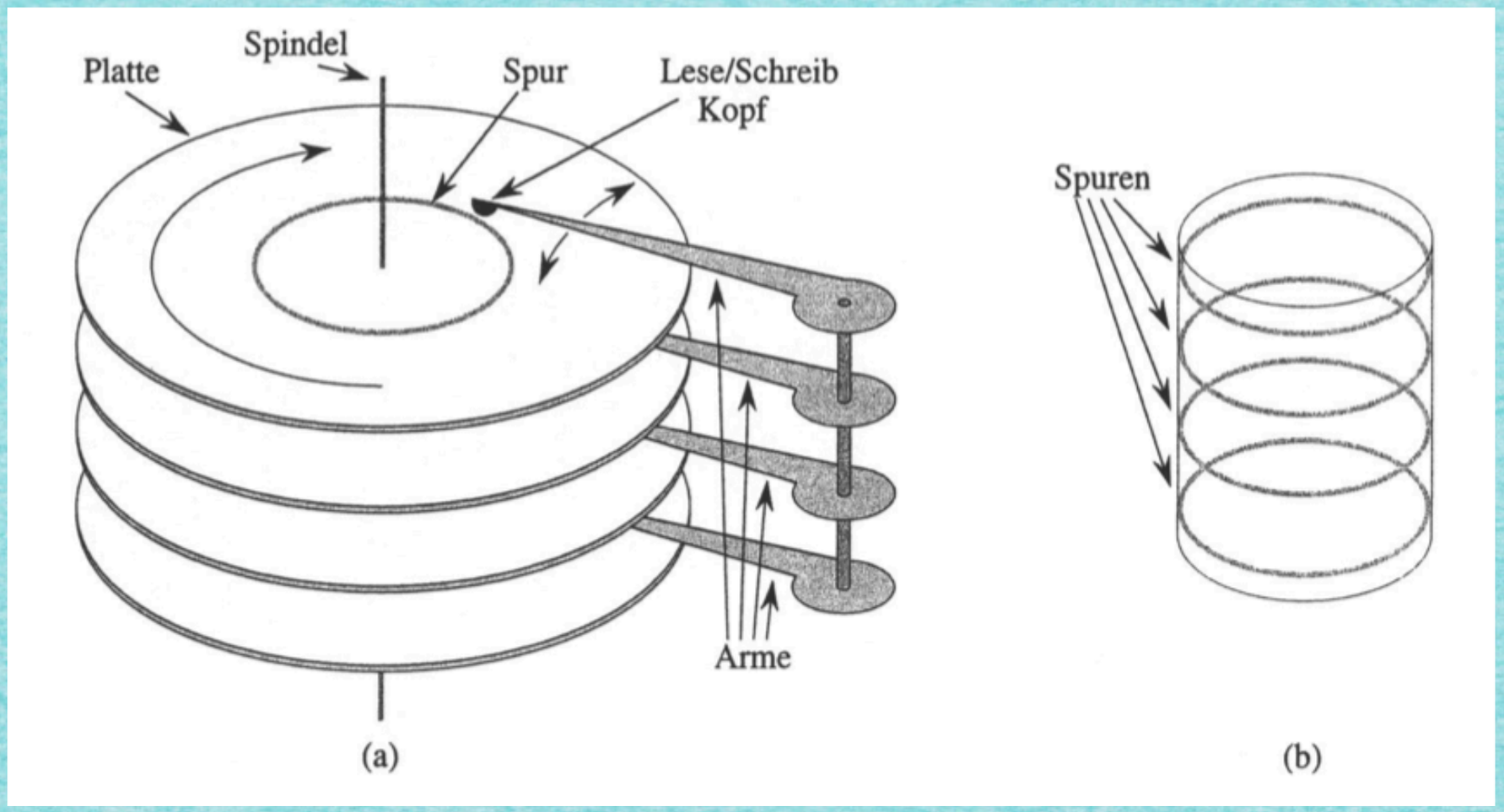
4.8 Rot-Schwarz-Bäume



(a)

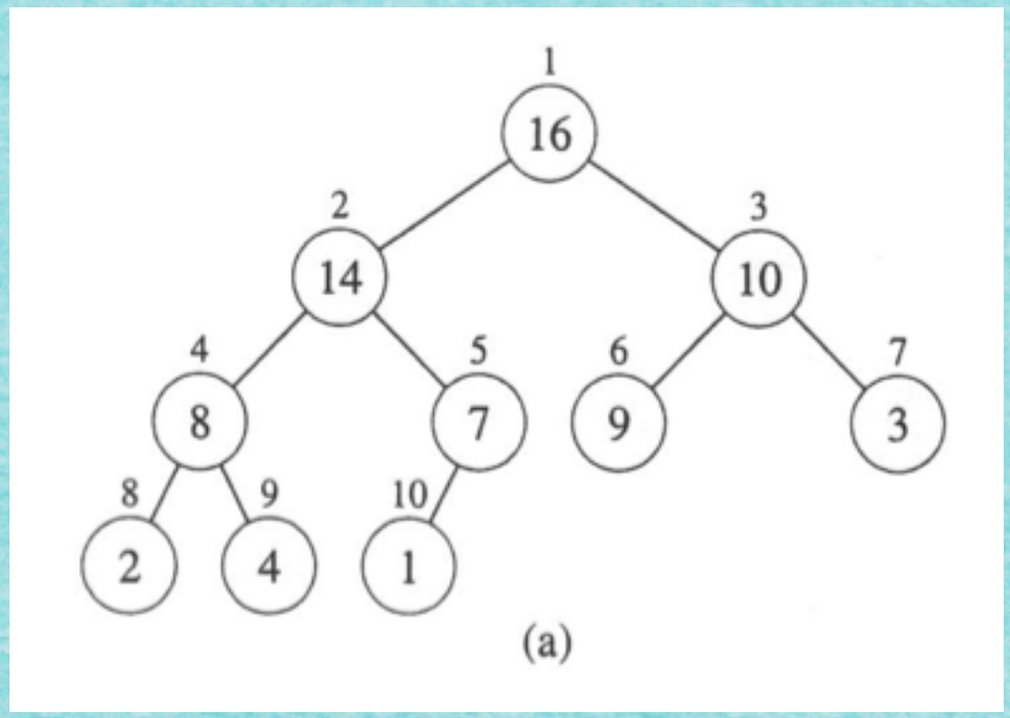
Idee: Verwende "Farben", um den Baum vertikal zu strukturieren und zu balancieren!

4.9 B-Bäume



Kontext: Speicherhierarchien mit unterschiedlich schnellen Zugriffszeiten

4.10 Heaps



(a)

Idee: Ordne Baum so, dass
größere Elemente immer oben stehen!

4.11 Andere Strukturen

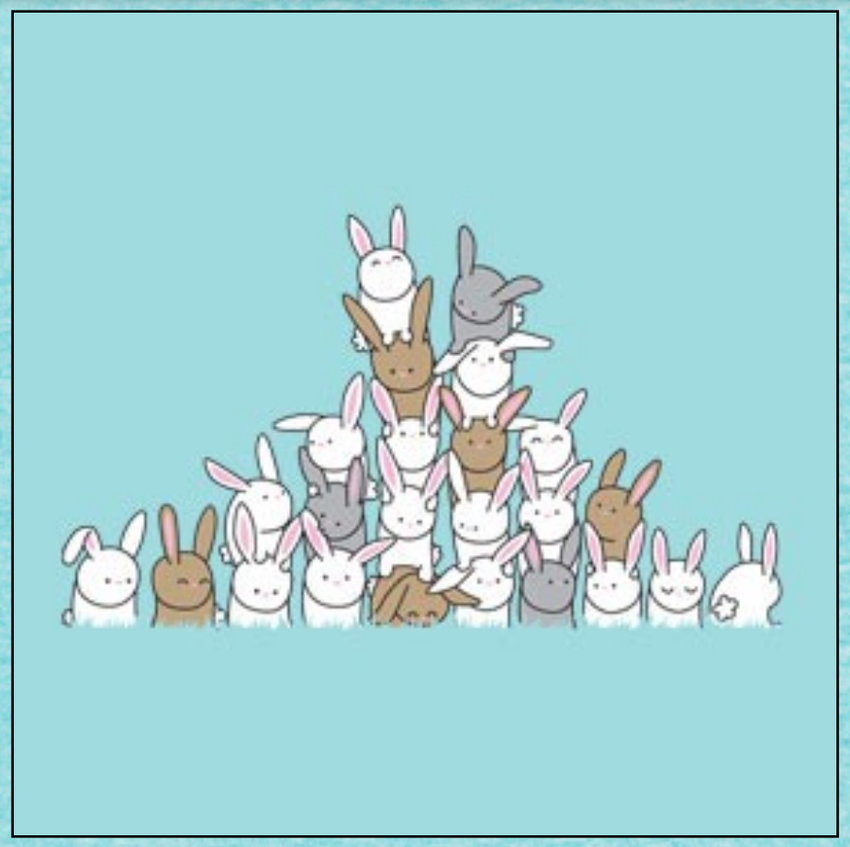
4.11.1 Fibonacci-Heaps: Heap-Struktur mit sehr schneller *amortisierter* Zugriffszeit.

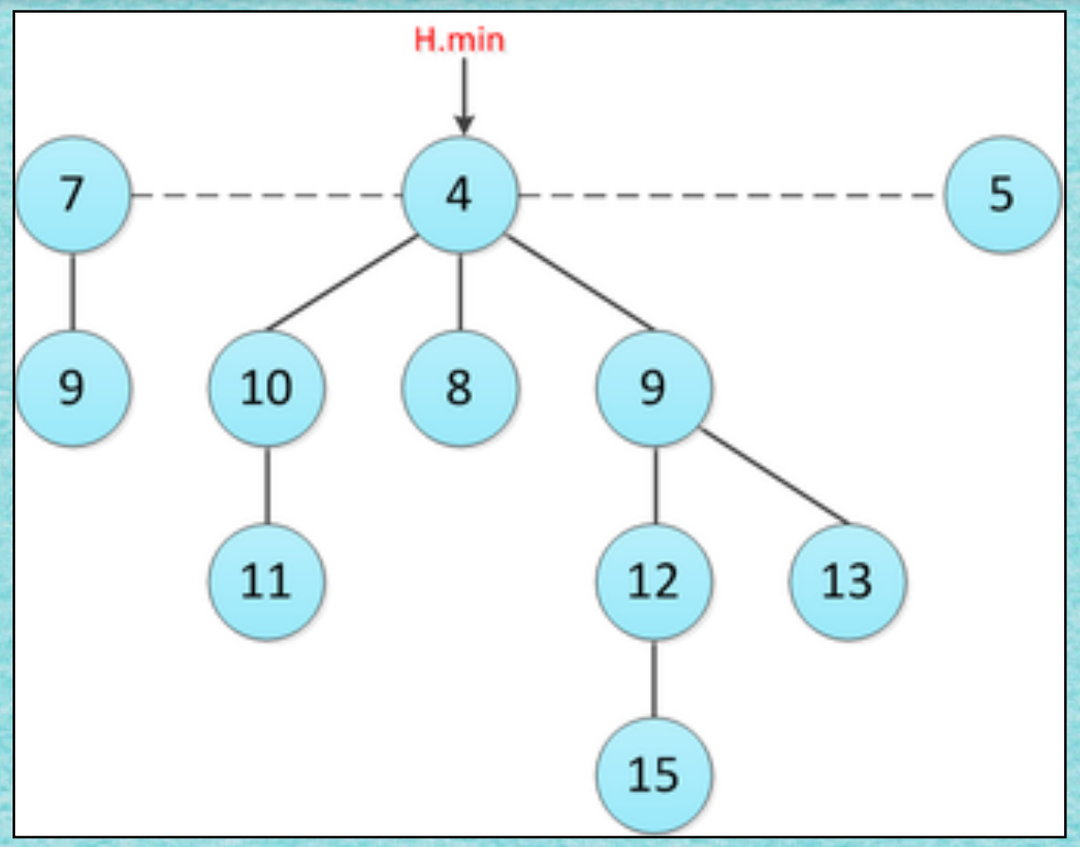
Common Operations	*Effect*	Unsorted Linked List	Self-balancing binary search tree	Binary heap	Binomial heap	Fibonacci heap
insert(data,key)	Adds *data* to the queue, tagged with *key*	O(1)	O(log n)	O(log n)	O(log n)	O(1)
findMin() -> key,data	Returns *key,data* corresponding to min-value *key*	O(n)	O(log n) or O(1) (**)	O(1)	O(log n)[4]	O(1)[1]
deleteMin()	Deletes *data* corresponding to min-value *key*	O(n)	O(log n)	O(log n)	O(log n)	O(log n)*
delete(node)	Deletes *data* corresponding to given *key*, given a pointer to the node being deleted	O(1)	O(log n)	O(log n)	O(log n)	O(log n)*
decreaseKey(node)	Decreases the *key* of a node, given a pointer to the node being modified	O(1)	O(log n)	O(log n)	O(log n)	O(1)*
merge(heap1,heap2) -> heap3	Merges two heaps into a third	O(1)	O(m log(n+m))	O(m + n)	O(log n)***	O(1)

(*)Amortized time
(**)With trivial modification to store an additional pointer to the minimum element
(***)Where n is the size of the larger heap

4.11 Andere Strukturen

4.11.1 Fibonacci-Heaps:
Heap-Struktur mit sehr schneller *amortisierter* Zugriffszeit.





4.11 Andere Strukturen

4.11.2 Cache-Oblivious B-Trees: Umgang mit großen Datenmengen bei unbekannter Cache-Größe

CACHE-OBLIVIOUS B-TREES*

MICHAEL A. BENDER†, ERIK D. DEMAINE‡, AND MARTIN FARACH-COLTON§

Abstract. This paper presents two dynamic search trees attaining near-optimal performance on any hierarchical memory. The data structures are independent of the parameters of the memory hierarchy, e.g., the number of memory levels, the block-transfer size at each level, and the relative speeds of memory levels. The performance is analyzed in terms of the number of memory transfers between two memory levels with an arbitrary block-transfer size of B; this analysis can then be applied to every adjacent pair of levels in a multilevel memory hierarchy. Both search trees match the optimal search bound of $\Theta(1+\log_{B+1} N)$ memory transfers. This bound is also achieved by the classic B-tree data structure on a two-level memory hierarchy with a known block-transfer size B. The first search tree supports insertions and deletions in $\Theta(1+\log_{B+1} N)$ amortized memory transfers, which matches the B-tree's worst-case bounds. The second search tree supports scanning S consecutive elements optimally in $\Theta(1+S/B)$ memory transfers and supports insertions and deletions in $\Theta(1+\log_{B+1} N+\frac{\log^2 N}{B})$ amortized memory transfers, matching the performance of the B-tree for $B=\Omega(\log N \log\log N)$.

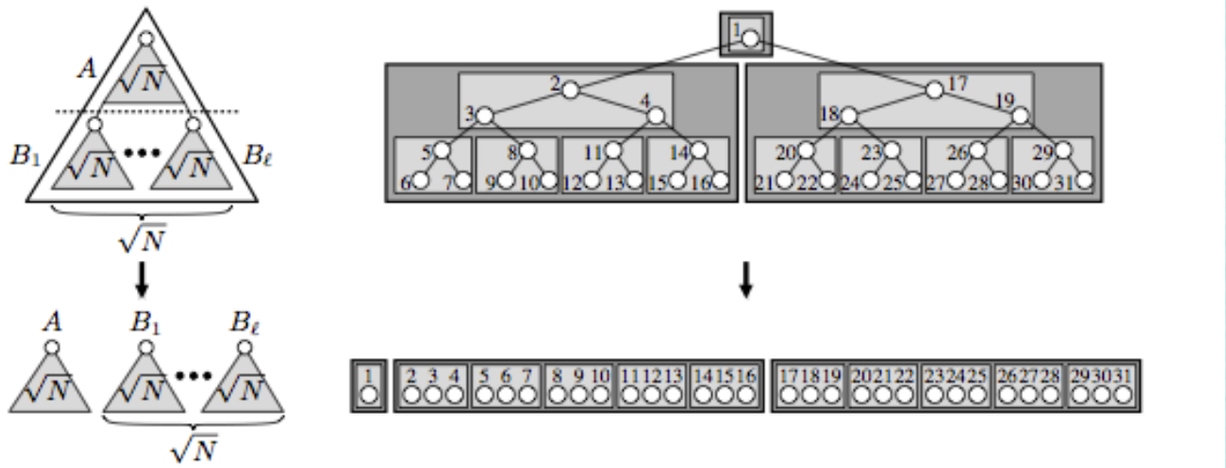
Key words. Memory hierarchy, cache efficiency, data structures, search trees

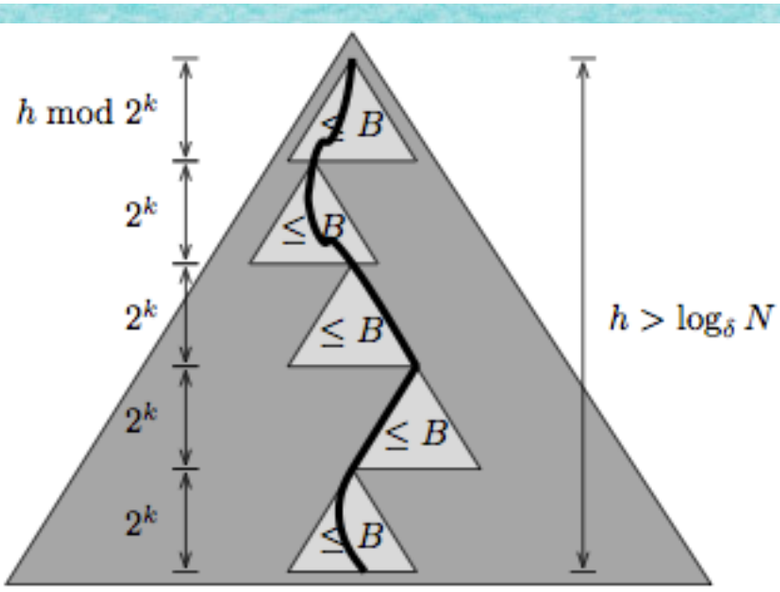
AMS subject classifications. 68P05, 68P30, 68P20

DOI. 10.1137/S0097539701389956

1. Introduction. The memory hierarchies of modern computers are becoming increasingly steep. Typically, an L1 cache access is two orders of magnitude faster than a main memory access and six orders of magnitude faster than a disk access [27]. Thus, it is dangerously inaccurate to design algorithms assuming a flat memory with uniform access times.

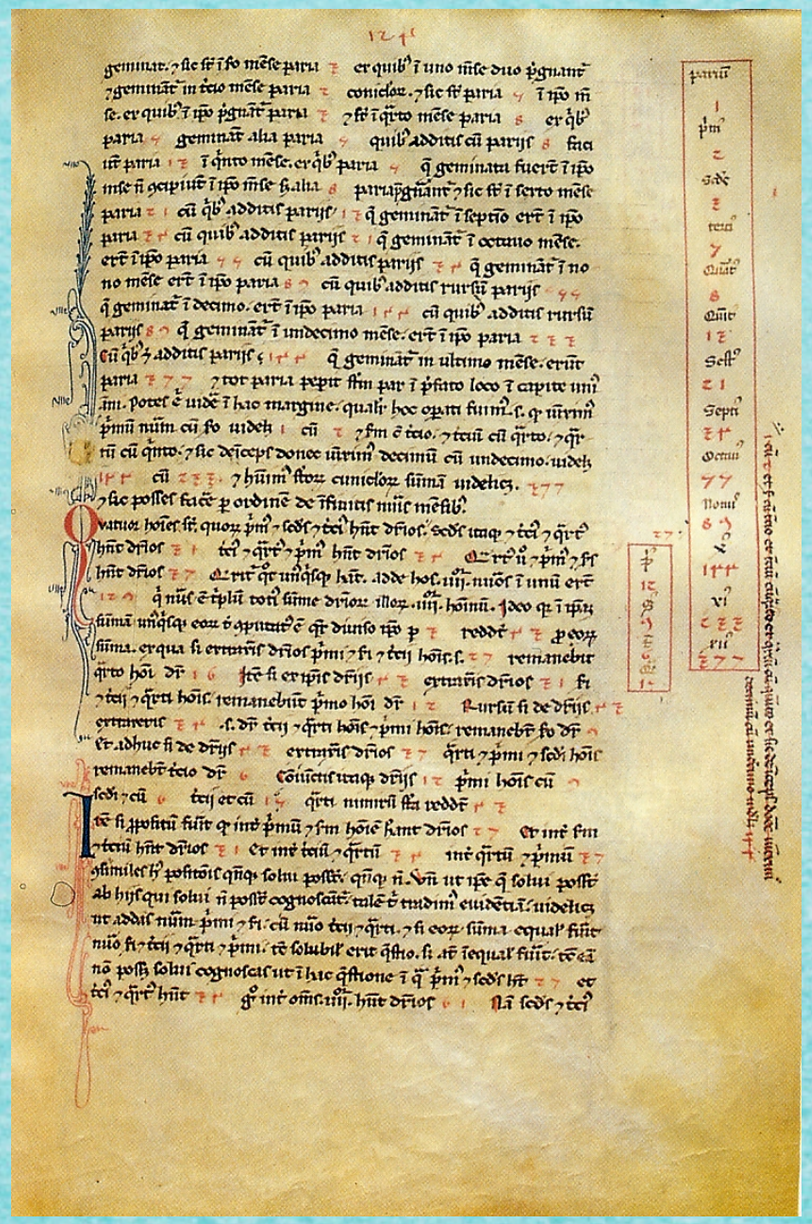
Many computational models attempt to capture the effects of the memory hier-





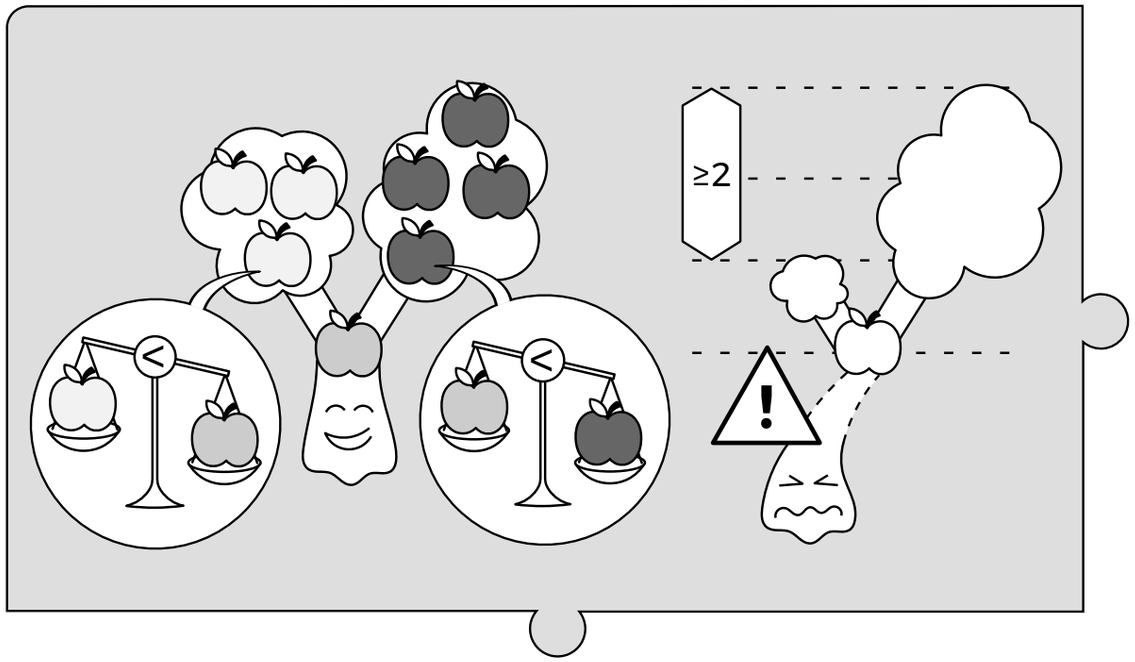
4.7 Fibonacci-Zahlen

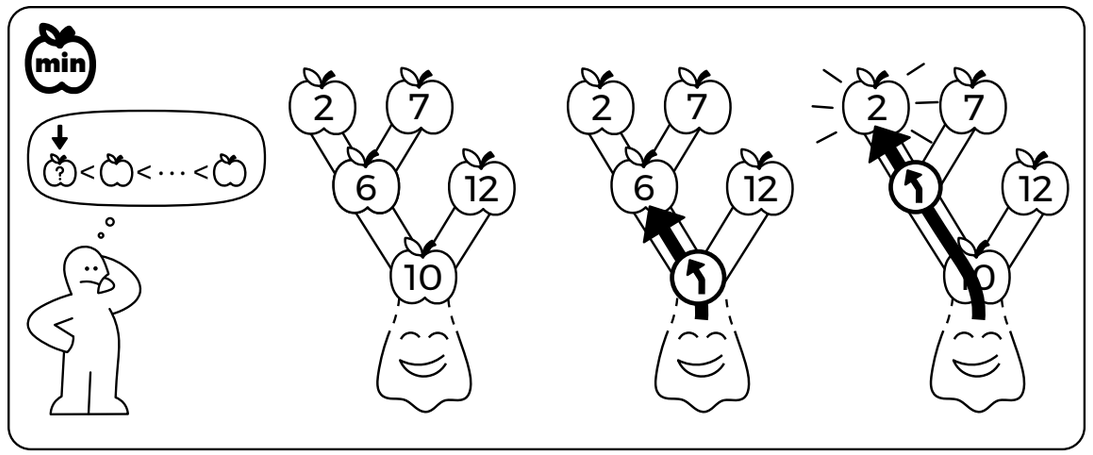
Leonardo da Pisa,
gen. Fibonacci
1180-1241

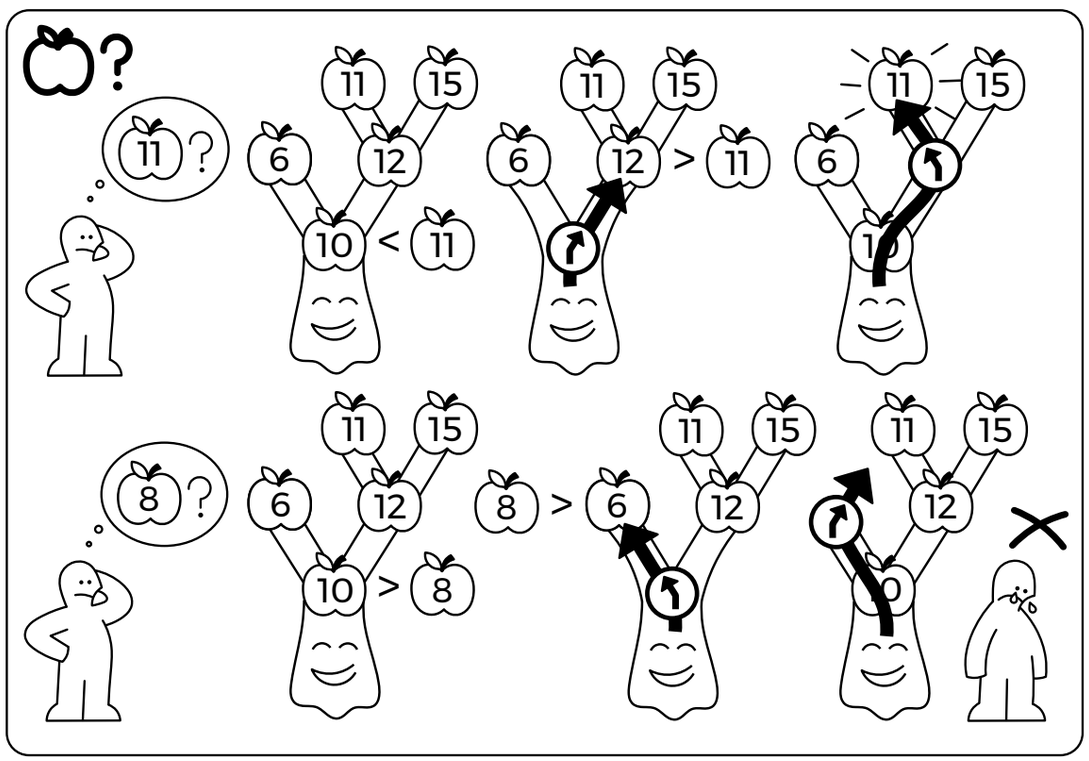


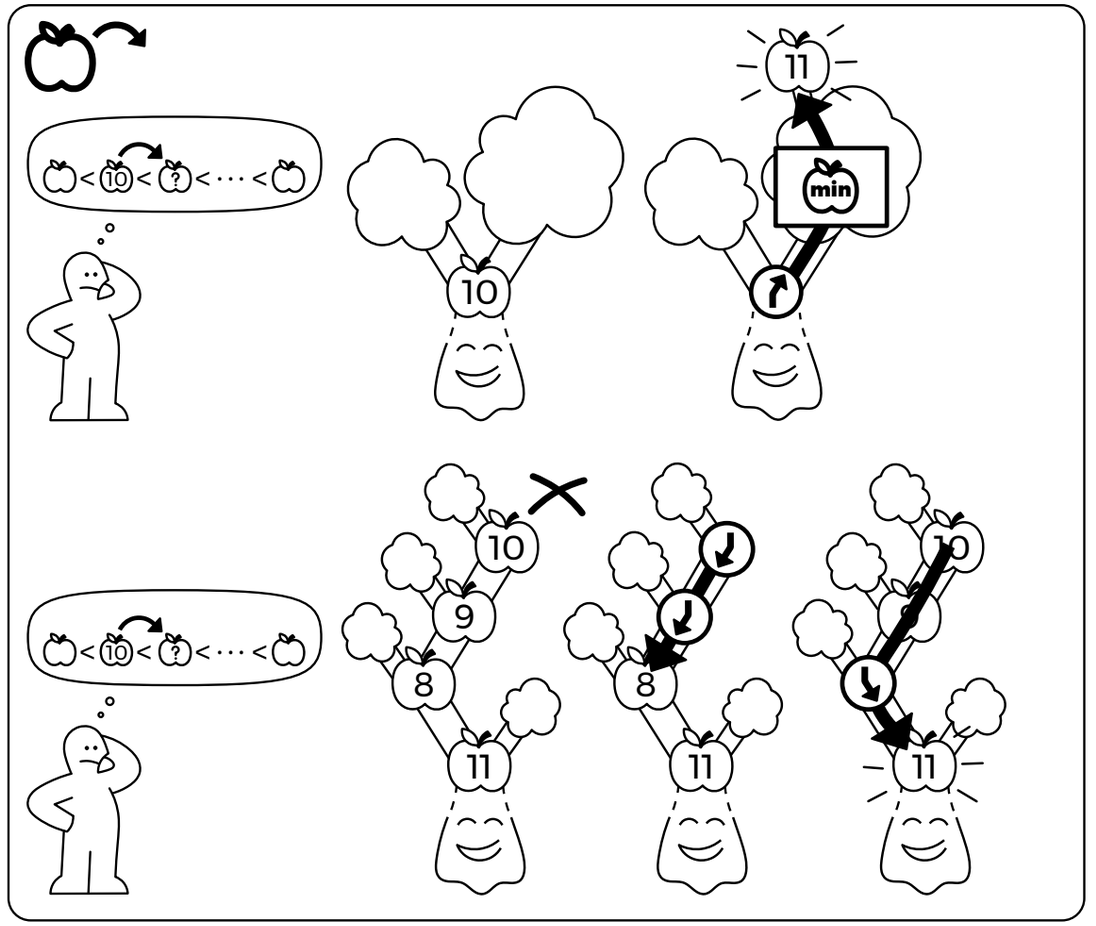
BÄLÄNCE TREE 1/2

idea-instructions.com/avl-tree/
v1.0, CC by-nc-sa 4.0





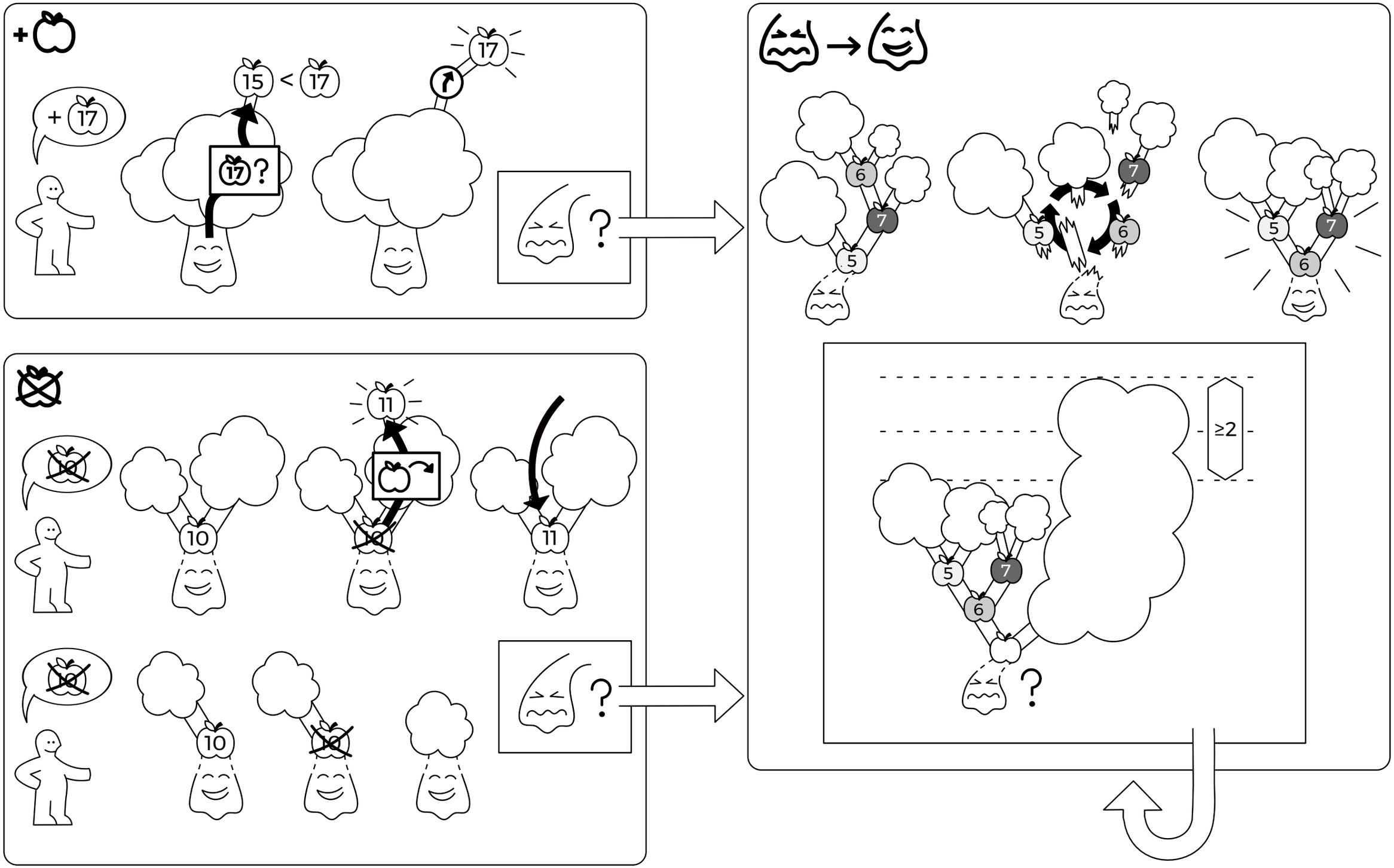




BÄLÄNCE TREE 2/2

idea-instructions.com/avl-tree/
v1.0, CC by-nc-sa 4.0

IDEA



Zusammenfassung Kapitel 4!

Kapitel 5: Sortieren

Algorithmen und Datenstrukturen
WS 2022/23

Prof. Dr. Sándor Fekete

Sortieren von Objekten

Gegeben: n Objekte unterschiedlicher Größe

23 17 13 19 33 28 15

Zahl der Vergleiche:

Gesucht: Eine Sortierung nach Größe

Sortieren von Objekten

Gegeben: n Objekte unterschiedlicher Größe

13 15 17 19 23 28 33

Zahl der Vergleiche:

Gesucht: Eine Sortierung nach Größe

Sortieren von Objekten

Gegeben: n Objekte unterschiedlicher Größe

13 15 17 19 23 28 33

Zahl der Vergleiche: $O(n^2)$

Gesucht: Eine Sortierung nach Größe

Sortieren von Objekten

Gegeben: n Objekte unterschiedlicher Größe

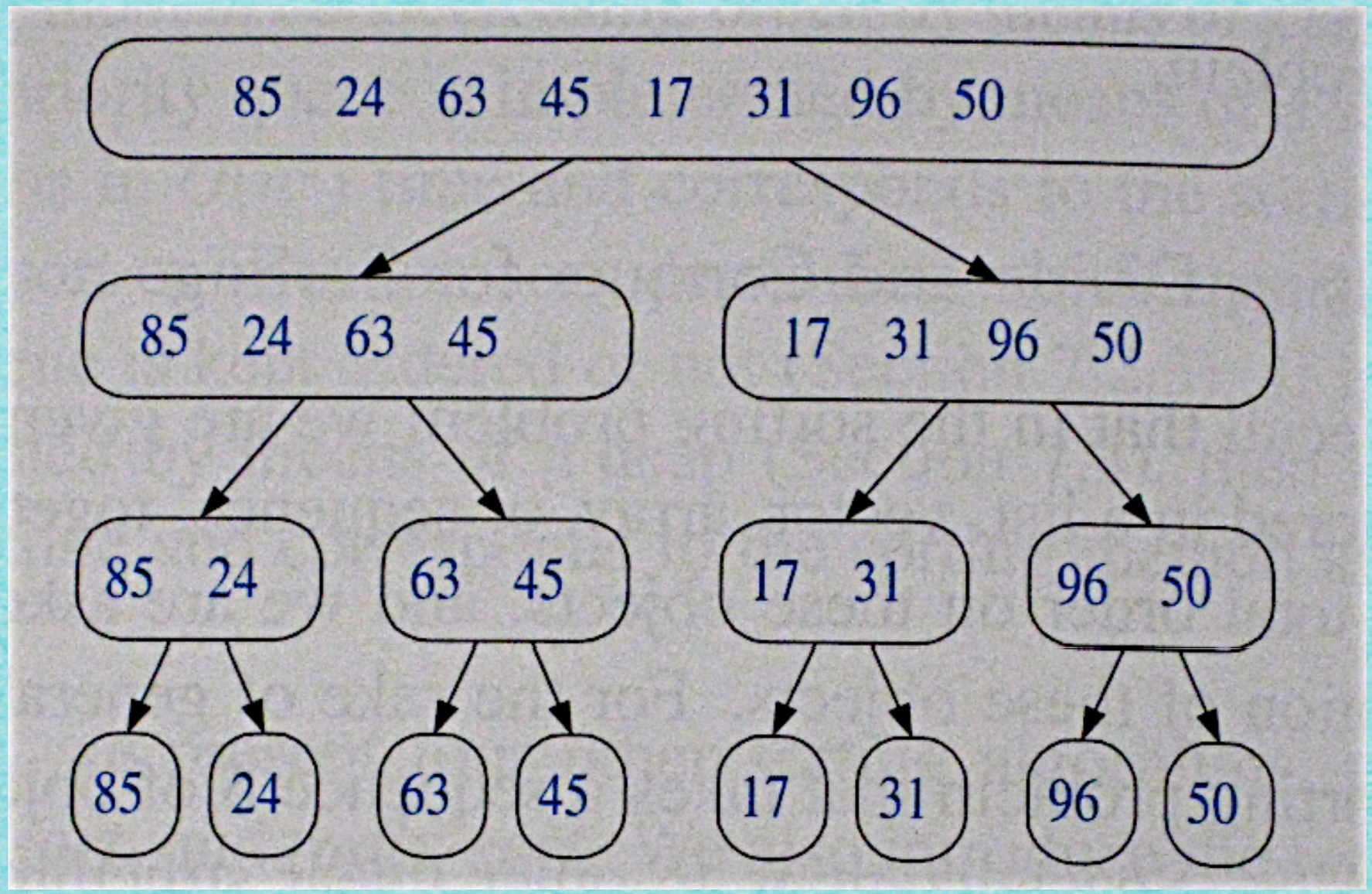
13 15 17 19 23 28 33

Zahl der Vergleiche: $O(n^2)$

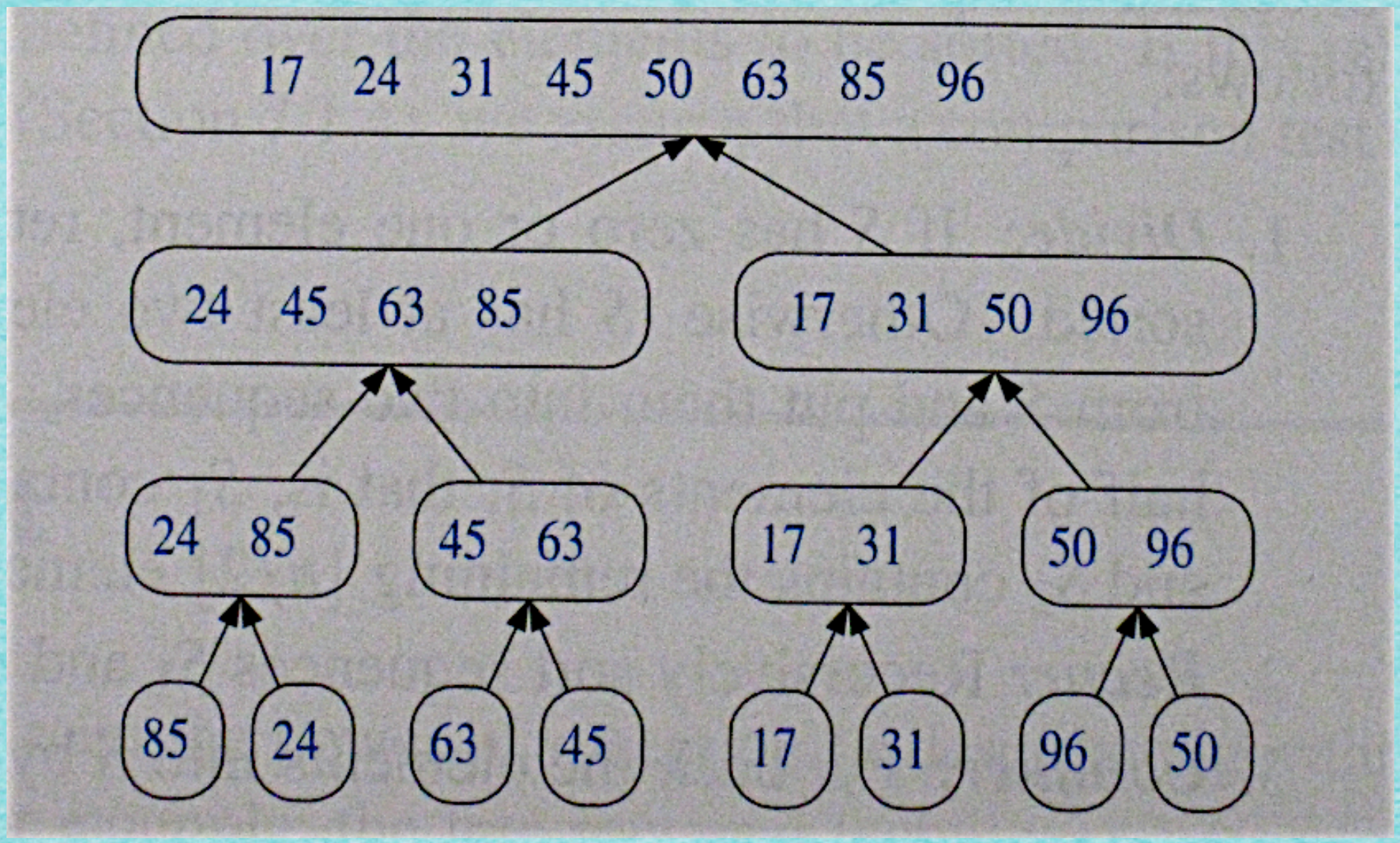
Gesucht: Eine Sortierung nach Größe

„Geht‘s nicht noch etwas schneller?“

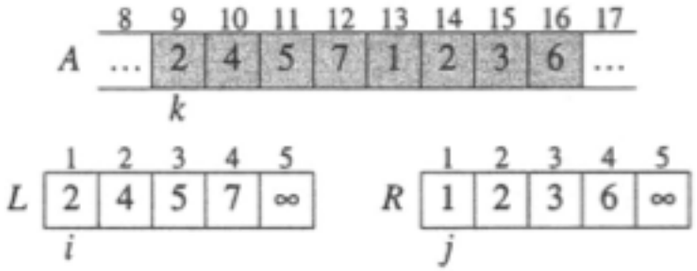
5.2 Mergesort



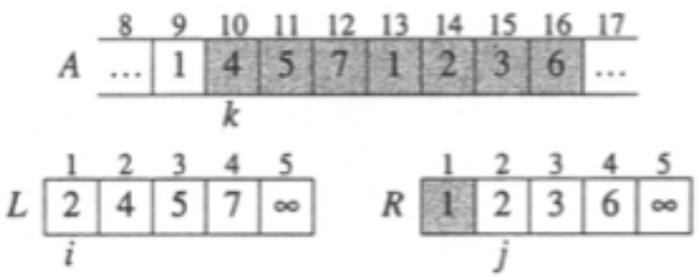
5.2 Mergesort



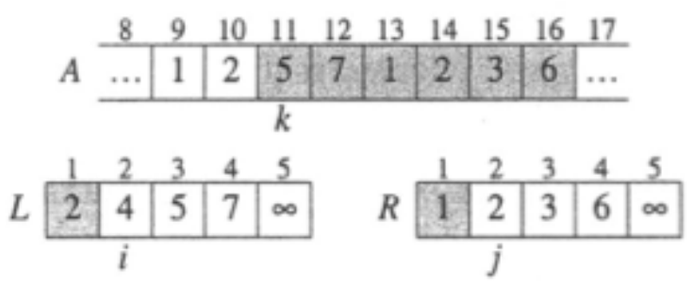
5.2 Mergesort



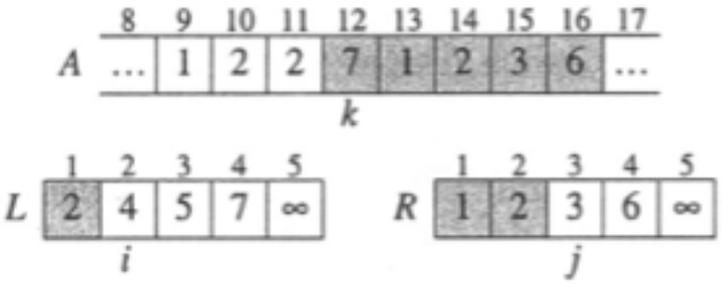
(a)



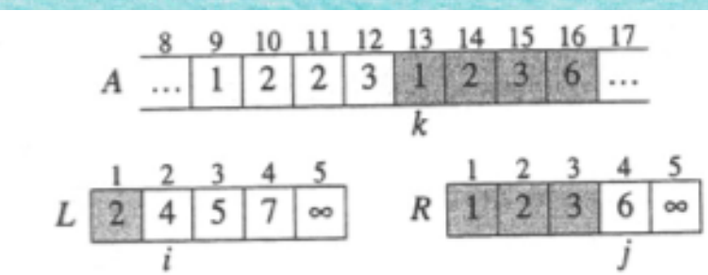
(b)



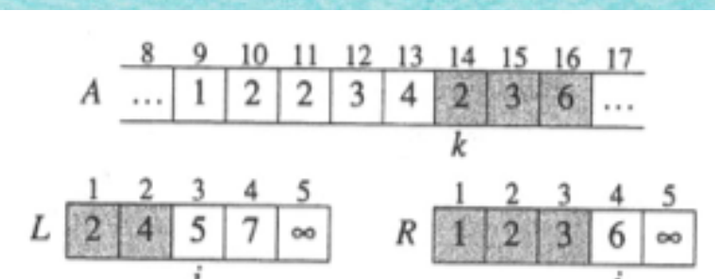
(c)



(d)



(e)



(f)

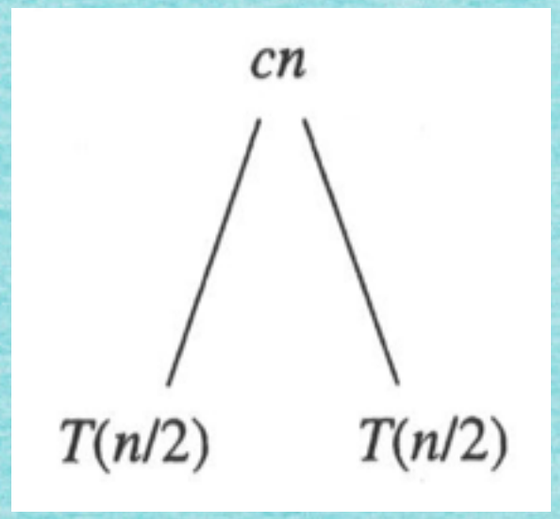
5.2.3 Laufzeit von Mergesort

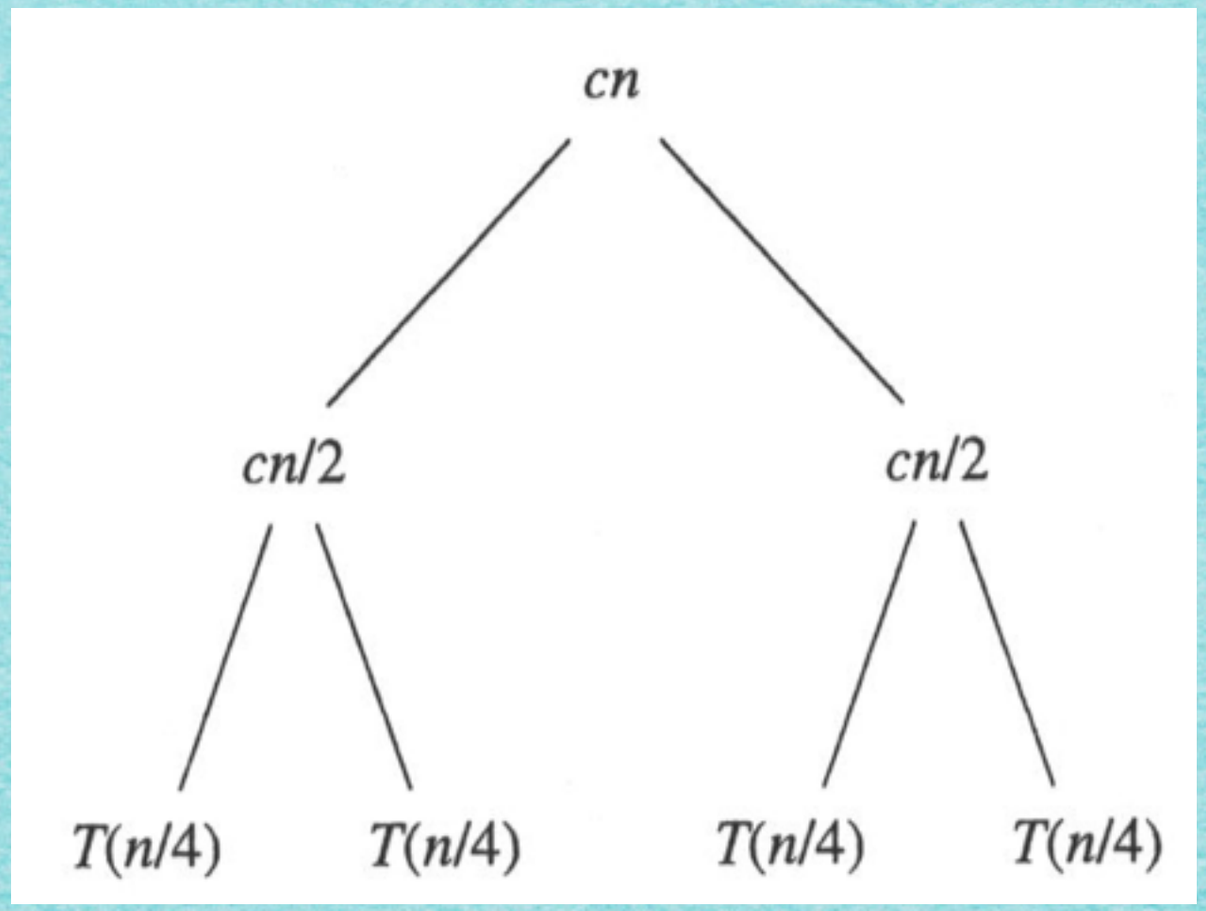
Wie viele Schritte benötigt Merge-Sort für einen Array der Länge n?

$T(n)$

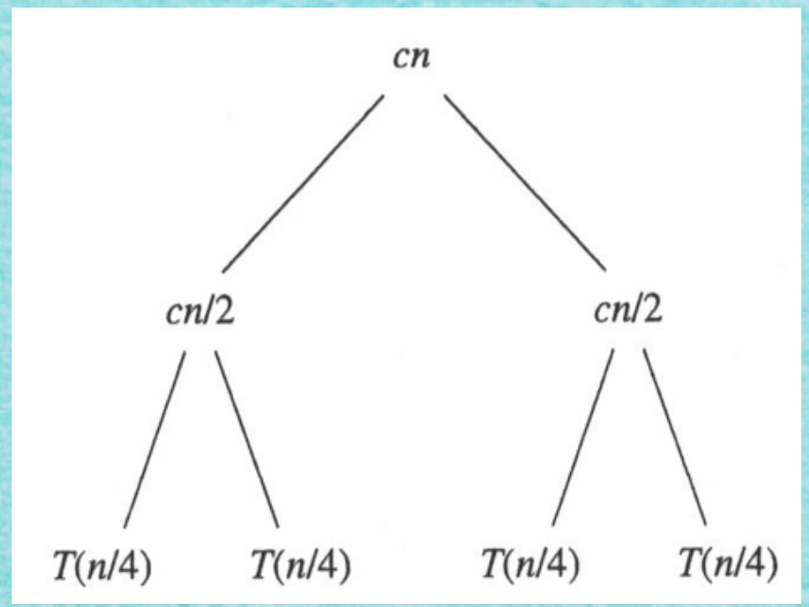
5.1.3 Laufzeit von Mergesort

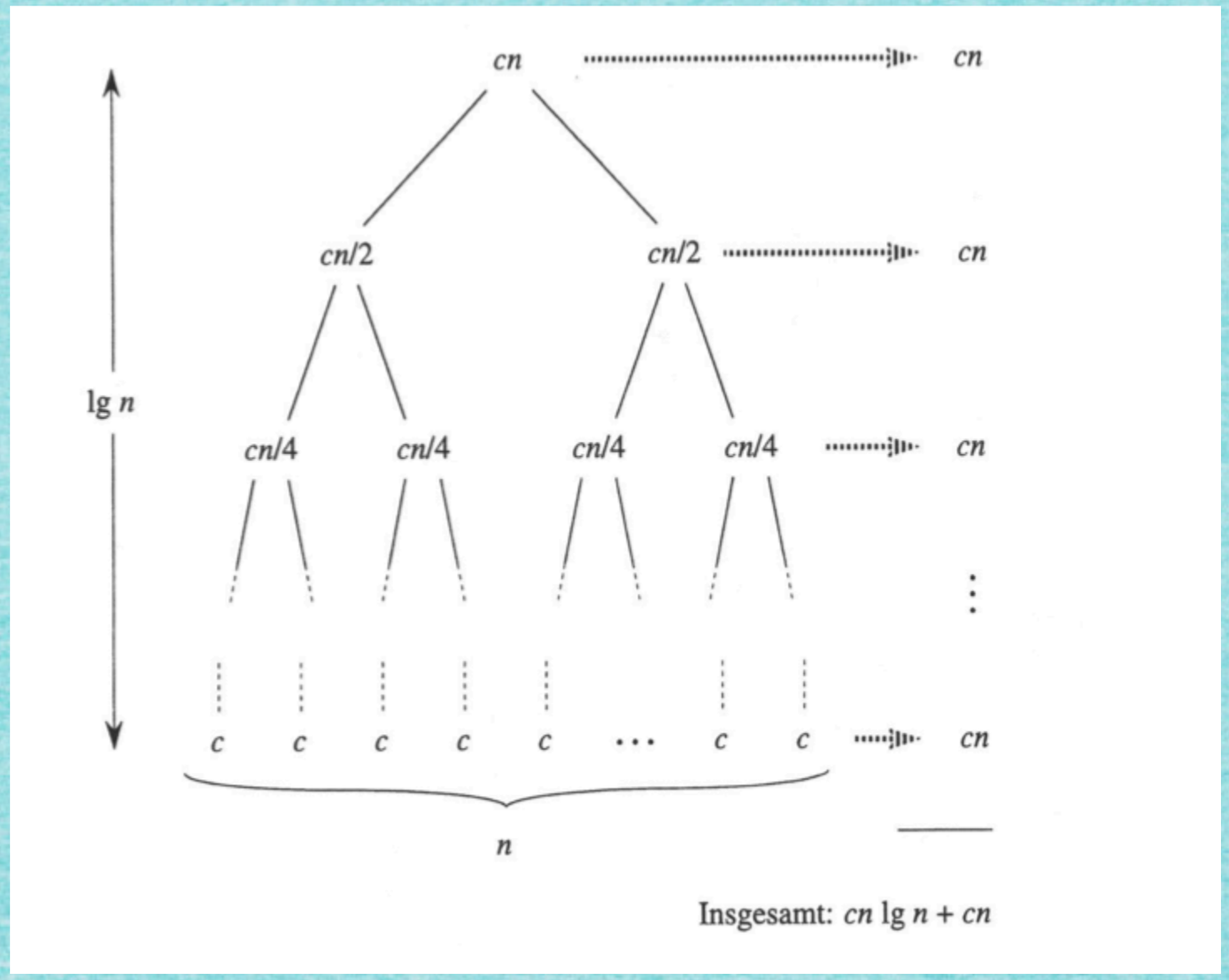
$T(n)$





5.2.3 Laufzeit von Mergesort





5.2.3 Laufzeit von Mergesort

<u>Satz 5.7 (Komplexität von Mergesort)</u>
Für einen n-elementigen Array A hat Mergesort eine Laufzeit von O(n log n).

Satz 5.6. *Für n Objekte $x_1, \ldots, x_n$*

benötigt man zum Sortieren mindestens $\Omega(n \log(n))$,

wenn man die Objekte nur paarweise vergleichen kann.

Kapitel 5.3: Behandeln von Rekursionen

Algorithmen und Datenstrukturen
WS 2022/23

Prof. Dr. Sándor Fekete

5.3.1 Substitutionsmethode

5.3 Behandeln von Rekursionen

Welche Möglichkeiten gibt es, Rekursionsgleichungen zu lösen?

5.3.1 Substitutionsmethode

Wie gesehen!

(1) Rate eine Lösung.
(2) Beweise die Richtigkeit per Vollständiger Induktion.

Das haben wir im Beweis von SATZ 5.7 angewendet.

Schwierigkeit: Gute Lösung finden!
(In der Regel ist man an einer möglichst genauen Lösung interessiert – das kann schwer bis unmöglich sein!)
Oft kann man aber Abschätzungen gewinnen, z.B.

$$T(n) \in \Omega(n)$$
$$T(n) \in O(n^2)$$

5.3.2 Erzeugende Funktionen

5.3.2 Erzeugende Funktionen

Man kann Rekursionen auch oft lösen, indem man sie in einen abstrakt-formalen Kontext einbettet und die dafür bekannten Rechenregeln anwendet.

Dafür betrachten wir

DEFINITION 5.8 (Erzeugende Funktion)

Sei $(a_n)_{n\in\mathbb{N}}$ eine Zahlenfolge mit $a_n \in \mathbb{R}$.

Dann heißt $f(x) = \sum_{n=0}^{\infty} a_n \cdot x^n$

die (gewöhnliche) erzeugende Funktion von $(a_n)_{n\in\mathbb{N}}$.

Beispiel

(a) $a_n = 2a_{n-1}$, $n = 0, 1, 2, \ldots$

$a_0 = 1$

$a_1 = 2$, $a_2 = 4$, $a_3 = 8$

5.3.2 Erzeugende Funktionen

$$F_n = \frac{1}{\sqrt{5}} \left(\varphi^n - \bar{\varphi}^n \right)$$

bzw. (ganz explizit!)

$$F_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right)$$

5.3.3 Master-Theorem: Lineare Rekursionen

5.3.3 Master-Theorem: Lineare Rekursionen

Satz 5.9 (Master-Theorem)

Sei $T: \mathbb{N} \to \mathbb{R}$ mit

$$T(n) = \sum_{i=1}^{m} T(\alpha_i \cdot n) + \Theta(n^k),$$

wobei $\alpha_i \in \mathbb{R} : 0 < \alpha_i < 1$, $m \in \mathbb{N}$, $k \in \mathbb{R}$.

Dann gilt

$$T(n) \in \begin{cases} \Theta(n^k) & \text{für } \sum_{i=1}^{m} \alpha_i^k < 1 \\ \Theta(n^k \log n) & \text{für } \sum_{i=1}^{m} \alpha_i^k = 1 \\ \Theta(n^c) \text{ mit } \sum_{i=1}^{m} \alpha_i^c = 1 & \text{für } \sum_{i=1}^{m} \alpha_i^k > 1 \end{cases}$$

Kapitel 5.5: Nichtlineare Rekursionen

Algorithmen und Datenstrukturen
WS 2022/23

Prof. Dr. Sándor Fekete

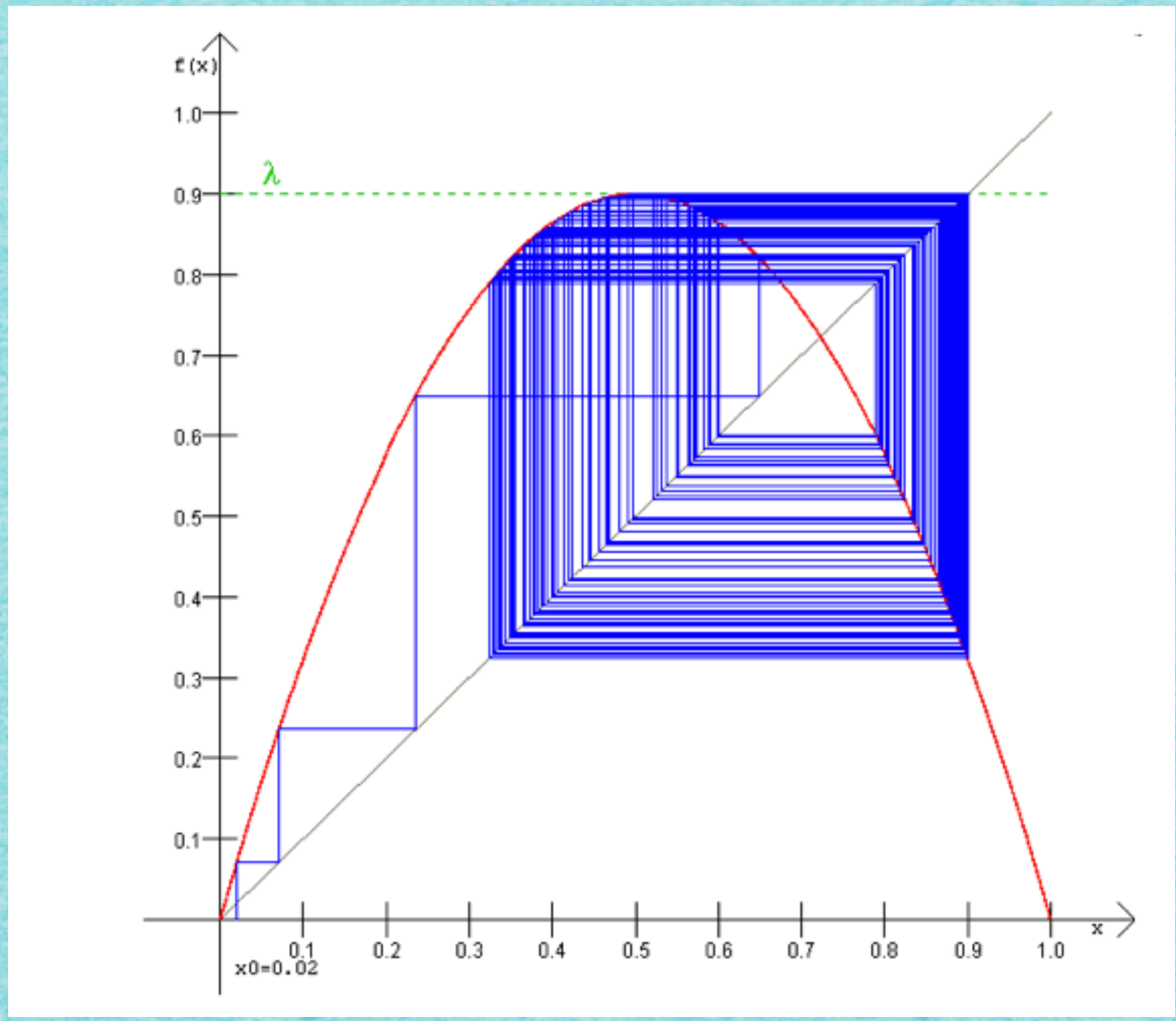
5.4.1 Logistische Rekursion

5.4.1 Logistische Rekursion

Keinerlei Fixpunkte - deterministisches Chaos:

5.4.1 Logistische Rekursion

Keinerlei Fixpunkte - deterministisches Chaos:



5.4.1 Logistische Rekursion

5.4.1 Logistische Rekursion

5.4.1 Logistische Rekursion

Edward Lorenz
(1917-2008)

5.4.1 Logistische Rekursion

Edward Lorenz
(1917-2008)

Grenzen der Kausalität!

5.4.1 Logistische Rekursion

Edward Lorenz
(1917-2008)

Grenzen der Kausalität!

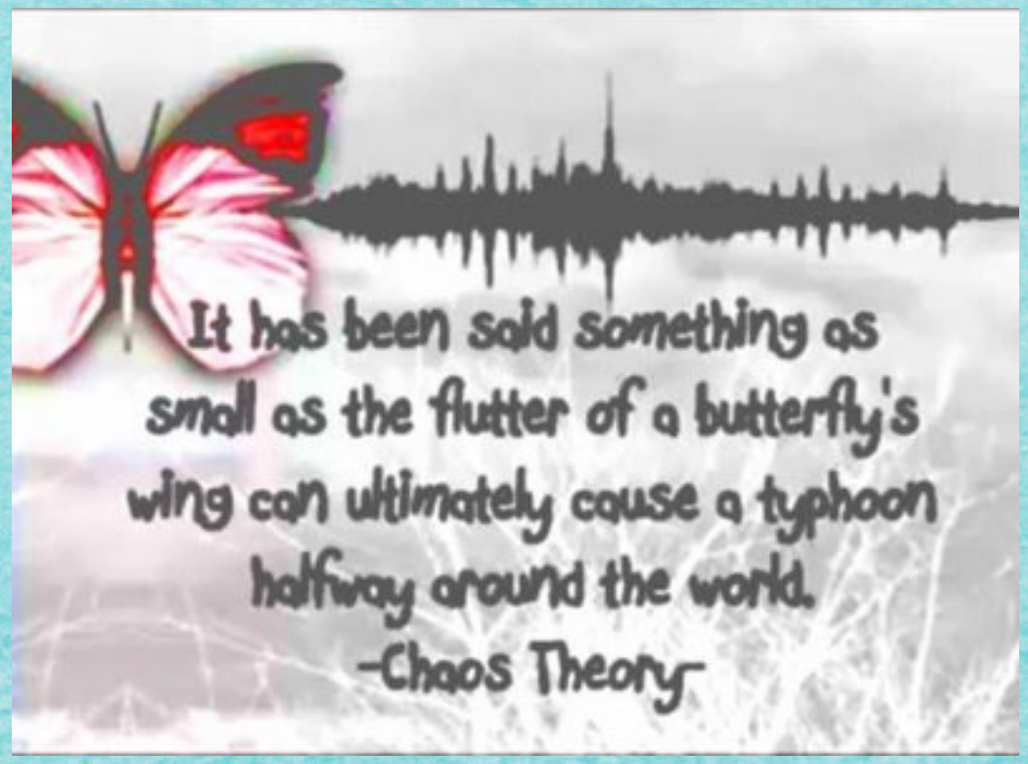
Predictability: Does the flap of a butterfly's wings in Brazil set off a tornado in Texas?

5.4.1 Logistische Rekursion

Edward Lorenz
(1917-2008)

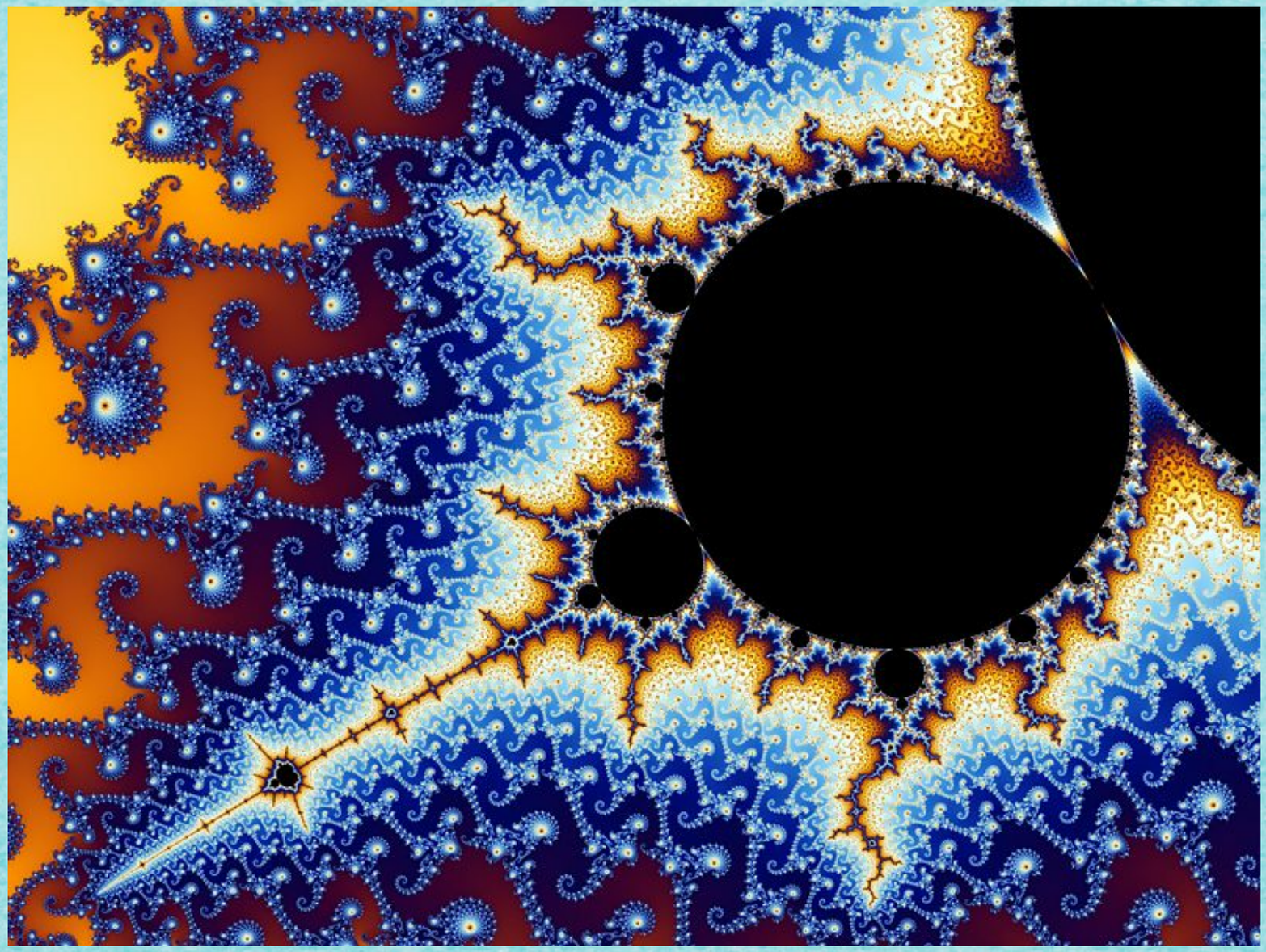
Grenzen der Kausalität!

Predictability: Does the flap of a butterfly's wings in Brazil set off a tornado in Texas?



54.2 Die Mandelbrotmenge

Ausschnitt:

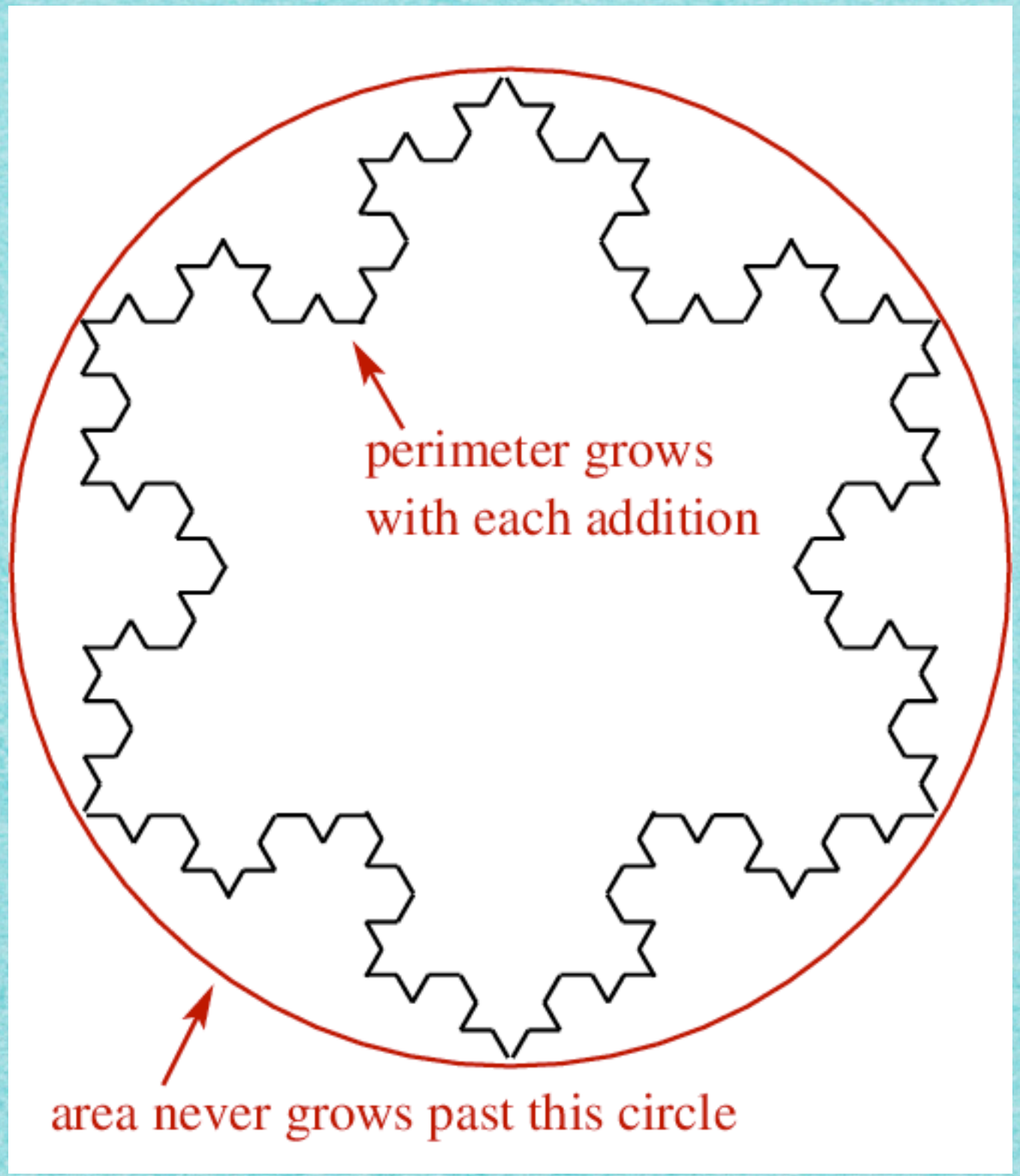


-> Filme!

5.4.3 Fraktale

Hausdorff-Dimension:
Wie wächst das Gesamtmaß in Abhängigkeit von der Größe?

5.4.3 Fraktale

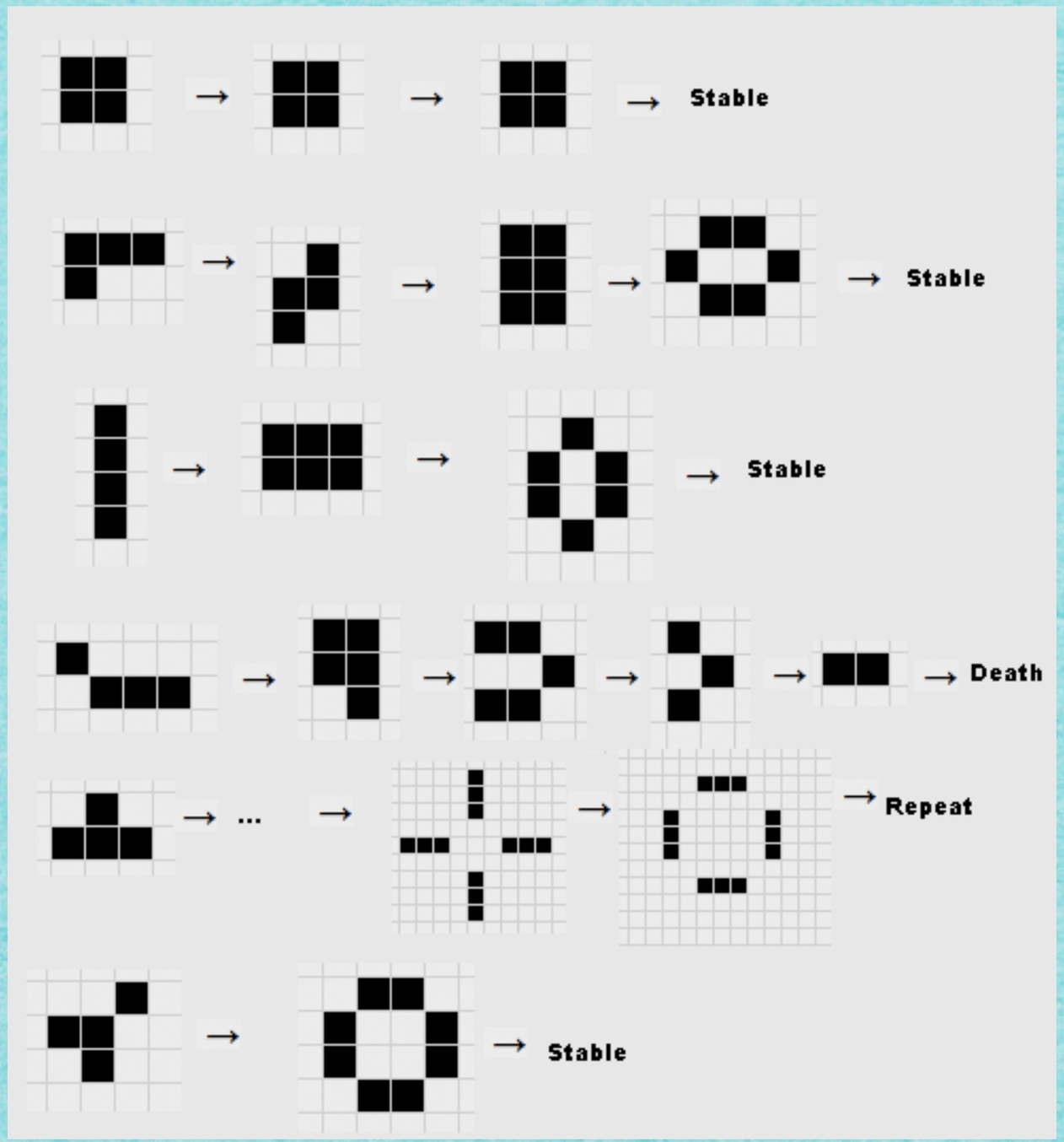


Pro Iteration:
Länge wächst um
Faktor 4/3

Hausdorff-Dimension
des Randes:
log(4)/log(3)
= 1.2618595...

Fläche:
Länge wächst um
Faktor 4/3

5.4.4 Zelluläre Automaten



Kapitel 5.5: Quicksort

Algorithmen und Datenstrukturen
WS 2022/23

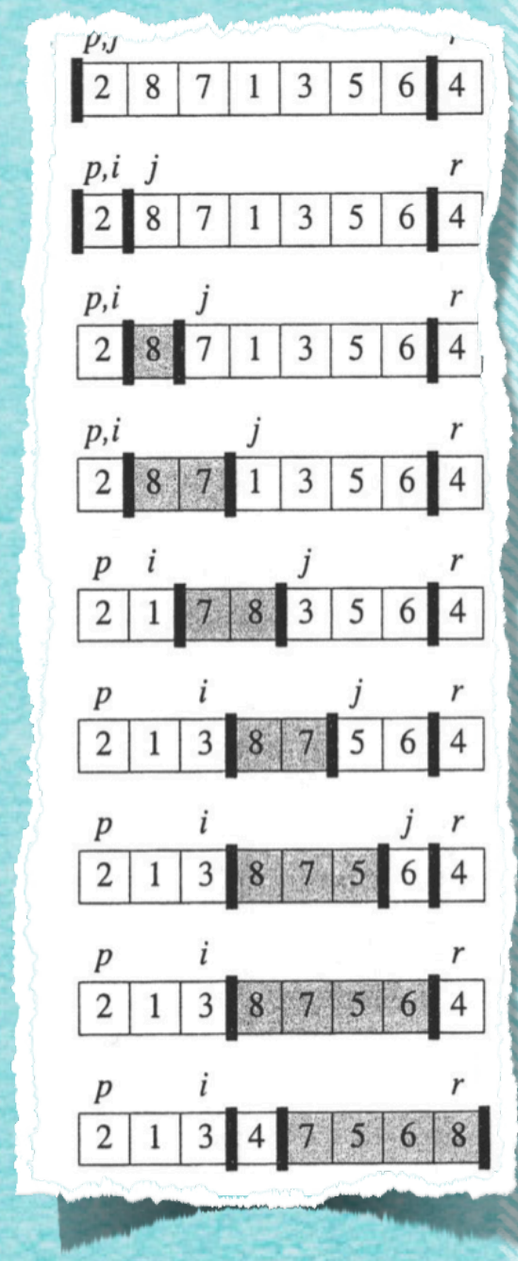
Prof. Dr. Sándor Fekete

Subroutine 5.1.2

INPUT: Subarray von A=[1,...,n], d.h. A[p,...,r]

OUTPUT: Zwei Subarrays A[p,...,q-1] und A[q+1,...,r]
mit A[i]≤A[q] und A[q]<A[j] für i=p,...,q-1 und j=q+1,...,r

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4        do if A[j] ≤ x
5              then i ← i + 1
6                   vertausche A[i] ↔ A[j]
7  vertausche A[i + 1] ↔ A[r]
8  return i + 1
```



5.5 Quicksort

Satz (Komplexität von Quicksort)
Für einen n-elementigen Array A hat Mergesort eine erwartete Laufzeit von O(n log n).

Wir müssen also die Wahrscheinlichkeit bestimmen, dass s_i und s_j verglichen werden.

Dafür betrachten wir

$$S_{ij} := \{s_i, \dots, s_j\}.$$

$|\overline{s_i \mid \dots \mid s_j}|$

Dann ist

$$P(s_i \text{ wird mit } s_j \text{ verglichen}) = P(s_i \text{ oder } s_j \text{ ist erster Pivot in } S_{ij}).$$

Denn wird s_i oder s_j gewählt, wird es mit allen anderen verglichen; wird aber zuerst ein Pivot dazwischen gewählt, werden s_i und s_j getrennt und nicht mehr verglichen!

Kapitel 5.7: Mediane

Algorithmen und Datenstrukturen
WS 2022/23

Prof. Dr. Sándor Fekete

Satz 5.23

Der Median für n Zahlen kann in $\mathcal{O}(n)$ berechnet werden.

Beweisidee:

$$X = \{1, 22, 10, 13, 24, 6, 18, 21, 4, 25, 11, 16, 2, 20, 8, 17, 5, 12, 19, 14, 3, 9, 15, 7, 23\}$$

- Gruppiere die Zahlen in Fünfergruppen.

1	6	11	17	3
22	18	16	5	9
10	21	2	12	15
13	4	20	19	7
24	25	8	14	23

$\mathcal{O}(n)$

- Sortiere die Fünfergruppen.

1	4	2	5	3
10	6	8	12	7
13	18	11	14	9
22	21	16	17	15
24	25	20	19	23

$\mathcal{O}(n)$

Beweisidee (Forts.):

- Berechne den Median der Mediane.

1	4	2	5	3
10	6	8	12	7
13	18	11	14	9
22	21	16	17	15
24	25	20	19	23

$$T(\frac{n}{5})$$

- Verwende den Median der Mediane als Pivot, um die Menge zu reduzieren.

≥ n/4 Zahlen

≥ n/4 Zahlen

3	2	1	5	4
7	8	10	12	6
9	11	13	14	18
15	16	22	17	21
23	20	24	19	25

$$T(\frac{3n}{4})$$

$$T(n) = T(\frac{n}{5}) + T(\frac{3n}{4}) + \Theta(n)$$

$$\sum_{i=1}^{m} \alpha_i^k = \frac{1}{5} + \frac{3}{4} = \frac{19}{20} < 1.$$

Beweisidee (Forts.):

- Berechne den Median der Mediane.

1	4	2	5	3
10	6	8	12	7
13	18	11	14	9
22	21	16	17	15
24	25	23	20	0

$$T(\frac{n}{5})$$

- Verwende den Median der M… eren.

$$T(n) \in \Theta(n^k) = \Theta(n)$$

1	4	13	14	18
15	16	22	17	21
23	20	24	19	25

Zahlen

$$T(\frac{3n}{4})$$

$$T(n) = T(\frac{n}{5}) + T(\frac{3n}{4}) + \Theta(n)$$

$$\sum_{i=1}^{m} \alpha_i^k = \frac{1}{5} + \frac{3}{4} = \frac{19}{20} < 1.$$

5.6.2 Laufzeit

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + \Theta(n)$$

FRAGE 5.24

Was passiert, wenn man Dreier- statt Fünfergruppen verwendet?
Was passiert bei Siebenergruppen?

Kapitel 5.8: Sortieren in Linearzeit

Algorithmen und Datenstrukturen
WS 2022/23

Prof. Dr. Sándor Fekete

5.7.1 Countingsort

5.7 Sortieren in linearer Zeit

In Satz 5.4 haben wir gezeigt, dass das Sortieren von n Objekten nicht schneller als in $\Omega(n \log n)$ möglich ist - vorausgesetzt, man kann dafür keine Zusatzinformation einsetzen und darf Objekte nur paarweise vergleichen (und ggf. vertauschen).

In vielen Situationen hat man aber zusätzliche Informationen!
Das wollen wir hier analysieren.

5.7.1 Countingsort

Im einfachsten Fall sind die n Objekte Zahlen aus dem Bereich $1, \ldots, k$.

5.7.2 Radixsort

5.7.2 Radixsort

Idee: Sortiere nicht gleich nach den Schlüsselwerten, sondern nach den Ziffern der Schlüssel!

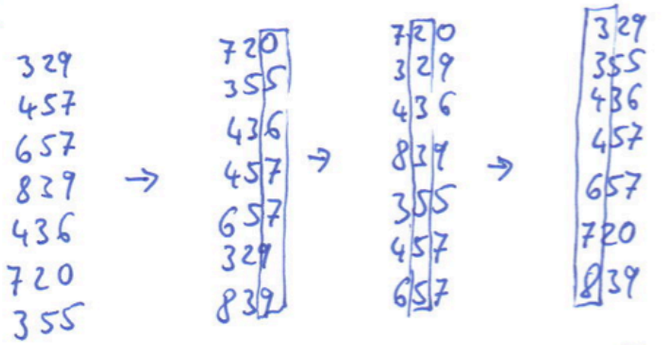
(Früher sehr häufig bei Lochkarten eingesetzt.)

Naiver Ansatz: Sortiere nach größter Ziffer, dann nach zweiter Ziffer, ...

Problem: Entweder benötigt man viele temporäre Zwischenmengen - oder die Vorsortierung geht kaputt!

Besserer Ansatz: Sortiere nach letzter Ziffer, dann nach vorletzter Ziffer, ...

Beispiel:



Wichtig: Jeweilige Sortierung darf Reihenfolge gleichwertiger Ziffern nicht verändern!

5.7.2 Radixsort

5.7.2 Radixsort

Idee: Sortiere nicht gleich nach den Schlüsselwerten, sondern nach den Ziffern der Schlüssel!

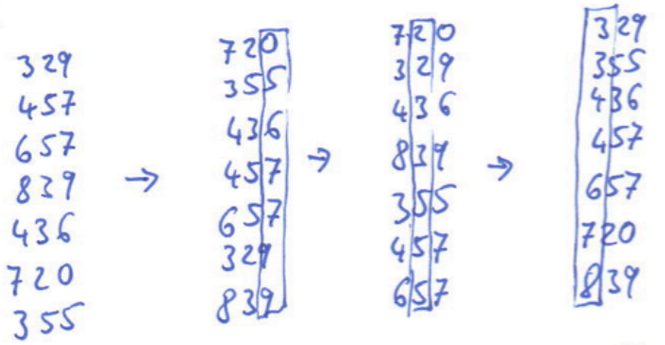
(Früher sehr häufig bei Lochkarten eingesetzt.)

Naiver Ansatz: Sortiere nach größter Ziffer, dann nach zweiter Ziffer, ...

Problem: Entweder benötigt man viele temporäre Zwischenmengen - oder die Vorsortierung geht kaputt!

Besserer Ansatz: Sortiere nach letzter Ziffer, dann nach vorletzter Ziffer, ...

Beispiel:

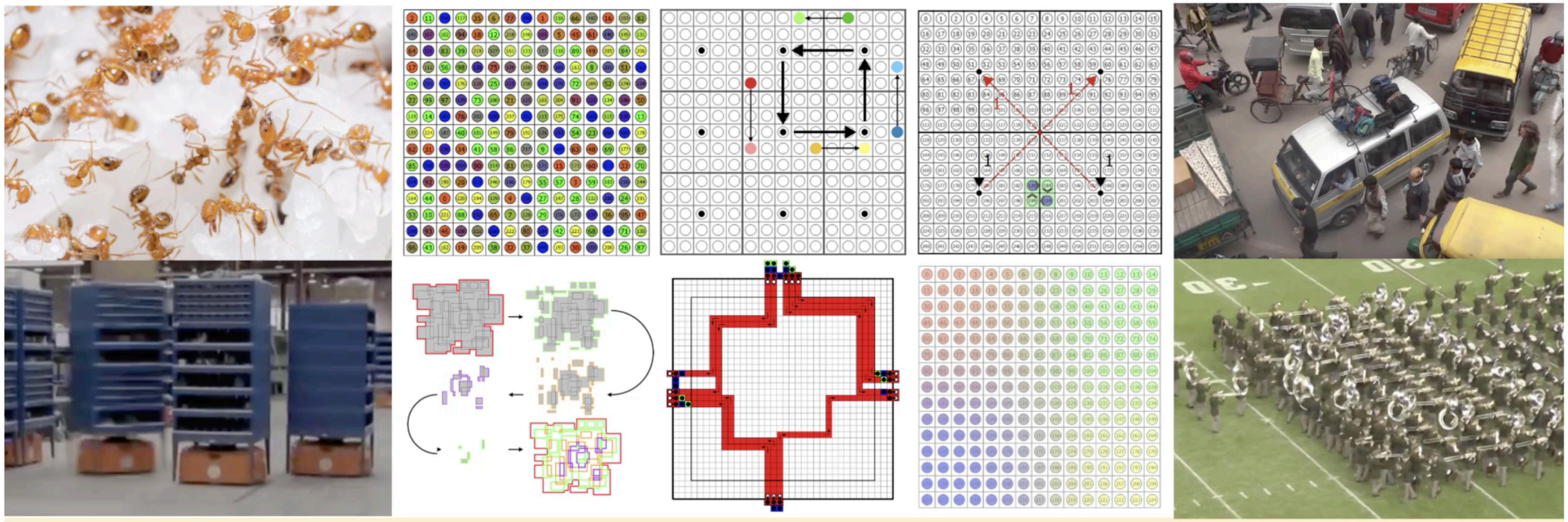


Wichtig: Jeweilige Sortierung darf Reihenfolge gleichwertiger Ziffern nicht verändern!

Kapitel 5.9: Paralleles Sortieren

Algorithmen und Datenstrukturen
WS 2022/23

Prof. Dr. Sándor Fekete



Coordinated Motion Planning: The Video

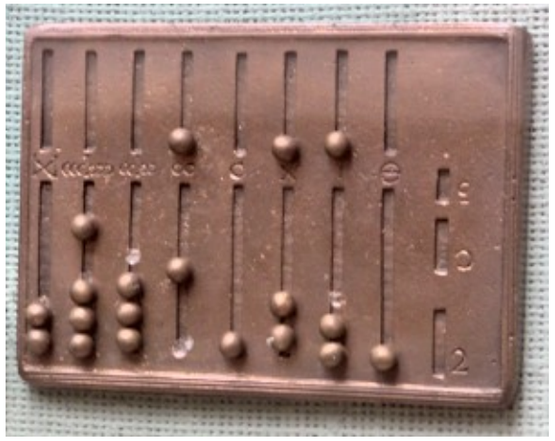
Aaron Becker, Sándor P. Fekete, Phillip Keldenich,
Matthias Konitzny, Lillian Lin, Christian Scheffer

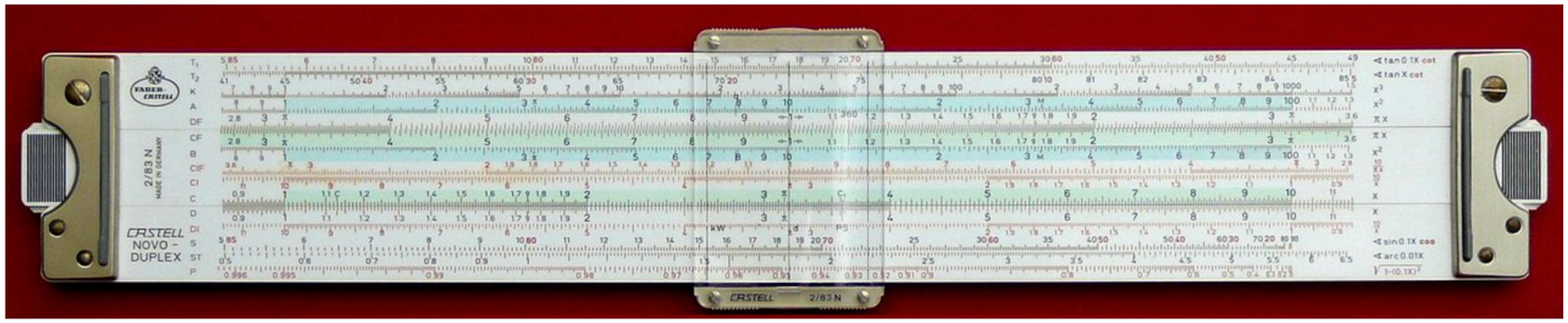
Kapitel 5.10: Analoge Verfahren

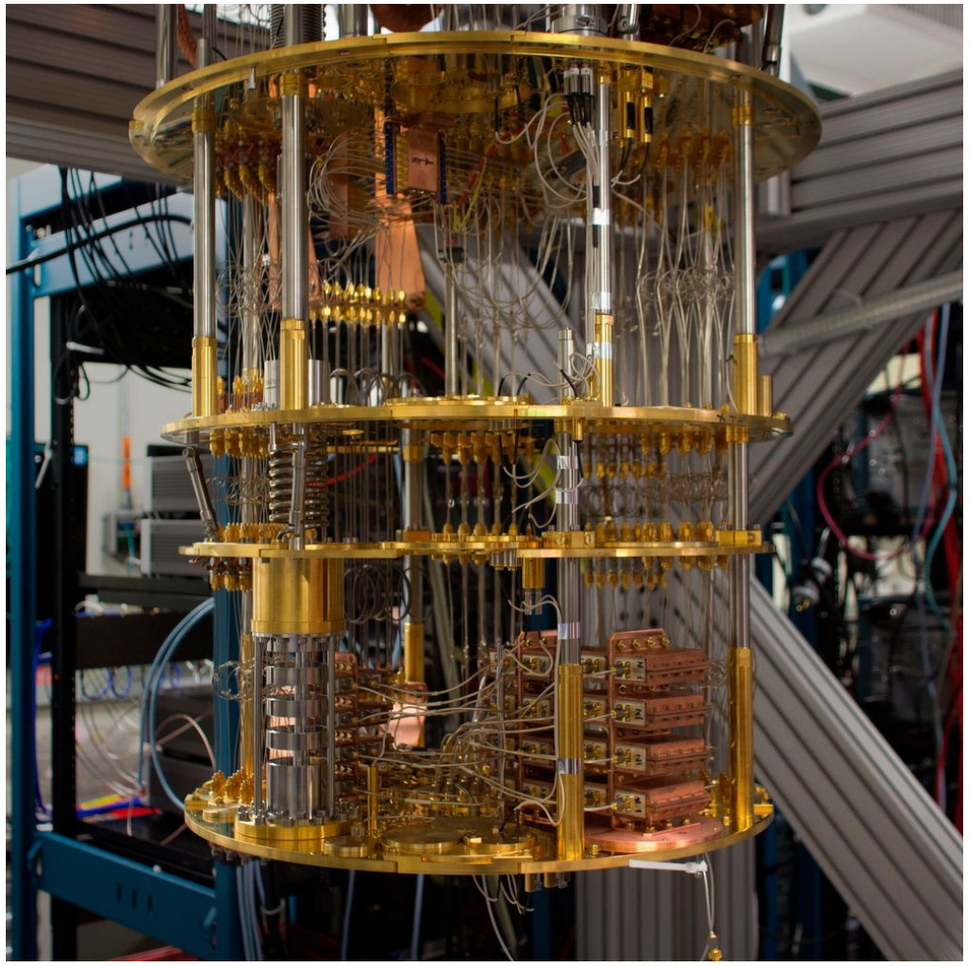
Algorithmen und Datenstrukturen
WS 2022/23

Prof. Dr. Sándor Fekete

Analogrechner



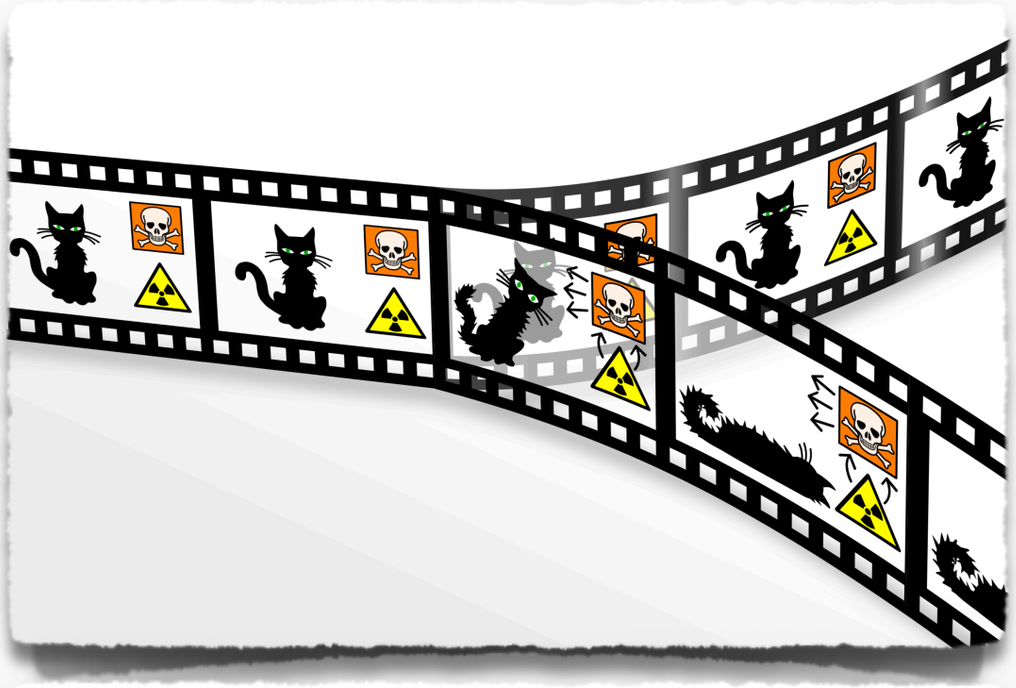


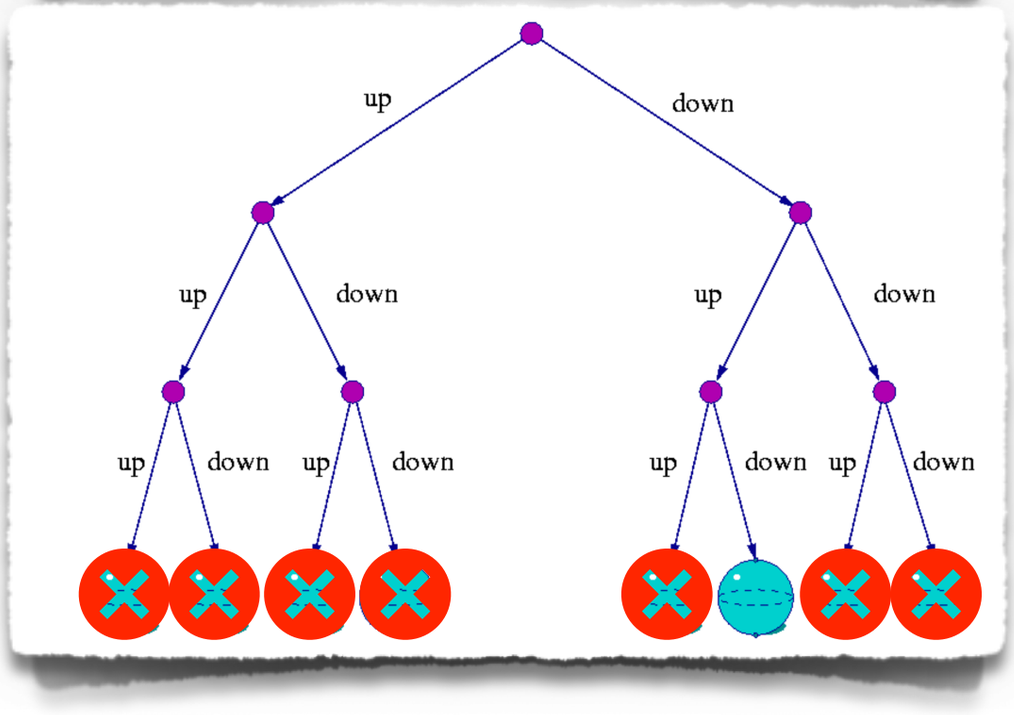


Kapitel 5.11: Unernste Sortierverfahren

Algorithmen und Datenstrukturen
WS 2022/23

Prof. Dr. Sándor Fekete



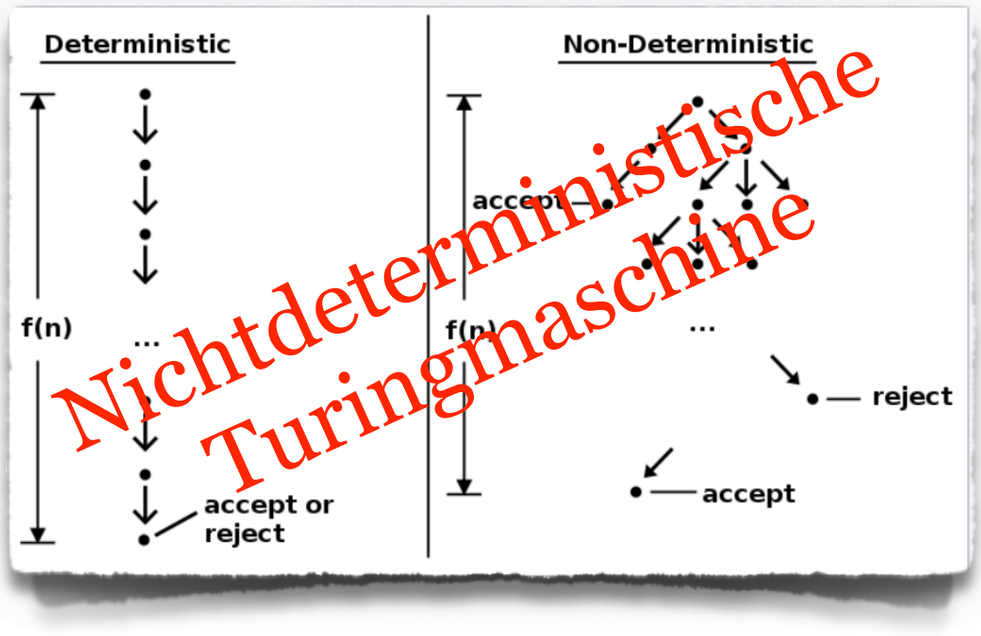


ALGORITHMUS 5.30 (Quantum Bogosort)

INPUT : n Zahlen $A[1], \ldots, A[n]$

OUTPUT : Sortierter Array in geeignetem Universum

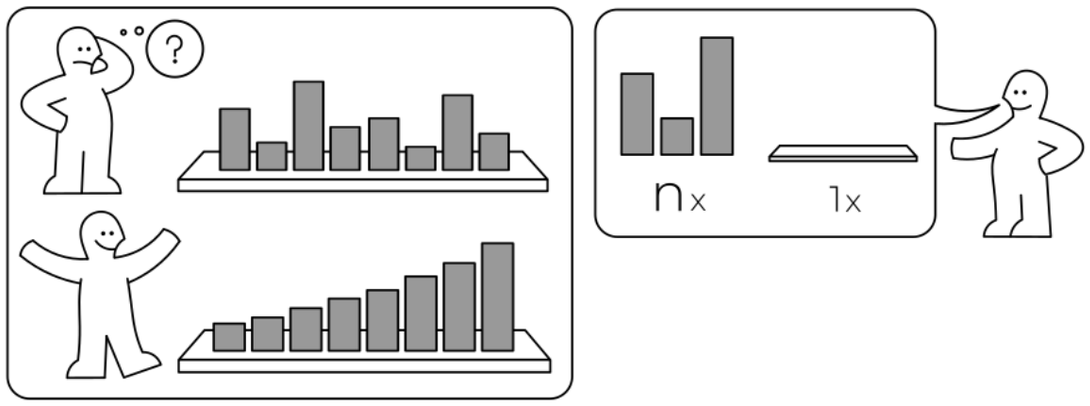
1. Permutiere zufällig
2. IF (Permutation unsortiert)
   2.1 Zerstöre Universum



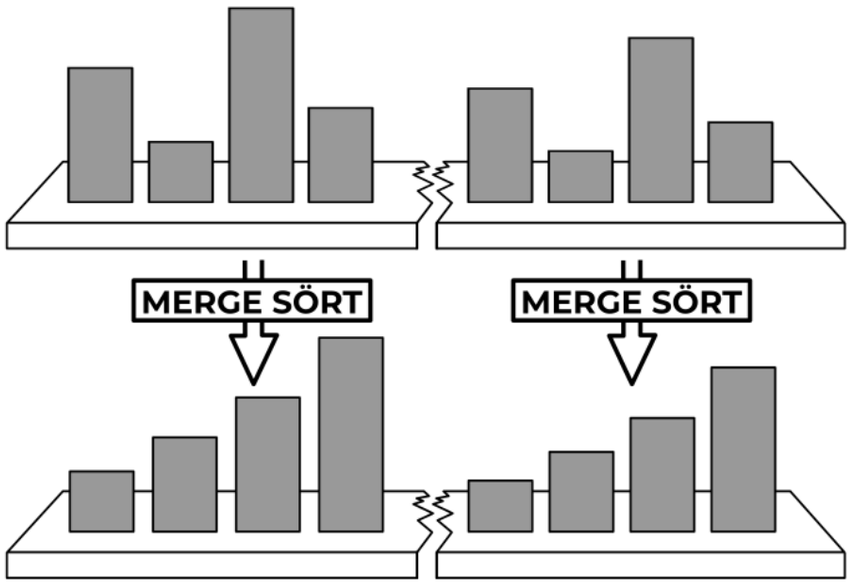
MERGE SÖRT

idea-instructions.com/merge-sort/
v1.2, CC by-nc-sa 4.0

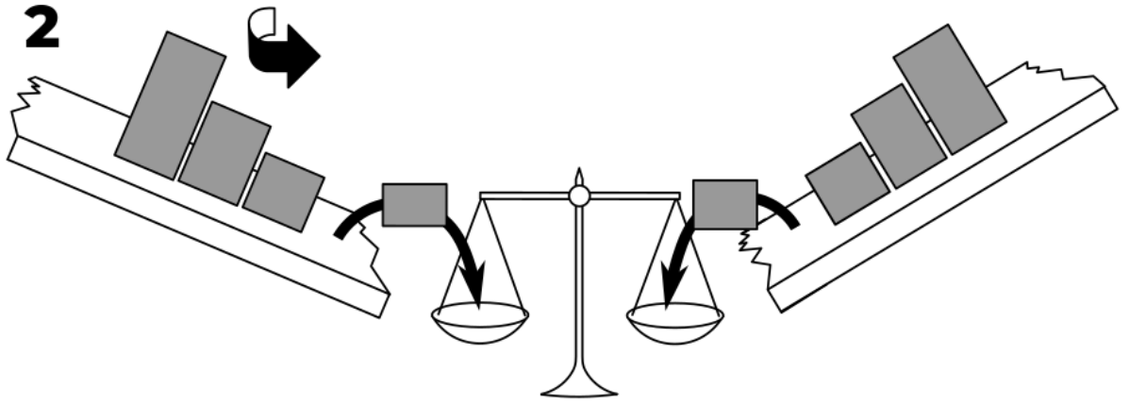
IDEA



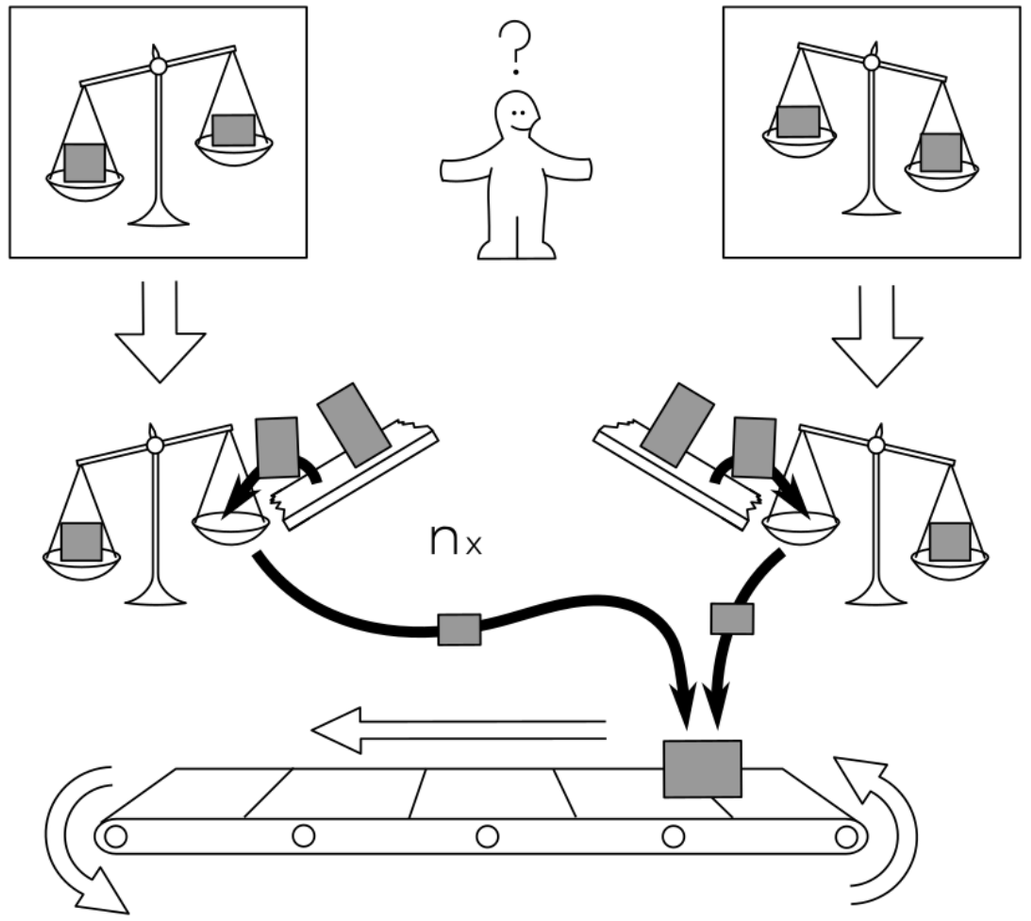
1



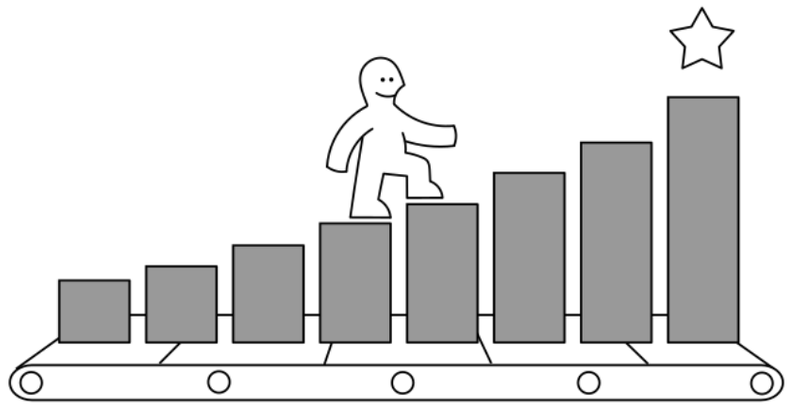
2



3



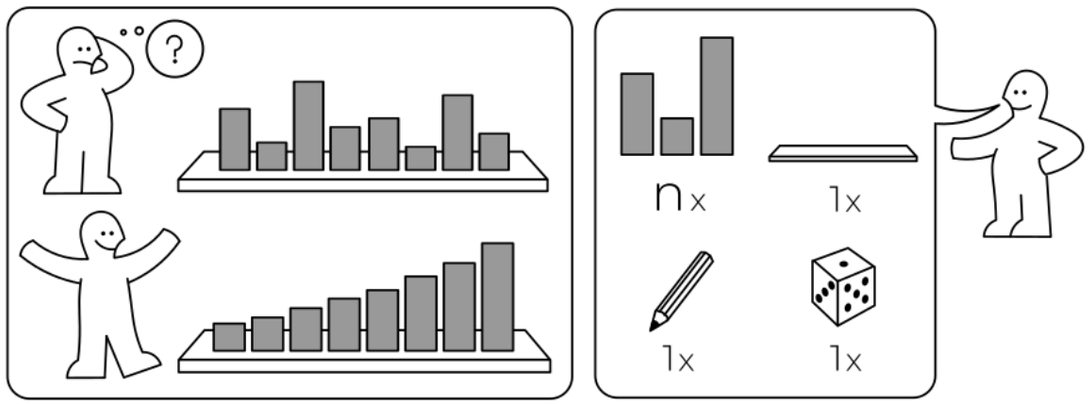
4

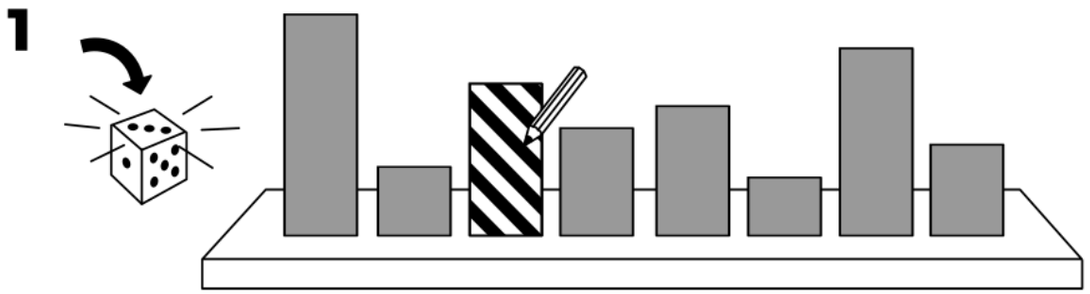


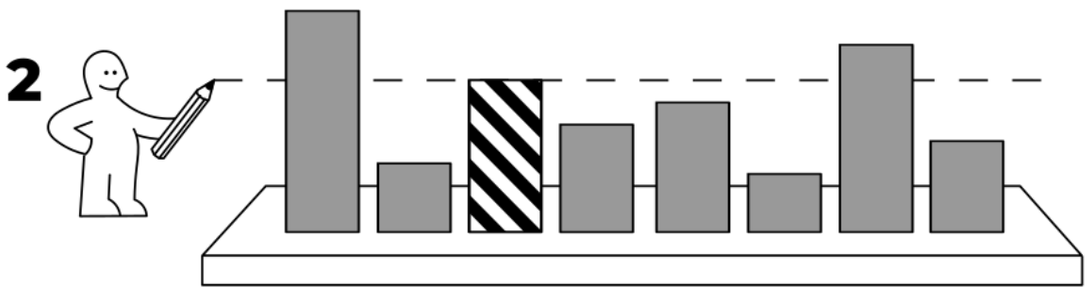
KVICK SÖRT

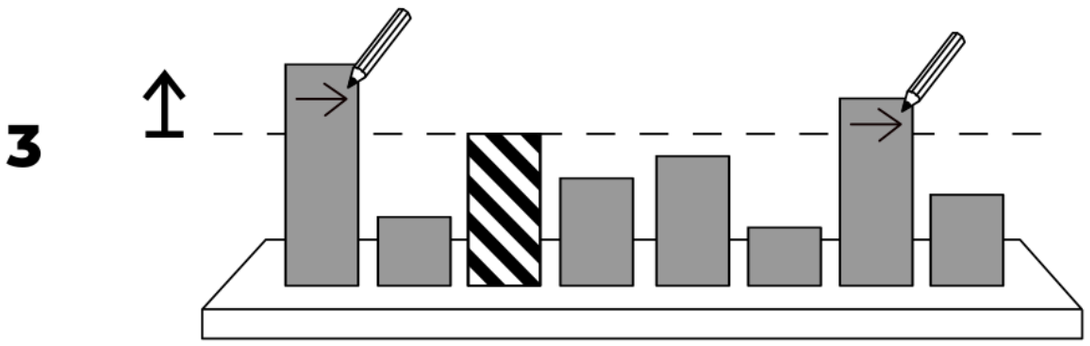
idea-instructions.com/quick-sort/
v1.2, CC by-nc-sa 4.0

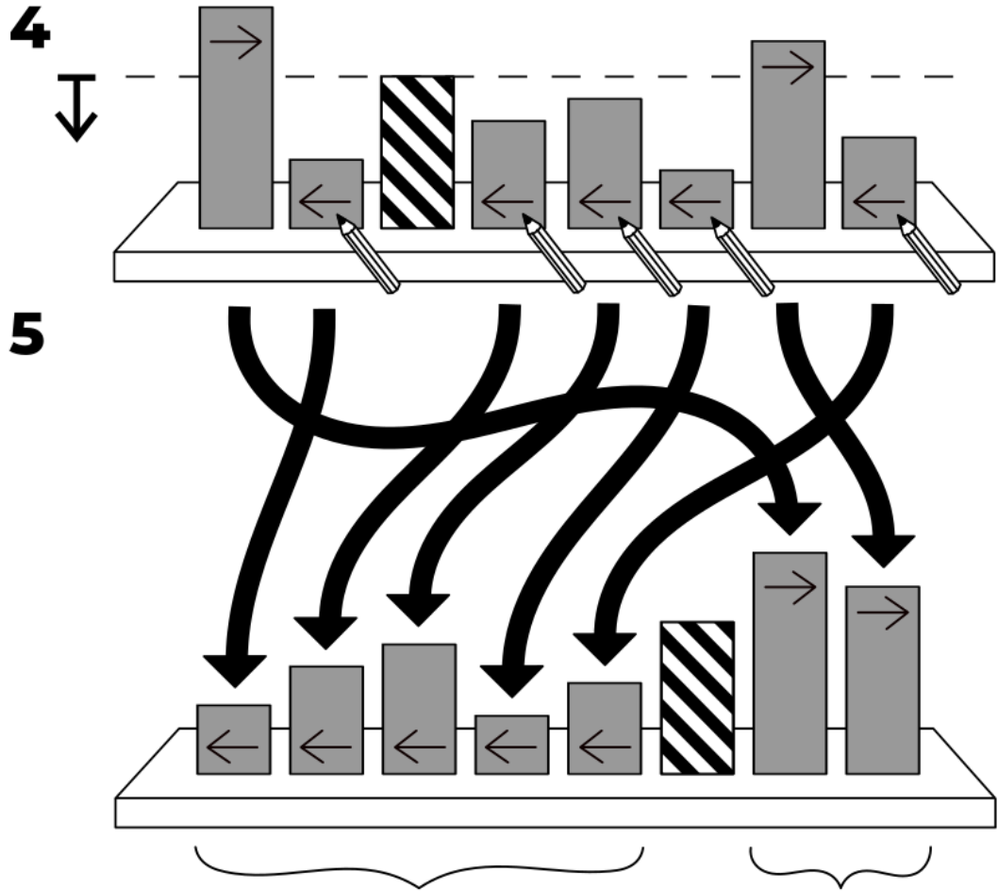
IDEA

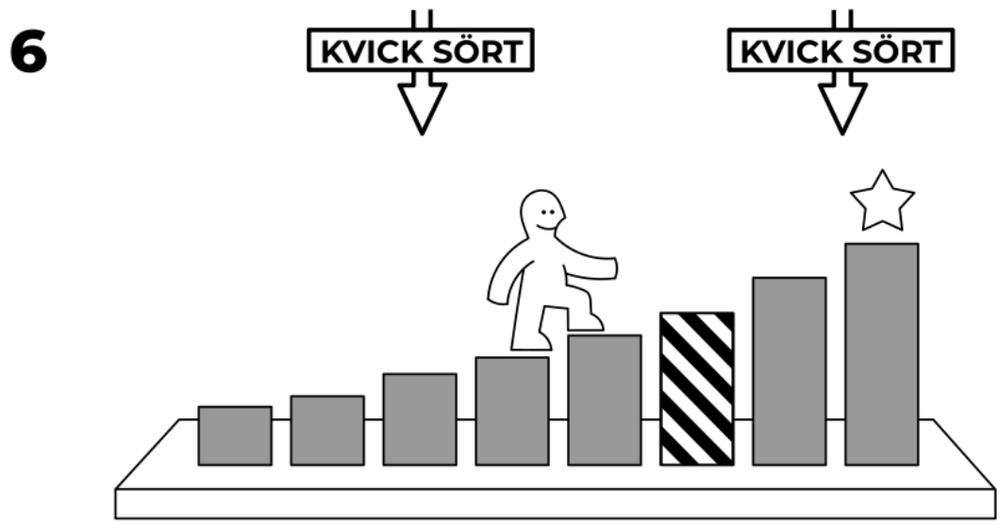






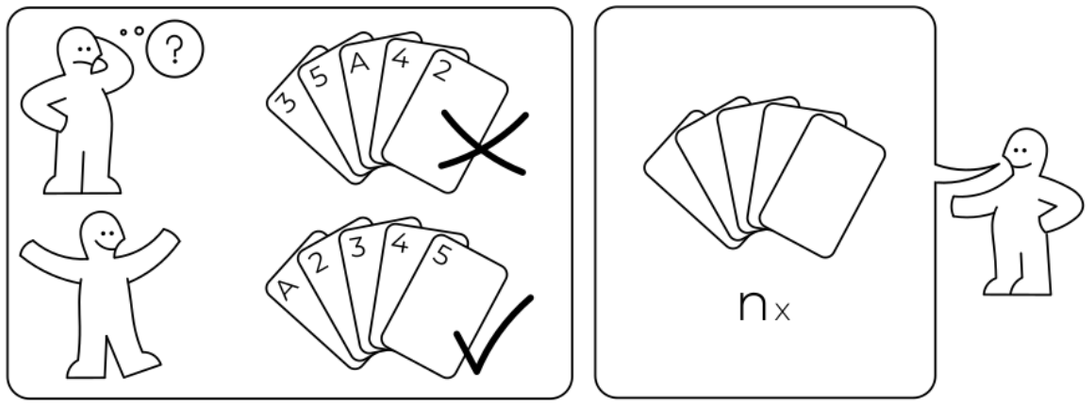


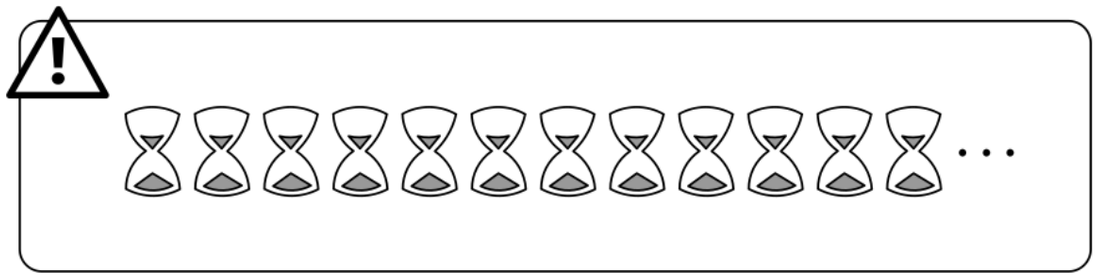




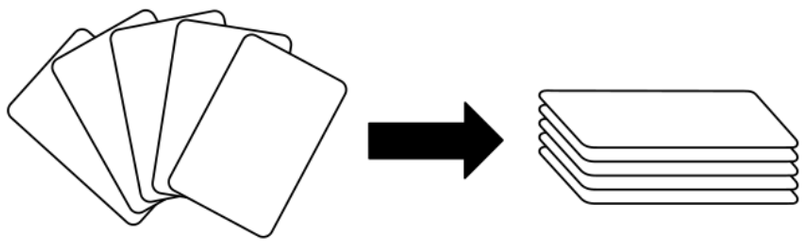
BOGO SÖRT

idea-instructions.com/bogo-sort/
v1.2, CC by-nc-sa 4.0



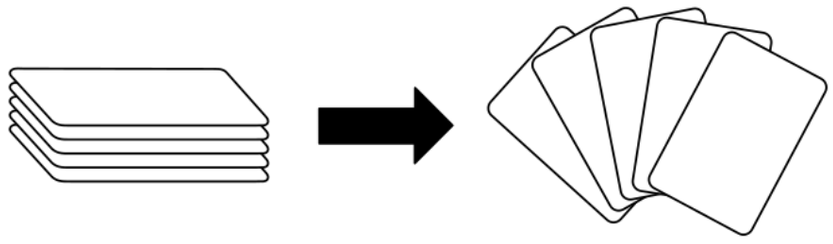


1

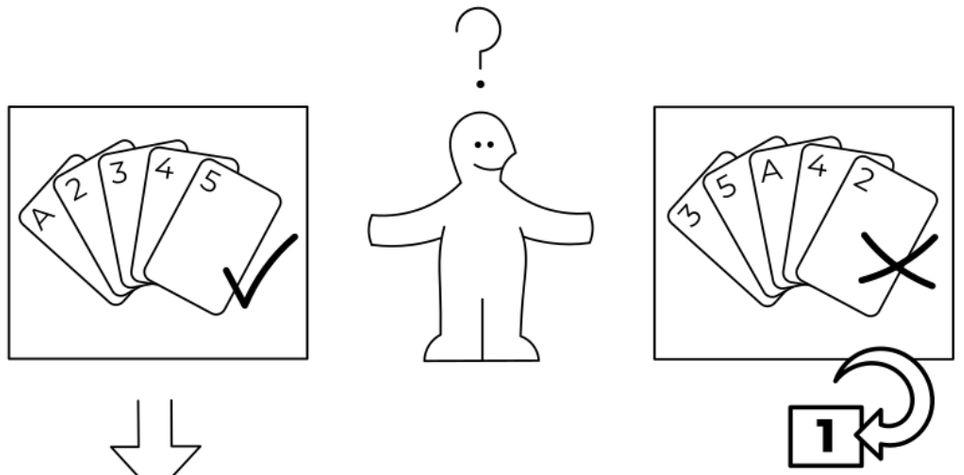


2

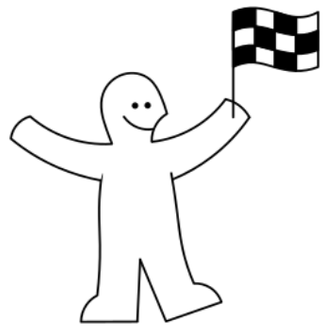
3

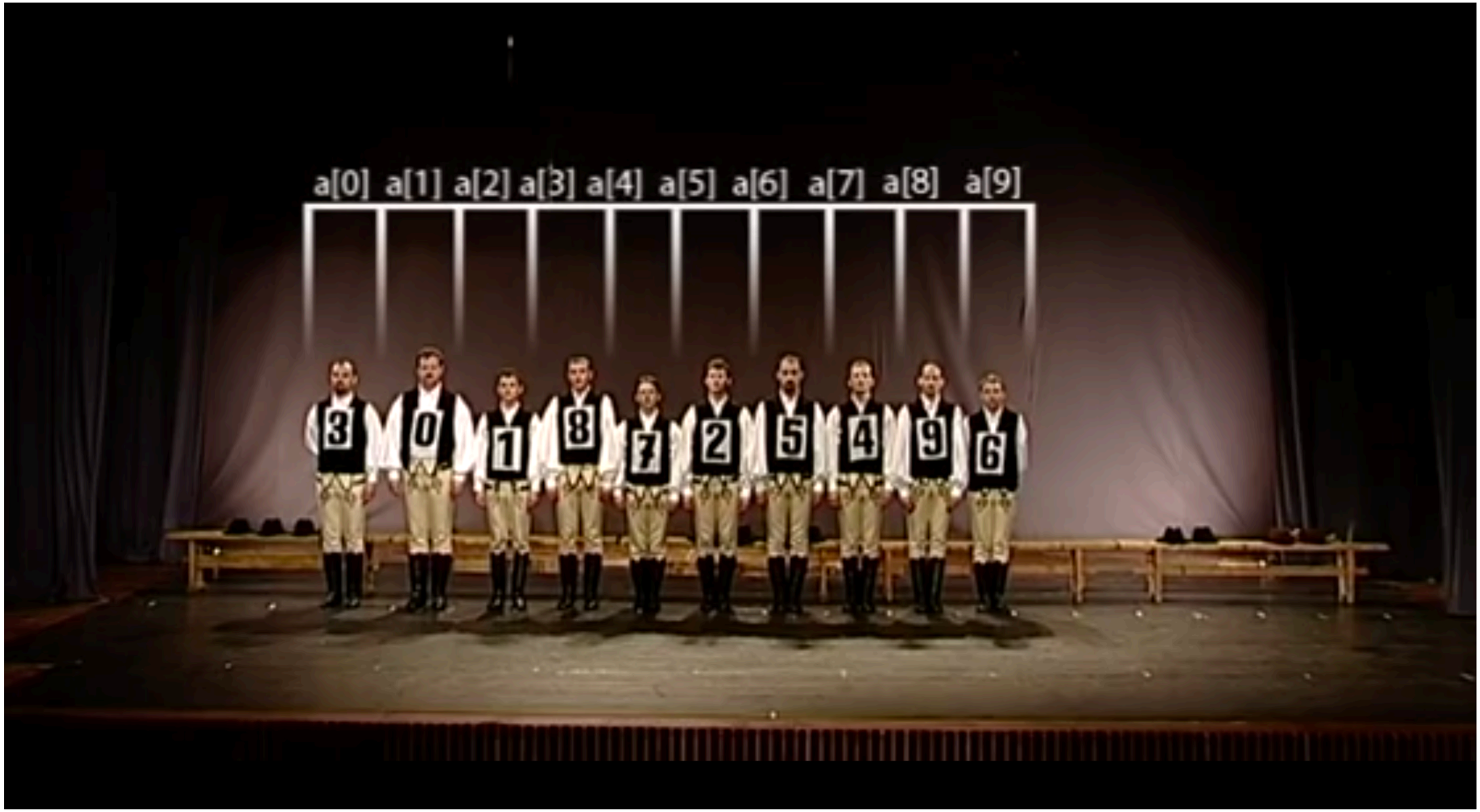


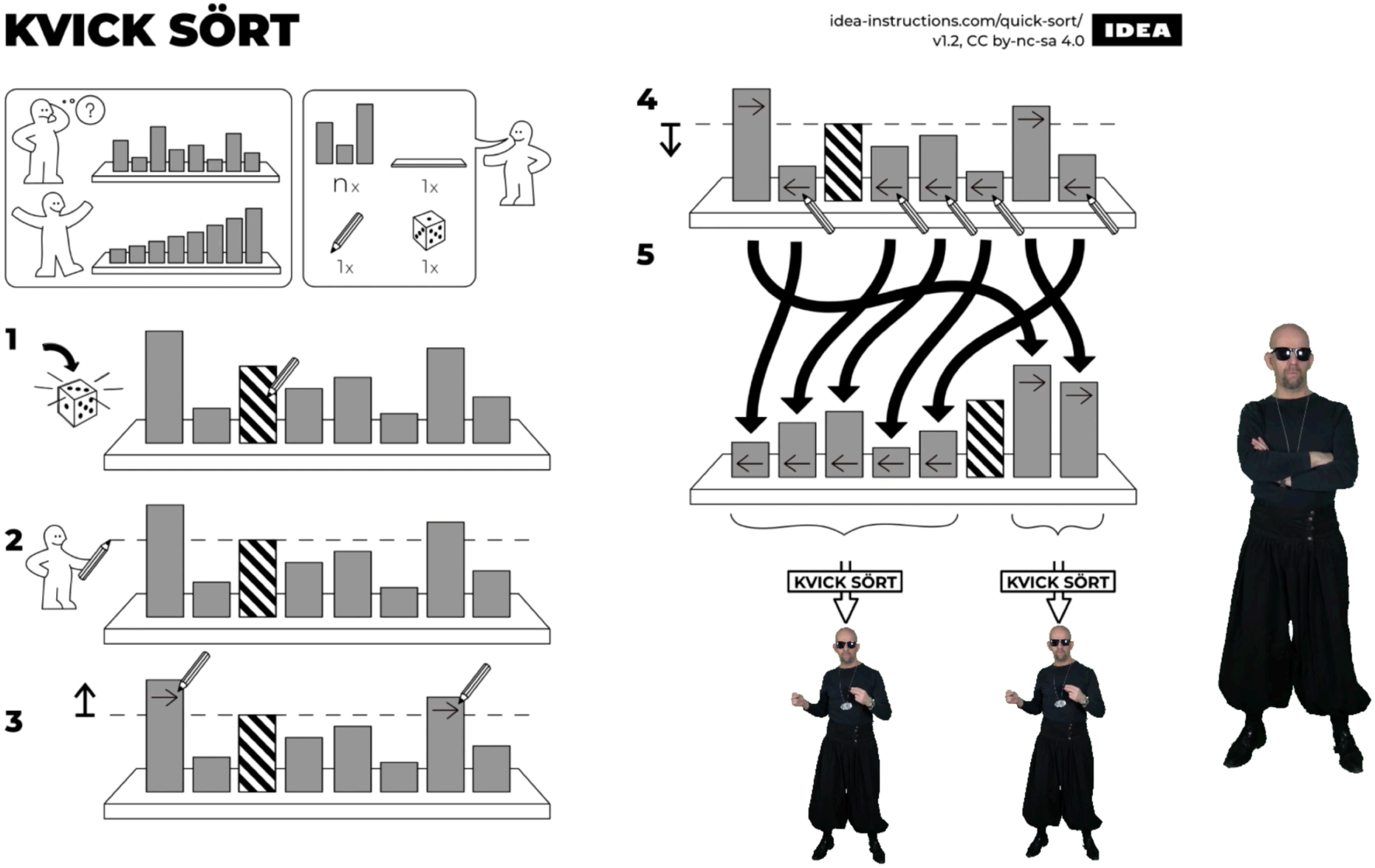
4



5




a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9]
3 0 1 8 7 2 5 4 9 6


KVICK SÖRT
idea-instructions.com/quick-sort/
v1.2, CC by-nc-sa 4.0
IDEA
n x
1x
1x
1x
1
2
3
4
5
KVICK SÖRT
KVICK SÖRT

Ausblick!

Technische Universität Braunschweig

Netzwerkalgorithmen

Linda Kleist + Arne Schmidt

Netzwerke

Stromnetz

Netzwerke

Stromnetz

Netzwerke

Stromnetz

Netzwerke

Stromnetz

Fragen:

Wie viel km Kabel werden benötigt, um alle Städte mit Strom zu versorgen?

Nebenbedingungen:

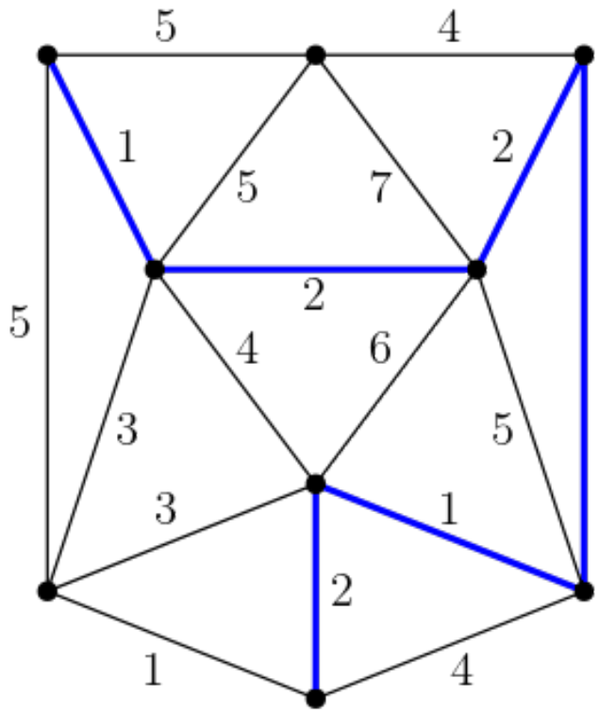
- Relaisstationen möglich?
- Geometrie?

Wie berechnet man das?

Netzwerke

Straßennetz

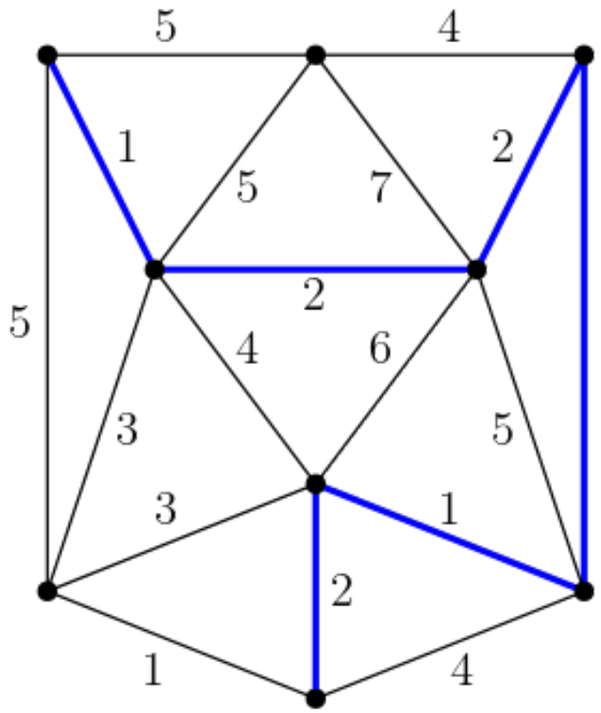
Netzwerke



Straßennetz

Netzwerke

Straßennetz



Frage:
Wie kommt man von A nach B?

Welche Nebenbedingungen existieren?
- Zeit?
- Weglänge?

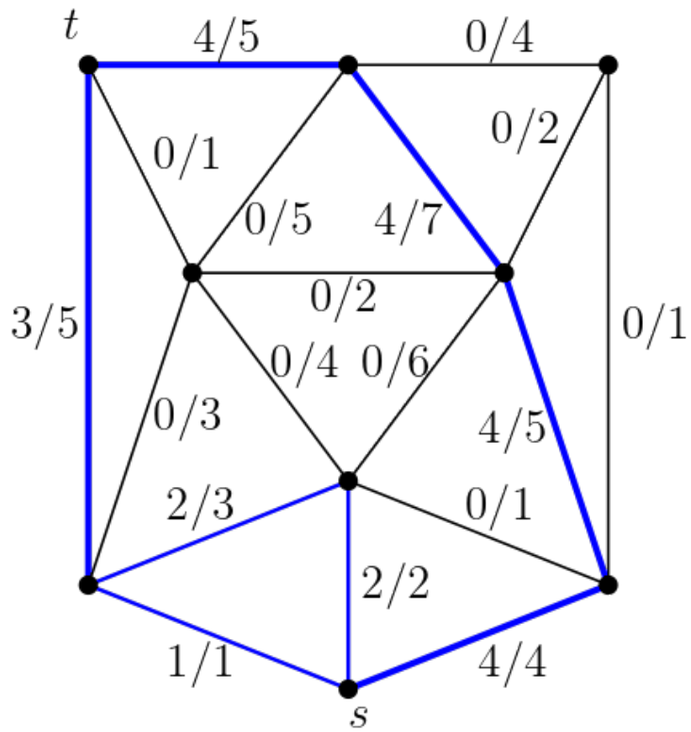
Wie berechnet man das?

Netzwerke

Pipelinenetz

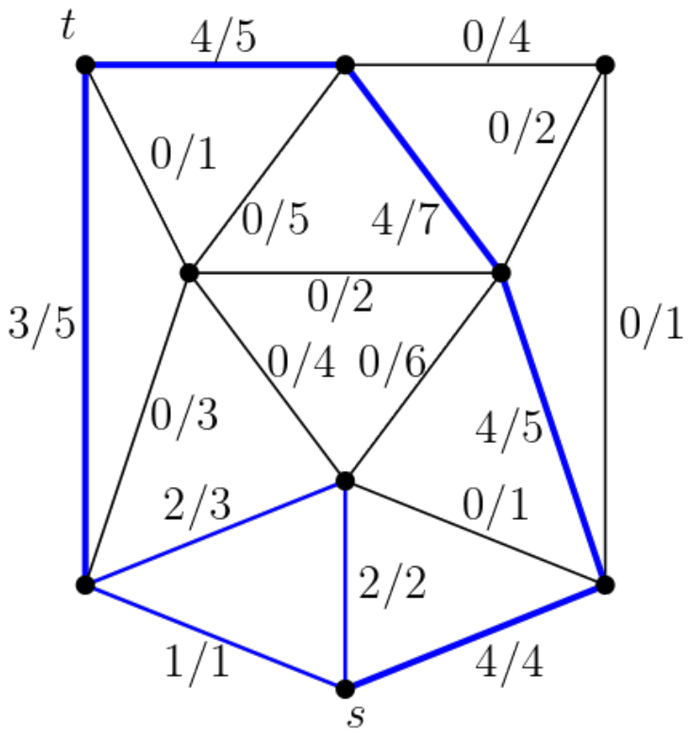
Netzwerke

Pipelinenetz



Netzwerke

Pipelinenetz

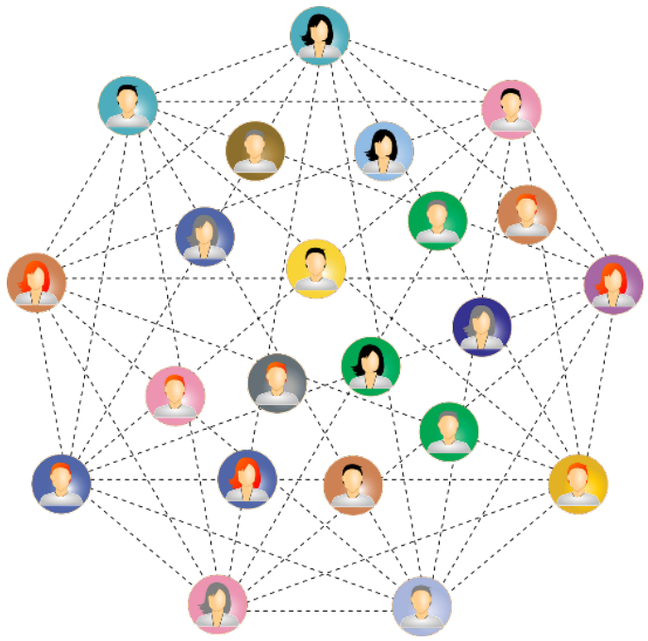


Frage:
Wie viel kann von A nach B transportiert werden?

Welche Nebenbedingungen existieren?
- Kapazität
- Maximaler Zufluss/ Abfluss

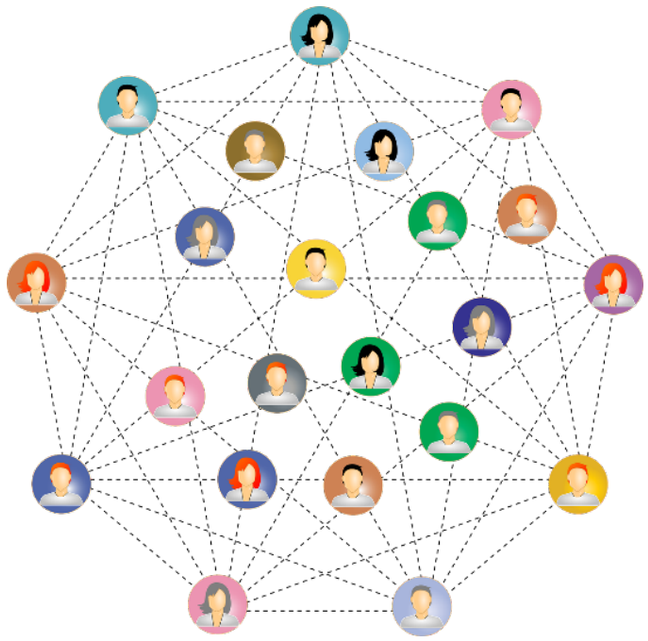
Wie berechnet man das?

Netzwerke

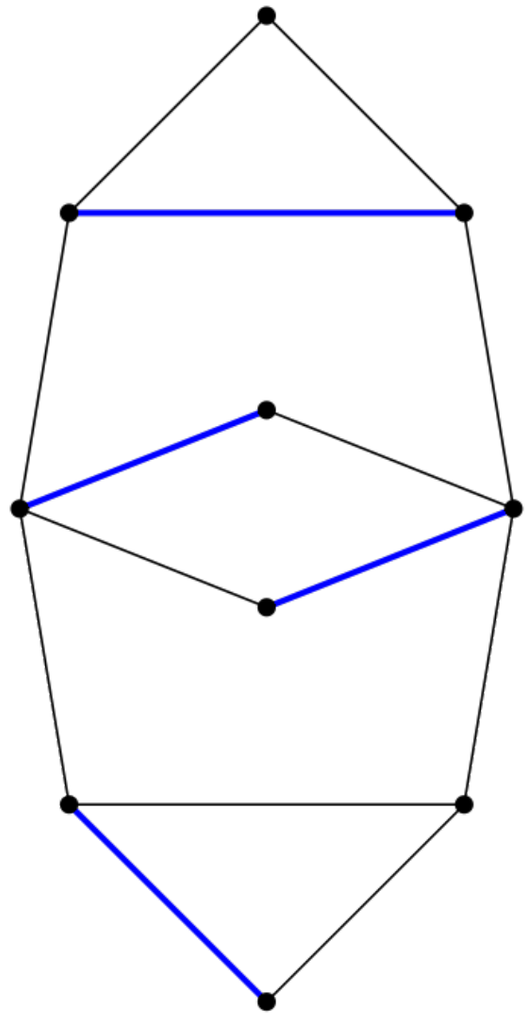


Soziales Netzwerk

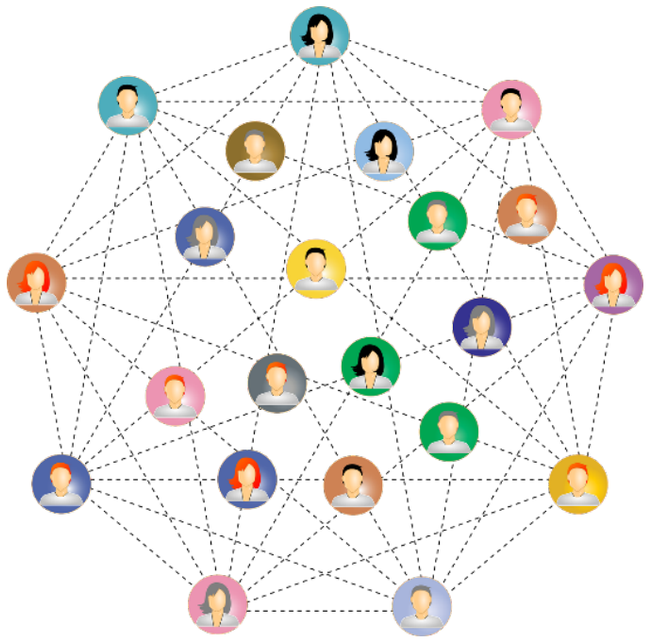
Netzwerke



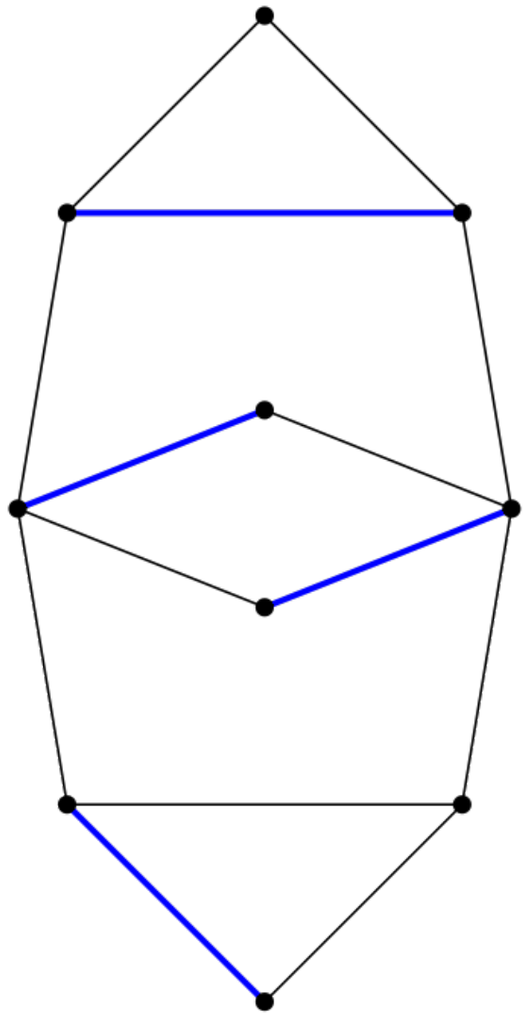
Soziales Netzwerk



Netzwerke



Soziales Netzwerk



Frage:
Kann jeder einen Arbeitspartner finden?

Welche Nebenbedingungen existieren?

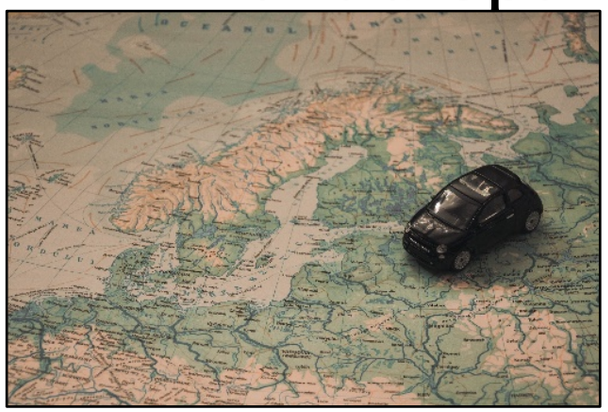
- Typ A nur mit Typ B?
- Jeder mit jedem?

Wie berechnet man das?

Inhalt der Vorlesung

<u>Kapitel 2:</u>

Minimal aufspannende Bäume

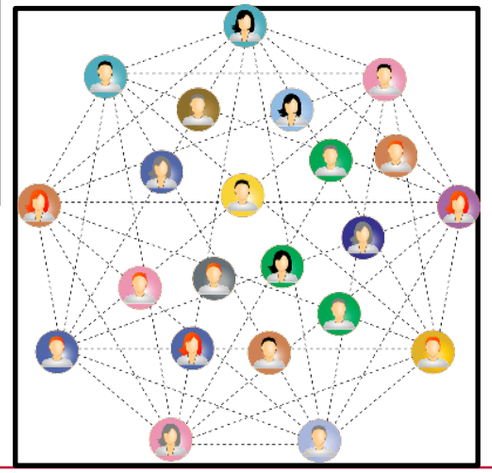


<u>Kapitel 3:</u>

Kürzeste Wege

<u>Kapitel 4:</u>

Maximale Flüsse



<u>Kapitel 5:</u>

Matching

Algorithmen und Datenstrukturen 2

Sommersemester 2021

Startseite | Vorlesungen | Organisation | Kapitel | Kontakt | Archiv

Startseite

Startseite / Algorithmen und Datenstrukturen 2

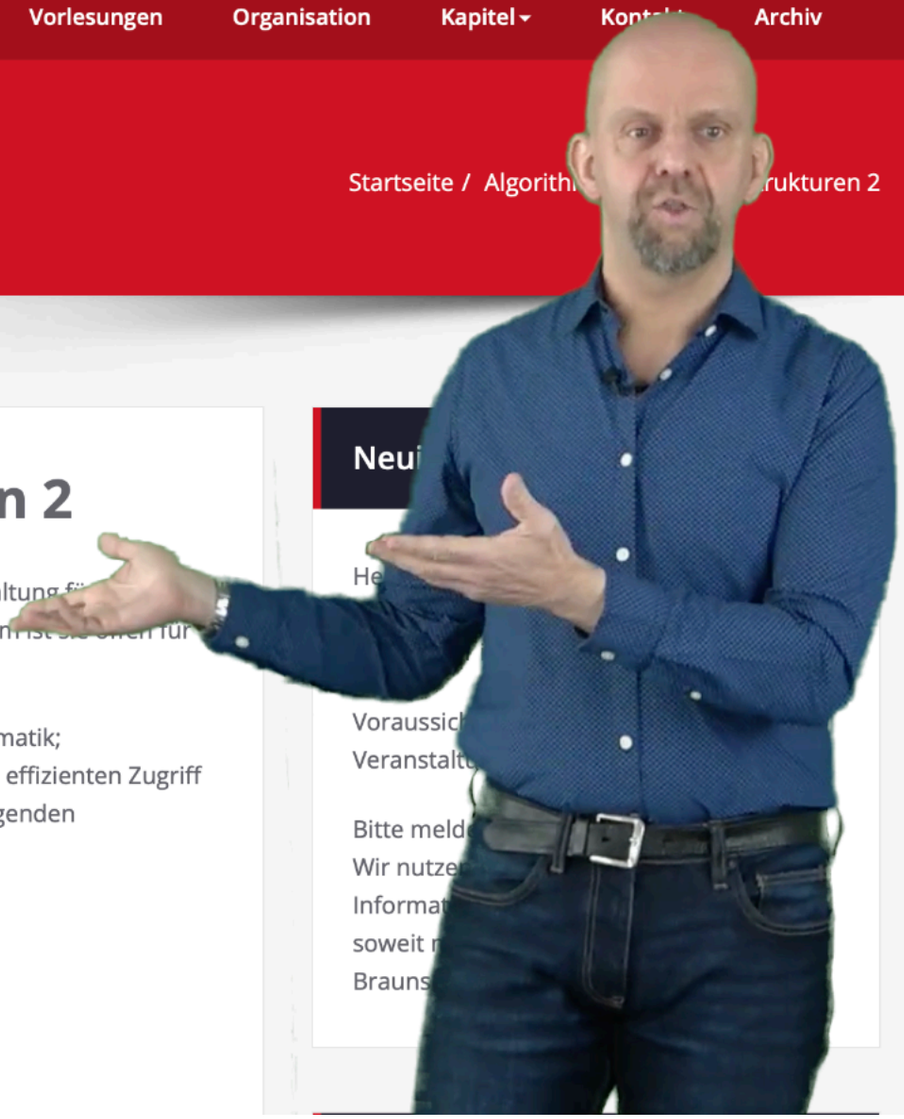
Algorithmen und Datenstrukturen 2

Die Vorlesung **Algorithmen und Datenstrukturen 2** ist eine Wahlpflichtveranstaltung für der Informatik, Wirtschaftsinformatik, Informations- und Systemtechnik; außerdem ist sie offen für interessierte Studierende anderer Studiengänge.

Algorithmen sind das methodische Herz der theoretischen und praktischen Informatik; Datenstrukturen ermöglichen die effiziente Umsetzung von Algorithmen und den effizienten Zugriff auf Input- und Outputdaten. In dieser weiterführenden Vorlesung werden die folgenden grundlegenden Begriffe erarbeitet:

- Elementare Aspekte zu Heuristiken
- Exakte Verfahren: Dynamic Programming, Branch-and-Bound
- Approximationsalgorithmen
- Komplexitätsaspekte

Neu



Technische Universität Braunschweig


TUNIGHT
EG-402
LehrLEO

Vielen Dank!

Vielen Dank!

s.fekete@tu-bs.de