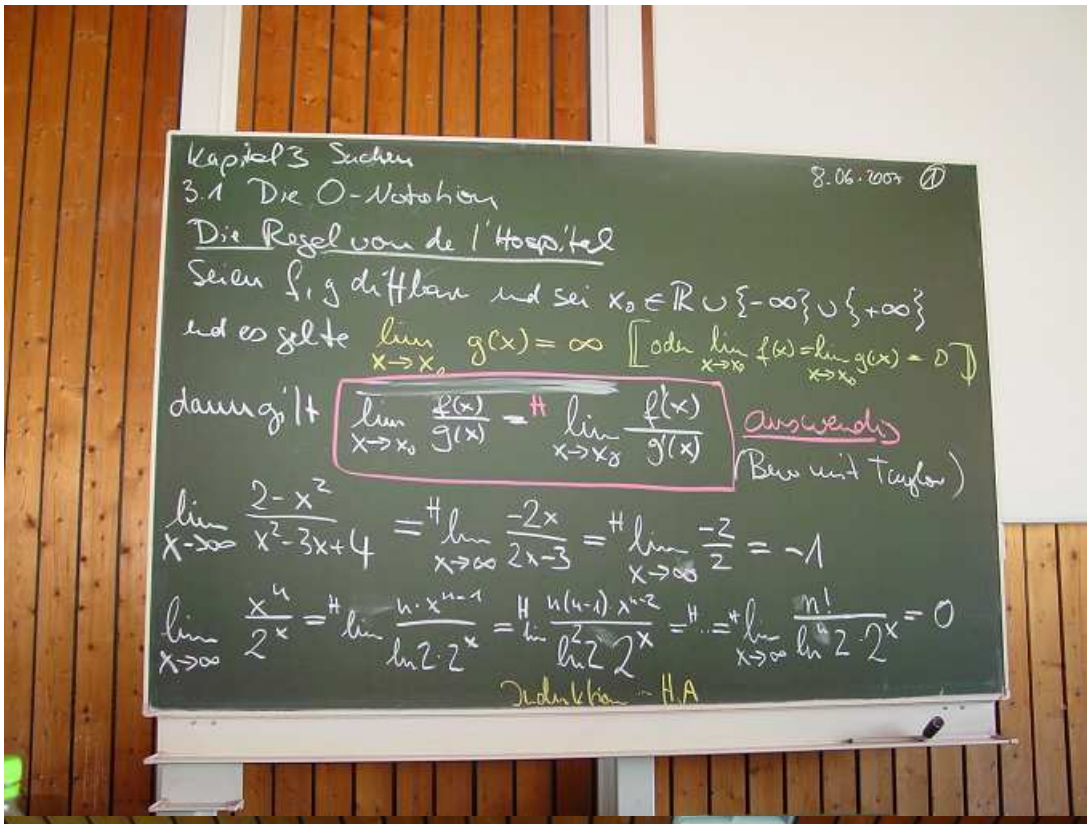
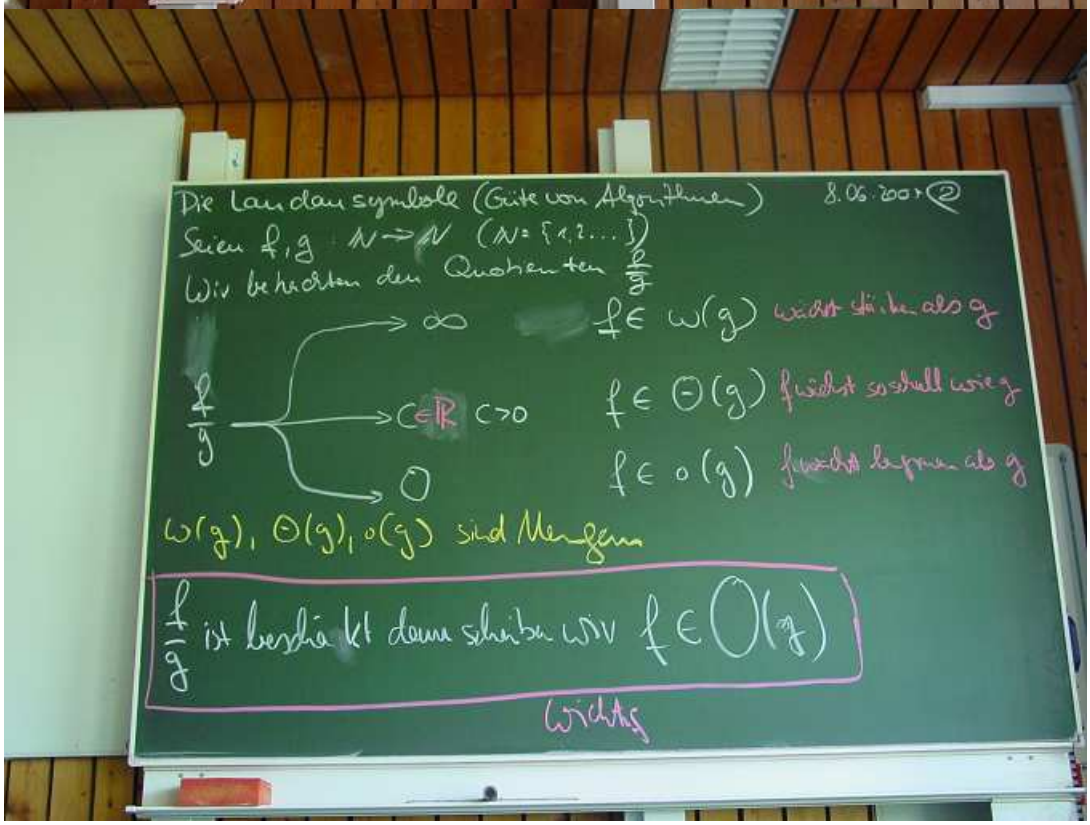
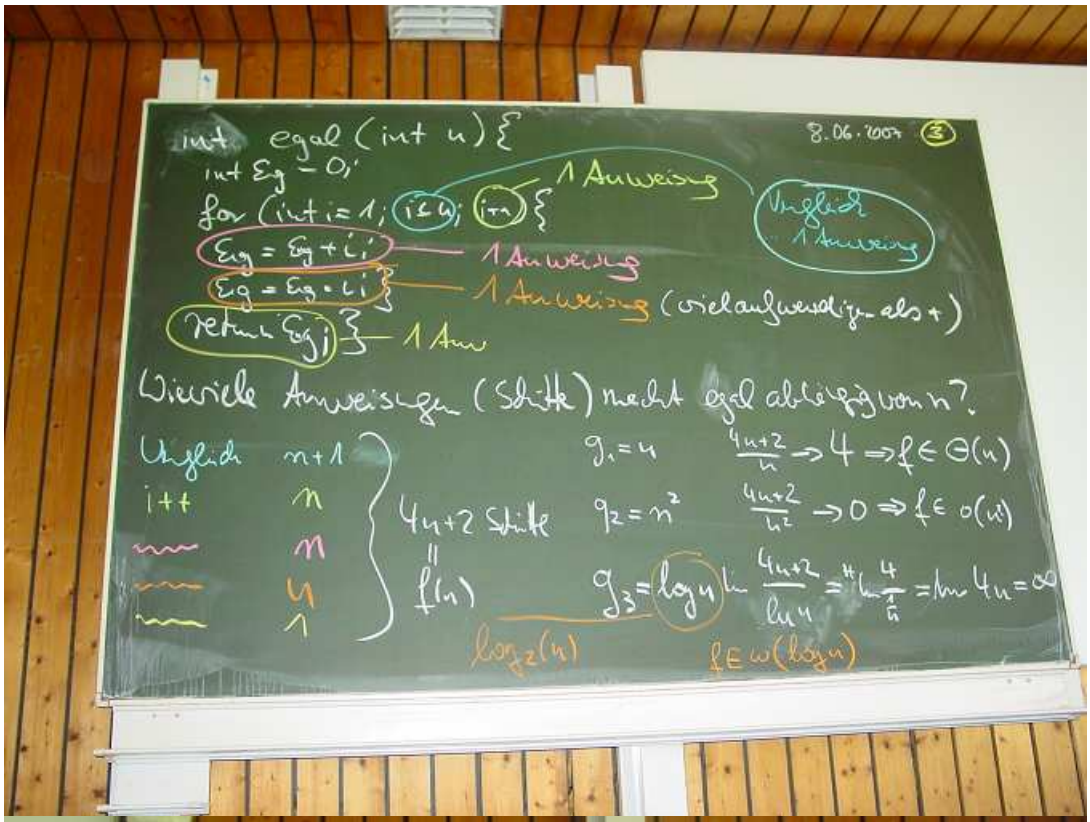
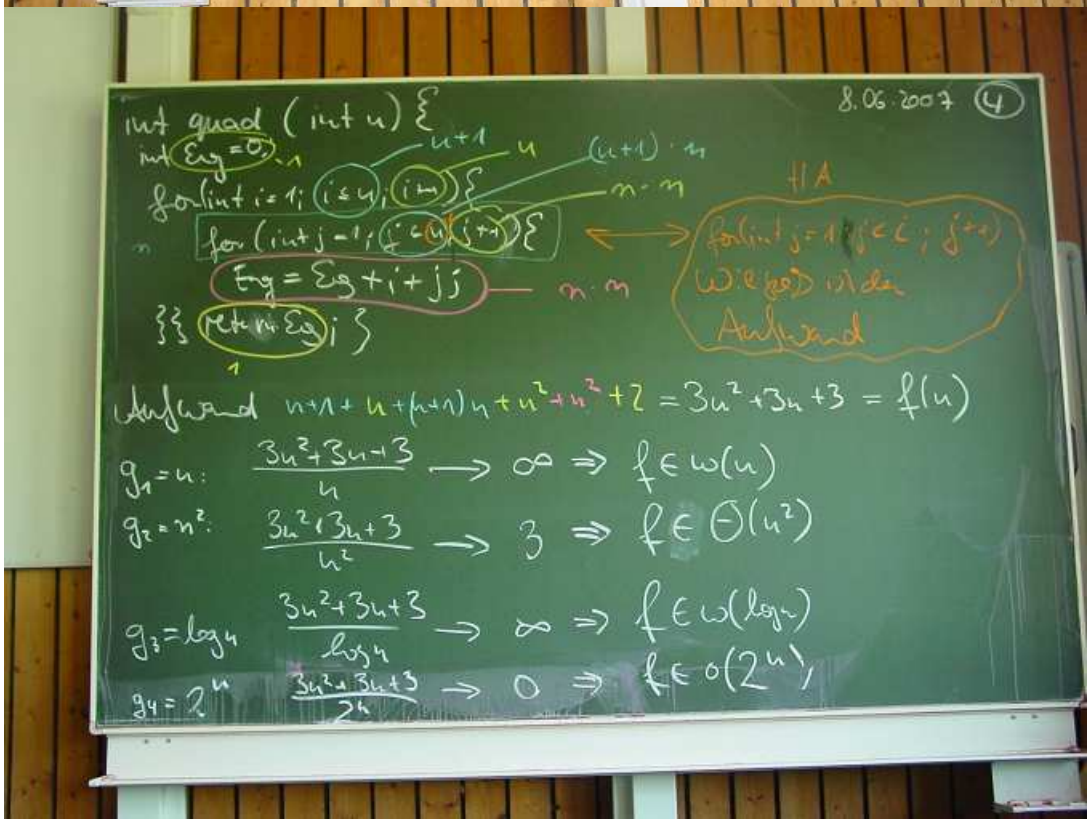
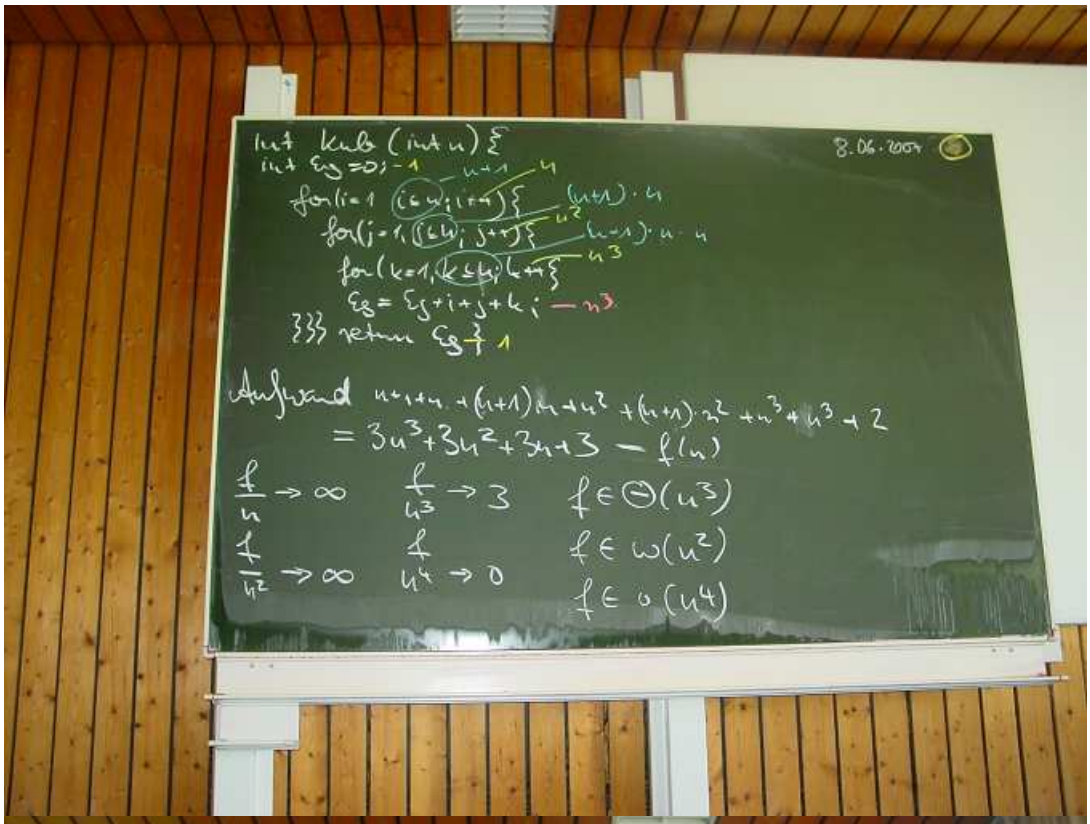
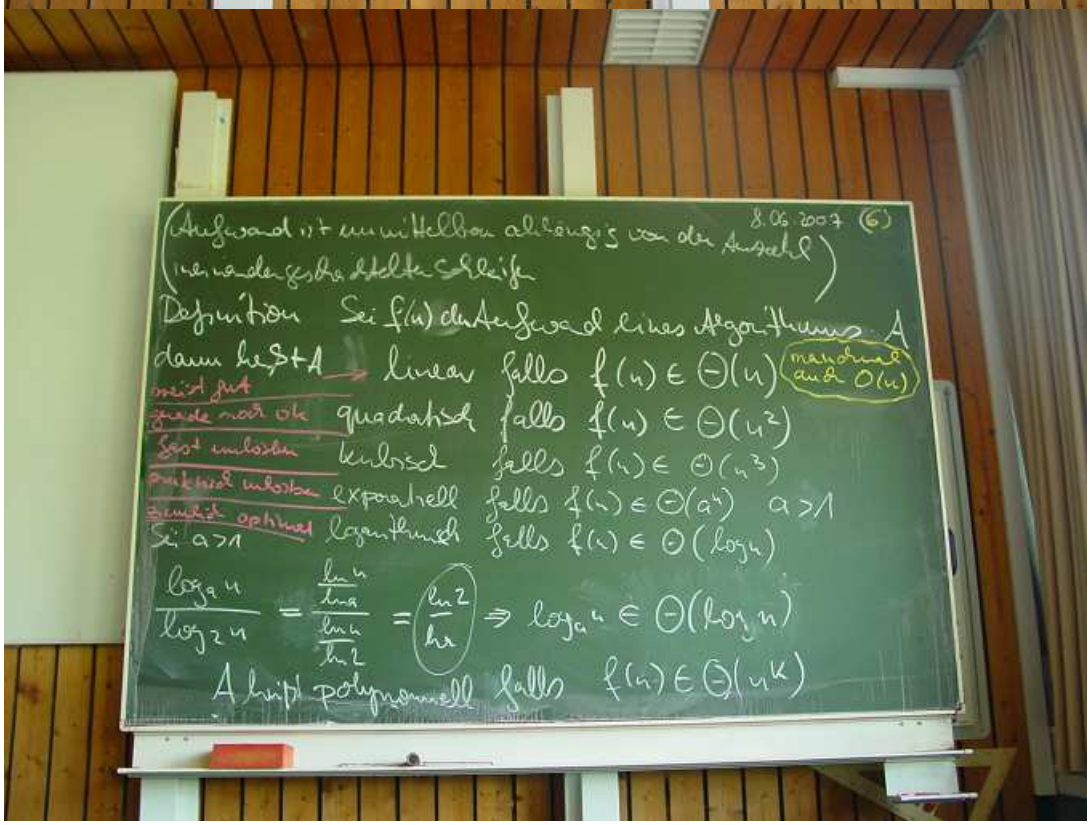

Kapitel 3 Suchen
8.06.2007 ①
3.1 Die O-Notation
Die Regel von de l'Hospital
Seien f, g diffbar und sei $x_0 \in \mathbb{R} \cup \{-\infty\} \cup \{+\infty\}$
und es gelte $\lim_{x \to x_0} g(x) = \infty$ [[oder $\lim_{x \to x_0} f(x) = \lim_{x \to x_0} g(x) = 0$]]
dann gilt $\lim_{x \to x_0} \frac{f(x)}{g(x)} \overset{H}{=} \lim_{x \to x_0} \frac{f'(x)}{g'(x)}$ anwendbar
(Beweis mit Taylor)
$\lim_{x \to \infty} \frac{2 - x^2}{x^2 - 3x + 4} \overset{H}{=} \lim_{x \to \infty} \frac{-2x}{2x - 3} \overset{H}{=} \lim_{x \to \infty} \frac{-2}{2} = -1$
$\lim_{x \to \infty} \frac{x^4}{2^x} \overset{H}{=} \lim \frac{4 \cdot x^{4-1}}{\ln 2 \cdot 2^x} \overset{H}{=} \lim \frac{4(4-1)x^{4-2}}{\ln^2 2 \, 2^x} \overset{H}{=} \ldots \overset{H}{=} \lim_{x \to \infty} \frac{4!}{\ln^4 2 \cdot 2^x} = 0$
Induktion - H.A

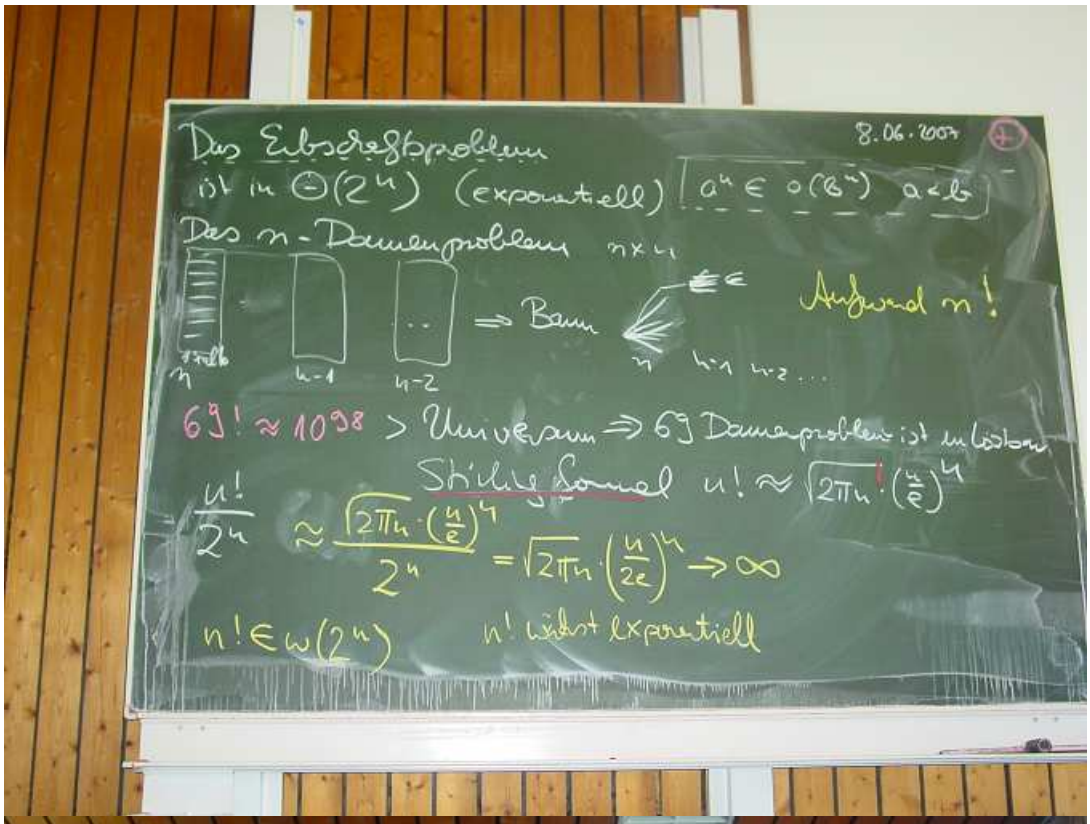

Die Landau symbole (Güte von Algorithmen) 8.06.2007 ②
Seien $f, g : \mathbb{N} \to \mathbb{N}$ ($\mathbb{N} = \{1, 2 \ldots\}$)
Wir betrachten den Quotienten $\frac{f}{g}$
$\frac{f}{g} \to \infty$ $f \in \omega(g)$ wächst stärker als g
$\frac{f}{g} \to c \in \mathbb{R}$, $c > 0$ $f \in \Theta(g)$ f wächst genauso schnell wie g
$\frac{f}{g} \to 0$ $f \in o(g)$ f wächst langsamer als g
$\omega(g)$, $\Theta(g)$, $o(g)$ sind Mengen
$\frac{f}{g}$ ist beschränkt dann schreiben wir $f \in O(g)$
Wichtig

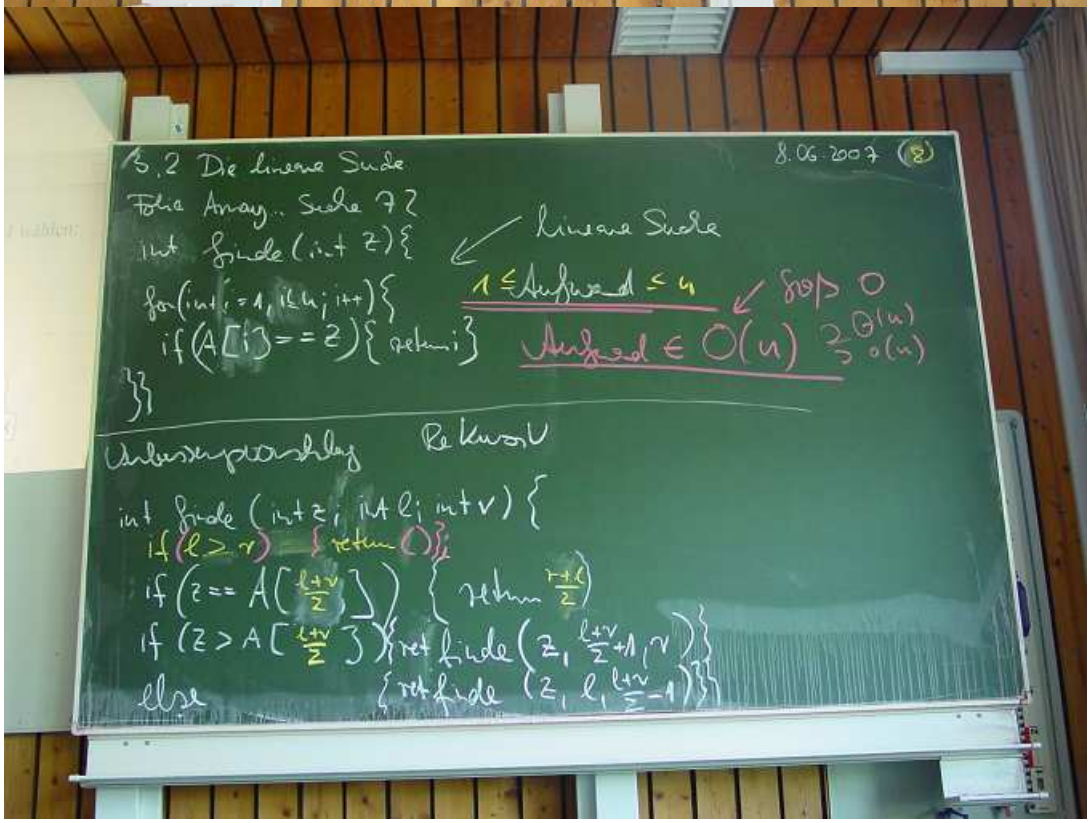

int egal (int n) {
8.06.2007 (3)
int Erg = 0;
for (int i = 1; i ≤ n; i++) {
1 Anweisung
Ungleich 1 Anweisung
Erg = Erg + i;
1 Anweisung
Erg = Erg · i;
1 Anweisung (vielaufwendiger als +)
return Erg; } 1 Anw
Wieviele Anweisungen (Schritte) macht egal abhängig von n?
Ungleich n+1
i++ n
n
n
1
4n+2 Schritte = f(n)
g1 = n
4n+2 / n → 4 ⇒ f ∈ Θ(n)
g2 = n²
4n+2 / n² → 0 ⇒ f ∈ o(n)
g3 = log n
4n+2 / log n = ∞ / (1/n) = lim 4n = ∞
log2(n)
f ∈ ω(log n)

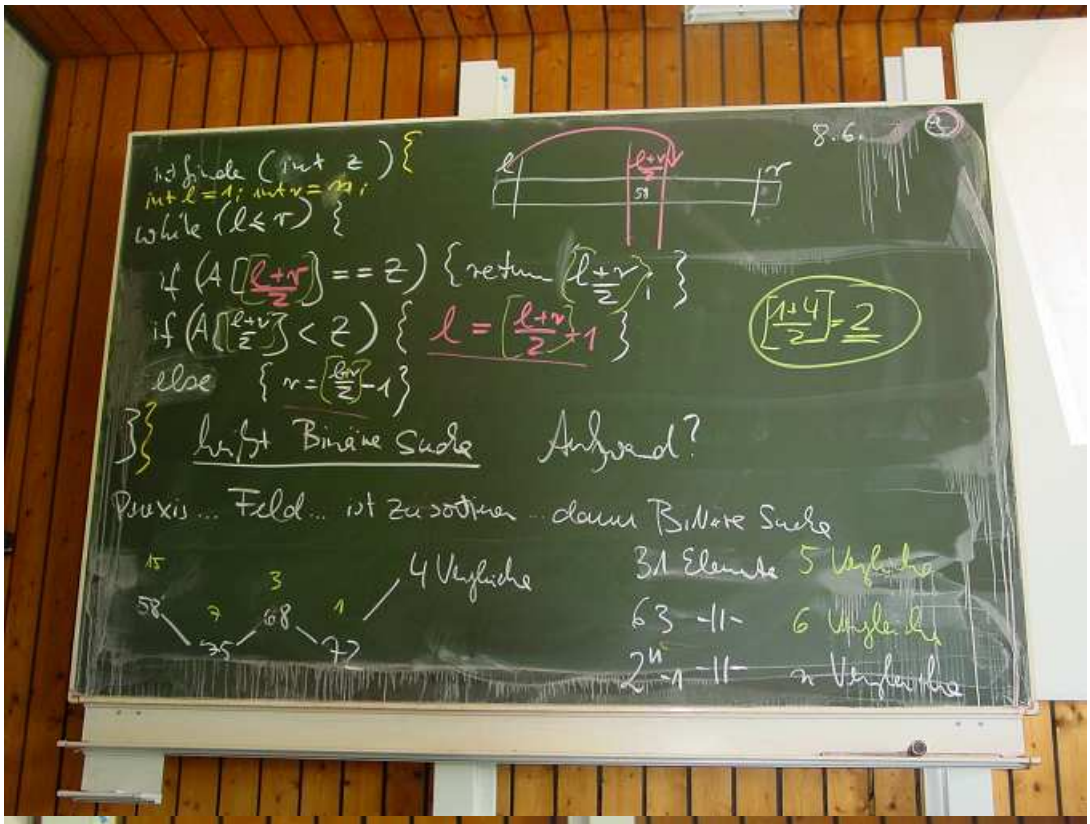

int quad (int n) {
8.06.2007 (4)
int Erg = 0; 1
for (int i = 1; i ≤ n; i++) {
n+1
n
for (int j = 1; j ≤ n; j++) {
(n+1)·n
n·n
Erg = Erg + i + j;
n·n
return Erg; }
1
+1 A
for (int j = 1; j ≤ i; j++)
Weg fällt in den Aufwand
Aufwand n+1+n+(n+1)n+n²+n²+2 = 3n²+3n+3 = f(n)
g1 = n: (3n²+3n+3)/n → ∞ ⇒ f ∈ ω(n)
g2 = n²: (3n²+3n+3)/n² → 3 ⇒ f ∈ Θ(n²)
g3 = log n: (3n²+3n+3)/log n → ∞ ⇒ f ∈ ω(log n)
g4 = 2ⁿ: (3n²+3n+3)/2ⁿ → 0 ⇒ f ∈ o(2ⁿ)


8.06.2007
int kub (int n) {
int sg = 0; - 1
for(i=1 (<n; i++) { n+1 n
for(j=1, (<n; j++) { n² (n+1)·n
for(k=1, k<n; k++) { n³ (n+1)·n·n
sg = sg+i+j+k; — n³
}}} return sg+1
Aufwand n+1+n·(n+1)·n+n²+(n+1)·n²+n³+n³+2
= 3n³+3n²+3n+3 — f(n)
f/n → ∞ f/n³ → 3 f ∈ Θ(n³)
f/n² → ∞ f/n⁴ → 0 f ∈ ω(n²)
f ∈ o(n⁴)

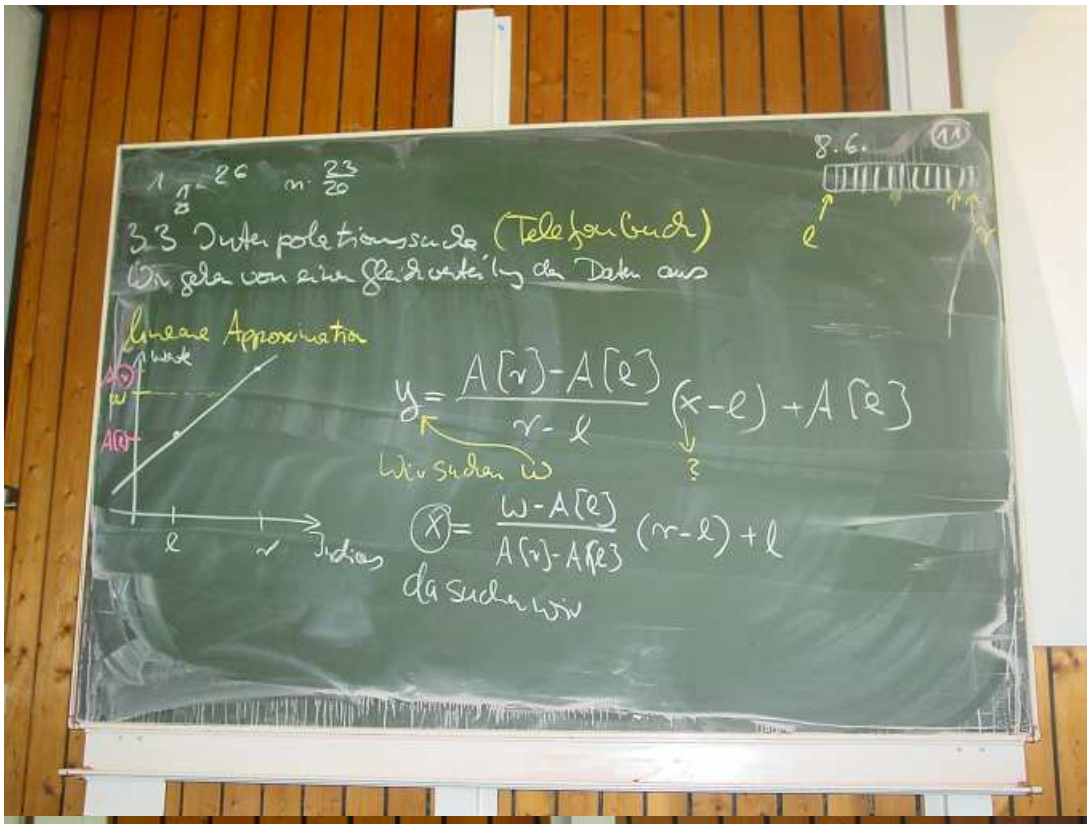

8.06.2007 (6)
(Aufwand ist unmittelbar abhängig von der Anzahl der ineinandergeschachtelten Schleifen)
Definition Sei f(n) der Aufwand eines Algorithmus A
dann heißt A → linear falls f(n) ∈ Θ(n) (manchmal auch O(n))
spricht gut
gerade noch ok quadratisch falls f(n) ∈ Θ(n²)
fast unlösbar kubisch falls f(n) ∈ Θ(n³)
praktisch unlösbar exponentiell falls f(n) ∈ Θ(aⁿ) a > 1
theoretisch optimal logarithmisch falls f(n) ∈ Θ(log n)
Sei a > 1
log_a n / log_2 n = (ln n / ln a) / (ln n / ln 2) = ln 2 / ln a ⇒ log_a n ∈ Θ(log n)
A heißt polynomiell falls f(n) ∈ Θ(n^k)

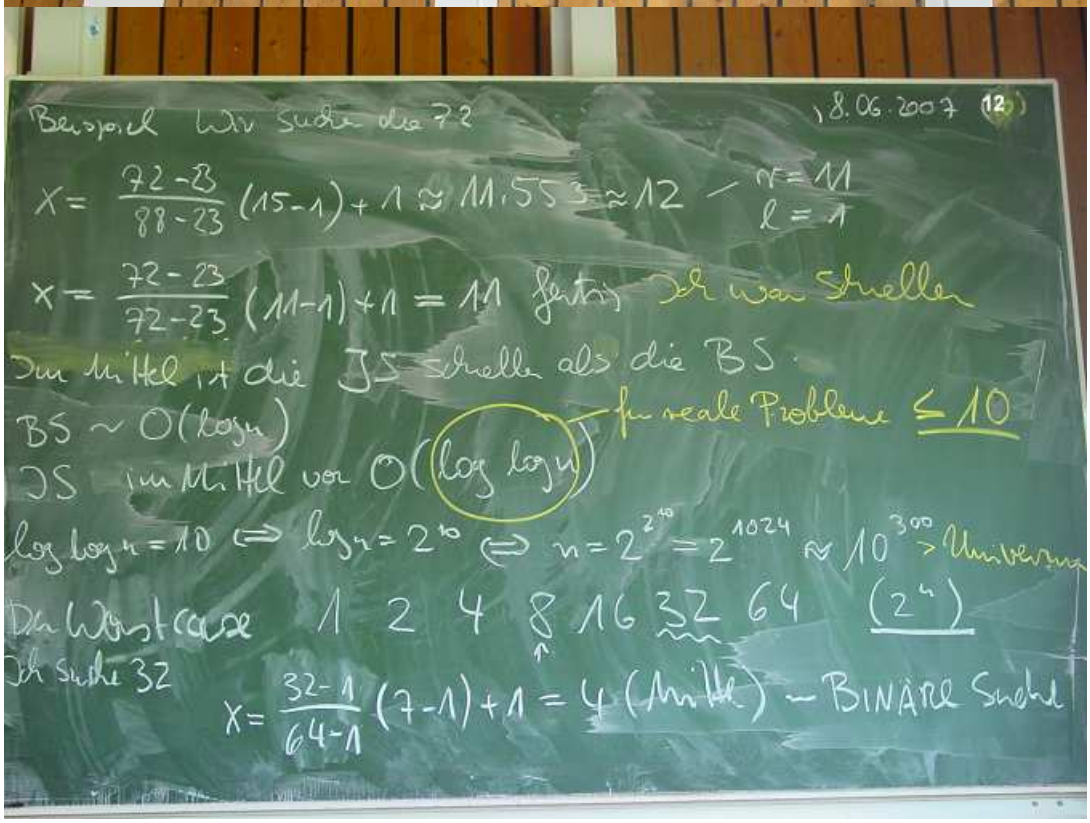

Das Erbsregalproblem
8.06.2007
ist in $\Theta(2^n)$ (exponentiell) $a^n \in o(b^n)$ $a<b$
Das n-Damenproblem $n \times n$
Aufwand n!
$\Rightarrow$ Baum
n n-1 n-2 ...
$69! \approx 10^{98}$ > Universum $\Rightarrow$ 69 Damenproblem ist unlösbar
Stirling Formel $n! \approx \sqrt{2\pi n}\cdot\left(\frac{n}{e}\right)^n$
$\frac{n!}{2^n} \approx \frac{\sqrt{2\pi n}\cdot\left(\frac{n}{e}\right)^n}{2^n} = \sqrt{2\pi n}\cdot\left(\frac{n}{2e}\right)^n \to \infty$
$n! \in \omega(2^n)$ n! wächst exponentiell


5.2 Die lineare Suche
8.06.2007
Folie Array... Seite 72
int finde(int z){
for(int i=1, i≤n; i++){
if(A[i]==z){return i}
}}
lineare Suche
$1 \le$ Aufwand $\le n$
$\delta\alpha > 0$
Aufwand $\in O(n)$ $\supseteq \Theta(n)$ $\supseteq o(n)$
Verbesserungsvorschlag Rekursiv
int finde(int z, int l, int r){
if(l>r){return ();}
if(z==A[$\frac{l+r}{2}$]){return $\frac{l+r}{2}$}
if(z>A[$\frac{l+r}{2}$]){ret finde(z, $\frac{l+r}{2}$+1, r)}
else {ret finde(z, l, $\frac{l+r}{2}$-1)}}


8.6.
int finde (int z) {
int l = 1; int r = n;
while (l ≤ r) {
if (A[(l+r)/2] == z) { return (l+r)/2; }
if (A[(l+r)/2] < z) { l = (l+r)/2 + 1 }
else { r = (l+r)/2 - 1 }
}
[(1+4)/2] = 2
heißt Binäre Suche Aufwand?
Praxis ... Feld ... ist zu sortieren ... dann Binäre Suche
4 Vergleiche
31 Elemente 5 Vergleiche
63 -11- 6 Vergleiche
2^n - 1 -11- n Vergleiche


bei n = 2^k - 1 Elemente k Vergleiche (worst case) 8.06.2007 (10)
k = log(n+1)
log(n+1)
⇒ BINÄRE Suche ist ein logarithmisches Verfahren (fast opt.)
Frage nach der mittleren Suchdauer O(log n)
(Folie)
In 2^(k-1) Fällen k Schleifendurchläufe
Mittlere Suchdauer bei 2^n - 1 Elementen
Duplo Vorlesung im Netz
(Σ_{k=1}^{n} 2^(k-1) · k) / (2^n - 1) ≈ n - 1 ≙ fast der worst case
fast ein Θ und kein O mehr
=
≤


8.6.
11
l
r
3.3 Interpolationssuche (Telefonbuch)
Wir gehen von einer Gleichverteilung der Daten aus
Lineare Approximation
Index
A(r)
w
A(l)
l
r
Index
$y = \frac{A(r)-A(l)}{r-l}(x-l)+A(l)$
wir suchen w
?
$(X) = \frac{w-A(l)}{A(r)-A(l)}(r-l)+l$
da suchen wir


Beispiel Wir suchen die 72
18.06.2007
12
$X = \frac{72-8}{88-23}(15-1)+1 \approx 11.55 \approx 12$
r = 11
l = 1
$X = \frac{72-23}{72-23}(11-1)+1 = 11$ Ja, er war schneller
Im Mittel ist die IS schneller als die BS.
BS ~ O(log n)
IS im Mittel von O(log log n)
für reale Probleme ≤ 10
log log n = 10 ⇔ log n = 2^10 ⇔ n = 2^(2^10) = 2^1024 ≈ 10^300 > Universum
Der Worstcase 1 2 4 8 16 32 64 (2^n)
Ich suche 32
$X = \frac{32-1}{64-1}(7-1)+1 = 4$ (Mittel) – BINÄRE Suche

Das schlechteste

8.6. 13/1

1 4 27 256 3125 6^6 7^7 n^n

$x = \frac{6^6 \cdot 1}{7^7 - 1}(7-1) + 1 \approx 1,3 < 2$ sieht also bei 1

.... Wir suchen immer ganz vorne

Bei n^n bei Suche von $(n-1)^{n-1}$ müssen vor $(n-1)$ mal suchen

$\Rightarrow$ Im Worst case ist LS nach $n-1$ Schritten fertig $\Rightarrow O(n)$

die BS wenn benutzt nach $\log n$ Schritten fertig

Kapitel 4 Der Suchbaum

18.06.2007 1/12

4.1 Der BINÄRBAUM

BB BB BB BB

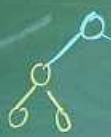
kein BB:

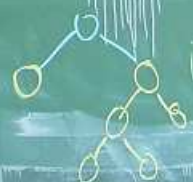
Definition (rekursiv) $\emptyset$ = leere Menge ist ein BB

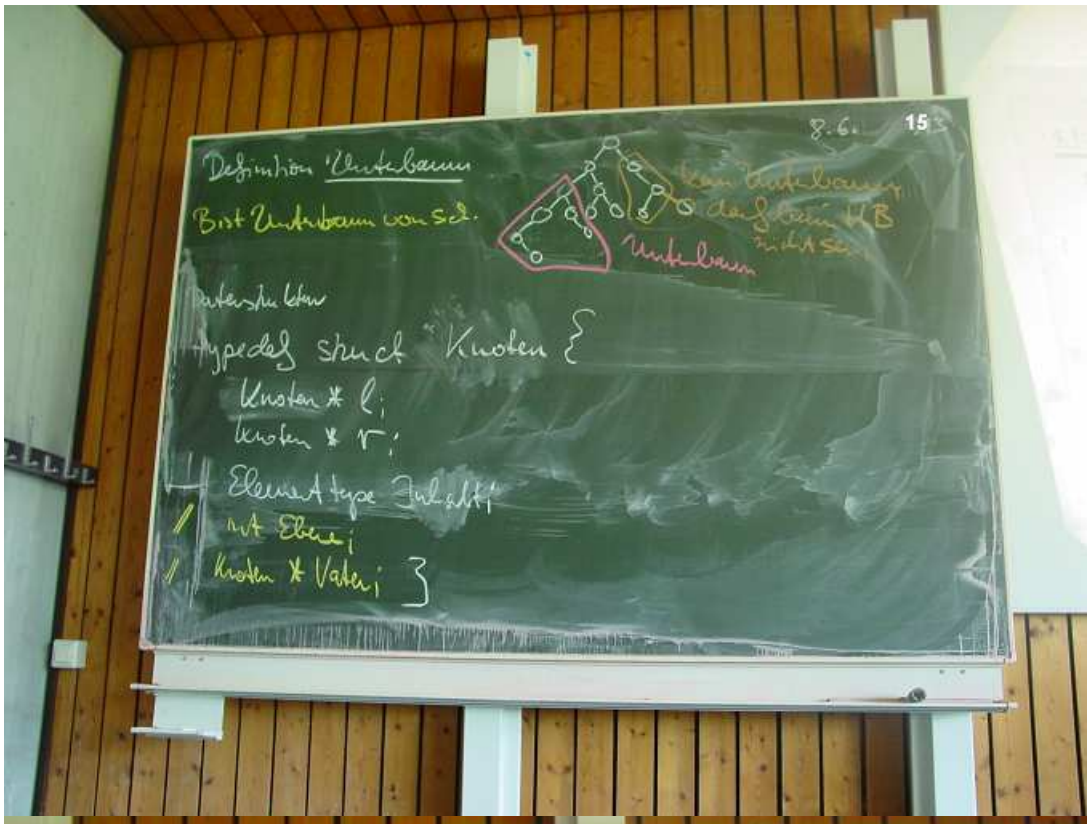
seien B_1 und B_2 BB dann ist auch ein BB

ist BB $\hat{=}$ 0 oder  (ist BB)

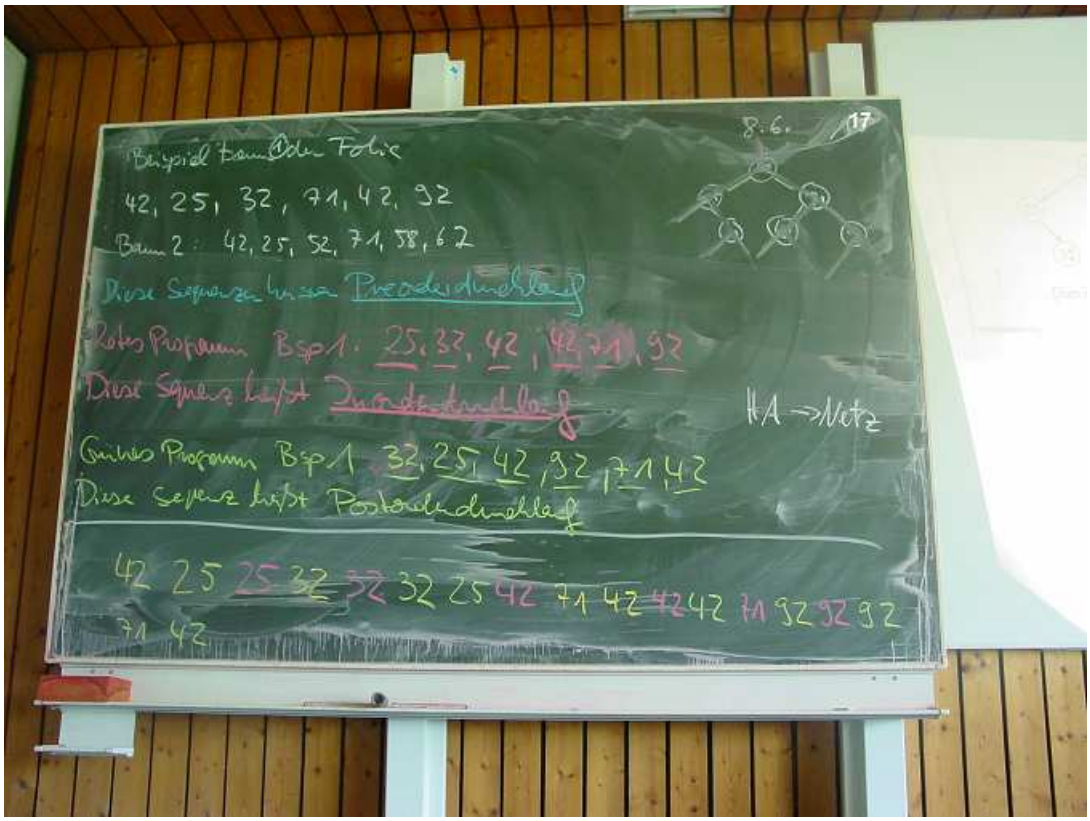
 ist BB

 ist BB

 ist BB


8.6. 15
Definition Unterbaum
Bist Unterbaum von Sk.
Unterbaum


4.2 Baum durchlaufen (Tiefensuche)
18.06.2007 16
void Durchlauf (Knoten &k){
cout << k.Inhalt; ← Preorderaktion
if (k.l != NULL){Durchlauf(k.l)}
cout << k.Inhalt ← Inorderaktion
if (k.r != NULL){Durchlauf(k.r)}
cout << k.Inhalt ← Postorderaktion
}


8.6.
17
Beispiel Baum Oder Folie
42, 25, 32, 71, 42, 92
Baum 2: 42, 25, 52, 71, 58, 62
Diese Sequenz heißt Preorderdurchlauf
Rotes Programm Bsp1: 25, 32, 42, 42, 71, 92
Diese Sequenz heißt Inorderdurchlauf
HA → Notz
Gelbes Programm Bsp1: 32, 25, 42, 92, 71, 42
Diese Sequenz heißt Postorderdurchlauf
42 25 25 32 32 32 25 42 71 42 42 42 71 92 92 92
71 42