

# ALGODATEN UE

$O_1$   $\Omega_2$   $\Theta_3$

$\log(n)$   
 $n^2$   $n^3$

---

---

---

---

# Algorithmen & Datenstrukturen

## ÜE 2 ARBEITSBLATT 2

2.1)  $T(n) = a \cdot T\left(\frac{n}{c}\right) + r(n)$   $r \in O(n^\alpha)$

- $a < c^\alpha$  :  $T \in O(n^\alpha)$
- $a = c^\alpha$  :  $T \in O(n^\alpha \cdot \log n)$
- $a > c^\alpha$  :  $T \in O(n^{\log a})$

$$T(n) = a \cdot T\left(\frac{n}{c}\right) + r(n) = T(c \cdot n) = a \cdot T(n) + r(n)$$

Aufgabe 1:

$$T(1) = 2000$$

$$T(3n) = 10 \cdot T(n) + n \cdot (\log(n))^2$$

$$a=10$$

$$c=3$$

Umformen:

$$(n \log(n^2))^2 = n^2 \cdot (\log(n^2))^2$$

$r(n) \notin O(n^2)$

Wie überprüft man dann die Funktion oben nach  $n^2$  beschränkt ist?

Schauen was schneller ist

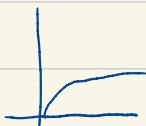
$$\frac{n^2 \cdot \log^2 n^2}{n^2} = \frac{\log^2 n^2}{1} : \text{geht gegen unendlich}$$
$$a < c^\alpha$$

$$r(n) \in O(n^3)$$

$$T \in O(n^3)$$

obere Schranke gefunden

andere obere Schranken:  $O(n^{2.5}), O(n^{2.01})$

  
log.

  
polynom

$$\frac{n^{2+\varepsilon}}{a=10, c=3} \quad \text{wobei } \varepsilon > 0 \quad r(n) \in O(n^{2+\varepsilon})$$

$$\alpha = 2 + \varepsilon$$

$$10 \quad 3^{2+\varepsilon} > = 9 + \varepsilon$$

groesser als 9 weil  $\varepsilon$  so klein

3. Fall

$$\text{Also: } T(n) \in O(n^{\log_3 10}) = (n^{2.1...})$$

$$b) a = c^\alpha \quad \text{Fall 2}$$

$$T(3n) = g \cdot T(n) + n^2$$
$$r(n) \in O(n^2)$$

$$\alpha = 2$$

$$a = 3$$

$$g = 3^2$$

$$T(n) \in O(n^2 \cdot \log n)$$

2.2

$$T_1(2n) = 8 T_1(n) + 18000 n \quad (\text{umgeformt})$$

$$c = 2$$

$$a = 8$$

$$\alpha = 1$$

$$T_1 \in O(n^3)$$

$$T_2(3n) = 2T_2(n) + \underbrace{(3n)^2}_{9n^2}$$

$$c = 3$$

$$\alpha = 2 \quad T_2 \in O(n^2)$$

$$\alpha = 2$$

$$T_3(g_n) = 3T_3(n) + (1 + \sqrt{g_n}) \quad c=9$$

$$\alpha=3$$

$$T_3 \in O(\sqrt{n} \log(n)) \quad \alpha=10,15$$

$$\text{Bsp3} \quad T'(2n) = a \cdot T'\left(\frac{n}{2}\right) + 4n^2$$

$$T(2n) = 8 \cdot T(n) + 4n^3$$

1) Laufzeit von T

◦ Fundamentalsatz anwenden: 1) Parameter

$$a=8 \quad \alpha=3$$

$$c=2$$

Fall:  $a = c^\alpha$

Daher:  $T \in (n^3 \log n)$

# Algo Daten

$$T' : c=4, a=2 \quad a > 17$$

Für  $a = 17$ :  $T' \in O(n^{\log_4(17)})$

Ziel:  $T' \in O(n^{3+\epsilon}) \rightarrow$

→ wie groß muss dann  $a$  sein?

Antwort:  $\forall a > 64$

$$\log_{10} a = 3 + \epsilon$$

also  $a$  darf max 64 sein, ab 65 ist die Funktion  $T'$  langsamer.

Bsp 4.: Matrixmultiplikation von quadratischen Matrizen

$$n \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix} \times n \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix} = n \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}$$

A                    B                    C

Wieviele Multiplikationen für 2 Werte:  $n$   
 $n^2 \cdot n = n^3$  naiver Algorithmus

$$\begin{bmatrix} n & & n & & n \\ A_{11} & | & A_{12} & & C_m \\ \hline A_{21} & | & A_{22} & & C_n \\ \hline & & & & C_{21} \\ & & & & C_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & | & B_{12} \\ \hline B_{21} & | & B_{22} \end{bmatrix} = \begin{bmatrix} C_m & C_n \\ C_{21} & C_{22} \end{bmatrix}$$

1) Zeigen Sie, dass der Algorithmus stimmt.

2) Laufzeit abschätzen

$$T_{\text{STR}}(n) = T\left(\frac{n}{2}\right)$$

Wie viele Additionen macht  $T_{\text{STR}}$ ?

es gibt 7 Multiplikationen pro Aufruf  
 ↳ selbst aufruft

Er macht 15 Additionen & jede Addition:  $n^2$  Operationen

$7 > 2^2$  : Fall 3

$$\text{Laufzeit } n^{\log_2 7} \quad T_{\text{STR}} \in O(n^{\log_2 7}) \\ = n^{2.8}$$

wichtig, wie man auf rekursive Laufzeitabschätzung kommt.

Aufgabe 5: diskte

a)  $O(n)$  man muss ganze diskte angelenken

b)  $O(n)$

1 dim.  
Array

$O(1)$  wie in gegeben mit Index

$O(n)$

## 1DIMENSIONAL

### ARRAY

$O(1)$

$O(n)$

$O(n)$

1. Lösung  
+ verschließen  
1. & 2. Hälfte in 1 neuer

	LISTE	pointer
c)	$O(n)$	alle durchgehen
d)	$O(1)$	
e)	$O(n)$	länge kann $O(\frac{n}{2})$ nicht: $O(n)$
f)	$O(n)$	neues Array 2 Listen immer kleinstes Element nehmen

# Algodaten NE

3.04.2019

## Übungsblatt 3



Sortiert: wenn man zwei nebeneinander liegende Daten vergleicht:  $0 > 1 > 2$

### 1) Suchverfahren

Worst Case

⊕ Selection Sort

$n^2$

• Bubble Sort

$n^2$

★ Insertion Sort

$n^2$

- ⊕  $O(n^2)$  kleinstes Ele in 1 Arr: n Schritte (1 mal durchgehen)
- $O(n)$  Abbruchbedingung: nichts mehr vertauschen
- ★  $O(n)$

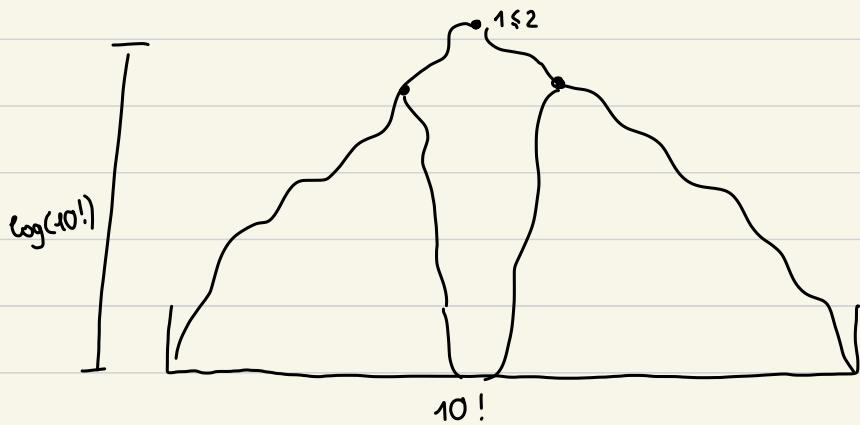
Welches ist nun das längsamste? Selection Sort  
Unterscheidet sich um den Faktor n.

NOTIZEN: • Liste mit 0 Element ist sortiert

b) Falsch,

W $\ddot{u}$ r Möglichkeiten, Array mit 10 Werten zu sortieren?

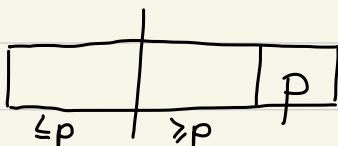
A: 10!



$\Rightarrow \log(10!)$  Vergleiche  $\Rightarrow$  mehr als 15 Vergleiche

3.2

## Quicksort



Für jedes der beiden Teilarrays wendet man wieder Quicksort an. (rekursiver Aufruf)

Abbruchbedingung: Array mit Länge 0/1



Donut

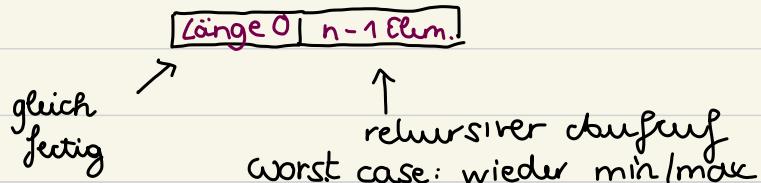
$\Rightarrow$ 

	p	
--	---	--

 Wenn beide Teile sortiert  $\Rightarrow$   
Array ist sortiert

Worst Case Laufzeit:  $O(n^2)$

schlimmster Fall : min / max haben



$\hat{=}$  WORST CASE



Bsp für Input: bereits sortiertes Array



Was passiert, wenn alles gleiche Elemente:

"best case" man hat immer Elemente, die  $>$ ,  $\leq$  sind.  
hängt davon ab, wie man's implementiert ( $n \log n$ )

Absteigend sortiert: auch worst case :  $O(n^2)$

c) worst case: (Median of 3)

wenn p das 2. kleinste / größte ist

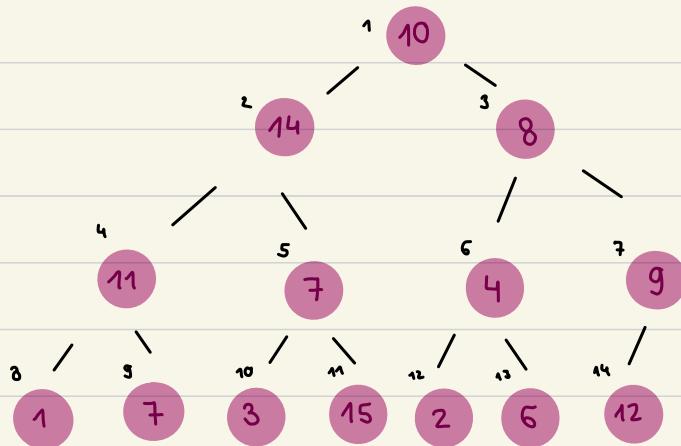
3.3

## Heap

Definition Array, kann man als Baum darstellen

(Binärbaum: jedes Element hat 2 Nachfolger)

a)



Baumdarstellung des Arrays

Bedingung:

$$a[h] \geq a[2h]$$

beschreibt, dass

$$a[h] \geq a[2h+1]$$

$a[h]$  (Elternknoten)

mit seinen 2 Kindknoten vergleicht.

Bedingung trifft in diesem Baum nicht zu.

$H(i, j)$        $\forall k : i \leq k \leq j$

Start                  Ende  
Teilbereich

In welchem Bereich gilt die H-Bedingung?

- (2, 6)
- (8, 14) <sup>NICHT</sup> (1, 14)

& Teilbereich, die keine Nachfolger haben.

$\Rightarrow$  Blattknoten (8, 14)

$\nearrow$  falls gerade

$\text{length}(a) = n$

$H\left(\frac{n}{2} + 1, n\right)$  ist immer 1 Heap

$H\left(\left\lfloor \frac{n}{2} + 1 \right\rfloor, n\right)$  falls  $n$  ungerade

Wie baut man nun einen vollständigen Heap?

Ziel: (1, n) bzw. (1, 14)

(8, 14) hat man schon. nun  $\underbrace{(7, 14)}$

Schauen, ob  $\text{arr}[7]$  größer als Kind.  
Nein, also tauschen

Dann  $(6, 14)$  mit Kinder vergleichen. 6 & 4 tauschen

$(5, 14)$  15 mit 7 tauschen

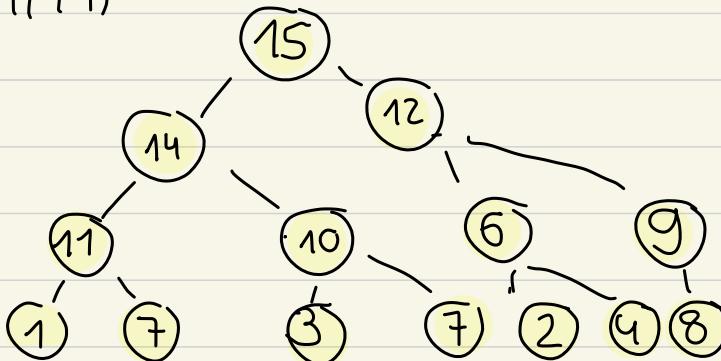
$(4, 14)$  ✓

$(3, 14)$  8 mit 12 tauschen

nun stimmb 8,9 nicht: also tauschen

$(2, 14)$  15, 14

$(1, 14)$



Immer zu gr. Knoten versichern

D.h. man kann sagen: gr. Element vom  
Array ist in der Wurzel. Im Array dann ganz vorne

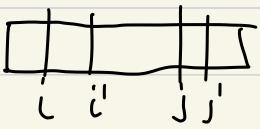
$\Rightarrow [15, 14, 12, \dots]$

d) Heapsort lernen

e): Heap so sortieren, dass Wurzel: (min)  
Also  $\leq$

### 3.4 Eigenschaften

b) wahr, dass max gr. weil Eltern immer gr. als Kind, Kind  $\geq$  unter Kind... Elemente werden immer kleiner

a)  gilt es in beiden Bereichen

Ja. gleiche Begr. wie oben

c) Für alle ungeraden Zahlen

d) Ja, alle gleich,  
1 Element

3.5a) Nein

b) versichern :  $O(m \cdot k)$

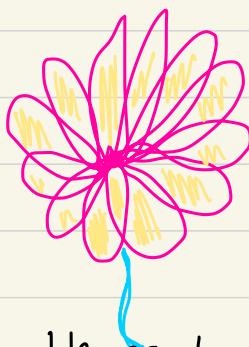
Arr. Länge m, Höhe:  $\log(m)$

Element an Stelle i :  $\log(i) = k$

Wv Schritte macht er dann ausgedrückt in  
i & j

$O(\log j - \log i) \leq \underbrace{O(\log n)}$

1 versichern



1. Element (gr) 1 Schritt (1 residuen)
2. gr. Elum (1 versichern)

↓  
Heapsort  $\leq O(n \log(n))$

Michi 

# Übungsblatt 4

## Aufgabe 1

$H(1, n)$

a)  $O(\log(n))$  

b) Max:  $O(1)$  Min:  $O(n)$  Allgemein:  $O(n)$

c) Max:  $O(\log(n))$  Min:  $O(1)$  Allgemein:  $O(n \cdot \log(n))$

↑  
da man wieder  
versichern muss

Blattknoten

↑  
wieder vertauschen  
& versichern

Schlimmster Fall: Wurzel  
(man müsste wieder alles  
versichern)

## Aufgabe 2:

Stack: push: Element einfügen

pop: Element entfernen

Stack ist generell

unsortiert

A

B

C (A B C)

↑ alle nach C verschieben

A	<u>a...a</u>
B	<u>bb...cc</u>
C	leer



A	a-a b...b
B	beer
C	c...c



A	a...a
B	b...b
C	c...c

Ja, es geht schneller wenn man mehrere Waffen gleichzeitig verschiebt.

### Aufgabe 4.3

#### Counting Sort

unterschied bzgl. Operationen: keine Vergleiche, nur Addition

Vorbedingung:

$$\forall 0 \leq i \leq n : 0 \leq a[i] \leq k$$

$$n=7$$

$$k=5$$

Bsp:

a: 1/ 4/ 2/ 5/ 3) 0/ 1/

new Array  
 $\downarrow c$

1	2	1	1	1	1
Index 0	1	2	3	4	5

$O(n)$

6 Elemente:

$c[0] = 1$ , da 0 genau 1 mal vorkommt.

$c[2] = 1$ , da 1 in a 2 mal vorkommt

1	3	4	5	6	2
0	2	3	9	5	6

$O(k)$

Ein Wert, der kleiner gleich 0 ist.

0	1	1	2	3	4	5
6						

$O(n)$

$$+ = O(2n+k)$$

$$= O(n+k)$$

$$= O(n), \text{ da } k \in O(n)$$



$$\forall 0 \leq i < 21 \quad a[i] = 2i$$

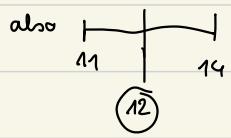
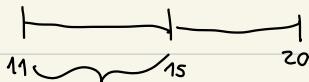
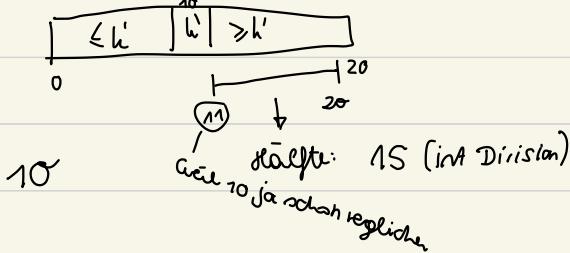
$k = 24$  aufst. sortiert

- durchlaufen bis 24 gy.
- wenn  $\text{zell} > 24$ : Abbruch

a) sequentielles Suchen:

Ind von 0, ..., 12 reziproken WC:  $O(n)$

b) binäres Suchen



$$\frac{11 + 14}{2} = 12$$

Allgemein:  $O(\log(n))$

FC:  $O(\log(n))$



$m(\text{int } a)$

$a = 10$

$10 \downarrow$

$\{\text{class } a$   
 $\text{int } b\}$

$m(\text{class } a) \{ a.b = 10$



# Algodat Übung

## Arbeitsblatt 5



### Hash Verfahren

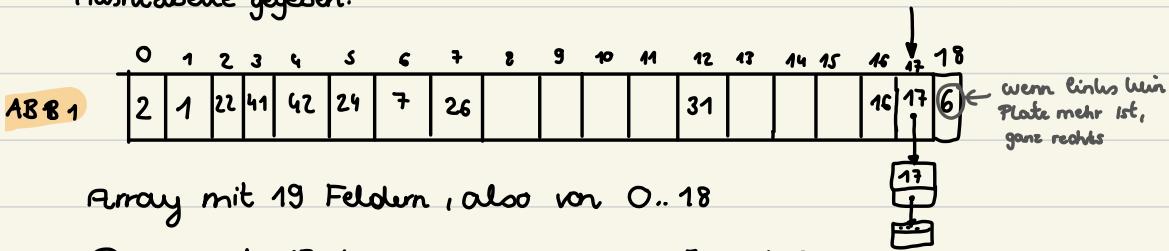
großen Wertebereich auf kleinen Wertebereich abbilden

Funktion: z.B. Hashtabellen: Set von Werten abspeichern

#### Aufgabe 1:

wird in diesem Fall auf einen Wertebereich von 0..19 abgebildet

Hashtabelle gegeben:



Array mit 19 Feldern, also von 0..18

Der Wert 17 kommt auf 17, da  $17 \bmod 19$

$$h(17) = 17 \bmod 19$$

Wenn einer schon besetzt, das nächste linke

$$h(1) = 1 \bmod 19 = 1$$

offenes Hash Verfahren: Wert wird direkt im Array gespeichert  
(es gibt keine Pointer)

- Sondierverfahren

## Aufg. 1 mit linearen Sortierverfahren

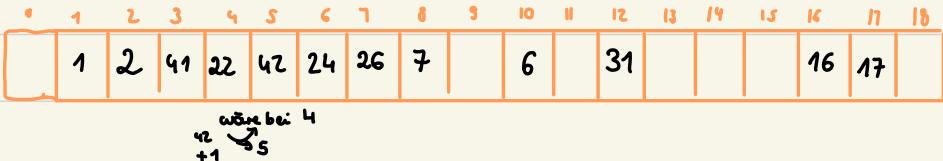
Siehe ABB 1

Was fällt auf?

Viel ganz links!!  $\Rightarrow$  Blöcke

Angenommen: aus. Element hinzufügen: man braucht lange, um Einfügeoperationen durchzuführen.

### Quadratisches Sortierverfahren



immer gesetztes +1

Regel:

- $1^2$  nach rechts gesetzt
- $1^2$  links gesetzt
- $2^2$  rechts
- $3^2$  ....

3 Sortierschritte, viel weniger als linear (7)

## Doppeltes Hashverfahren

$$h'(k) = 1 + (k \bmod 17)$$

0	1	2	3	4	5	6	7	8	9	10	K	12	13	M	15	K	17	4
1	2	41	42	24	6	26		7	31					22	11	16		

Erste Kollision mit 22 nach Hashfunktion würde Stelle 3, aber schon besetzt

Also  $1 + (22 \bmod 17) = 6$  von 3: 6 Schritte nach links 3-6

neuer Hashwert von Position abziehen

Hashfunktion: 8

um 8 Stellen links, allerdings besetzt nochmal 8, ... 18-8

2. Suchschritte benötigt !!

## Aufgabe 5.2

Gegeben: Array mit Werte

0 1 2 3 4 5 6

1	164	8	21	73	22	98
---	-----	---	----	----	----	----

HW	1	4	1	3	3	4	3
----	---	---	---	---	---	---	---

1. Schritt: Hashwerte HW berechnen

Ziffernsumme % 7

2. Welche Werte an richtiger Stelle: 21

⇒ 21 ↑ 73 ← 98  
↑  
eingefügt vor

8 ← 98

73 ← 21 ← 164 ← 8 ← 1 ← 98

}

21 4 73 4 32 4 164 4 8 4 1 4 98

totale Ordnung:

In diesem Fall nur eine Ordnung



## Aufgabe 5.3

### Perfekte Hashfunktion

- eindeutig
- keine Kollisionen
- Eingabe | Ausgabedomäne : AUF mind. so groß wie EF
  - 1..15      1..10
  - wäre keine perf. Hashf.

mod zumindet 12 !! für perf. Hashfunktion

- Jaenner: 60
  - Juli : 48
- } 12 gemeinsamer Hashwert:  
⇒ Kollision

Was ist der kleinste Hashwert dann?

$m = 27$  liefert perfekte Zahl

oder

mod größen: 97 (Sept)

- b) Finde 2 Worte, die den selben Hashwert haben

Ampel  
dampe

} gleiche  
Buchstaben

---

## Aufgabe 5.4

### dauzeitanalyse für Einfügeoperation

Worst Case (WC):  $\in O(n)$  wenn Hashwerte gleich sind bzw.  $h(k)=1$

Einfügeop. von 1 Element für offene Tabelle

Kollision  
außer bei 1.

Wenn disk voll: n mal durchgehen

WC tritt nur selten ein

### average case:

2 Grundannahmen

n ... Elemente

m ... Größe der disk

Verhältnis

$$\alpha = \frac{n}{m} \leq 0.5$$

HF, die ideal streut

$\hat{=} R(h)$   gleich verteilt

# Average Case AC

1. Hashwert ausrechnen:  $O(1)$

2. Ist die Stelle frei? Fallunterscheidung

ja

$\Downarrow$

$O(1)$  nur einfügen

ist Aufwand  $O$



alles addieren  $\hat{=}$  AC

wieder  
Fallunterscheidung

$$AC : \in O(1 + 0 \cdot \underbrace{\frac{m-n}{m}}$$

Wahrscheinlichkeit  
eines Felds zu treffen

$$+ \frac{n}{m} \cdot \left( 1 + \underbrace{\frac{n-1}{m-1}}_{\text{Sondierschritt}} \cdot \left( 1 + \frac{n-2}{m-2} \dots \right) \right)$$

durch nachst. Stelle:  $O(1)$

$$= O \left( 1 + \underbrace{\frac{n}{m}}_{\alpha} + \underbrace{\frac{n}{m} \cdot \frac{n-1}{m-1}}_{\leq \alpha^2} + \underbrace{\frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2}}_{\leq \alpha^3} + \dots \right)$$

$$\subseteq O \left( 1 + \alpha + \alpha^2 + \alpha^3 + \dots \right) = O \left( \sum_{i=0}^{\infty} \alpha^i \right)$$

Falls  $\alpha \leq 1$  in  $\Sigma$  von  
geometr. Reihe

$$\rightarrow = O \left( \frac{1}{1-\alpha} \right) = O(1)$$

AC ist konstant

dann gilt für suchen

## Aufgabe 5.5

Alg. zum Überpr. auf Duplicata

- ▶ naiver Algorithmus
- ▶ Hashtables
- ▶ Counting Sort

Worst Case

$O(n^2)$

$O(n^2)$

$O(n) + \text{Speicher}$

average case

$O(n^2)$

$O(n)$

$O(n)$

Michi du Bro

# Aufgabenblatt 6

## Aufgabe 6.1

ABB 1:

1	2	3	4	5	
1				1	1
1		1	1		2
	1		1	1	3
	1	1			4
1		1			5

} Knoten

ungerichteter Graph: Es gibt keine Richtung zw. Knoten

Matrix: an der Diagonalen gespiegelt

= Adjazentmatrix

Es gibt noch eine Inzidentmatrix

bei einer Kante ist  
wenn der Knoten (ein n Punkt einer  
Kante ist)

ABB 2

1	2	3	4	5	6	
1					1	1
1	1	1				2
1			1	1	1	3
		1	1			4
				1	1	5

Der Rest: 0

Spalte: Kante

Zeilen: Knoten



In einer Spalte / Zeile  
sind 2 Einser (2 Knoten)

= Incidentmatrix

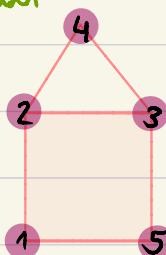
Was passiert, wenn man die Adjunktmatrizen multipliziert?

$A \times A$  (Matrizenmultiplikation)

	2	3
1	0	2

Möglichkeiten

Es gibt  
2 Wege  
die mit  
dänge 2



Graph skizziert:

- Wir Wege gibt es, von dem Knoten 1 zu 3 zu kommen

- $A \cdot A \cdot A$  : dänge 3 / 3 Schritte | Anzahl der Wege mit dänge 3
- $A^k$  Anzahl der Wege der dänge k, um von A nach B zu kommen.

Ergebnis bei Matrizenmult.

Inzidentmatrix mit sich selbst:

$$5 \times 6 \times 6 \times 5$$

keine quadrat. Matrix  $\Rightarrow$  also  $I \times I^T$

Ergebnis:  $5 \times 5$

	1	2	3	4	5
1	2	1			1
2	1	3	1	1	
3	0		3	1	1
4				2	
5					2

$\Rightarrow$  1er in Zeilen zählen

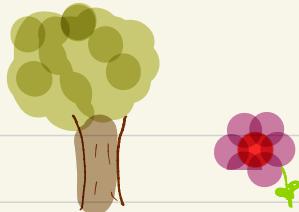
sagt aus: bei wr Kanten  
der Knoten dazw ist  $\hat{=}$  Grad  
Diagonale  $\hat{=}$  Knotengrad

Wie oft kommen Knoten in einer Kante vor (Anzahl)

$\Rightarrow 0 \backslash 1$  Mal

alle anderen Felder gleich wie in der Adjacentmatrix

# Bäume



- minimal zusammenhängend
- kreisfrei
- ungerichtet

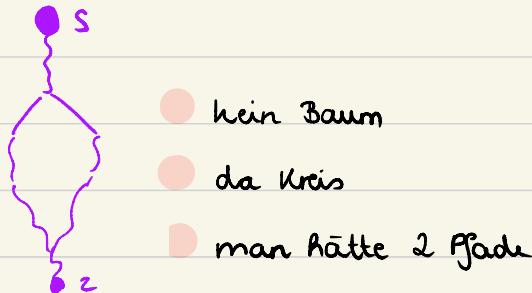
↙ wäre kein Baum

a) wahr

zusammenhängend  $\Rightarrow$  Es gibt einen Pfad (U. Def)

indirekter Beweis: man nimmt das Gegenteil an.

In diesem Fall: es gibt mehrere Wege



a)  $h \dots$  Höhe

$m \dots |\text{Blätter}| \}$  Anzahl

Blätter leg. sich auf Ebene  $h$  (ganz unten)

$$h > \log_2(m) \quad =$$

$$m \leq 2^h \mid \log$$

$$\underbrace{h \cdot \log_2}_1^2 \geq \log_2 m$$

$$h \geq \log_2 m$$

$2^h$ : Blätter am Ende

b)  $h + 1 \geq \log_2(|V| + 1)$

-1  
 $\log$

$|V|$ : Im Baum gilt:  $|V| \leq 2^h + 2^{h-1} + \dots + 2^0 = 2^{h+1} - 1$

6.3

WRL  
LRW

4 3 2 1 10 5 8 11 9 6 7  
7 6 9 11 8 5 10 1 2 3 4

LWR :

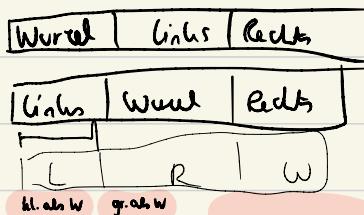


7 5 6 8 11 9 4 3 10 2 1

6.4



a) Preorder:



üller Preorder: Wurzel: 4

b)

- 11 -

Sortierter Baum  
geord. Baum

c) links: kleiner } als Wurzel  
rechts: größer }

Postorder gegeben

Wurzel:

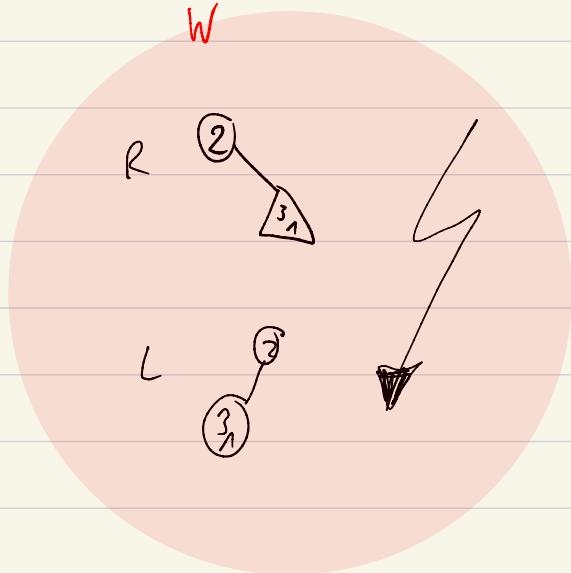
Kann man es  
wieder herstellen?

# Mit Gegebeispiel

postad:

3	1	2
---	---	---

W



6.5



Gegebeispiel

1 & links 1.5

