

(K1) Problem: Beschreibung Eingabe/abhängig Ausgabe / Gibt nicht los

Determiniertheit: gleiche Eingabe → gleiche Ausgabe

(Determiniert)

Determinismus: gleiche Eingabe → die Ausführung stets identisch.  
(Deterministisch)

Terminierung = läuft jede Eingabe nur endlich lange

Korrektheit ↗ Partielle: berechnet stets erwünschte, spezifizierte Ausgabe

Total: partielle Korrektheit + Terminierung.  
Effizienz: Sparsamkeit im Ressourcenverbrauch (Zeit, Speicher, Energie...)  
 $\Leftarrow k_2$

## Datenstrukturen

Elementare: byte, short, int, long, boolean, char, float, double, ...

Patentypen → Zusammengesetzte Typen: Struk., Record, Set, Array.

homogene Daten: Array.

- gleichartigen Elementen
- Elemente ⚡ keine eigenen Namen.
- Anzahl der Elemente ⚡ Erzeugung

heterogene Daten: Klassen.

- verschiedenartigen Elementen
- Elemente ⚡ eigenen Namen.
- Anzahl der Elemente ⚡ Deklaration static.

Beispiel: Sieb des Eratosthenes.

Problem: Suche alle Primzahlen kleiner n.

$$a[i] = \begin{cases} 1 & \text{falls } i \text{ prim ist} \\ 0 & \text{sonst.} \end{cases}$$

Algorithmus:

1.  $a[1] - a[n] := 1$

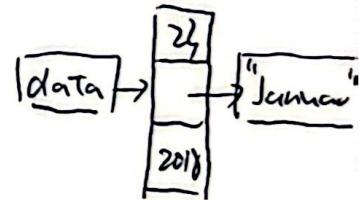
2. Setze Vielfache sukzessive auf 0

3. Arrayeinträge = 1 → prim.

\*  $1 < i < \sqrt{n}$

class Date {

int day  
String month  
int year



}

- Knoten werden zur Laufzeit (dynamisch) erzeugt und verketten.
- Strukturen können dynamisch wachsen & schrumpfen.

## Dynamische Datenstrukturen.

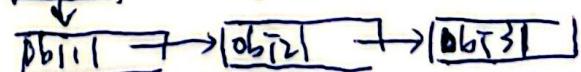
① LIST (Zyklusfrei)

Größe ⚡ Speicherplatz.

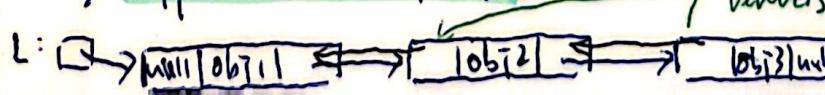


Einfach verankerte Liste

First ← Zeiger.

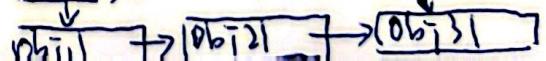


→ Doppelt verkettete Liste: zusätzlich Verweis.



Doppelt verkettete Liste

First ← Zeiger → Last

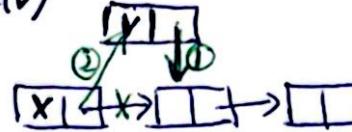


• Einfügen insert(v)

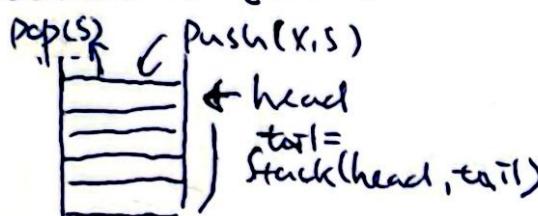
$l = \text{Entry}(v)$

$e.\text{next} = x.\text{next}$

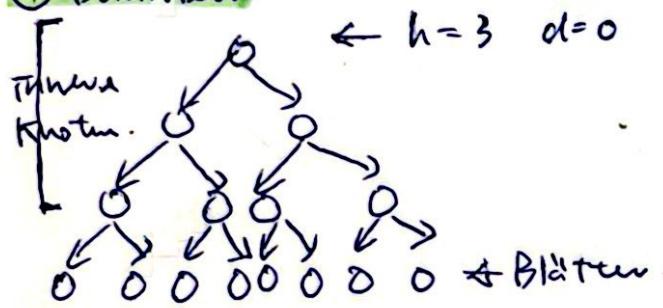
$x.\text{next} = l$        $\left\{ \begin{array}{l} \text{s. pop(1) Entfernen Letztes} \\ \text{.append(v) Hinzufügen am Ende} \end{array} \right.$



② Stack  $\leftrightarrow$  LIFO



③ Queue  $\leftrightarrow$  FIFO



maximal:  $2^{h+1} - 1$  Knoten.

minimum  $h+1$  Knoten.

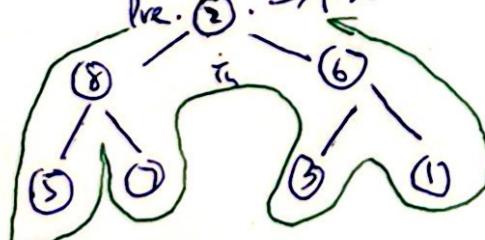
maximal  $2^h - 1$  Innere Knoten.

maximal  $2^h$  Blätter.

Baumtraversierung.  $\rightarrow$  Breitendurchlauf

Tiefendurchlauf.

Preorder      Postorder Knoten  
 (Präfix)      (Postfix)      (Infix)  
 2 8 5 7 6 3 1      5 7 8 3 6 2      1 8 7 2 3 6 1

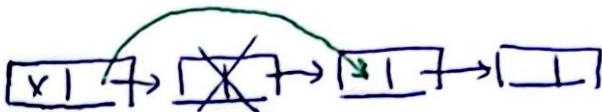


Algorithmus

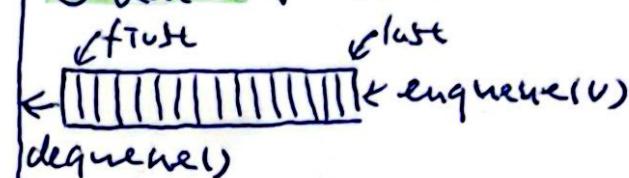
eine endliche Sequenz von Handlungsvorschriften, die eine Eingabe in eine Ausgabe transformiert.

• Löschen remove(v)

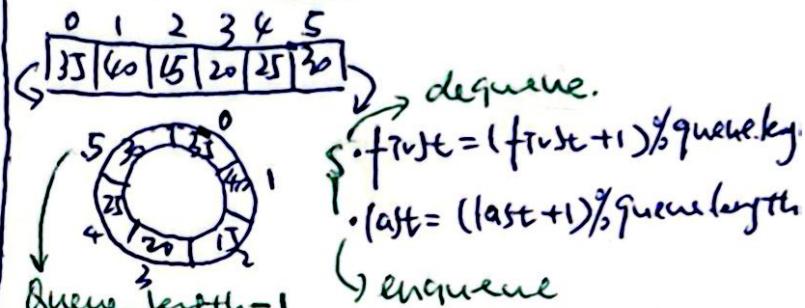
$x.\text{next} = x.\text{next}.\text{next}$



④ Queue  $\leftrightarrow$  FIFO

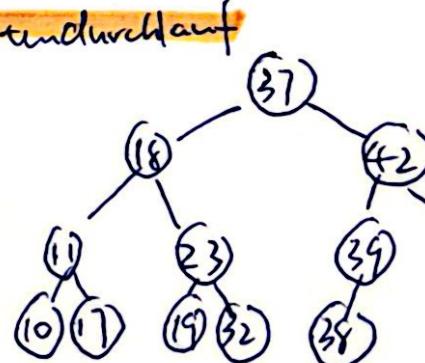


Queue als zyklisches Array.

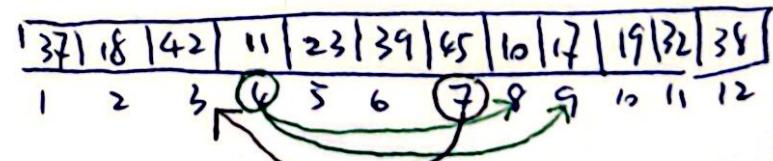


- Full:  $\text{first} == (\text{last} + 1) \% \text{queue.length}$
- empty  $\text{first} == \text{last}$

Priority Queue: mit Prioritätswerten.  
 $\Leftarrow$  Heapstruktur.



- Kinder von  $i$   
 $2i, 2i+1$
- Vorgänger von Knoten  $i$   
 $\lfloor \frac{i}{2} \rfloor$
- Blätter.  $\frac{n}{2} < i \leq n$
- $2i > n$  existieren



- Array beginnt bei 1
- link vollständig  $\Rightarrow$  keine Lücken.

## K2 Asymptotische Komplexitätsklassen.

Laufzeit  $T(n)$  sehr große Eingabe  $n \in \mathbb{N} \Rightarrow$  Komplexität unabhängig von Konstanten & Summanden sein.

$$O(f) = \{ g: \mathbb{R} \rightarrow \mathbb{R} \mid \exists c > 0 : \exists x_0 > 0 : \forall x > x_0 \quad |g(x)| \leq c \cdot |f(x)| \}$$

- $f$  ist obere Schranke von  $g$ .
- $g$  wächst höchstens so schnell wie  $O(f)$  &

### Komplexitätsklassen.

Klasse	Beispiel	Struktur / Vergleich
$O(1)$	Neue Liste initialisieren / Einfügen eines Stackelements / Value-Atribute setzen	$\neq O(j=7) \dots$
$O(\log n)$	Binärsuchbaum (balanced) / Suche in AVL-Baum	
$O(n)$	sequentielle Suche (lineare Suche in Array) / Summe $\text{Bis}(n) + 2 \dots + n$	
$O(n \log n)$	Merge Sort / Heap Sort	
$O(n^2)$	Matrixaddition	
$O(n^3)$	Matrixmultiplikation	
$O(n^k)$		
$O(2^n)$	Edit-Distanz naiv, Fibonacci-Folge (Naiver, rekursiver Algorithmus)	
$O(n!)$	Permutationen	
$O(n^u)$		

### $O$ -Notation Rechenregeln

1. Konstante:  $g(x) = a \in \mathbb{R} \Rightarrow g \in O(1)$   $* 2^{n+a} = 2^a \cdot 2^n \in O(2^n)$
2. Skalare Multiplikation:  $g(x) \in O(f) \quad a \in \mathbb{R} \Rightarrow ag(x) \in O(f)$   $O(\log_b n) = O(\log n)$
3. Addition:  $g_1(x) \in O(f_1) \wedge g_2(x) \in O(f_2) \Rightarrow g_1 + g_2 \in O(\max(f_1, f_2))$
4. Multiplikation:  $g_1 \in O(f_1) \wedge g_2 \in O(f_2) \Rightarrow g_1 \cdot g_2 \in O(f_1 \cdot f_2)$   
 $\Rightarrow O(2^n) \cdot O(2^m) = O(2^{n+m}) / O(n \log \log n) \subset O(n \cdot \log n) \subset O(n^2) \subset O(n^2 \log n)$

Beispiel: Folge von Anweisungen  $x=0; y=0; z=0$

Bedingte Anweisungen:  $\text{if}(a \% == 0) \quad a = a/2; \text{else } a++;$

Fallunterscheidungen:  $\text{switch}(n)$  case: ... break;

- Endständige Rekursion:
- Führt Ergebnis Rekurrenz mit  $n$  Schritte bis zum Ende.
  - Rekursionsaufruf ist letzte Aktion.
  - alte Werte auf dem Stack werden nicht mehr benötigen

# Master-Theorem Analyse von Rekursionsgleichungen

## ① Divide-and-Conquer.

mit  $c > 0, a > 0, b > 1, f(n) \in O(n^d), d \geq 0$  gilt

$$T(n) \leq \begin{cases} c & n \leq 1 \\ a \cdot T\left(\frac{n}{b}\right) + f(n) & n > 1 \end{cases}$$

$$T(n) = \begin{cases} O(n^d) & d > \log_b a \\ O(n^d \log n) & d = \log_b a \\ O(n \log_b a) & d < \log_b a \end{cases}$$

## ② Subtract and Conquer

$$T(n) \leq \begin{cases} c & n \leq 1 \\ aT(n-b) + f(n) & n > 1 \end{cases}$$

mit  $c > 0, a > 0, b > 0, f(n) \in O(n^d), d \geq 0$  gilt

$$T(n) = \begin{cases} O(n^d) & a < 1 \\ O(n^{d+1}) & a = 1 \\ O(n^d a^{\lfloor \log_b n \rfloor}) & a > 1 \end{cases}$$

Beispiel 1:

① Merge Sort  $T(n) = 2T\left(\frac{n}{2}\right) + n$

$$a=2 \quad b=2 \quad c=1 \quad d=1$$

$$\log_2 b = 1 = d$$

$$\Rightarrow O(n \log n)$$

Binär Suche  $T(n) = T\left(\frac{n}{2}\right) + 1$

$$a=1 \quad b=2 \quad c=1 \quad d=0$$

$$\log_2 b = \log_2 2 = 1 = d$$

$$\Rightarrow O(n^0 \log n) = O(\log n)$$

Beispiel 2: ① Fibonacci

①  $T(n) = T(n-2) + T(n-1) + 1$

$$\Leftrightarrow T(n) \leq 2T(n-1) + 1$$

$$a=2 \quad b=1 \quad c=1 \quad d=0$$

$$\Rightarrow O(n^0 2^n) = O(2^n)$$

③  $T(n) = 9T\left(\frac{n}{3}\right) + n$

$$a=9 \quad b=3 \quad c=1 \quad d=1$$

$$\log_3 a = \log_3 9 = 2 \geq 1 = d$$

$$\Rightarrow O(n \log_3 a) = O(n^2)$$

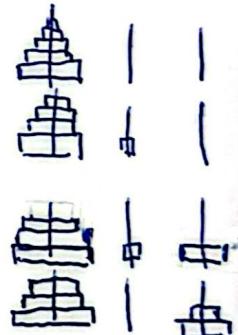
④  $T(n) = 3T\left(\frac{n}{2}\right) + n^2$

$$a=3 \quad b=2 \quad c=1 \quad d=2$$

$$\log_2 3 < 2 = d$$

$$\Rightarrow O(n^2)$$

② Towers of Hanoi Puzzle



$$T(n) = 2T(n-1) + 1$$

$$a=2 \quad b=1 \quad c=1 \quad d=0$$

$$\Rightarrow O(n^0 2^n) = O(2^n)$$

Beispiel 3: Substitution  $\star$

①  $T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \log_2 n$

$$\text{Substitution: } \log_2 n = m \Leftrightarrow 2^m = n$$

$\rightarrow T(n) = 2T\left(\frac{m}{2}\right) + m$

$$S(m) = T(2^m) \Leftrightarrow S\left(\frac{m}{2}\right) = T\left(2^{\frac{m}{2}}\right)$$

$$S(m) = 2S\left(\frac{m}{2}\right) + m$$

$$a=2 \quad b=2 \quad c=1 \quad d=1 \quad \log_2 2 = 1 = d \Rightarrow O(m \log m)$$

$$S(n) \in O(m \log m)$$

$$\Rightarrow T(n) = T(2^m) = S(m) \in O(m \log m)$$

$$= O(\log n \cdot \log \log n)$$

$$\textcircled{2} \quad T(n) = T(\sqrt{n}) + 1$$

$$= T(n^{\frac{1}{2}}) + 1$$

Substitution:  $m = \log_2 n \Leftrightarrow 2^m = n$

$$T(n) = T(2^m) = T(2^{\frac{m}{2}}) + 1$$

$$S(m) = T(2^m) \Leftrightarrow S\left(\frac{m}{2}\right) = T(2^{\frac{m}{2}})$$

$$S(m) = S\left(\frac{m}{2}\right) + 1$$

$$a=1 \quad b=2 \quad c=1 \quad d=0$$

$$\log_2 1 = 0 = d$$

$$\Rightarrow S(m) \in O(\log m)$$

$$\Rightarrow T(n) = T(2^m) = S(m) \in O(\log m)$$

$$\Rightarrow T(n) \in O(\log \log n)$$

$$\textcircled{3} \quad T(n) = T(4\sqrt{n}) + 128$$

$$= T(n^{\frac{1}{4}}) + 128$$

Substitution:  $m = \log_4 n \Leftrightarrow 4^m = n$

$$T(n) = T(4^m) = T(4^{\frac{m}{4}}) + 128$$

$$S(m) = T(4^m) \Leftrightarrow S\left(\frac{m}{4}\right) = T(4^{\frac{m}{4}})$$

$$S(m) = S\left(\frac{m}{4}\right) + 128$$

$$a=1 \quad b=4 \quad c=1 \quad d=0$$

$$\log_4 1 = 0 = d$$

$$\Rightarrow S(m) \in O(\log m)$$

$$\Rightarrow T(n) = T(4^m) = S(m) \in O(\log m)$$

$$\Rightarrow T(n) \in O(\log \log n)$$

$$\textcircled{4} \quad T(n) = a \cdot T\left(\frac{x}{c}n\right) + \log_x(a) \quad x \cdot y > 1 \quad \log_x(a) < 1$$

Substitution:  $m = \log_y(n) \Leftrightarrow y^m = n$

$$T(n) = T(y^m) = a \cdot T\left(y^{\frac{m}{x}}\right) + m$$

$$S(m) = T(y^m) \Leftrightarrow S\left(\frac{m}{x}\right) = T\left(y^{\frac{m}{x}}\right)$$

$$S(m) = aS\left(\frac{m}{x}\right) + m$$

$$a=a \quad b=x \quad c=1 \quad d=1$$

$$\log_x a < 1 = d$$

$$\Rightarrow S(m) \in O(m)$$

$$\Rightarrow T(n) = T(y^m) = S(m) \in O(m)$$

$$\Rightarrow T(n) \in O(\log n)$$

### Sukzessiven Einsetzen Analyse von Rekursionsgleichungen.

$$\textcircled{1} \quad T(n) = T(n-1) + T(n-2) + 2 \underset{\textcircled{2}}{\cancel{T(n)}} + T(0) = T(1) = 1$$

$$T(n) \leq 2T(n-1) + 2$$

$$\leq 2(2T(n-2) + 2) + 2 = 4T(n-1) + 6$$

$$\leq 4 \cdot (2T(n-3) + 2) + 6 = 8T(n-1) + 14$$

$$\leq \dots \leq 2^n T(n-n) + (2+4+8+16+\dots+2^n)^*$$

$$= 2^n T(0) + (2^{n+1} - 2) \in O(2^n)$$

$$\textcircled{*} \quad \text{Sum} = a \cdot \frac{1-q^n}{1-q}$$

### K3 Sortieren

#### Partielle Ordnung Beispiel: Teiler

- Reflexivität  $x \leq x$
- Transitivität  $x \leq y \wedge y \leq z \Rightarrow x \leq z$
- Antisymmetrie  $x \leq y \wedge y \leq x \Rightarrow x = y$

	B.S	S.S	I.S	MS	Q.S	H.S
B.C	$O(n^2)$	$n^2$	$n$	$n \log n$	$n \log n$	$n \log n$
A.C	$n^2$	$n^2$	$n^2$	$n \log n$	$n \log n$	$n \log n$
W.C	$n^2$	$n^2$	$n^2$	$n \log n$	$n^2$	$n \log n$



Bubble Sort — find the bigest Element

pair of adjacent elements are compared, and the elements swapped if they are not in order.

```
public void bubblesort(int[] a) {
    int n = a.length;
    for (int i=0; i<n; i++) {
        for (int j=0; j<n-i-1; j++) {
            if (a[j] > a[j+1])
                swap(a, j, j+1);
        }
    }
}
```

	Swap	Compare
i=0	8 2 7 3 6 4 5	8
i=1	1 8 2 7 3 6 4 5 9	6
i=2	1 2 7 3 6 4 5 8 9	4
i=3	1 2 3 6 4 5 7 8 9	2
i=4	1 2 3 4 5 6 7 8 9	0
i=5	1 2 3 4 5 6 7 8 9	0
i=6	1 2 3 4 5 6 7 8 9	0
i=7	1 2 3 4 5 6 7 8 9	0
i=8	1 2 3 4 5 6 7 8 9	0

#### Insertion Sort — sortierte Teil liste

start from array[1] put it in temp, check the left if it larger than array[1] the move it one position right.

```
public void insertionsort (int[] a) {
    for (int i=1; i<a.length; i++) {
        int key = a[i];
        int j = i - 1;
        while (key < a[j]) {
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = key;
    }
}
```

swap	compare
B.C: 0	B.C: $O(1)$
A.C: $O(n^2)$	A.C: $O(n^2)$
W.C: $O(n^2)$	W.C: $O(n^2)$

#### totale Ordnung

- Totalität  $x \leq y \vee y \leq x$

#### Beispiel:

- alphabetische Ordnung
- lexikographische Ordnung.

#### Stabilität: Beispiel: 4454B321

Erhält die Reihenfolge von gleichwertigen Schlüsseln.

#### In-place:

An algorithm that does not need an extra space and produces an output in the same memory that contains the input in place.

	Swap	Compare
i=0	8 2 7 3 6 4 5	8
i=1	1 8 2 7 3 6 4 5 9	6
i=2	1 2 7 3 6 4 5 8 9	4
i=3	1 2 3 6 4 5 7 8 9	2
i=4	1 2 3 4 5 6 7 8 9	0
i=5	1 2 3 4 5 6 7 8 9	0
i=6	1 2 3 4 5 6 7 8 9	0
i=7	1 2 3 4 5 6 7 8 9	0
i=8	1 2 3 4 5 6 7 8 9	0

$$\sum_{i=0}^{n-1} (n-i) = O(n^2)$$

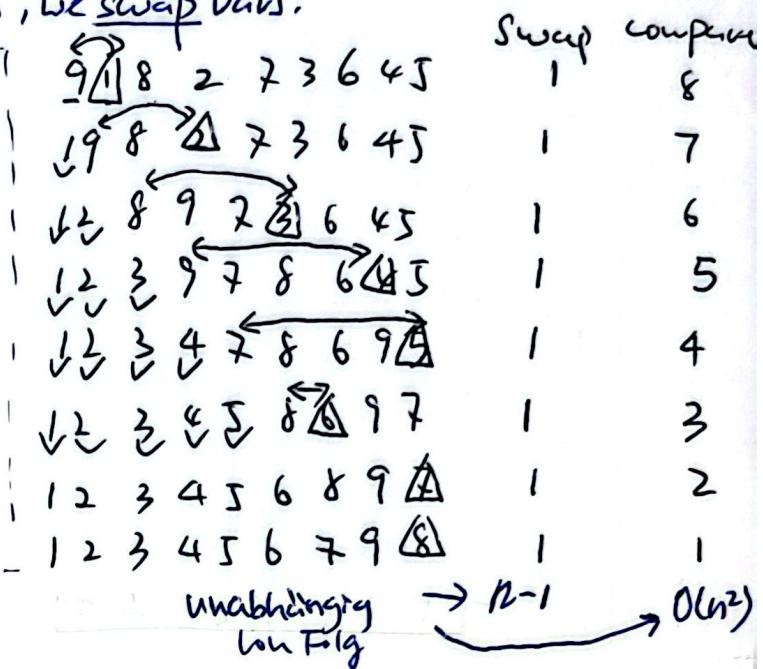
unabhängig von Folge

	Swap	Compare
i=1	6 1 7 4 2 9 8 5 3	1
i=2	1 6 7 4 2 9 8 5 3	0
i=3	1 6 7 4 2 9 8 5 3	2
i=4	1 4 6 7 2 9 8 5 3	3
i=5	1 2 4 6 7 9 8 5 3	0
i=6	1 2 4 6 7 9 8 5 3	1
i=7	1 2 4 6 7 8 9 5 3	4
i=8	1 2 4 5 6 7 8 9 3	6

## Selection Sort $\rightarrow$ Iteration k-sto kleinste Zahl.

Search through an array and keep track the minimum value during each iteration. At the end of each iteration, we swap vars.

```
public void SelectionSort(int[] a) {
    for (int i = 0; i < a.length - 1; i++) {
        int min = i;
        for (int j = i + 1; j < a.length; j++) {
            if (a[j] < a[min])
                min = j;
        }
        swap(a, i, min);
    }
}
```



## MergeSort - Divide & Conquer

MergeSort(L):

Smallest case = 1 element

1.  $n = \#$  of elements of L

2. If  $n > 2$ :

a) MergeSort( $\text{first half of } L$ )  $a = \frac{n}{2}$

b) MergeSort( $\text{Second half of } L$ )  $b = \frac{n}{2}$

c) Merge first & second half of L

\*  $n$  is odd



[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

def mergeSort(array):  $T(n)$

If  $\text{len}(\text{array}) > 1$ :

mid =  $\text{len}(\text{array}) / 2$

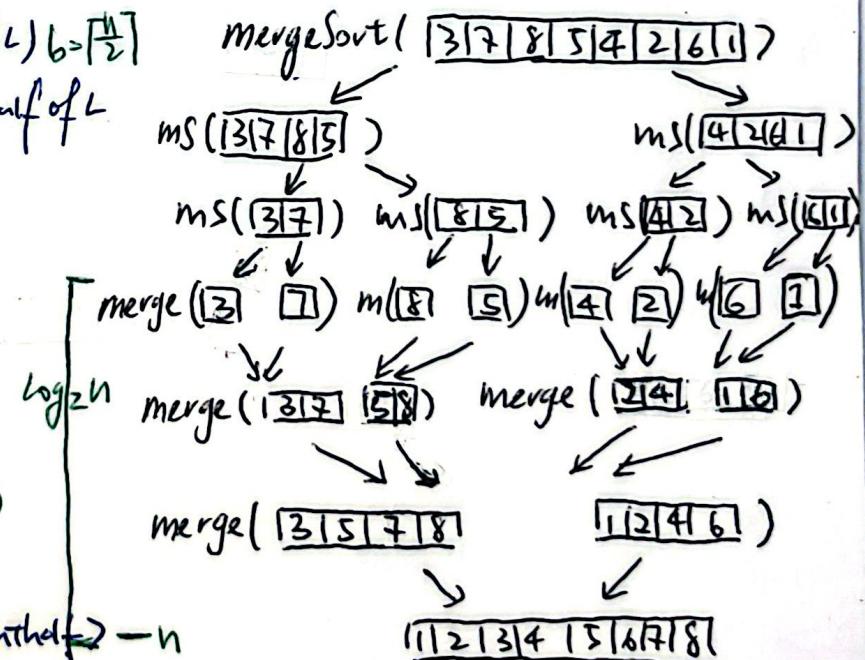
lefthalf = array[:mid]

righthalf = array[mid:]

mergeSort(lefthalf)  $- T(\frac{n}{2})$

mergeSort(righthalf)  $- T(\frac{n}{2})$

merge(array, lefthalf, righthalf)  $- n$



def merge(array, lefthalf, righthalf)

i, j, k = 0, 0, 0

while i < len(lefthalf) and j < len(righthalf):

if lefthalf[i] <= righthalf[j]:

array[k] = lefthalf[i]

i = i + 1

else:

array[k] = righthalf[j]

$$T(n) = 2T\left(\frac{n}{2}\right) + h$$

$$a = 2 \quad b = 2 \quad c = 1 \quad d = 1$$

$$\log_2 2 = 1 = d$$

$$\Rightarrow O(n \log n)$$

merge(array, lefthalf, righthalf)

j = j + 1

## Quicksort — Divide & Conquer / Pilot

Split into 2 pieces and recursively solve each piece, but instead of doing merging, we are going to assume once we doing splitting, all of the elements at least in some kind of order.

Quicksort(A):

base case: just one element

1. if the length(A) ≥ 2

- Pick a **pilot element** of P
- Move all elements of A > P to the right of p and all elements of A ≤ P to the left.
- Quicksort(set of > P)
- Quicksort(set of ≤ P)

def quicksort(arr, low, high) - T(n)

if low < high // base case: low=high

pi = partition(arr, low, high) - O(n-1)

quicksort(arr, low, pi-1) - T( $\frac{n-1}{2}$ )

quicksort(arr, pi+1, high) - T( $\frac{n-1}{2}$ )

def PARTITION(arr, low, high):

pilot = arr[high]

i = low

for j in range(low, high)

if arr[j] ≤ pilot:

arr[i], arr[j] = arr[j], arr[i]

i += 1

arr[i], arr[high] = arr[high], arr[i]

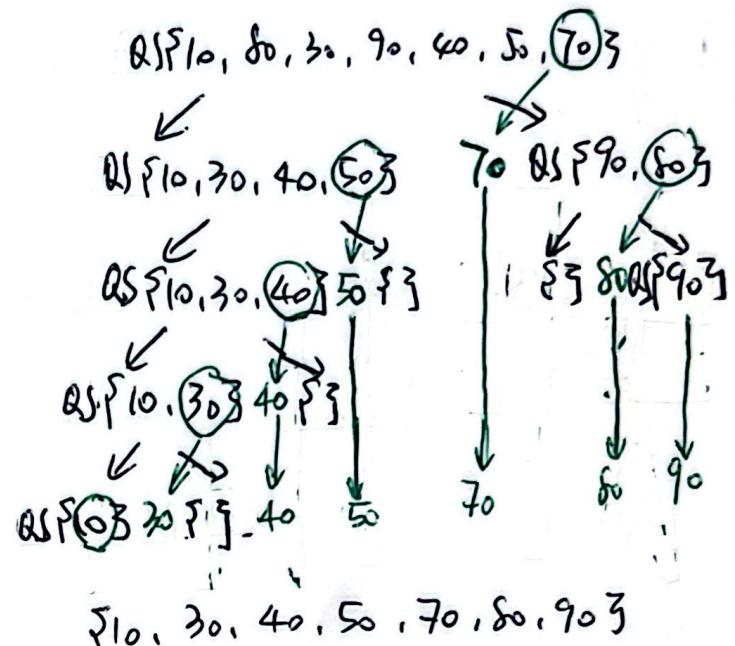
### Wahl des Pilotelements

• einfache: erst, letzte, mittler  $p = x_{\lceil \frac{n}{2} \rceil}$

Median ∈ nicht parktisch 2. Sort 2.  $p = x_{\lceil \frac{n}{2} \rceil}$

• Median of Three: median( $x_1, x_{\lceil \frac{n}{2} \rceil}, x_n$ )

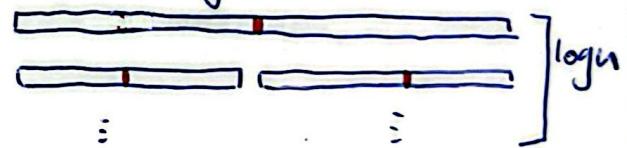
• zufällig Element



**Best case:** Die Fliege zerfällt immer in zwei gleich große Teile.

$$T(n) = 2T\left(\frac{n-1}{2}\right) + n-1 < 2T\left(\frac{n}{2}\right) + n$$

$$\Rightarrow T(n) \in O(n \log n)$$

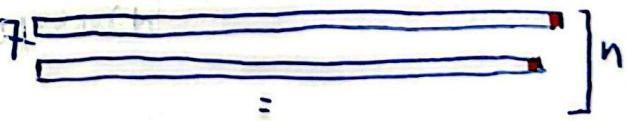


**Worst case:** eine Folge der Längen  $n-1$

$$T(n) = T(n-1) + n-1 < T(n-1) + n$$

$$a=1 \quad b=1 \quad c=1 \quad d=1$$

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

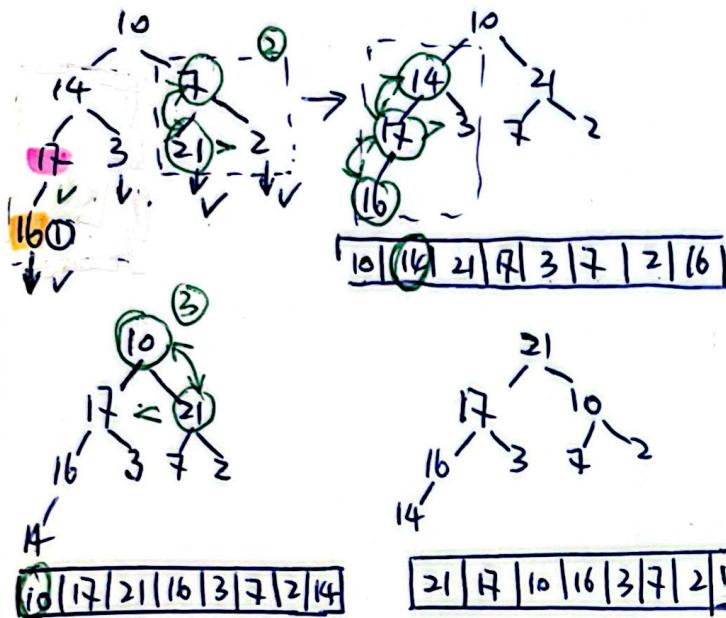


Worst case vermeiden.

Heapsort — Suche nach min/max in  $O(\log n)$

**Heap**: Inkomplettständigen Binärbaum  
representation in Array: Breitendurchlauf.

## 1. Heaps aufbaum (Heapify) ↑



Komplexität: ① Heapsort  
 $T(n) \leq 2^{h+1} = h+1 \in O(n)$   
② Sortphase  
 $O(h \log n)$

(Allgemein)

Entscheidungsbasis  $\Leftarrow$  Vergleichen bestehende

- Unterschiede Suchstrategie  $\Rightarrow$  von Unterschiede Knoten bis zu einem Blatt
  - untere Schrank (worst case)  $\Rightarrow \Omega(n \log n)$

Sortieralgorithmus

## Counting Sort 一个类 Tim place

- $m \leq n$
  - ↑      ↑  
~~Max~~ Schlüssel
  - nicht gleichweisen  
 Vergleichen.

① Initialisierung des Histogramms  $\rightarrow O(n)$

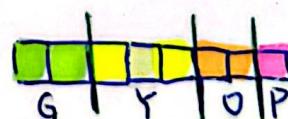
② Zählen der Schlüssel, Schleife über alle Schlüssel  $\rightarrow O(n)$

③ Bestimmung der Adressen für alle m Werte  $\rightarrow O(m)$

④ Zusammenstellung des finalen sortierten Arrays  $T(n)$

$$\Rightarrow O(\max(m, n))$$

Beispiel:  $\xrightarrow{②}$



1 2 3 2 1 ... Histogramm  
G Y O P

# Suchen K4

	B.C	A.C	W.C	Speicher	Vorteil	Nachteil
Sequentielle Suche	$O(1)$	$O(n)$	$O(n)$	$O(n)$	Keine Initialisierung	Hohe Suchkosten
Binäre Suche	$O(1)$	$O(\log n)$	$O(\log n)$	$O(n)$	W.C. $O(\log n)$	Sortiertes Array
Interpolationssuche	$O(1)$	$O(\log \log n)$	$O(n)$	$O(n)$	Schnelle Suche	Sortiertes Array Annahme über Datenverteilung.

$\downarrow \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

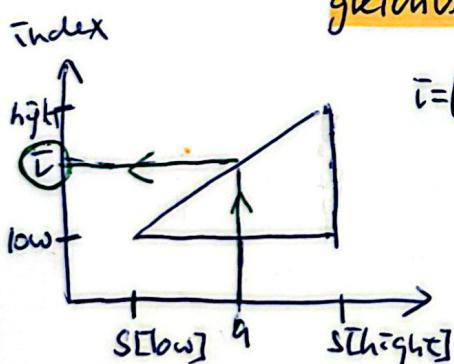
## Sequentielle Suche

```
def linearSearch(S, searchkey):
    for key, value in S:
        if key == searchkey:
            return value
```

$$S = \begin{pmatrix} \text{key} \\ \text{value} \end{pmatrix} = \begin{pmatrix} 92 & 54 & 73 & 14 & 23 & 61 \\ A & C & D & G & H & B \end{pmatrix}$$

## Interpolationssuche - Schätzen

Voraussetzung: 1. Sortiertes Array  
2. Elemente sind gleichverteilt.



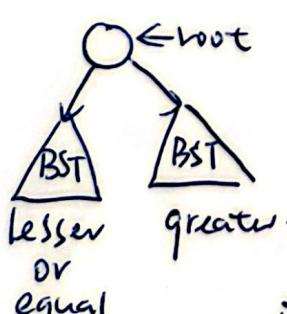
$$i = \text{low} + \frac{(q - S[\text{low}])(\text{high} - \text{low})}{(S[\text{high}] - S[\text{low}])}$$

## \* Suffix Array

- lexicographisch sortieren.
- $O(\log |S|)$  Schritte für die Suche (B.S.)
- $O(|P|)$  Zeichenvergleiche in jedem einzelnen Schritte
- $\Rightarrow O(|P| \log |S|)$

## Suchbaum.

### Binäre Suchbaum.

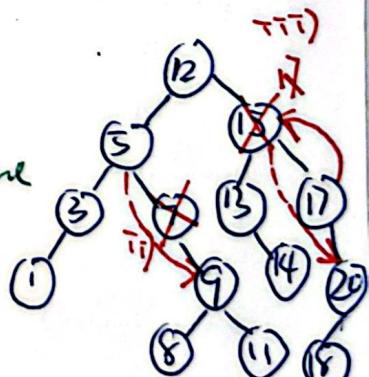


immer Knoten

$O(n)$  W.C.

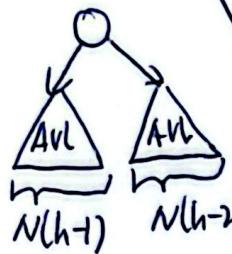
$\rightarrow O(\log n)$  B.S.  $\leftarrow$  balancierte Suchbaum

- ① **Insert:** put(x)  $\rightarrow$  Blatt
- ② **Remove:** remove(x)
  - : leaf node - direkt löschen
  - i) with 1 child - link mit Parent
  - ii) with 2 children - ① replace with min value in LST, then delete duplicate.
  - ② replace with max value in LST, then delete duplicate



Über Datenverteilung.

## AVL-Tree

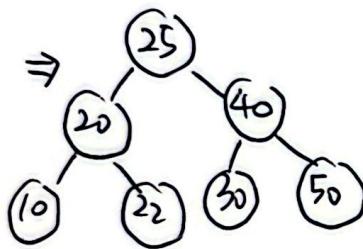
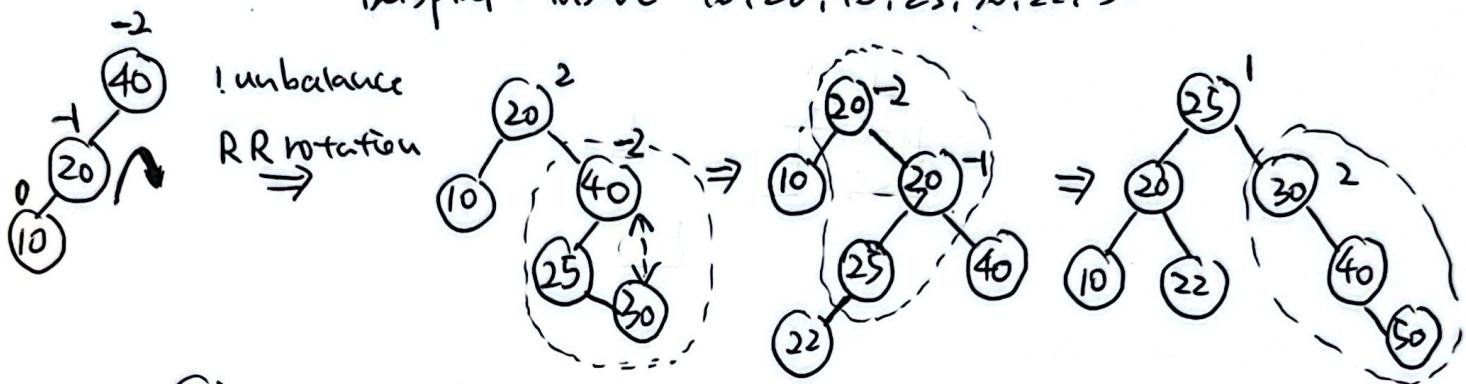


② AVL-Label  $\{ -1, 0, 1 \}$

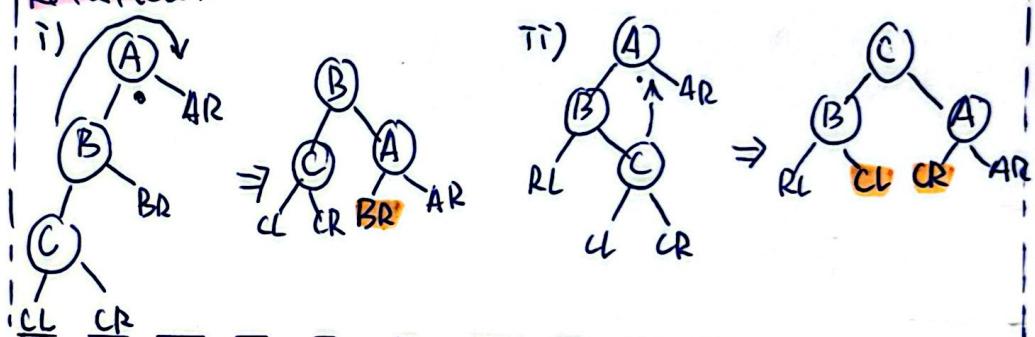
$b = \text{Höhe (R Teilbaum)} - \text{Höhe (L Teilbaum)}$

①  $\text{insert}(x)$  — Einfügen      Rotation  
 $O(h) + O(1) = O(\log n)$

Beispiel =  $\text{insert}: 40, 20, 10, 25, 30, 22, 50$



Rotation:



②  $\text{delete}(x)$  — Entfernen      Rotation  
 $O(h) + O(h) = O(\log n)$

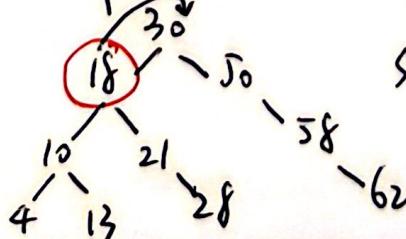
1) Zuerst „normal“ Löschen wie BST

2) Rotation.

## Splay-Baum

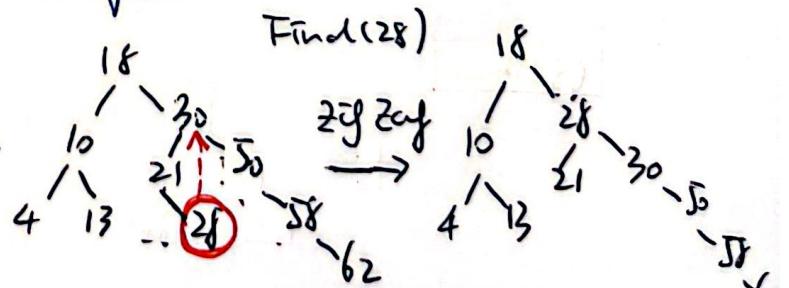
- Dynamische optimale Suchbaum
- Bei jeder Suche nach einem Schlüssel wird dieser durch Rotationen zur Wurzel des Suchbaums (Splay)
- Setzen angefragte Schlüssel  $\rightarrow$  tiefer
- Häufig aufgefragte Schlüssel  $\rightarrow$  sinkt wenig ab
- Suchen(x) & Einfügen(x) Sind mit Operation splay gekoppelt.
- Keine strukturelle Invarianz.  $\leftarrow$  Worst-Case-Sequence.
- amortisierte Laufzeitkomplexität:  $O(\log n)$

Bei Spiel.



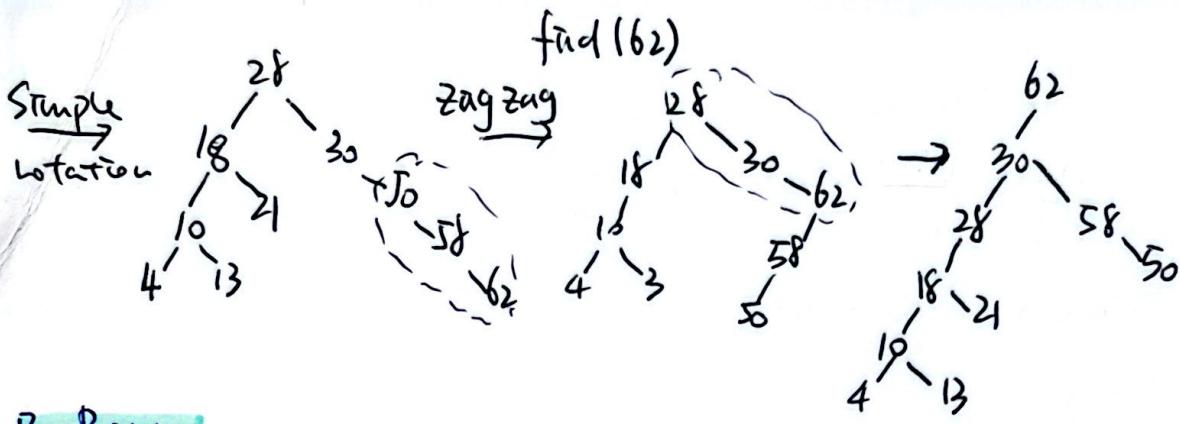
Find(18)

Simple Rotation



Find(28)

Zig-Zag



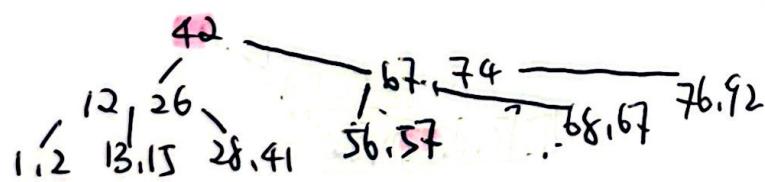
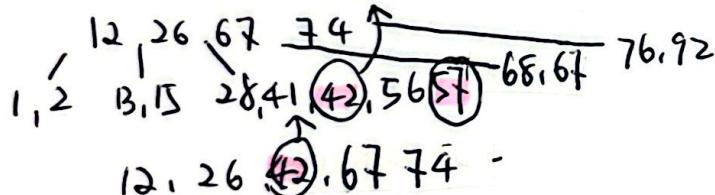
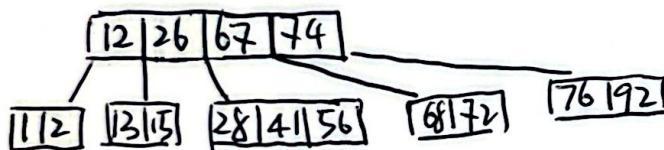
### B-Baum

- Mehrwege-Suchbaum.
- balanciert  $\rightarrow$  Alle Blätter haben denselben Abstand zur Wurzel
- Ordnung  $k \rightarrow$  Wurzel:  $1 \rightarrow 2k$  Schlüssel  
 $k > 1 \downarrow$  andere Knoten:  $k \rightarrow 2k$  Schlüssel

### ① Insert(x)

Beispiel:  $k=2$   $1 \rightarrow 4$

Insert 42, 57

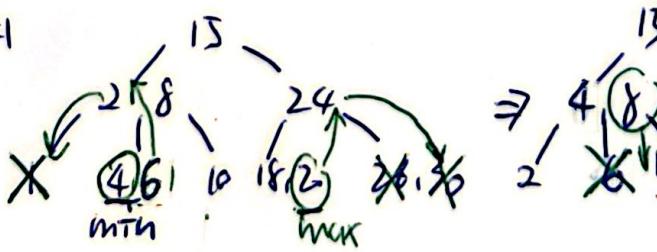


### ② delete(x)

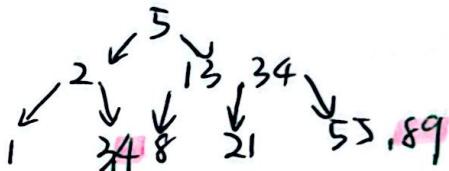
i) leaf delete 1, 26, 36, 6

① Ausgleich: Nachbarknoten mit mehr als  $k$  Schlüsseln

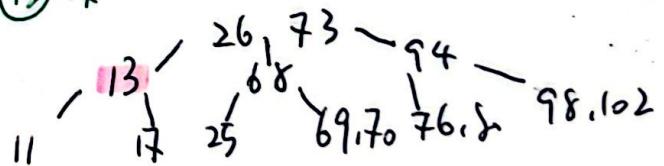
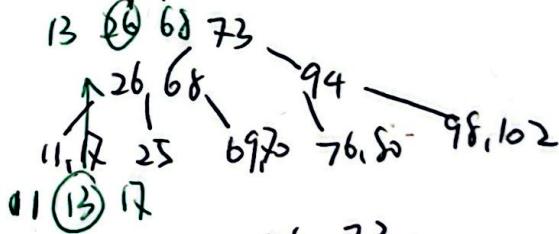
$k=1$



Beispiel 2  $k=1$   $1 \rightarrow 2$  Insert 4, 89

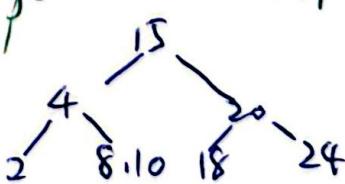


Beispiel 3  $k=1$  Insert 13



⑤ Verschmelzen: Nachbarknoten mit genau  $k$  Schlüsseln

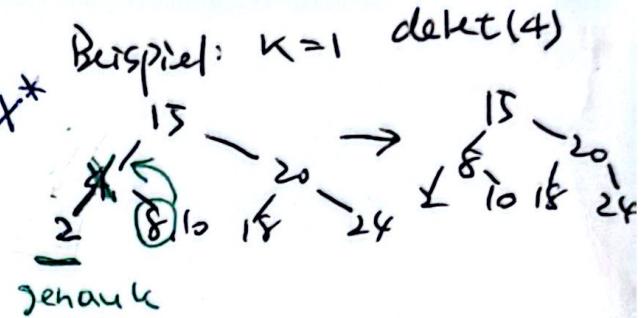
merge the child with parents



## ② Internal node

- größten Schlüssel im Teilbaum links löschen\*
- ersetze  $x$  im Knoten  $N$  durch  $x'$
- Löschen  $x'$

\* OR smallest element in the right side



## \* Höhenabschätzung

min Schlüsselzahl  $s_{\min}$

$$S \geq s_{\min} = 1 + 2k \sum_{i=0}^{h-1} (k+1)^i = 2(k+1)^h - 1$$

max Höhe

$$h \leq \log_{k+1} \frac{S+1}{2} \approx \log_{k+1} S$$

max Schlüsselzahl  $s_{\max}$

$$S \leq s_{\max} = 2k \sum_{i=0}^{h-1} (2k+1)^i = (2k+1)^h - 1$$

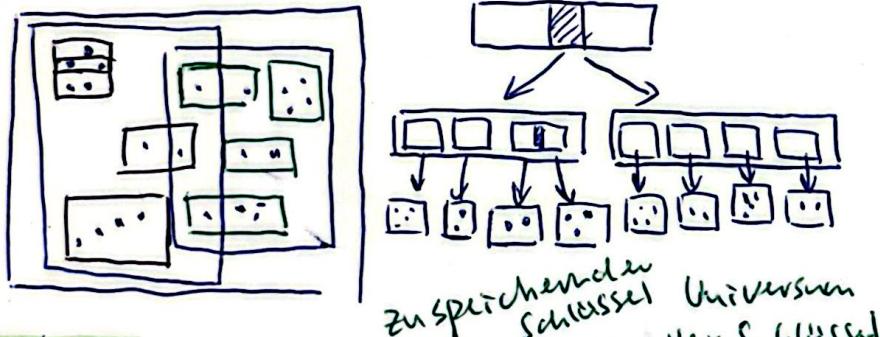
min Höhe

$$h \geq \log_{2k+1}(S+1) - 1 \approx \log_{2k+1} S$$

## B+-Baum

- innere Knoten enthalten keine Daten, nur Wegweiser-Werte
- Blattknoten enthalten Paarsatz mit zugehörigen Schlüsselwerten.

## R-Baum



## Hashing

- $|U|$  sehr groß ist und  $|S| << |U|$
- Perfekt Hashfunktion:  $h(k_i) = h(k_j) \Leftrightarrow i=j \Rightarrow$  Kein Kollisionen.
- Minimal Hashfunktion:  $m=n$ , genau so viele Plätze wie Elemente benötigt werden

## Offenes Hashing (mit geschlossener Adressierung)

$8 \rightarrow 8$	0 → [8] ↗
$3 \rightarrow 3$	1 → [ ] ↗
$13 \rightarrow 3$	2 → [ ] ↗
$6 \rightarrow 6$	3 → [3] ↗ → [ ] ↗
$4 \rightarrow 4$	4 → [ ] ↗
$10 \rightarrow 1$	5 → [ ] ↗
	6 → [6] ↗
	7 → [ ] ↗
	8 → [8] ↗
	9 → [ ] ↗

hashing funktion :=

$$x \% 10$$

↑ size of hashing Table

⇒ Komplexität  $> O(1)$

## Geschlossenes Hashing (mit offener Adressierung)

- Linear sondieren.  $h(x) = x \bmod m$

$$h(x, j) = (h(x) + c_1 j) \bmod m$$

↑ jump

5	24	4	11	15	16	26	2
5	2	4	0	4	5	4	2

Index	0	1	2	3	4	5	6	7	8	9	10
Hashtabelle	11	24		4	5	15	16	26		2	
Sondierung	0	0		0	0	1	1	2		4	

- Quadratisches Sondieren mit  $c=2$

$$h(x, j) = (h(x) + c_1 j^2) \bmod m$$

Index	0	1	2	3	4	5	6	7	8	9	10
Hashtabelle	11	26	24		4	5	15	16		2	
Sondierung	0	2	0		0	0	1	1		2	

⇒ Problem Clusturbildung = für gleiche Schlüssel werden dieselben Positionen sondiert.

⇒ Doppelhashing.

- 2 unabhängige Hashfunktion
- If mod of index is full, insert original number into 2. equation
- The second equation give you the number of buckets to jump

Beispiel 1 9 5 0 8 9 3 9

$$h_1(x) = x \% 5 \quad h_2 = 3 - (x \% 3)$$

0	50
1	89
2	39
3	
4	19

$$3 - (89 \% 3) = 1$$

$$3 - (39 \% 3) = 3$$

Beispiel 2 4 9 1 4 1 1 9

$$h_1(x) = x \% 5 \quad h_2 = 3 - (x \% 3)$$

0	14
1	1
2	9
3	19
4	4

$$3 - (9 \% 3) = 3$$

$$3 - (14 \% 3) = 1$$

$$3 - (1 \% 3) = 2$$

⇒ min Hashfunktion.

## (T5) Graph

**Gerichtete Graphen**  $G(V, E)$  :  $|E| \leq |V|^2$

Vorgänger Nachfolger



adjazent

**ungerichtete Graphen**: Kantenrelation  $E$  symmetrisch  $\Rightarrow (v, w) \in E \Rightarrow (w, v) \in E$

**Grad**: Anzahl der ein- und ausgehenden Kanten



**Pfad**: eine Folge von Knoten  $v_0, \dots, v_{n-1}$  mit  $(v_i, v_{i+1}) \in E$  für  $0 \leq i \leq n-1$ .

**Länge eines Pfad**: Anzahl der Kanten auf dem Pfad.

**einfacher Pfad**: einen Knoten höchstens einmal enthält

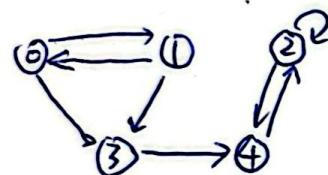
**Zyklus**: ein Pfad mit  $v_0 = v_{n-1}$  und Länge  $n \geq 2$  (gerichtet) |  $n \geq 3$  (ungerichtet)

**Teilgraph**  $G' = (V', E')$  eines Graphen  $G = (V, E)$  ist ein Graph mit  $V' \subseteq V$  und  $E' \subseteq (V' \times V') \cap E$

**Repräsentation von Graphen**:

### Adjazenzmatrix

$A_{ij} = \begin{cases} \text{true} & \text{falls } (v_i, v_j) \in E \\ \text{false} & \text{sonst} \end{cases}$

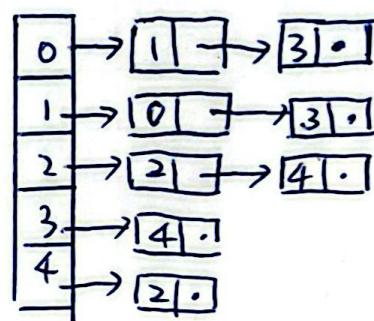


- $V$ : Entscheidung, ob  $(i, j) \in E$  gehört in Zeit  $O(1)$
- $N$ : Platzbedarf:  $O(|V|^2)$

		nach				
		0	1	2	3	4
von	0	0	1	0	1	0
	1	1	0	0	1	0
2	0	0	1	0	1	0
3	0	0	0	0	1	0
4	0	0	1	0	0	0

### Adjazenzliste

eine Liste der Nachbarknoten



$V$ : gering Platzbedarf  
 $O(|V| + |E|)$

Initialisierung

In Zeit  $O(|V| + |E|)$

$N$ : Entscheidung, ob  $(i, j) \in E$  benötigt  
 $O(\frac{|E|}{|V|})$  Zeit in A.C.

### Minimale Spannbaum

Prim - Algo.      Kruskal - Algo.      } Greedy Method.

- **ungerichteter, zusammenhängender Graph**  $G = (V, E)$  mit Kantengewichten  $c: E \rightarrow \mathbb{R}$   $\Rightarrow$  **ungerichteter Subgraph**  $G' = (V, E')$  mit  $E' \subseteq E \wedge G'$  zyklenfrei & zusammenhängend (Spannbaum)  $\wedge G'$  ist minimal.

### Prim - Algo.

$V' \leftarrow v_0$  beliebig

$E' \leftarrow \emptyset$

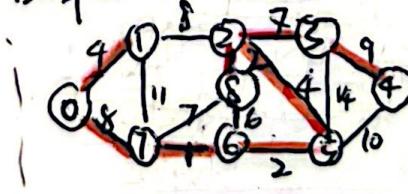
solang  $V' \neq V$

$$(u, v) = \arg \min_{e \in E \setminus E'} (e)$$

$$V' = V' \cup v$$

$$E' = E' \cup \{(u, v)\}$$

Beispiel:



Laufzeitkomplexität:  
 $O(|V|^2)$

$V'$	$E'$
{0}	$\emptyset$
{0, 1, 3}	{(0, 1), (0, 3)}
{0, 1, 7, 3}	{(0, 1), (0, 7), (7, 3)}
{0, 1, 7, 6, 3}	{(0, 1), (0, 7), (7, 6), (6, 3)}
{0, 1, 7, 6, 5, 3}	{(0, 1), (0, 7), (7, 6), (6, 5), (5, 3)}
{0, 1, 7, 6, 5, 2, 3}	{(0, 1), (0, 7), (7, 6), (6, 5), (5, 2), (2, 3)}
{0, 1, 7, 6, 5, 2, 8, 3}	{(0, 1), (0, 7), (7, 6), (6, 5), (5, 2), (2, 8), (8, 3)}
{0, 1, 7, 6, 5, 2, 8, 13}	{(0, 1), (0, 7), (7, 6), (6, 5), (5, 2), (2, 8), (8, 13)}
{0, 1, 7, 6, 5, 2, 8, 13, 12}	{(0, 1), (0, 7), (7, 6), (6, 5), (5, 2), (2, 8), (8, 13), (12, 13)}

## Kruskal - Algo

$$E' \leftarrow \emptyset$$

Sortiere E auf aufsteigend nach c(E)

Solang  $E \neq \emptyset$

$$e \leftarrow \min(E)$$

$$E = E - \{e\}$$

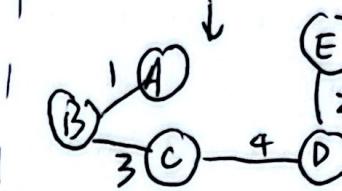
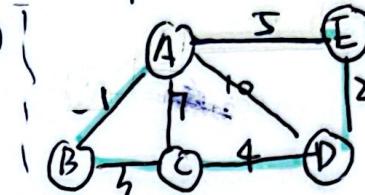
Falls  $G[V, E' \cup \{e\}]$  zyklusfrei

$$E' = E' \cup \{e\}$$

Rauf-Zerzkompakt:  $O(|E| \log |E|)$

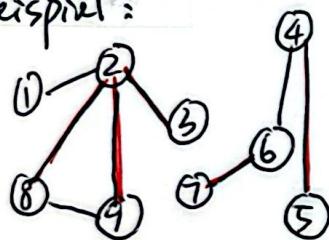
\* Union-Find - Algorithmus  
(Disjoint Set)

## Beispiel



E
(A,B) $\rightarrow 1$
(E,D) $\rightarrow 2$
(B,C) $\rightarrow 3$
(C,D) $\rightarrow 4$
(A,E) $\rightarrow 5$
(A,C) $\rightarrow 7 \rightarrow$ Zyklus
(D,E) $\rightarrow 10$

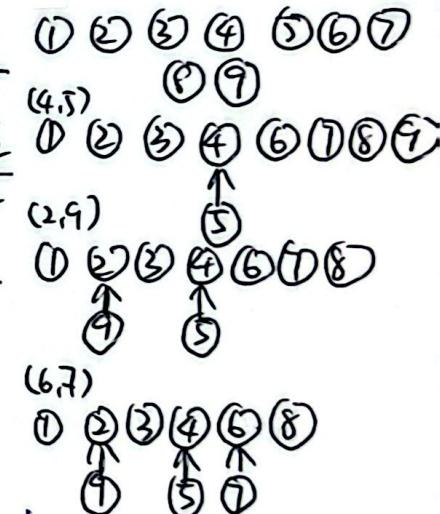
Beispiel:



\* find kann  $O(n)$  bzw  
 $O(\log n) \rightarrow$  Heuristiken  
 $\Rightarrow$  Pfadkompression.

$$\text{find}(8), \text{find}(9) \rightarrow 8, 9$$

Kante	1	2	3	4	5	6	7	8	9
Init	1	2	3	4	5	6	7	8	9
4-5	1	2	3	4	4	6	7	8	9
2-9	1	2	3	4	4	6	7	8	2
6-7	1	2	3	4	4	6	6	8	2
2-3	1	2	2	4	4	6	6	8	2
2-8	1	2	2	4	4	6	6	2	2
4-6	1	2	2	4	4	4	6	2	2
1-2	1	1	2	4	4	4	6	2	2
8-9	1	1	2	4	4	4	6	1	1



Suche nach kürzesten Wegen

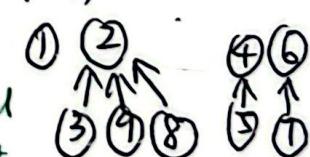
(2,3)

Vollständig  
Wenn es Lösung gibt, so wird diese auch gefunden.

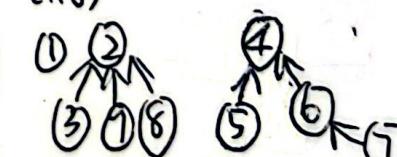
Optimal  
Es wird immer eine optimale Lsg.

Optimal effizient

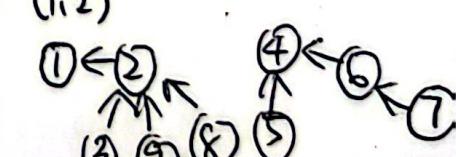
krum Algo, der die gleiche Heuristik und weniger Knoten benötigt



(4,6)



(1,2)



All Pairs Shortest Path  $\leftarrow$  allen Knoten  $\Rightarrow$  allen anderen Floyd  
Single Source Shortest Path  $\leftarrow$  einem Knoten  $\leftarrow$  Dijkst  
Single Destination Shortest Path  $\leftarrow$  allen anderen  $\leftarrow$  Ax  
Single Destination Shortest Path  $\leftarrow$  einem Ziel



\* Heuristik  $\rightarrow$  Schätzfunktion

- Sie darf die tatsächlichen Kosten zum Ziel nicht überschätzen

- Wenn die Heuristik = 0, d.h. man verwendet in diesem Fall keine Heuristik

$$\Rightarrow A^* = \text{Dijkst.}$$

Gegeben: Gerichteter Graph  $G$  mit Kostenfunktion (Adjazenzmatrix)  
 $\Rightarrow$  Pfad von  $v_0$  zu jedem Knoten  $w$  mit minimalen Gesamtkosten.

Dijkstra-Algo. — Greedy method.

Dijkstra  $(G, v_0)$

$S \leftarrow \{v_0\}$

for all  $v \in V$ :  $D[v] \leftarrow C[v_0, v]$

while  $(V - S) \neq \emptyset$

$w_{\min} \leftarrow \arg \min_{w \in V - S} D[w]$

$S \leftarrow S \cup \{w_{\min}\}$

for all  $v \in V - S$

$D[v] = \min(D[v], D[w_{\min}] + C[w_{\min}, v])$

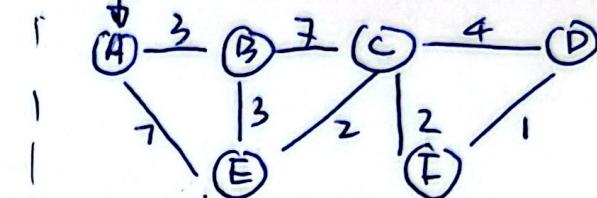
• liefert optimale Lösung, nicht nur Nächste

• negativen Kosten  $\Rightarrow$  kein eindeutigen Pfad.

• Komplexität: zusammenhängend mit Adjazenzmatrix  $O(|V|^2)$

Einsatz als „All Pair Shortest Path“ prinzipiell möglich  $\Rightarrow O(|V|^3)$

Beispiel



K	V_k	$D_k(B)$	$D_k(C)$	$D_k(D)$	$D_k(E)$	$D_k(F)$
1	A	3	$\infty$	$\infty$	7	$\infty$
2	B	3	$\infty$	$\infty$	6	$\infty$
3	E	3	8	0	6	$\infty$
4	C	3	8	12	6	10
5	F	3	8	11	6	10
6	D	3	8	11	6	10

Floyd -Algorithmus. — Dynamische Programmierung.

Für alle Knotenpaare  $i, j \in \{1, \dots, |V|\}$

$d[i, j] = c[i, j]$

Für alle  $k \in \{1, \dots, |V|\}$

Für alle  $j \in \{1, \dots, |V|\}$

$d[i, j] = \min(d[i, j], d[i, k] + d[k, j])$

$K=A$

$$\begin{pmatrix} 0 & 5 & 7 & \infty \\ 0 & 0 & 1 & \infty \\ \infty & 0 & 0 & 2 \\ 3 & 8 & 10 & 0 \end{pmatrix}$$

$K=B$

$$\begin{pmatrix} 0 & 5 & 6 & \infty \\ 0 & 0 & 1 & \infty \\ \infty & 0 & 0 & 2 \\ 3 & 8 & 9 & 0 \end{pmatrix}$$

$K=C$

$$\begin{pmatrix} 0 & 5 & 6 & 8 \\ 0 & 0 & 1 & 3 \\ \infty & \infty & 0 & 2 \\ 3 & 8 & 9 & 0 \end{pmatrix}$$

$K=D$

$$\begin{pmatrix} 0 & 5 & 6 & 8 \\ 6 & 0 & 1 & 3 \\ 5 & 10 & 0 & 2 \\ 3 & 8 & 9 & 0 \end{pmatrix}$$

Komplexität: Zeitkomplexität  $O(|V|^3)$

Platzkomplexität  $O(|V|^2)$

\* Warshall - Algo.

• berechnet die transitive Hülle eines Graphen.

• ähnlich wie Floyd.

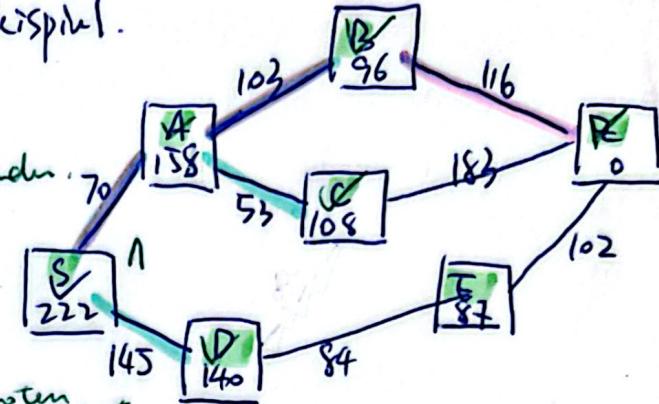
$\rightarrow$  Statt der Weglänge  $d[v, w]$   $\Rightarrow$  die Existenz einer Verbindung  $a[v, w]$

→ kleinste transitive Relation  $R^+$

# A\* - Algo

g(x) → h(x) → nicht überschreiten.  
 das sonst die optimal  
 Lsgung vielleicht nicht gefunden.  
 tatsächl. + geschätzte  
 Kosten = Gesamt  
 Kosten.  
 (Kürzestenweg (lon x bis  
 von Start zu x) zum Ziel)

## Beispiel.



Schritt | closed List | Der kürzeste Weg zum Knoten wurde gefunden, nicht Verdacht  
 openList - eine Weg zum Knoten ist bekannt, aber es können kürzere Wege.

0.	-	(S, 222)
1.	(S, 0)	(A, 228) (D, 285)
2.	(S, 0) (A, 70)	(C, 231) (B, 269) (D, 285)
3.	(S, 0) (A, 70) (C, 123)	(B, 269) (D, 285) (Z, 306)
4.	(S, 0) (A, 70) ((, 123) (B, 173))	(D, 285) (Z, 286)
5.	(S, 0) (A, 70) ((, 123)   B, 173) (D, 145)	(Z, 286) (E, 316)
6.	S → A → B → Z	

## Flussnetz

### Ford - Fulkerson - Method

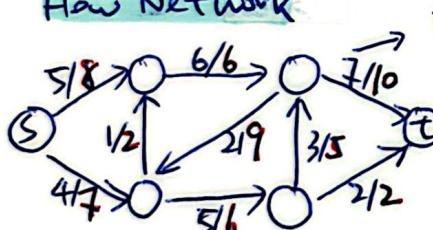
Laufzeit

beziiglich schlecht sein.

→ sehr langsam Erhöhung des Gesamtflusses.

To find the maximum flow (and min-cut) as a by product, the FF Method finds augmenting paths through the residual graph and augments the flow until no more augmenting path can be found.

### Flow Network



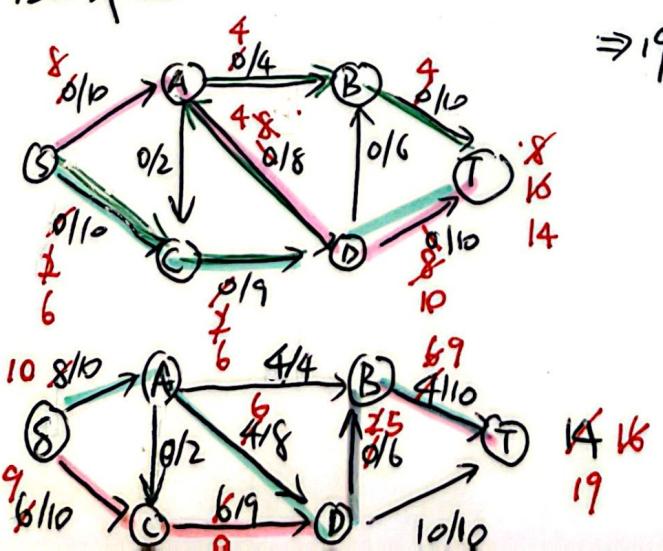
$$\text{Flow} = \text{① } f(u,v) \leq c(u,v)$$

$$\text{② } f(u,v) = f(v,u) = 0$$

$$\text{③ } f^{\text{in}}(u) = f^{\text{out}}(u)$$

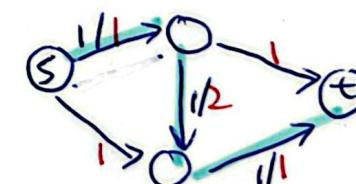
$$2+4=1+5$$

### Beispiel:



$$\text{Residual Capacity } (u,v) \in E$$

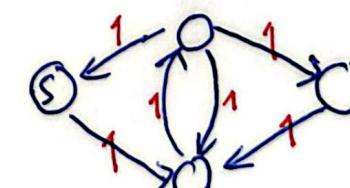
$$f(u,v) = \begin{cases} c(u,v) - f(u,v) & (v,u) \in E \\ f(v,u) & \text{unsent.} \\ 0 & \end{cases}$$



### Residual Graph.

$$V_f = V$$

$$E_f = \{(u,v) \in V \times V : f(u,v) > 0\}$$



⇒ Edmonds - Karp - Alg. (shortest

# Algorithmische Paradigmen. (K6)

**Backtracking.** → Damensproblem/Sudoku, Solitär-Brettspiele/Wegsuche A → B

• Lösung eines Problems durch **Versuch und Irrtum**.

• Schrittweise „herantasten“ an die Gesamtlösung

• Wenn die Teillösung **nicht** zu einer Gesamtlösung erweitert werden kann, setze letzten Schritt **zurück** und probiere **alternative Schritte**.

• Diese Vorgehensweise entspricht einer Tiefensuche im Suchbaum.

**n-Damenproblem**:

•  $n \times n$  Schachfeld  $\leftarrow n$  Damen • dieselbe Zeile / Spalte / Diagonale X

**Branch and Bound** → (NP vollständig) Optimierungsproblem → Traveling Salesperson problem (TSP)

• Wie bei Backtracking (Trial & error)

• Schrittweise Annähern an die Gesamtlösung

• Für die Teillösung werden **Schranken** (Bounds) berechnet.

• Teillösung werden dann **verworfen**, wenn Optimum nicht mehr erreicht werden kann.

## TSP

• Ein Handelsreisender plant eine **Rundreise** durch mehrere Städte.

• Start und Ziel der Rundreise ist eine vorgegebene Stadt.

• Jede Stadt soll nur einmal besucht werden.

• Die Kosten der Reise sollen minimal sein.

• Reihenfolge?

Beispiel:

	A	B	C	D	min
A	-	10	15	20	10
B	5	-	9	10	5
C	6	13	-	12	6
D	8	8	9	-	8

Untere Schranke.

-	10	15	20	15
5	-	9	10	5
6	13	-	12	12
8	8	9	-	8

$\Sigma 40$

-	10	15	20	10
5	-	9	10	5
6	13	-	12	12
8	8	9	-	8

$\Sigma 33$

-	10	15	20	15
5	-	9	10	5
6	13	-	12	12
8	8	9	-	8

$\Sigma 40$

-	10	15	20	10
5	-	9	10	5
6	13	-	12	12
8	8	9	-	8

$\Sigma 33$

-	10	15	20	10
5	-	9	10	5
6	13	-	12	12
8	8	9	-	8

$\Sigma 40$

-	10	15	20	10
5	-	9	10	5
6	13	-	12	12
8	8	9	-	8

$\Sigma 40$

-	10	21	10
5	-	9	10
6	13	-	12
8	8	9	-

$\Sigma 34$

-	10	21	10
5	-	9	10
6	13	-	12
8	8	9	-

$\Sigma 34$

-	10	21	10
5	-	9	10
6	13	-	12
8	8	9	-

$\Sigma 34$

10	20	10
5	-	9
6	-	12
8	-	11

ohne(2,4)

10	21	10
5	-	9
6	-	12
8	-	11

mit(2,4)

10	-	10
5	-	9
6	-	12
8	-	11

mit(2,4)

10	-	10
5	-	9
6	-	12
8	-	11

mit(2,4)

10	-	20
5	-	9
6	-	12
8	-	11

mit(2,4)

=obere Schranke

eine Lösung.

W/II Standig

Umg.

Divide-and-Conquer → Merge Sort / Binary Search / Quick Sort

Greedy (gierige) Algo. → Prim-Algo / Kruskal-Algo / Dijkstra-Algo

Schrittweise die in lokaler Situation beste Entscheidung treffen.  
→ Rucksackproblem.

Rucksackproblem.  $O(n)$

- definiert GröÙe bzw maximalen Gewicht G
- Objekte → Größe/Gewicht, Wert  
⇒ Maximaler Wert des Behälterinhaltes.

Beispiel.

Höchsten Bewertung, das noch in den Rucksack passt und packe dieses hinzufügen.  
Je kleiner desto besser  
Je wertvoller desto besser  
Je größer das Verhältnis Wert zu Größe desto besser.

Rucksack	
Object 1 (23€)	✓ Object 2 (15€)
03 (5€)	04 (15€)

Dynamische Programmierung. → Floyd-Algo / Edit Distance.

Erst kleinste Probleme lösen und dann deren Lösung sukzessive nutzen um größere Problemlösungen zu konstruieren.

Beispiel: Edit Distance  $O(m \cdot n)$

	H	A	L	L	E
H	0	1	2	3	4
A	1	0	1	2	3
L	2	1	0	1	2
V	3	2	1	1	2
S	4	3	2	2	2

Sweep-Line — Schnitt von Liniensegment.

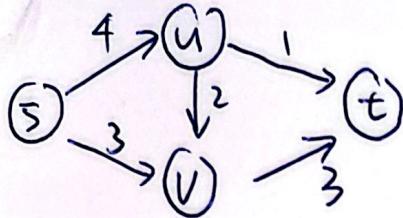
Teste nur die Segmentpaare, die gleichzeitig von einer horizontalen Sweep Line geschnitten werden.

Speicherkomplexität  $O(n+r)$  mit  $n$  Segmente,  $r$  Schnittpunkte.

Verbesserter Sweep-Line Algo

Laufzeitkomplexität  $O((n+r) \log n)$

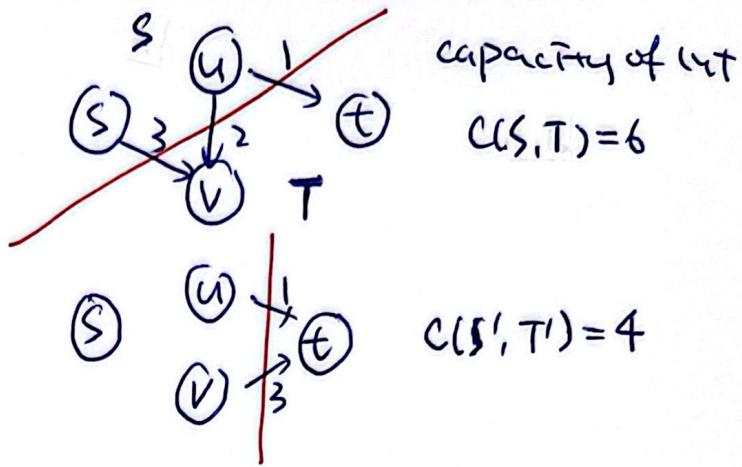
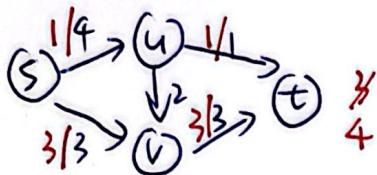
## Min-Cut-Max Theorem



$$S \cup T = V$$

$$S \cap T = \emptyset$$

min cut = max flow



## Eulerweg, Eulerkreis

Ein Weg heißt **Eulerweg**, wenn jede Kante aus  $E$  genau einmal vorkommt. Falls Anfang = Endknoten ist es so handelt es sich um einen **Eulerkreis**.

## Konvexe Menge

Eine Teilmenge  $S \subseteq \mathbb{R}^d$  ist KM, wenn für jedes Paar von Punkten  $q, p \in S$  das Intervallsegment  $\overline{qp}$  vollständig in  $S$  enthalten ist.

## Konvexe Hülle

kleinste KM, die  $S$  enthält.  
Jarvis's March Algorithm