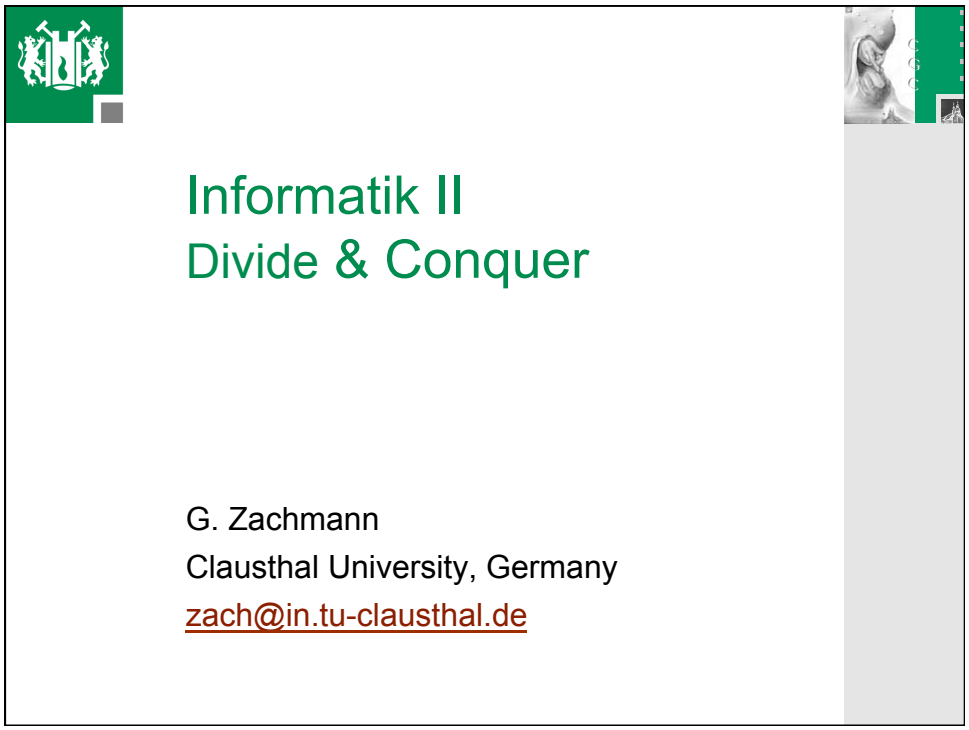
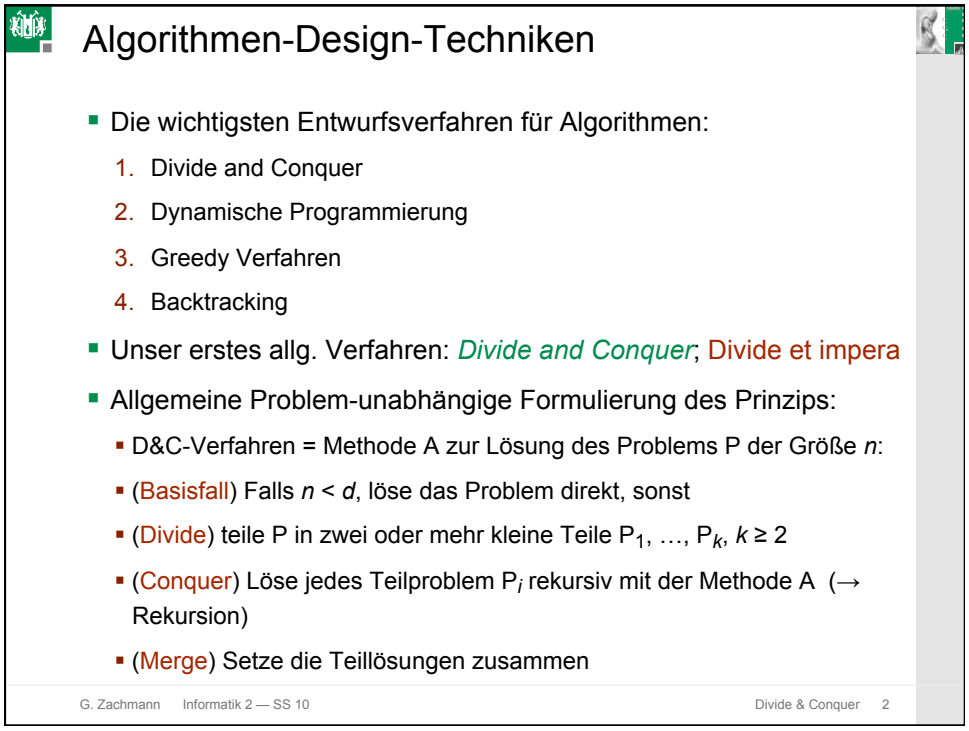


# Informatik II
## Divide & Conquer

G. Zachmann
Clausthal University, Germany
zach@in.tu-clausthal.de



## Algorithmen-Design-Techniken

- Die wichtigsten Entwurfsverfahren für Algorithmen:
  1. Divide and Conquer
  2. Dynamische Programmierung
  3. Greedy Verfahren
  4. Backtracking
- Unser erstes allg. Verfahren: *Divide and Conquer*; Divide et impera
- Allgemeine Problem-unabhängige Formulierung des Prinzips:
  - D&C-Verfahren = Methode A zur Lösung des Problems P der Größe $n$:
  - (Basisfall) Falls $n < d$, löse das Problem direkt, sonst
  - (Divide) teile P in zwei oder mehr kleine Teile $P_1, \ldots, P_k$, $k \geq 2$
  - (Conquer) Löse jedes Teilproblem $P_i$ rekursiv mit der Methode A (→ Rekursion)
  - (Merge) Setze die Teillösungen zusammen

## Beispiel für Divide-and-Conquer: Quicksort

- Quicksort ist ein klassisches Divide-and-Conquer-Verfahren:
  1. Divide = Partitionierung mittels Pivot
  2. Conquer = rekursiver Aufruf (kleineres Arrays sortieren)
  3. Combine (merge) = keine Arbeit hier nötig, da die Teilarrays *in-situ* sortiert wurden
  4. Basisfall = Array mit weniger als 1 Element



## Schnelle Multiplikation

- Schulverfahren (nach Adam Riese):

$$
\begin{array}{l}
x_{n-1}x_{n-2} \cdots\cdots x_1x_0 \;\times\; y_{n-1}y_{n-2} \cdots\cdots y_1y_0 \\
\hline
\qquad\qquad\qquad\qquad\qquad\qquad c^0{}_n c^0{}_{n-1} \cdots\cdots\cdots\; c^0{}_0 \\
\qquad\qquad\qquad\qquad\qquad c^1{}_n c^1{}_{n-1} \cdots\cdots\cdots\; c^1{}_0 \\
\qquad\qquad\qquad\qquad\quad \vdots \qquad\qquad\qquad \vdots \\
\qquad\qquad\qquad\qquad\quad \ddots \qquad\qquad\quad \ddots \\
\qquad\qquad\qquad\qquad\qquad \ddots \qquad\qquad \ddots \\
c^{n-1}{}_n c^{n-1}{}_{n-1} \cdots\cdots\cdots\; c^{n-1}{}_0 \\
\hline
z_{2n}\, z_{2n-1} \cdots\cdots\cdots\cdots\cdots\cdots\cdots\; z_1\, z_0
\end{array}
$$

- Anzahl Ziffern-Multiplikationen: $n_2$
- Anzahl Ziffern-Additionen: ~ $n_2$

# Algorithmus von Karatsuba und Ofman

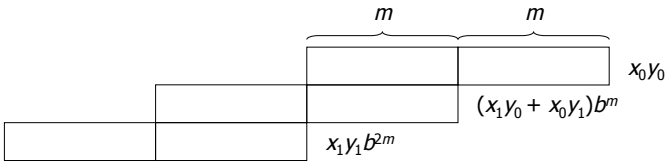
- Sei $n = 2m$ und $b$ die Basis einer $b$-adischen Darstellung (z.B. dezimal → $b = 10$):

$x = x_0 + x_1 b^m$ $\qquad$ $y = y_0 + y_1 b^m$

Ann.: shift ist gratis!

$xy = x_0y_0 + (x_1y_0 + x_0y_1)\, b^m + x_1y_1\, b^{2m}$

3 Add. der Länge n=2m *(?)* $\qquad$ 4 Mult. der Länge $m$




2 Additionen der Länge $\frac{n}{2} = m$



# Aufwand

- Annahme: $n = 2^k$

$$
\begin{aligned}
T(n) &= 4T\left(\tfrac{n}{2}\right) + (2\tfrac{n}{2} + n) \\
&= 4T\left(\tfrac{n}{2}\right) + 2n \\
&= 4\left(4T\left(\tfrac{n}{4}\right) + 2\tfrac{n}{2}\right) + 2n \\
&= 16T\left(\tfrac{n}{4}\right) + 4n + 2n \\
&\;\;\vdots \\
&= \underbrace{4^k}_{2^{2k}} T(1) + 2n \underbrace{\sum_{i=0}^{k-1} 2^i}_{\approx 2^k = n} \\
&\approx c(n^2 + 2n^2) \in \Theta\left(n^2\right)
\end{aligned}
$$

Additionen

Ziffern-Additionen

Ziffern-Multiplikationen

- Fazit: noch keine Verbesserung

## Der Trick von Karatsuba und Ofman

- Zu berechnen:

$$xy = x_0y_0 + (x_1y_0 + x_0y_1)b^m + x_1y_1b^{2m}$$

- Umformung:

$$(x_1y_0 + x_0y_1) = (x_1 + x_0)(y_0 + y_1) - x_0y_0 - x_1y_1$$

- Berechne also nur noch 3 Produkte:

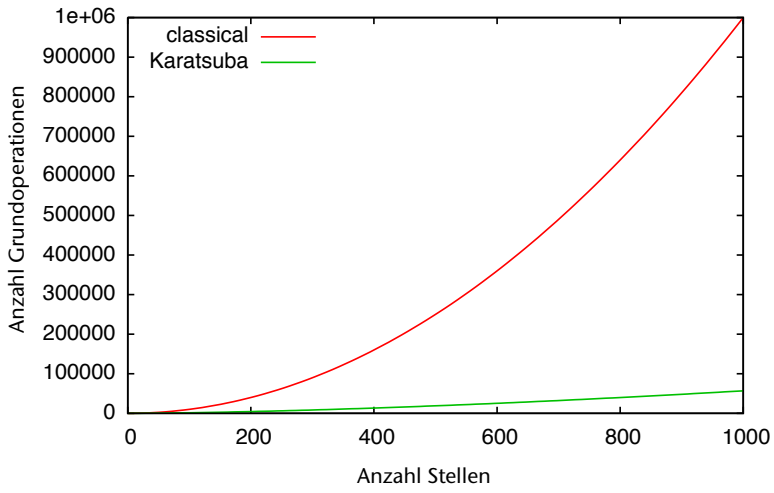
$$P_1 = x_0y_0$$
$$P_2 = x_1y_1$$
$$P_3 = (x_1 + x_0)(y_0 + y_1)$$



## Aufwand

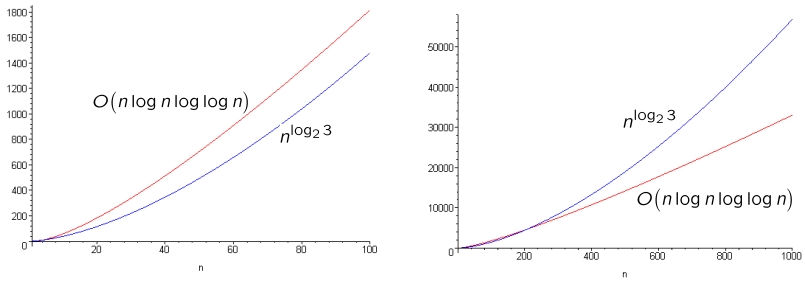
$$\begin{aligned} T(n) &= 3T\left(\tfrac{n}{2}\right) + cn \\ &= 3\left(3T\left(\tfrac{n}{4}\right) + c\tfrac{n}{2}\right) + cn \\ &\vdots \\ &= 3^k T(1) + cn\sum_{i=0}^{k-1}\left(\tfrac{3}{2}\right)^i \\ &\in O\left(3^{\log_2 n}\right) = O\left(n^{\log_2 3}\right) \end{aligned}$$

- Übungsaufgabe: Anzahl Additionen besser abschätzen
- Der Algorithmus von Karatsuba und Ofman ist schneller, als der bekannte Algorithmus zur Multiplikation, die rekursiven Aufrufe benötigen allerdings mehr Speicherplatz (*space-time trade-off*)






- Asymptotische Verbesserung von Schönhage und Strassen
  - Schnelle Multiplikation mit einer Laufzeit von $O(n \log n \log \log n)$



## Schnelle Matrix-Multiplikation

- Aufgabe: berechne $C = A \cdot B, \quad A, B \in \mathbb{R}^{n \times n}$
- Normale Matrix-Multiplikation (MM) benötigt O($n^3$) Multiplikationen (und etwa genauso viele Additionen)
- Geht es schneller?
- Idee: zerlege *A*, *B*, *C* in Blöcke:

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

mit $a_{ij}, b_{ij}, c_{ij} \in \mathbb{R}^{\frac{n}{2} \times \frac{n}{2}}$

- Ausmultiplizieren ergibt:

$$c_{11} = a_{11}b_{11} + a_{12}b_{21}$$
$$\vdots$$
$$c_{22} = a_{21}b_{11} + a_{22}b_{22}$$

→ 8 MM der Größe $\frac{n}{2} \times \frac{n}{2}$



- Idee: berechne diese rekursiv durch Zerlegung in Blöcke
  → Divide-and-conquer-Algorithmus für o.g. Aufgabe
- Aufwand:

$$T(n) = 8T\left(\tfrac{n}{2}\right) + cn^2 \quad \text{(für Additionen)}$$
$$= 8\left(8T\left(\tfrac{n}{4}\right) + c\left(\tfrac{n}{2}\right)^2\right) + cn^2$$
$$= 8^2 T\left(\tfrac{n}{2^2}\right) + \tfrac{8}{2^2}cn^2 + cn^2$$
$$= \vdots$$
$$= 8^k T(1) + cn^2 \sum_{i=0}^{k-1} 2^i$$
$$\approx c'n^3 + 2^k cn^2 \quad \text{(für Additionen)}$$
$$\approx c'n^3 + cn^3 \in O\left(n^3\right)$$

(für Multiplikationen)

## Idee von Strassen [1969]

- Berechne zunächst etwas umständliche Zwischenprodukte:

$$Q_1 \equiv (a_{11} + a_{22})(b_{11} + b_{22})$$
$$Q_2 \equiv (a_{21} + a_{22})b_{11}$$
$$Q_3 \equiv a_{11}(b_{12} - b_{22})$$
$$Q_4 \equiv a_{22}(-b_{11} + b_{21})$$
$$Q_5 \equiv (a_{11} + a_{12})b_{22}$$
$$Q_6 \equiv (-a_{11} + a_{21})(b_{11} + b_{12})$$
$$Q_7 \equiv (a_{12} - a_{22})(b_{21} + b_{22})$$

- Damit kann man die $c_{ij}$ so ausrechnen:

$$c_{11} = Q_1 + Q_4 - Q_5 + Q_7$$
$$c_{12} = Q_2 + Q_4$$
$$c_{21} = Q_3 + Q_5$$
$$c_{22} = Q_1 + Q_3 - Q_2 + Q_6$$



## Aufwand

$$\begin{aligned} T(n) &= 7T\left(\tfrac{n}{2}\right) + cn^2 \\ &= 7\left(7T\left(\tfrac{n}{4}\right) + c\left(\tfrac{n}{2}\right)^2\right) + cn^2 \\ &= \cdots \\ &= 7^k T(1) + cn^2 \underbrace{\sum_{i=0}^{k-1} \left(\tfrac{7}{4}\right)^i}_{\approx \left(\frac{7}{4}\right)^{\log_2 n} = n^{\log_2\left(\frac{7}{4}\right)}} \\ &\approx c' n^{\log_2 7} + cn^{2+\log_2\left(\frac{7}{4}\right)} \\ &\approx c' n^{2.8\ldots} + cn^{2.8\ldots} \in O\left(n^{2.8\ldots}\right) \end{aligned}$$

$c' n^{2.8\ldots}$: für Multiplikationen; $cn^{2.8\ldots}$: für Additionen

## Bemerkungen

- Aktuell kleinster Exponent = 2.376
  [Coppersmith & Winograd, 1990]
- Untere Schranke für Exponent = 2 (klar, da $n^2$ viele Elemente)
- Eine ähnliche Formel gibt es für Matrix-Inversion

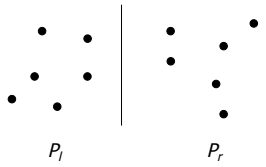


## Das Closest Points Problem

- Gegeben: Menge von $n$ 2D-Punkten
- Gesucht: das Paar, das am dichtesten beieinander liegt
- Offensichtlich gibt es $\frac{n(n-1)}{2}$, die Komplexität eines naïven Algo ist also $O(n^2)$
- Bemerkung: die 1D-Version ist einfach
  - Lösungsansatz: Sortieren
  - Komplexität $O(n \log n)$
- Mit Divide-and-Conquer lässt sich auch für den 2D-Fall $O(n \log n)$ erreichen

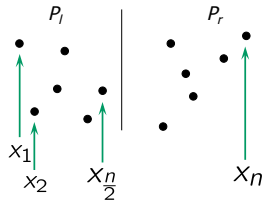
## Ein Divide-and-Conquer-Algorithmus

- Idee:
  - Sortiere Punkte nach ihrer x-Koordinate und teile zur Hälfte (= Median bestimmen)
  - Das dichteste Paar ist:
    - entweder in einer der Hälften,
    - oder hat in jeder Hälfte ein Mitglied




- Phase 1: sortiere die Punkte nach ihrer x-Koordinate

$$p_1, p_2, \ldots, p_{\frac{n}{2}}, p_{\frac{n}{2}+1}, \ldots, p_n$$

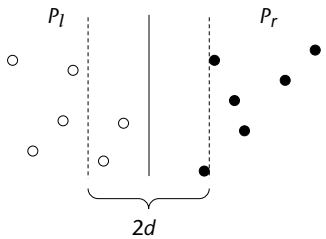


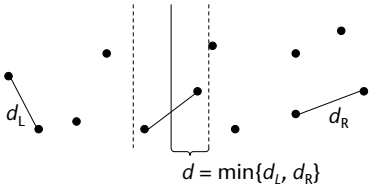



- Phase 2: berechne rekursiv das dichteste Paar und die minimalen Abstände $d_l$ und $d_r$ in

$$P_l = \{p_1, p_2, \ldots, P_{\frac{n}{2}}\} \text{ und}$$
$$P_r = \{P_{\frac{n}{2}+1}, \ldots, p_n\}$$

- Phase 3: finde das dichteste Paar (∘,•) im "Band" um die Mitte mit der Breite $2d$, wobei bekannt ist, daß kein (∘,∘)- oder (•,•)-Paar dichter zusammen ist als $d$



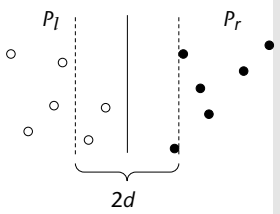



```
def close_pts( A ):
  sortiere A nach der x-Koordinate
  n = len( A )
  dL = close_pts( A[1:n/2] )          # T(n/2)
  dR = close_pts( A[n/2+1:n] )        # T(n/2)
  d = min( dL, dR )
  dM = suche Lösung in der Mitte      # ?
  return min( dL, dR, dM )
```



## Phase 3

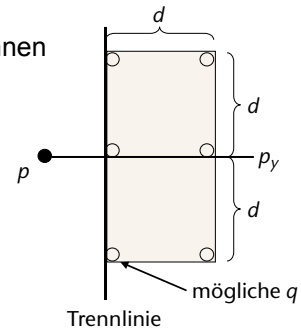
- Sortiere Punkte innerhalb des "Bandes" nach y-Koordinaten
- Gehe Punkte im Band der Reihe nach durch
- Für jeden solchen Punkt *p* betrachte alle Punkte *q* aus dem Band, die
  1. auf der "anderen" Seite der Trennlinie liegen
  2. deren y-Koordinate im Intervall $[p_y\text{-}d, p_y\text{+}d]$ liegen
- Pointer für *p* geht linear, Pointer für *q* "oszilliert"
- Bestimme alle Abstände und wähle den kleinsten
- Behauptung: zu jedem *p* aus dem Band kommen nur maximal 6 Punkte *q* aus dem Band in Frage → Aufwand für das gesamte Band ist $O(n) + O(n \log n)$



## Beweis

- Ann.: *p* und *q* sind potenzielle dichteste Punkte aus dem Band

1. Es gilt: *p* muß (oBdA) links und *q* rechts von der Trennlinie liegen
2. *q* kann nicht rechts außerhalb des Bandes liegen
3. Nur Punkte mit *y*-Koordinate im Intervall $[p_y\text{-}d, p_y\text{+}d]$ können Partner von *p* sein
4. Keine 2 Punkte im grauen Rechteck können dichter als *d* aneinander liegen!
5. Max 6 Punkte mit dieser Eigenschaft können in das Rechteck gepackt werden

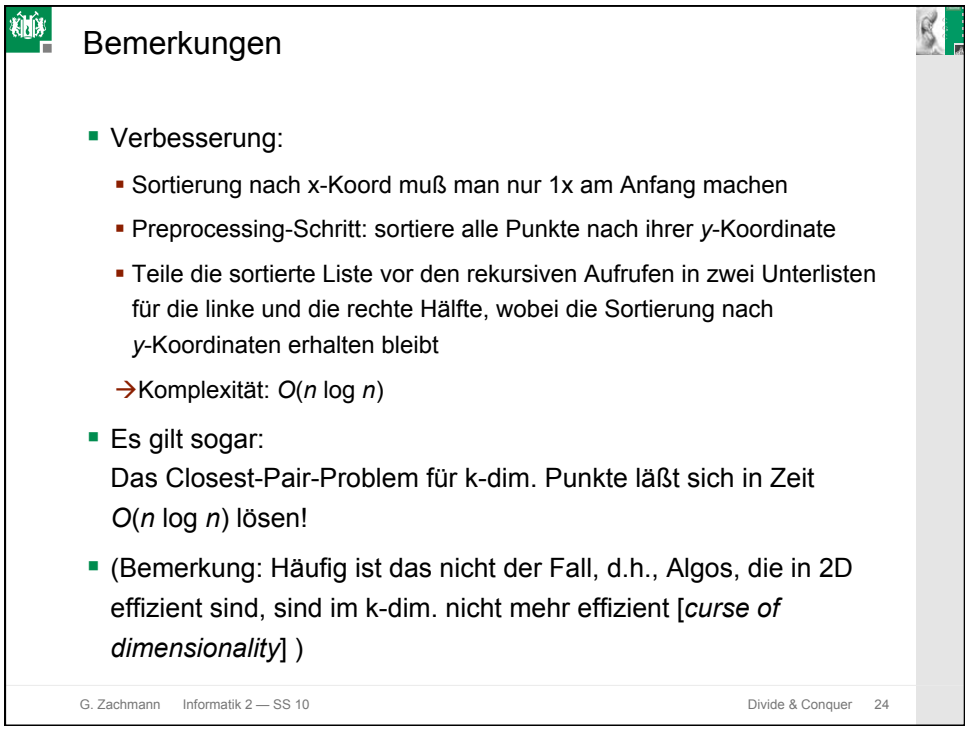





## Zeit-Komplexität

- Einzelschritte:
  1. Divide = Partitionieren (Sortieren nach x-Koordinate) → O(n log n)
  2. Rekursion (Conquer) $= 2T\left(\frac{n}{2}\right)$
  3. Merge = Sortieren nach y-Koordinaten + konstanter Aufwand pro Punkt im Band → *O(n* log *n)* + *O(n)*
- Annahme: $n = 2^k$

$$
\begin{aligned}
T(1) &= 1 \\
T(n) &= 2T\left(\tfrac{n}{2}\right) + 2n\log n \\
&= 4T\left(\tfrac{n}{4}\right) + 4\left(\tfrac{n}{2}\right)\log\left(\tfrac{n}{2}\right) + 2n\log n \\
&= 4T\left(\tfrac{n}{4}\right) + 2n(\log n - 1) + 2n\log n \\
&\quad \cdots \\
&= 2^k T\left(\tfrac{n}{2^k}\right) + 2n\big(\log n + (\log n - 1) + \cdots + (\log n - k + 1)\big) \\
&= n + 2n(1 + 2 + \cdots + \log n) = n + 2n\tfrac{(\log n+1)\log n}{2} \\
&\in O\big(n\log^2 n\big)
\end{aligned}
$$



## Bemerkungen

- Verbesserung:
  - Sortierung nach x-Koord muß man nur 1x am Anfang machen
  - Preprocessing-Schritt: sortiere alle Punkte nach ihrer *y*-Koordinate
  - Teile die sortierte Liste vor den rekursiven Aufrufen in zwei Unterlisten für die linke und die rechte Hälfte, wobei die Sortierung nach *y*-Koordinaten erhalten bleibt

  →Komplexität: $O(n \log n)$
- Es gilt sogar:
  Das Closest-Pair-Problem für k-dim. Punkte läßt sich in Zeit $O(n \log n)$ lösen!
- (Bemerkung: Häufig ist das nicht der Fall, d.h., Algos, die in 2D effizient sind, sind im k-dim. nicht mehr effizient [*curse of dimensionality*] )