




# Informatik II

## Divide & Conquer



Divide  
et  
Impera

G. Zachmann  
Clausthal University, Germany  
[zach@in.tu-clausthal.de](mailto: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  $P_1, \dots, P_k, k \geq 2$
  - (**Conquer**) Löse jedes Teilproblem  $P_i$  rekursiv mit der Methode A (→ Rekursion)
  - (**Merge**) Setze die Teillösungen zusammen

G. Zachmann Informatik 2 — SS 11 Divide & Conquer 2

## 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

G. Zachmann Informatik 2 — SS 11
Divide & Conquer 3

## Schnelle Multiplikation

- Schulverfahren (nach Adam Riese):

$$\begin{array}{r}
 x_{n-1}x_{n-2} \dots \dots x_1x_0 \quad \times \quad y_{n-1}y_{n-2} \dots \dots y_1y_0 \\
 \hline
 c_n^0 c_{n-1}^0 \dots \dots c_0^0 \\
 c_n^1 c_{n-1}^1 \dots \dots c_0^1 \\
 \cdot \\
 \cdot \\
 \cdot \\
 c_{n-1}^{n-1} c_{n-1}^{n-1} \dots \dots c_0^{n-1} \\
 \hline
 z_{2n} z_{2n-1} \cdot \dots \cdot z_1 z_0
 \end{array}$$

- Anzahl Ziffern-Multiplikationen:  $n^2$
- Anzahl Ziffern-Additionen:  $\sim n^2$

G. Zachmann Informatik 2 — SS 11
Divide & Conquer 4

## Algorithmus von Karatsuba und Ofman

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

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

Ann.: shift ist gratis!

$$xy = x_0 y_0 + (x_1 y_0 + x_0 y_1) b^m + x_1 y_1 b^{2m}$$

3 Add. der Länge  $n=2m$  (?)

4 Mult. der Länge  $m$

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

G. Zachmann Informatik 2 — SS 11
Divide & Conquer 5

## Aufwand

- Annahme:  $n = 2^k$

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

↑

Ziffern-Additionen

↑

Ziffern-Multiplikationen

- Fazit: noch keine Verbesserung

G. Zachmann Informatik 2 — SS 11
Divide & Conquer 6

## 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:
 
$$P_1 = x_0y_0$$

$$P_2 = x_1y_1$$

$$P_3 = (x_1 + x_0)(y_0 + y_1)$$

G. Zachmann Informatik 2 — SS 11 Divide & Conquer 7

## Aufwand

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

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

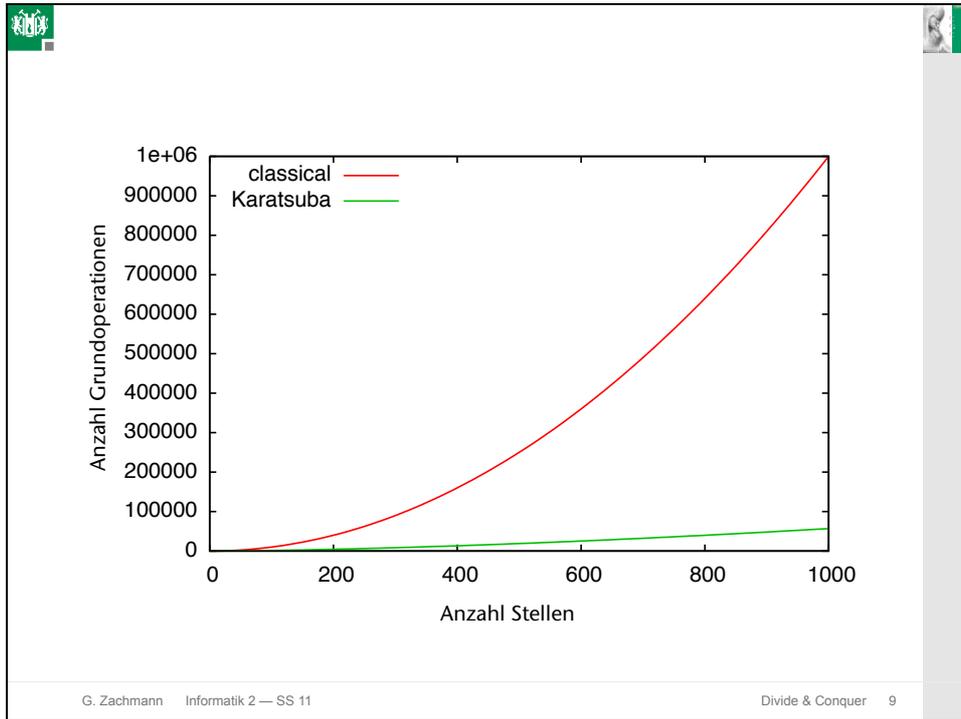
$$\vdots$$

$$= 3^k T(1) + cn \sum_{i=0}^{k-1} \left(\frac{3}{2}\right)^i$$

$$\in O(3^{\log_2 n}) = O(n^{\log_2 3})$$

- 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*)

G. Zachmann Informatik 2 — SS 11 Divide & Conquer 8



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

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

$n^{\log_2 3}$

$n^{\log_2 3}$

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

G. Zachmann Informatik 2 — SS 11 Divide & Conquer 10

## Schnelle Matrix-Multiplikation

- Aufgabe: berechne  $C = A \cdot B$ ,  $A, B \in \mathbb{R}^{n \times n}$
- 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:
 
$$\begin{aligned} c_{11} &= a_{11}b_{11} + a_{12}b_{21} \\ &\vdots \\ c_{22} &= a_{21}b_{11} + a_{22}b_{22} \end{aligned} \quad \rightarrow 8 \text{ MM der Größe } \frac{n}{2} \times \frac{n}{2}$$

G. Zachmann Informatik 2 — SS 11 Divide & Conquer 11

- Idee: berechne diese rekursiv durch Zerlegung in Blöcke  
→ Divide-and-conquer-Algorithmus für o.g. Aufgabe
- Aufwand:
 
$$\begin{aligned} T(n) &= 8T\left(\frac{n}{2}\right) + \overbrace{cn^2}^{\text{für Additionen}} \\ &= 8\left(8T\left(\frac{n}{4}\right) + c\left(\frac{n}{2}\right)^2\right) + cn^2 \\ &= 8^2T\left(\frac{n}{2^2}\right) + \frac{8}{2^2}cn^2 + cn^2 \\ &= \vdots \\ &= 8^k T(1) + cn^2 \sum_{i=0}^{k-1} 2^i \\ &\approx c'n^3 + 2^k cn^2 \quad \text{für Additionen} \\ &\approx \underbrace{c'n^3}_{\text{für Multiplikationen}} + \overbrace{cn^3}^{\text{für Additionen}} \in O(n^3) \end{aligned}$$

G. Zachmann Informatik 2 — SS 11 Divide & Conquer 12

## 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:
 
$$c_{11} = Q_1 + Q_4 - Q_5 + Q_7$$

$$c_{12} = Q_2 + Q_4$$

$$c_{21} = Q_3 + Q_5$$

$$c_{22} = Q_1 + Q_3 - Q_2 + Q_6$$

G. Zachmann Informatik 2 — SS 11
Divide & Conquer 13

## Aufwand

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

für Additionen

für Multiplikationen

G. Zachmann Informatik 2 — SS 11
Divide & Conquer 14

## 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

G. Zachmann Informatik 2 — SS 11 Divide & Conquer 15

## Das Closest Points Problem

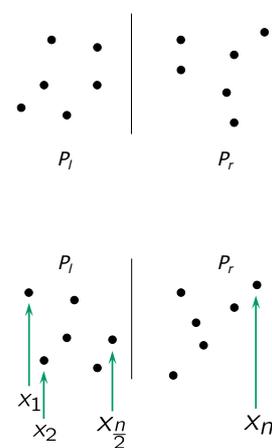
- Gegeben: Menge von  $n$  2D-Punkten
- Gesucht: das Paar, das am dichtesten beieinander liegt
- Offensichtlich gibt es  $\frac{n(n-1)}{2}$ , die Komplexität eines naiven Algo ist also  $O(n^2)$
- Bemerkung: die 1D-Version ist einfach
  - Lösungsansatz: Sortieren
  - Komplexität  $O(n \log n)$
- Mit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

G. Zachmann Informatik 2 — SS 11 Divide & Conquer 16

## 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
 

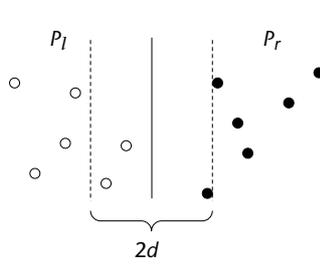
$p_1, p_2, \dots, p_{\frac{n}{2}}, p_{\frac{n}{2}+1}, \dots, p_n$



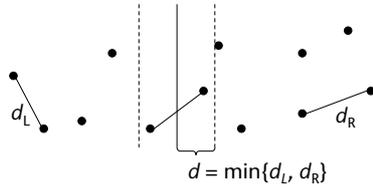
G. Zachmann Informatik 2 — SS 11 Divide & Conquer 17

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

$P_l = \{p_1, p_2, \dots, p_{\frac{n}{2}}\}$  und  
 $P_r = \{p_{\frac{n}{2}+1}, \dots, p_n\}$
- Phase 3: finde das dichteste Paar  $(\circ, \bullet)$  im "Band" um die Mitte mit der Breite  $2d$ , wobei bekannt ist, daß kein  $(\circ, \circ)$ - oder  $(\bullet, \bullet)$ -Paar dichter zusammen ist als  $d$



G. Zachmann Informatik 2 — SS 11 Divide & Conquer 19



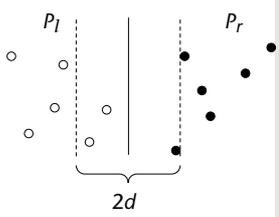
```

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 )
  
```

G. Zachmann Informatik 2 — SS 11 Divide & Conquer 20

### 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  $[p_y-d, p_y+d]$  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  $O(n) + O(n \log n)$

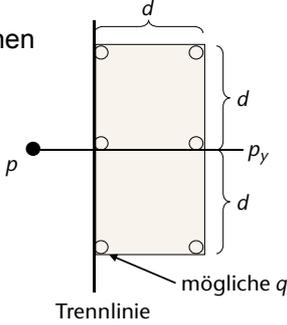


G. Zachmann Informatik 2 — SS 11 Divide & Conquer 21

## 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  $[p_y-d, p_y+d]$  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



Trennlinie

mögliche  $q$

G. Zachmann Informatik 2 — SS 11 Divide & Conquer 22

## Zeit-Komplexität

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

$$T(1) = 1$$

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

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

$$= 4T\left(\frac{n}{4}\right) + 2n(\log n - 1) + 2n \log n$$

$$\dots$$

$$= 2^k T\left(\frac{n}{2^k}\right) + 2n(\log n + (\log n - 1) + \dots + (\log n - k + 1))$$

$$= n + 2n(1 + 2 + \dots + \log n) = n + 2n \frac{(\log n + 1) \log n}{2}$$

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

G. Zachmann Informatik 2 — SS 11 Divide & Conquer 23

## 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*])

G. Zachmann Informatik 2 — SS 11 Divide & Conquer 24