



Algorithmentheorie

01 - Einleitung

Robert Elsässer



Vorlesung: Di 16.00 – 18.00 Uhr, HS 00-026, Geb.101
Mi 16.00 – 18.00 Uhr, HS 00-026, Geb.101
alle 14 Tage

Übungen: alle 14 Tage
Anmeldung bis 22. Oktober 2009, 15 Uhr
- Mo 09-11, SR 00-006, Geb. 051
- Mo 14-16, SR 00-031, Geb. 051
- Mo 14-16, SR 02-017, Geb. 052
- Do 11-13, SR 02-017, Geb. 052
- Do 16-18, SR 00-014, Geb. 078

<http://lak.informatik.uni-freiburg.de/>
→ Teaching → WS09/10 → Algorithm Theory

Klausur:

Teilnahmevoraussetzung:

- 1-mal in den Übungen vorgerechnet

Bonus bei bestandener Klausur:

Verbesserung um

- 1/3 Notenstufe, wenn 50% der Übungspunkte erzielt
- 2/3 Notenstufe, wenn 80% der Übungspunkte erzielt

Literatur

Th. Ottmann, P. Widmayer :
Algorithmen und Datenstrukturen
4. Auflage, Spektrum Akademischer Verlag,
Heidelberg, 2002

Th. Cormen, C. Leiserson, R. Rivest, C. Stein:
Introduction to Algorithms, Second Edition
MIT Press, 2001

Originalliteratur

Genauere Angaben zum Thema werden jeweils im
Web veröffentlicht

Entwurfsprinzipien und Analysetechniken für Algorithmen

- Divide and Conquer
- Greedy Prinzip
- Dynamische Programmierung
- Randomisierung
- Amortisierte Analyse

Algorithmen und Datenstrukturen

Problem-/Anwendungsbereiche

- Geometrische Algorithmen
- Algebraische Algorithmen
- Graphenalgorithmen
- Datenstrukturen
- Internet-Algorithmen
- Optimierungsverfahren
- Zeichenkettenverarbeitung



Divide and Conquer





Teile und Herrsche

Divide et impera (Caesar)



L'état, c'est moi ! (Louis XIV.)



Das Divide - and - Conquer Prinzip

- Quicksort
- Formulierung und Analyse des Prinzips
- Geometrisches Divide – and – Conquer
 - Closest-Pair
 - Segmentschnitt
 - Voronoi-Diagramm

Quicksort: Sortieren durch Teilen



```
function Quick (F:Folge): Folge;  
{liefert zu unsortierter Folge F die sortierte}  
begin  
  if  $|F| = 1$  then Quick:=F  
  else { wähle Pivotelement  $v$  aus F;  
        teile F in  $F_l$  mit Elementen  $< v$ ,  
        und in  $F_r$  mit Elementen  $> v$   
        Quick:=  $Quick(F_l) v Quick(F_r)$  }  
end;
```

Formulierung des D&C Prinzips

Divide-and-Conquer Verfahren zur Lösung eines Problems der Größe n

1. Divide:

$n > c$: Teile das Problem in k Teilprobleme der Größe n_1, \dots, n_k auf ($k \geq 2$)

$n \leq c$: Löse das Problem direkt

2. Conquer:

Löse die k Teilprobleme auf dieselbe Art (rekursiv)

3. Merge :

Füge die berechneten Teillösungen zu einer Gesamtlösung zusammen

$T(n)$ – Maximale Anzahl von Schritten, um ein Problem der Größe n zu lösen

$$T(n) = \begin{cases} a & n \leq c \\ T(n_1) + \dots + T(n_k) \\ \quad + \text{Divide- und Mergeaufwand} & n > c \end{cases}$$

Best Case: $k = 2, n_1 = n_2 = n/2$

Divide- und Mergeaufwand : $DM(n)$

$$T(1) = a$$

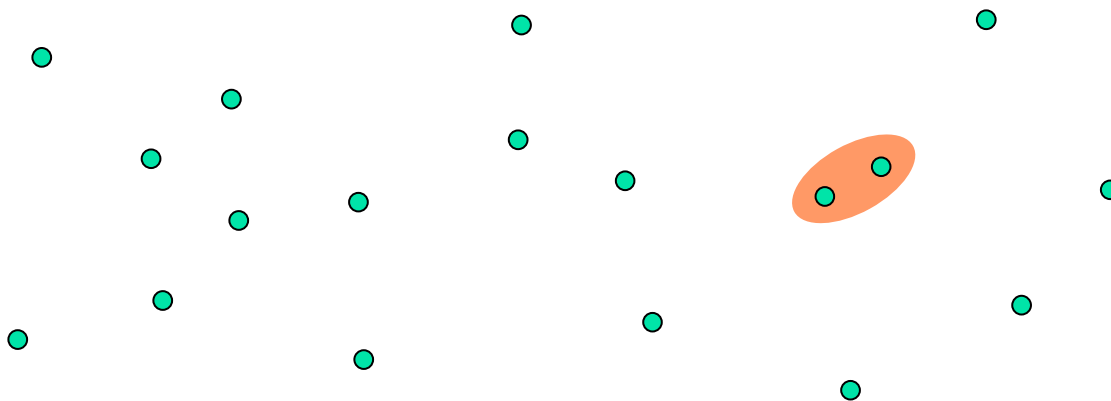
$$T(n) = 2T(n/2) + DM(n)$$

Nächste Paare

Geometrisches Problem:

- Problem: Nächstes Paar
- Eingabe: n Punkte in der Ebene
- Ausgabe: Das Paar q, r mit geringstem Abstand

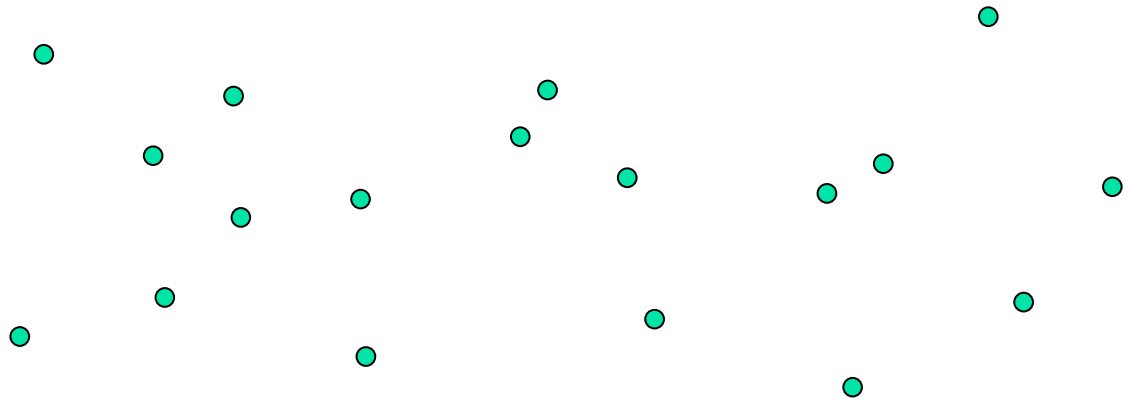
Beispiel:



Nächste Paare

Notation:

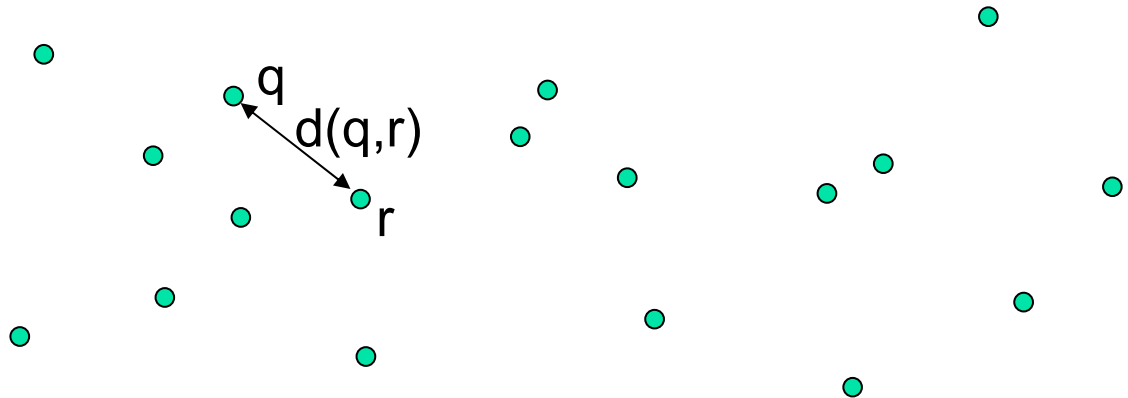
- $d(q,r)$ bezeichnet Abstand zwischen q und r



Nächste Paare

Notation:

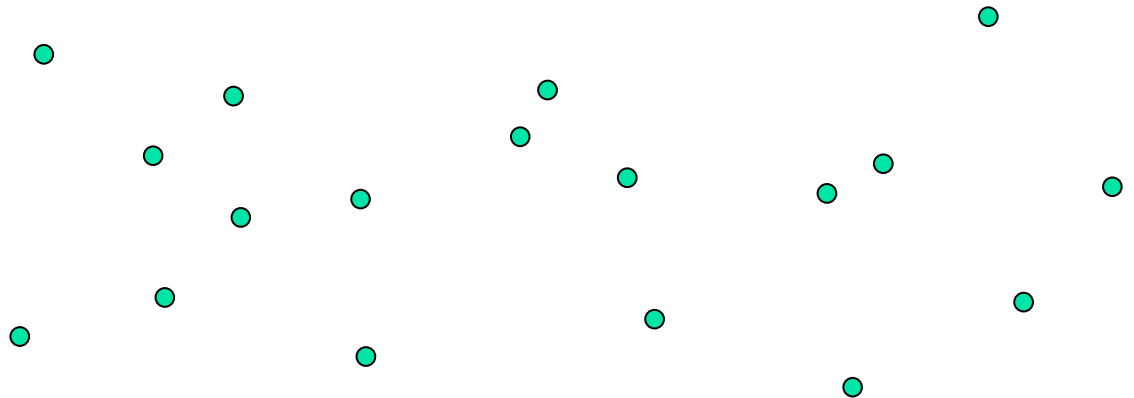
- $d(q,r)$ bezeichnet Abstand zwischen q und r



Nächste Paare

Plan für 2D:

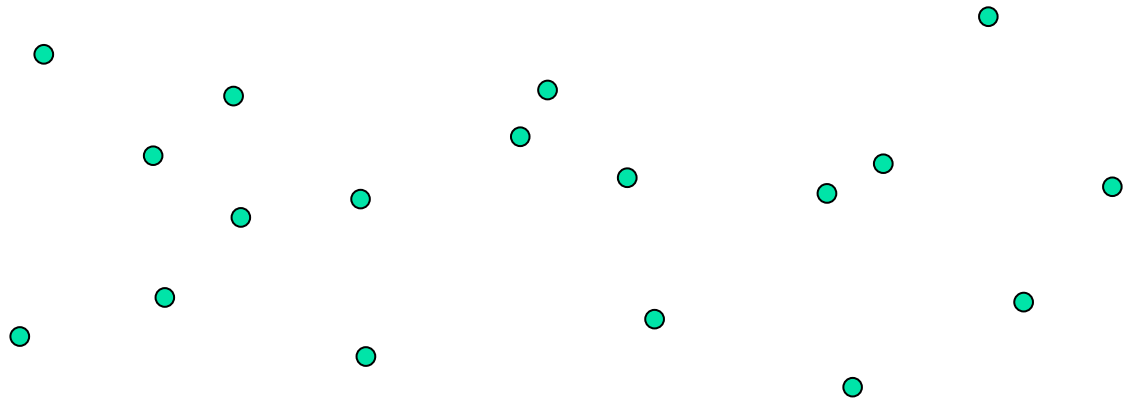
- Wie MergeSort nur im 2D



Nächste Paare

Plan:

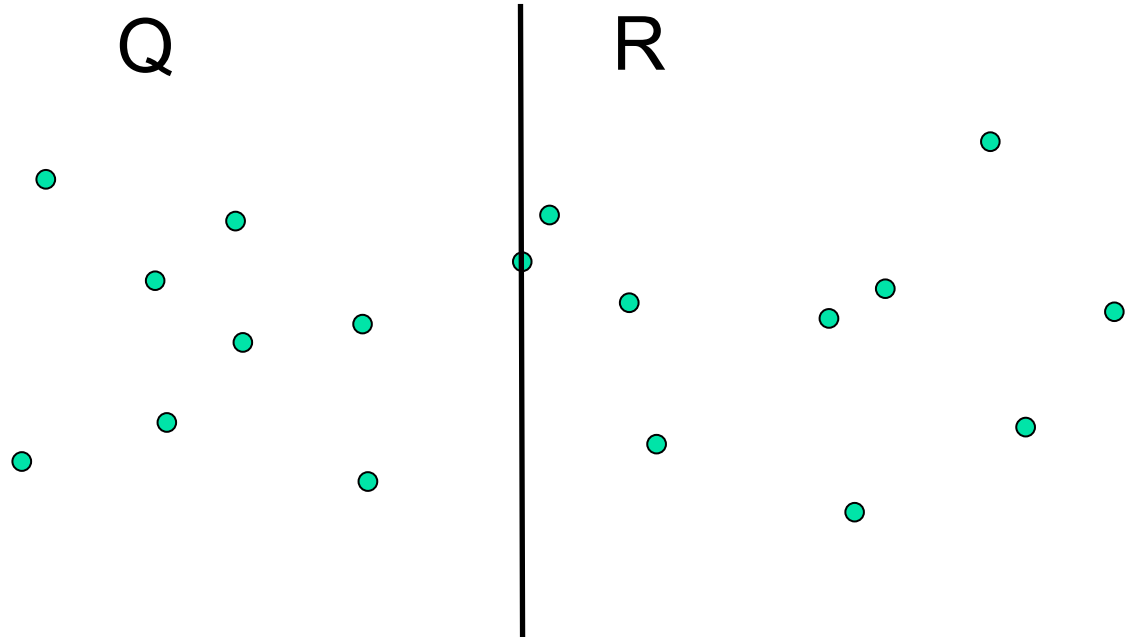
- Wie MergeSort nur im 2D
- Sortiere Punktmenge nach x-Koordinate



Nächste Paare

Plan:

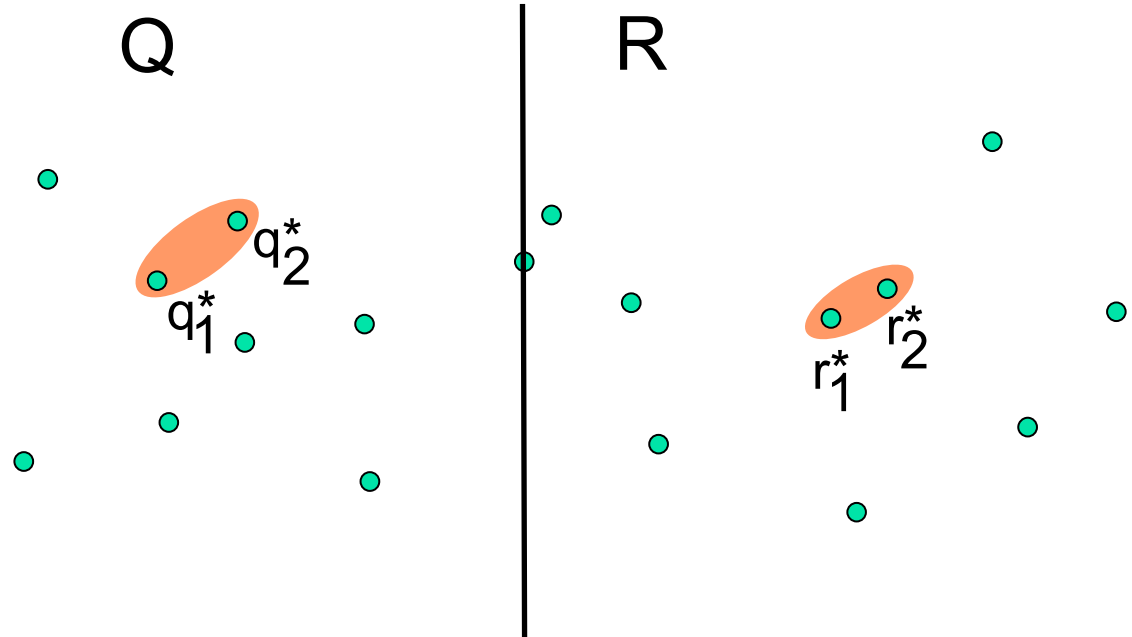
- Wie MergeSort nur im 2D
- Sortiere Punktmenge nach x-Koordinate
- Teile in der Mitte



Nächste Paare

Plan:

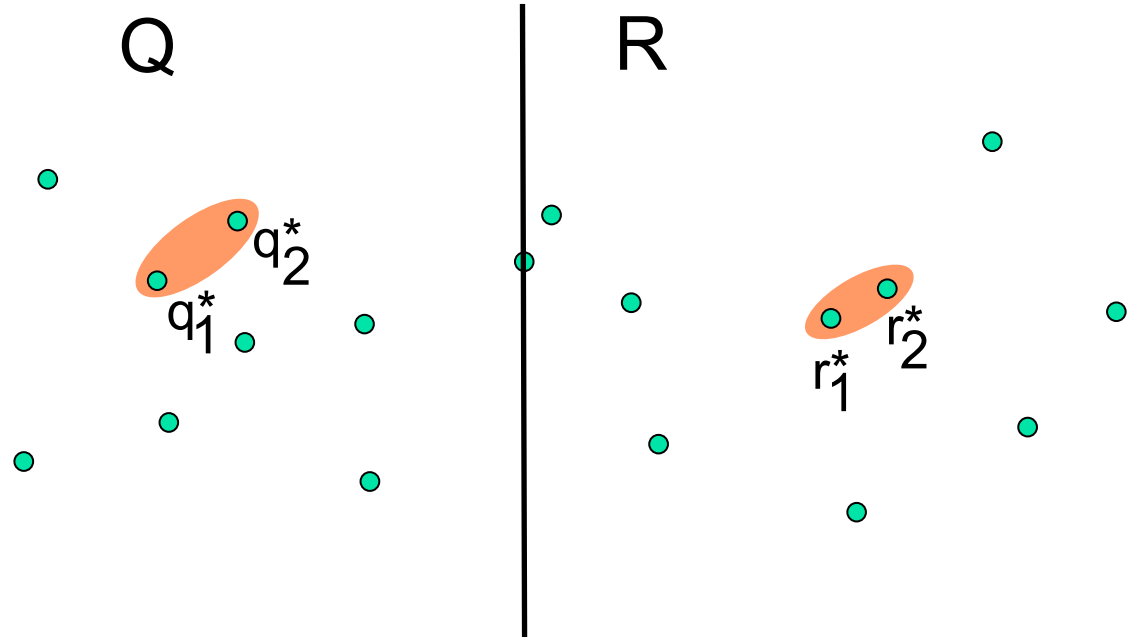
- Wie MergeSort nur im 2D
- Sortiere Punktmenge nach x-Koordinate
- Teile in der Mitte
- Löse rekursiv



Nächste Paare

Plan:

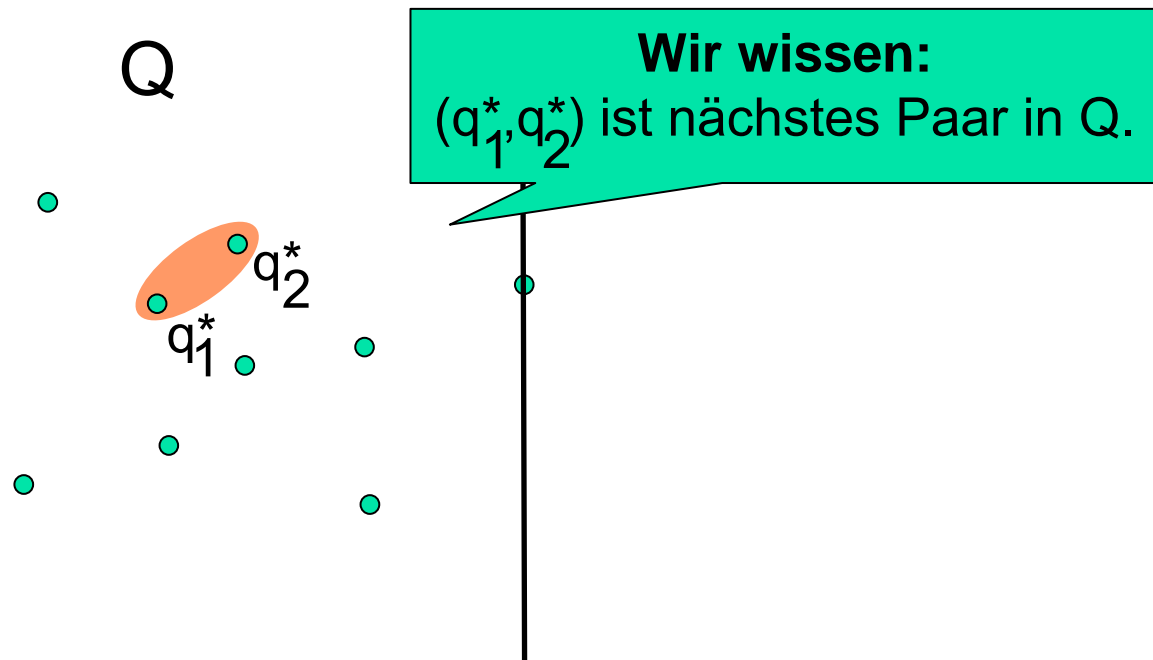
- Wie MergeSort nur im 2D
- Sortiere Punktmenge nach x-Koordinate
- Teile in der Mitte
- Löse rekursiv
- Füge zusammen



Nächste Paare

Plan:

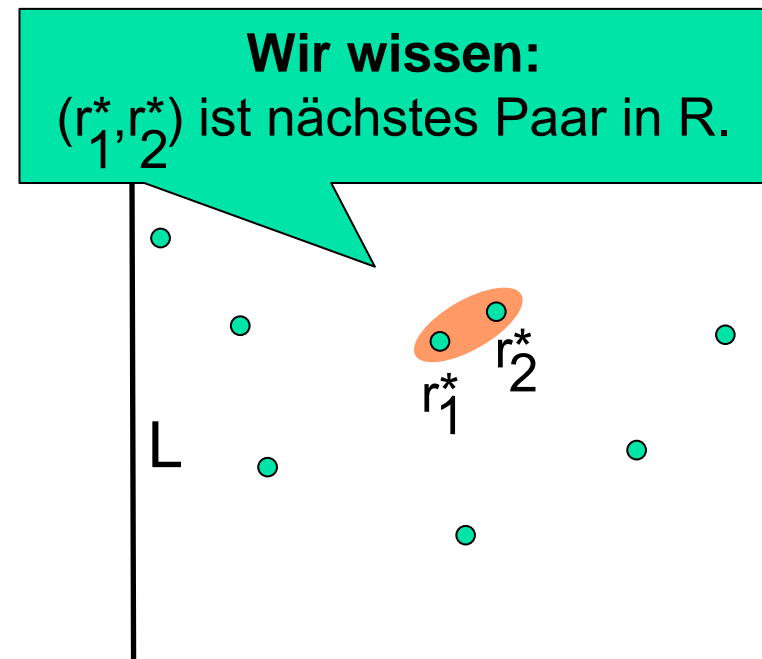
- Wie MergeSort nur im 2D
- Sortiere Punktmenge nach x-Koordinate
- Teile in der Mitte
- Löse rekursiv
- Füge zusammen



Nächste Paare

Plan:

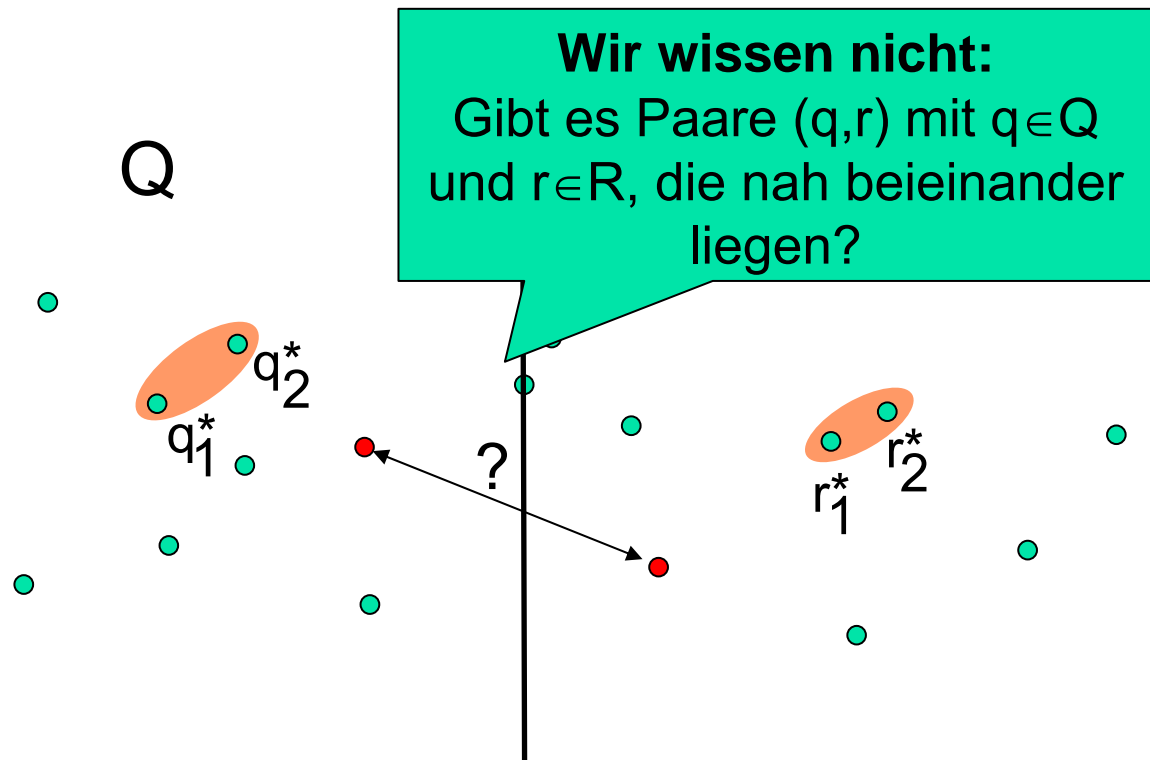
- Wie MergeSort nur im 2D
- Sortiere Punktmenge nach x-Koordinate
- Teile in der Mitte
- Löse rekursiv
- Füge zusammen



Nächste Paare

Plan:

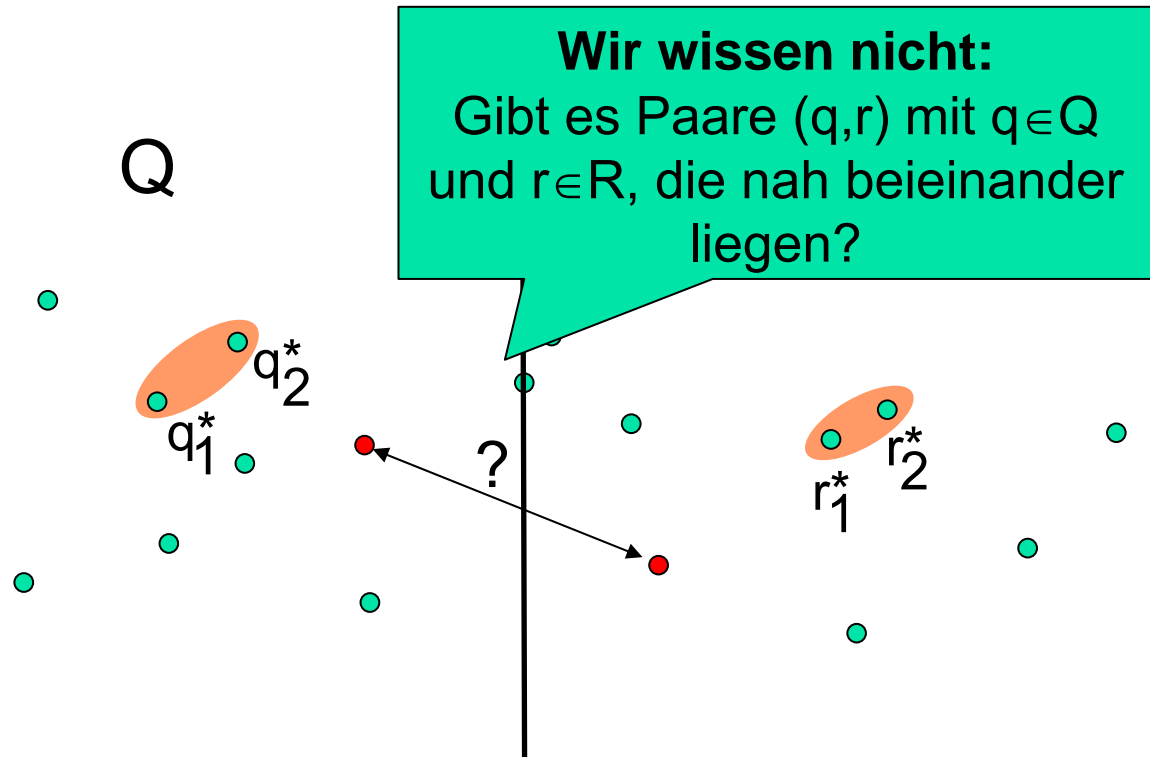
- Wie MergeSort nur im 2D
- Sortiere Punktmenge nach x-Koordinate
- Teile in der Mitte
- Löse rekursiv
- Füge zusammen



Nächste Paare

Plan:

- Wie MergeSort nur im 2D
- Sortiere Punktmenge nach x-Koordinate
- Teile in der Mitte
- Löse rekursiv
- Füge zusammen



Wir wissen nicht:
Gibt es Paare (q, r) mit $q \in Q$ und $r \in R$, die nah beieinander liegen?

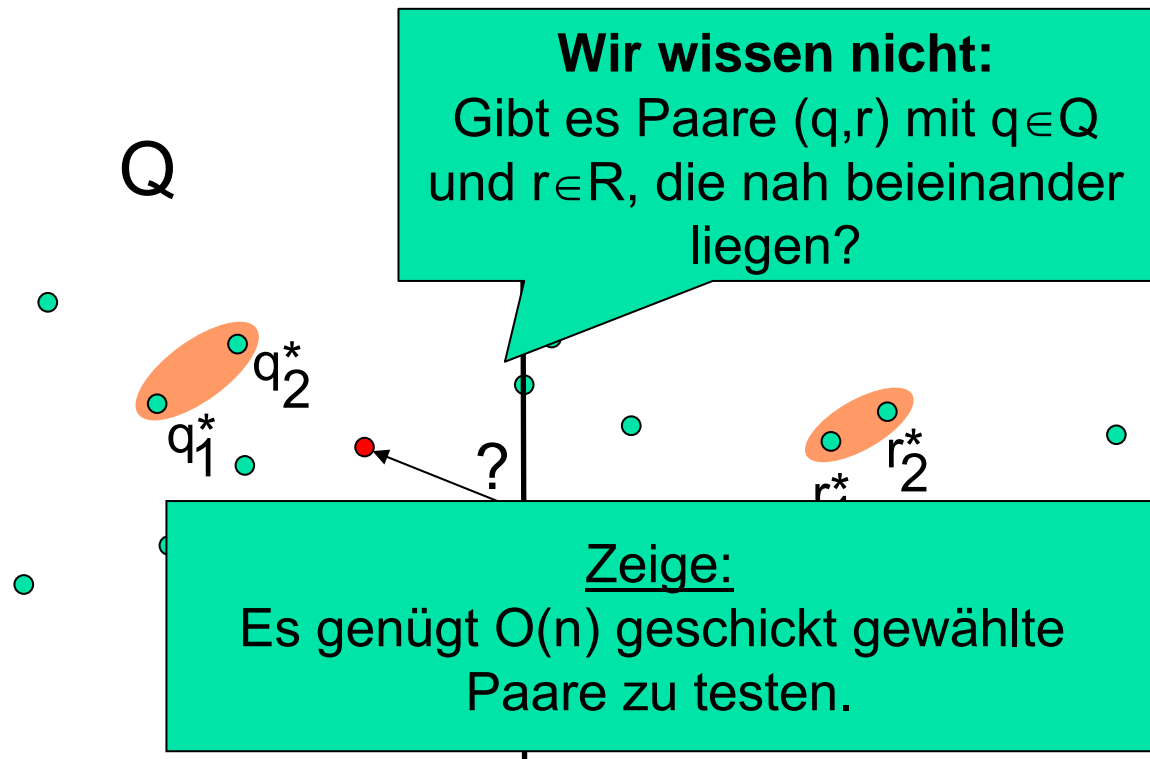
Problem:
Es gibt sehr viele solche Paare. ($\Theta(n^2)$ viele).

Nächste Paare

Plan:

- Wie MergeSort nur im 2D
- Sortiere Punktmenge nach x-Koordinate
- Teile in der Mitte
- Löse rekursiv
- Füge zusammen

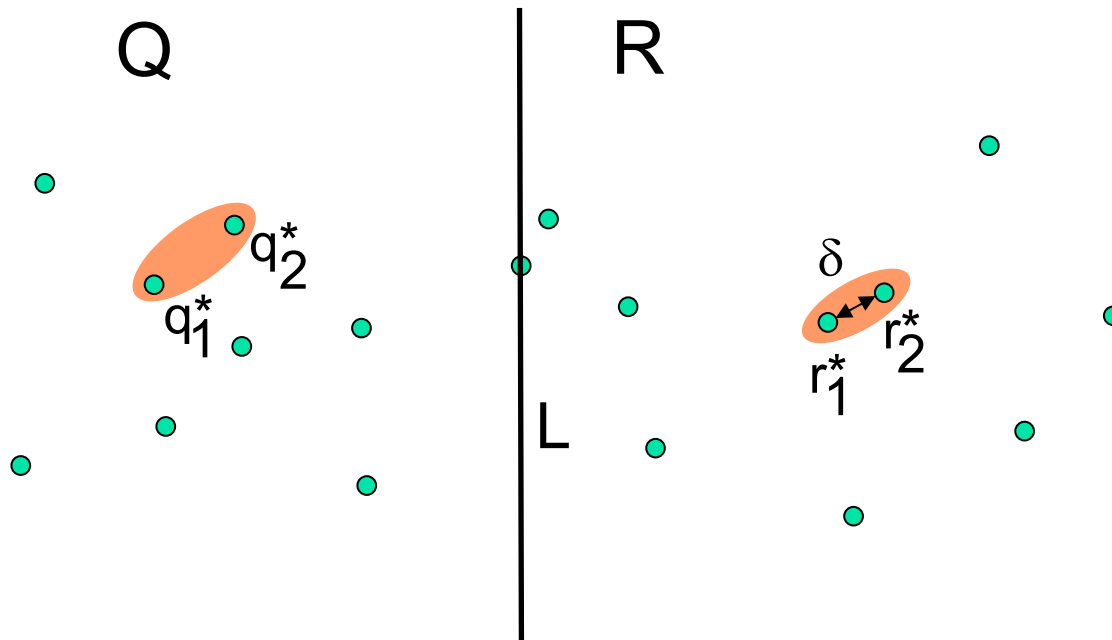
Problem:
Es gibt sehr viele solche Paare. ($\Theta(n^2)$ viele).



Nächste Paare

Definition:

$$\delta = \min\{d(q_1^*, q_2^*), d(r_1^*, r_2^*)\}$$

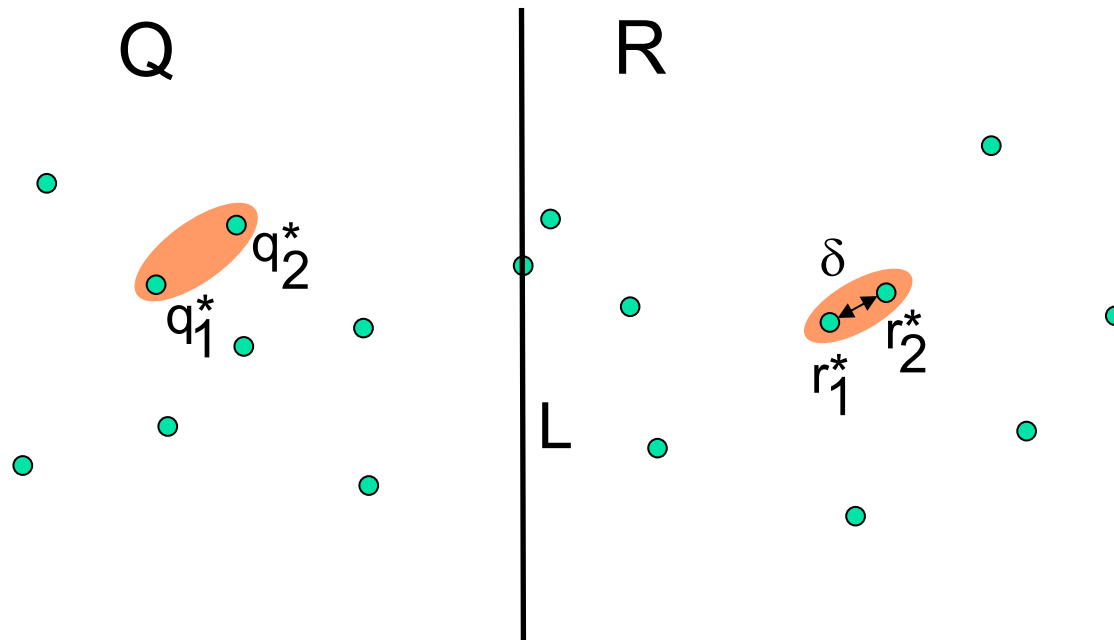


Nächste Paare

Wichtige Beobachtung:

Sind $q, r \in Q$, dann gilt $d(p, r) \geq \delta$.

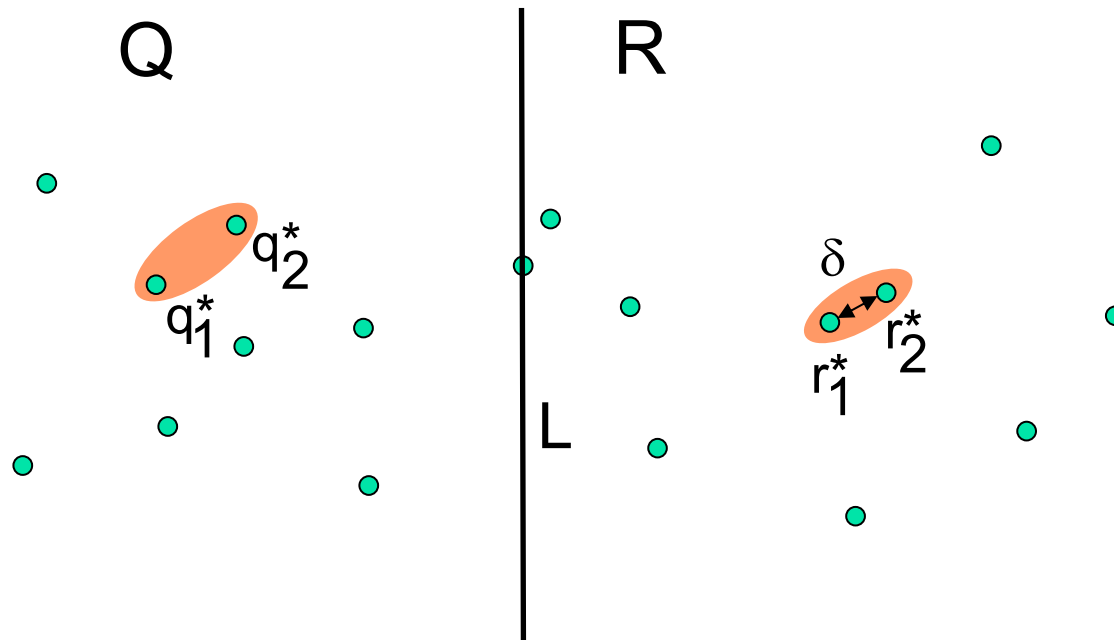
Sind $q, r \in R$, dann gilt $d(p, r) \geq \delta$.



Nächste Paare

Lemma 1.1

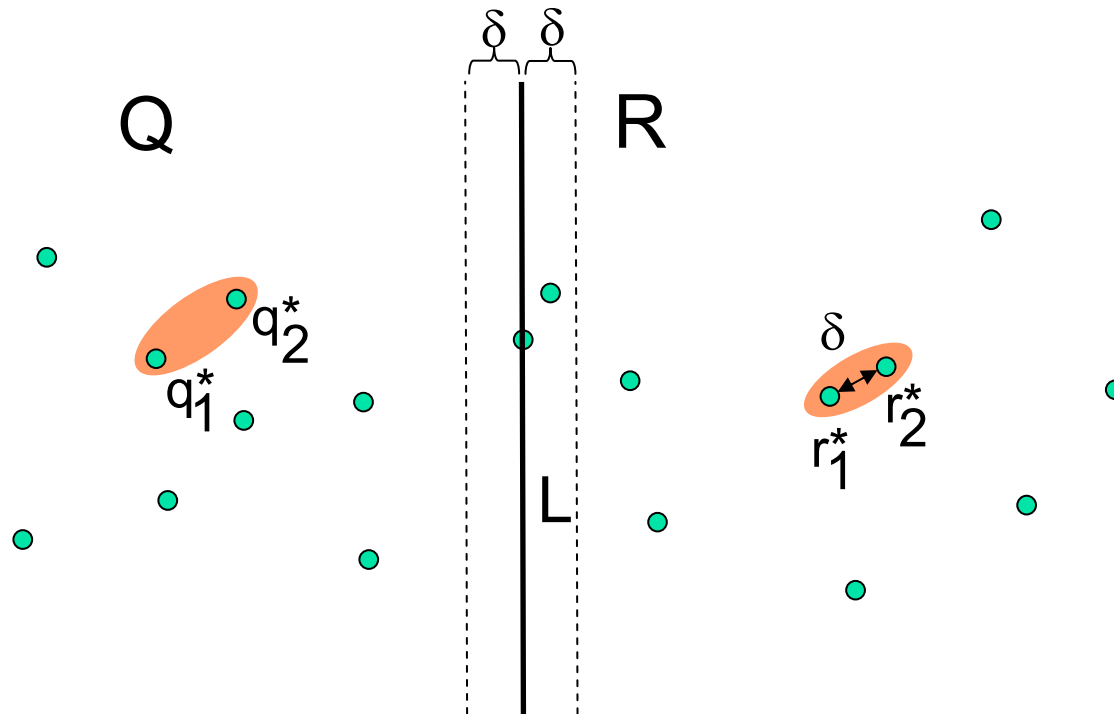
Gibt es $q \in Q$ und $r \in R$ mit $d(q,r) < \delta$, dann sind sowohl q als auch r höchstens δ von L entfernt.



Nächste Paare

Lemma 1.1

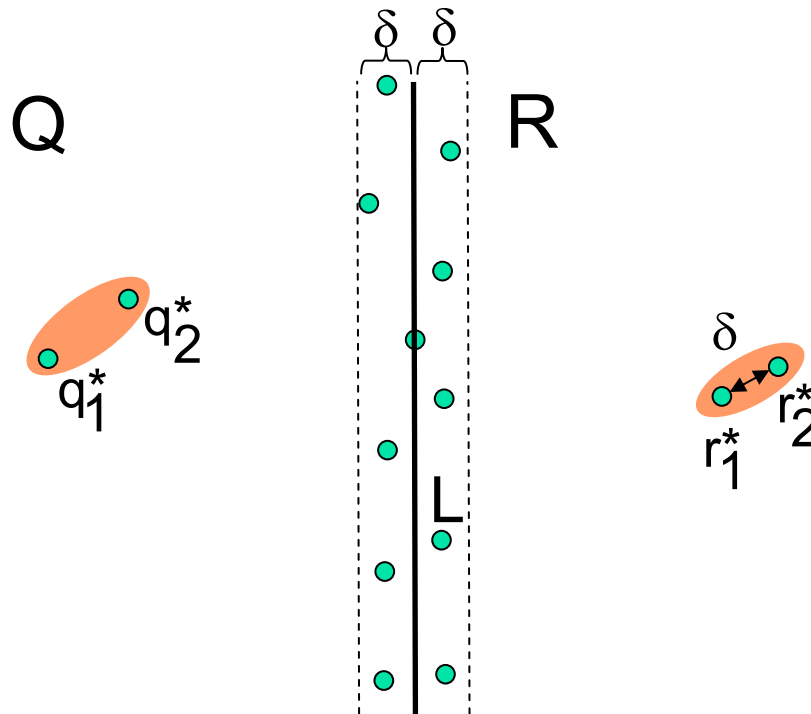
Gibt es $q \in Q$ und $r \in R$ mit $d(q,r) < \delta$, dann sind sowohl q als auch r höchstens δ von L entfernt.



Nächste Paare

Was gewinnen wir?

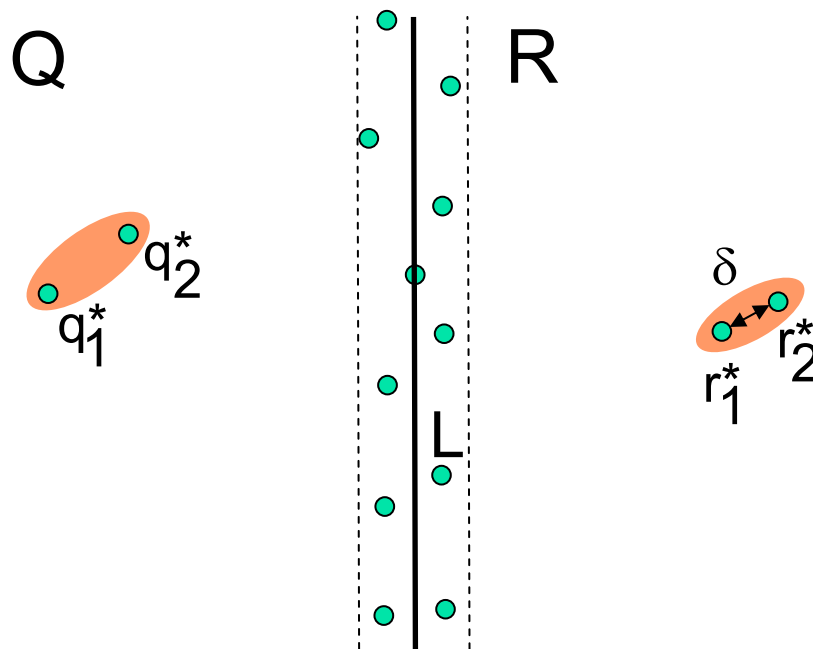
- Alle Punkte können nahe an L liegen



Nächste Paare

Abhilfe:

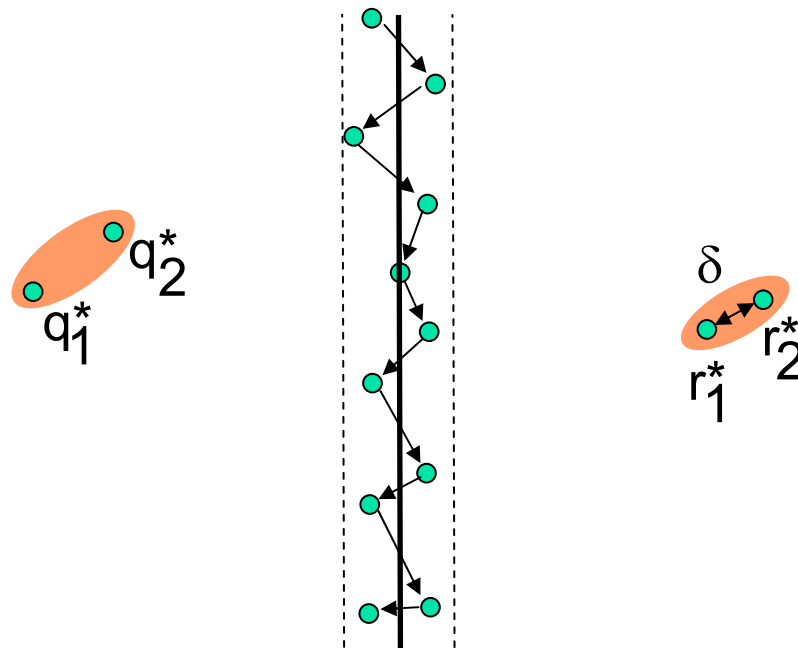
- Wir wollen nur Paare (q,r) testen, wenn
 - q und r nah an L liegen **und**
 - q und r kleinen Abstand in y -Richtung haben



Nächste Paare

Idee:

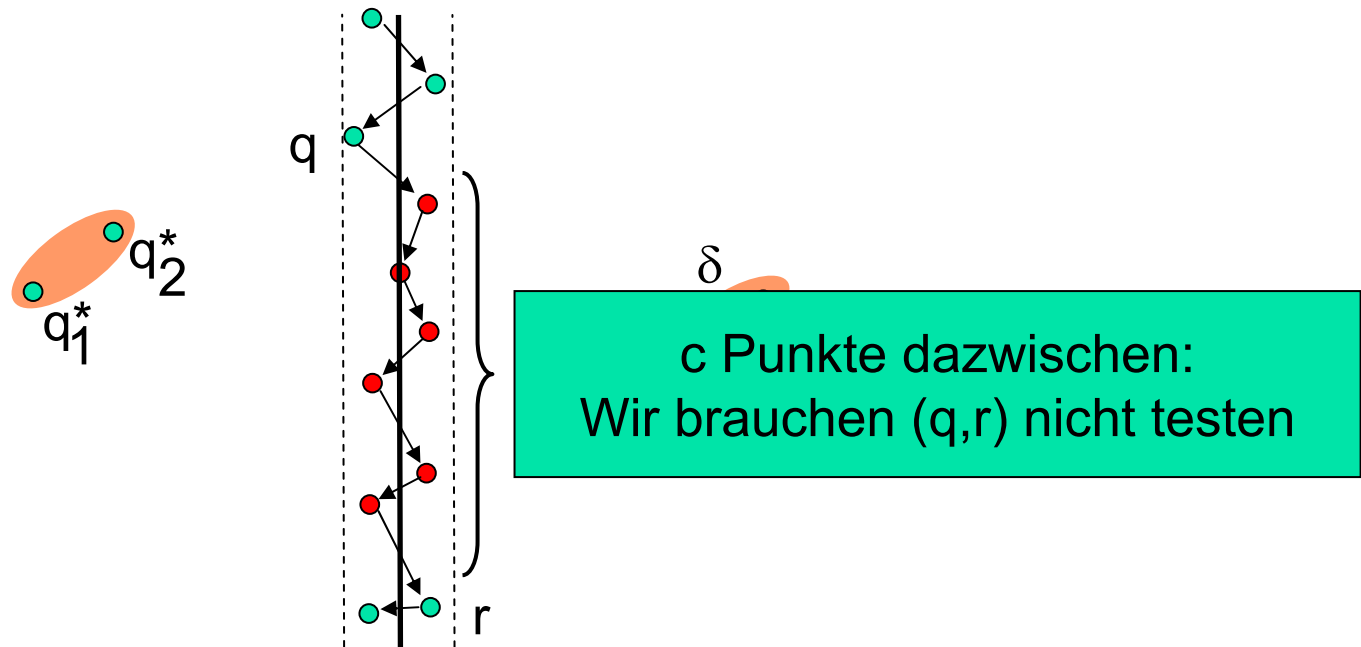
- Sortiere nahe Punkte nach y-Koordinate
- Zeige: Liegen in dieser Sortierung mehr als c Punkte zwischen q und r , dann kann (q,r) nicht nächstes Paar sein



Nächste Paare

Idee:

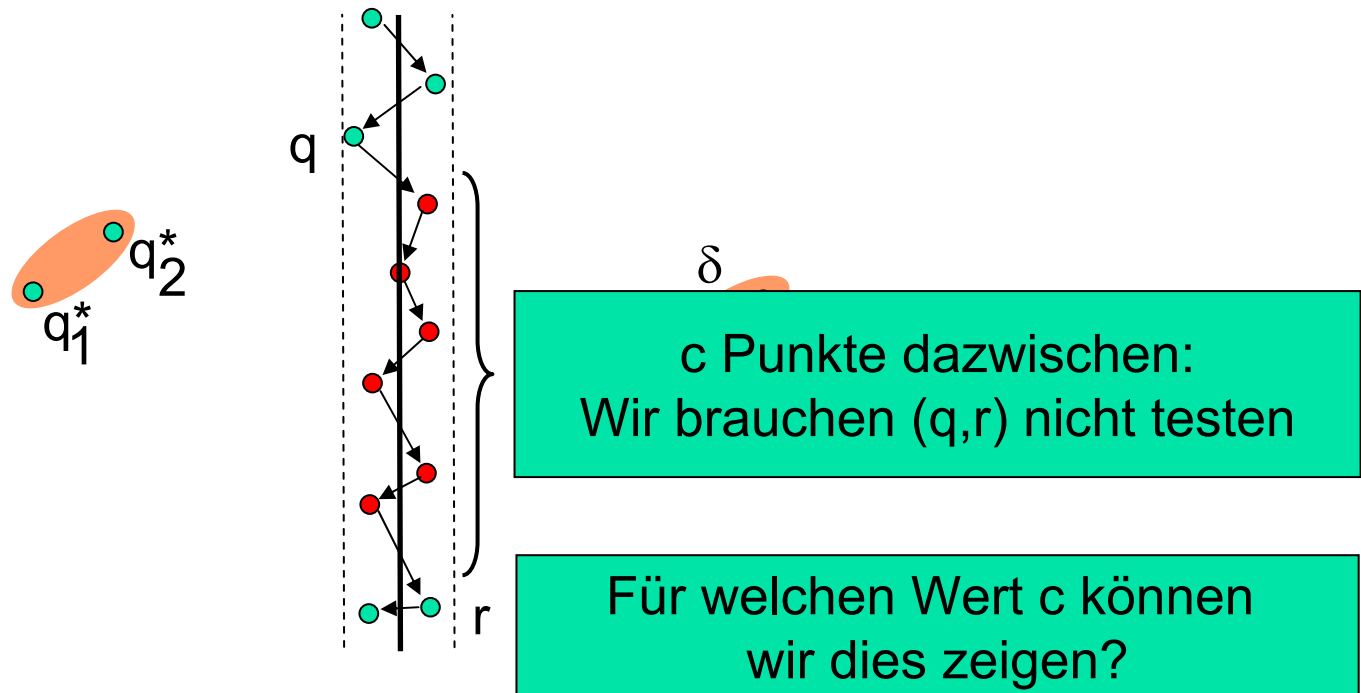
- Sortiere nahe Punkte nach y-Koordinate
- Zeige: Liegen in dieser Sortierung mehr als c Punkte zwischen q und r , dann kann (q,r) nicht nächstes Paar sein



Nächste Paare

Idee:

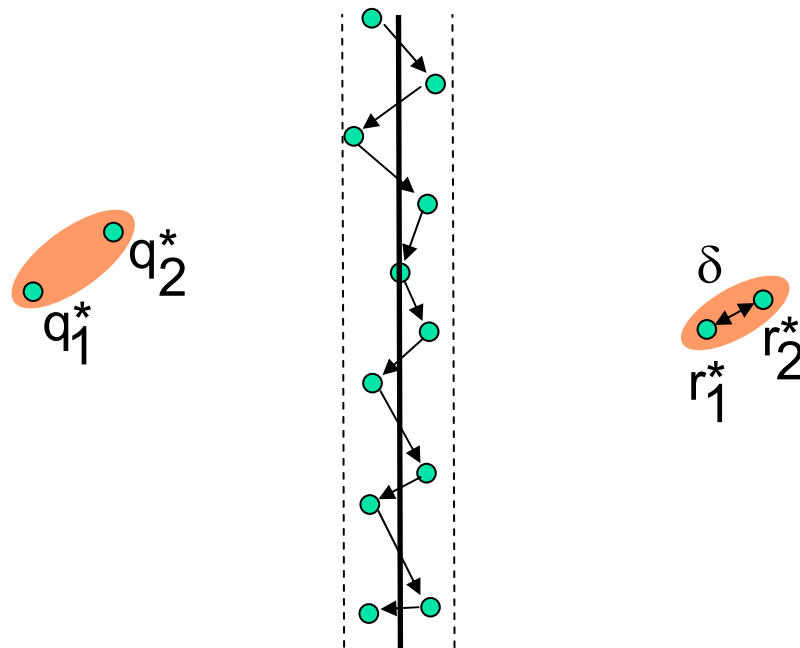
- Sortiere nahe Punkte nach y-Koordinate
- Zeige: Liegen in dieser Sortierung mehr als c Punkte zwischen q und r , dann kann (q,r) nicht nächstes Paar sein



Nächste Paare

Lemma 1.2

Liegen in der Sortierung mindestens 15 Punkte zwischen q und r , dann ist $d(q,r) > \delta$.



Nächste Paare

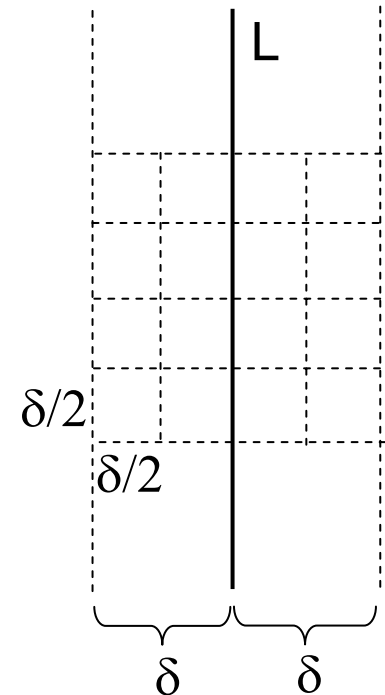
Lemma 1.2

Liegen in der Sortierung mindestens 15 Punkte zwischen q und r , dann ist $d(q,r) > \delta$.

Beweis:

- Teile Nahbereich in Quadrate der Seitenlänge $\delta/2$ auf

Beh.: In jedem Quadrat ist nur ein Punkt.



Nächste Paare

Lemma 1.2

Liegen in der Sortierung mindestens 15 Punkte zwischen q und r , dann ist $d(q,r) > \delta$.

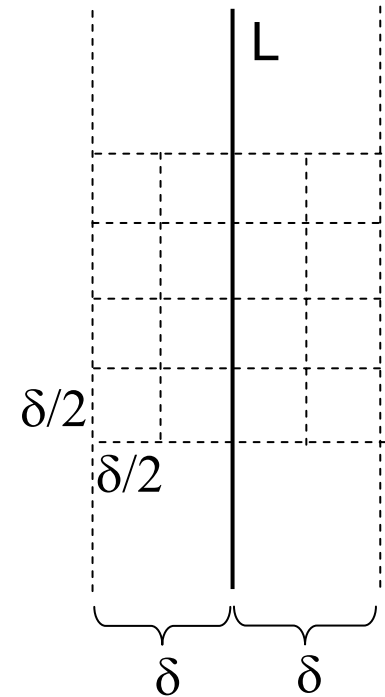
Beweis:

- Teile Nahbereich in Quadrate der Seitenlänge $\delta/2$ auf

Beh.: In jedem Quadrat ist nur ein Punkt.

Beweis:

- Annahme: Zwei Punkte q,r in einem Quadrat



Nächste Paare

Lemma 1.2

Liegen in der Sortierung mindestens 15 Punkte zwischen q und r , dann ist $d(q,r) > \delta$.

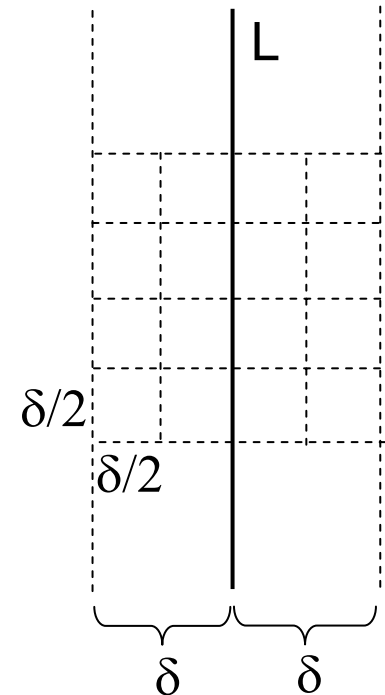
Beweis:

- Teile Nahbereich in Quadrate der Seitenlänge $\delta/2$ auf

Beh.: In jedem Quadrat ist nur ein Punkt.

Beweis:

- Annahme: Zwei Punkte q,r in einem Quadrat
- Dann ist $d(q,r) \leq \delta \cdot \sqrt{2}/2 < \delta$



Nächste Paare

Lemma 1.2

Liegen in der Sortierung mindestens 15 Punkte zwischen q und r , dann ist $d(q,r) > \delta$.

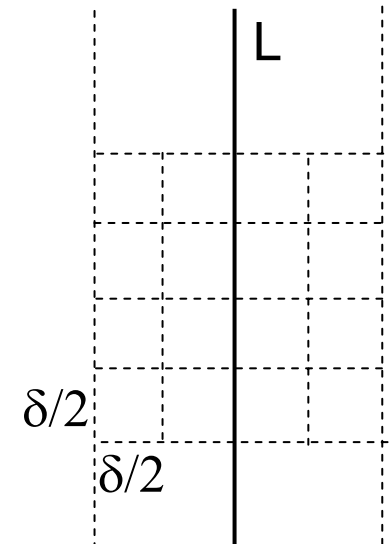
Beweis:

- Teile Nahbereich in Quadrate der Seitenlänge $\delta/2$ auf

Beh.: In jedem Quadrat ist nur ein Punkt.

Beweis:

- Annahme: Zwei Punkte q,r in einem Quadrat
- Dann ist $d(q,r) \leq \delta \cdot \sqrt{2}/2 < \delta$ ⚡



Ein Quadrat ist entweder komplett links oder rechts von L .
Also sind q,r beide in Q oder beide in R . Damit gilt $d(q,r) \geq \delta$.

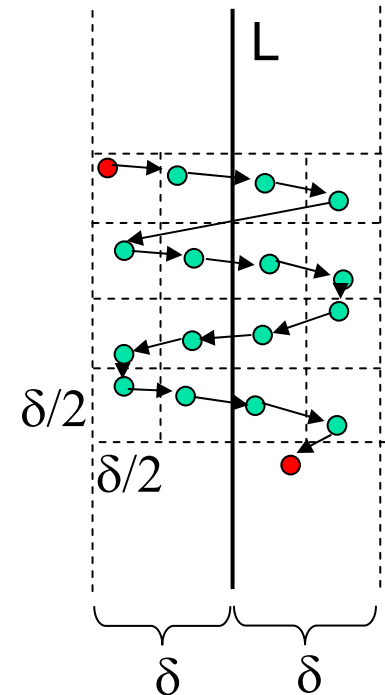
Nächste Paare

Lemma 1.2

Liegen in der Sortierung mindestens 15 Punkte zwischen q und r , dann ist $d(q,r) > \delta$.

Beweis:

- Seien mind. 15 Punkte zwischen q und r



Nächste Paare

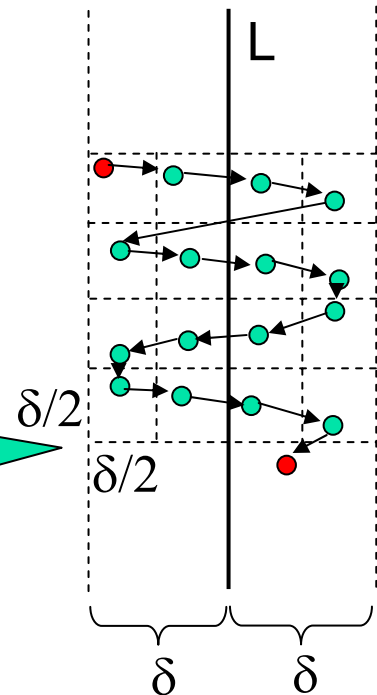
Lemma 1.2

Liegen in der Sortierung mindestens 15 Punkte zwischen q und r , dann ist $d(q,r) > \delta$.

Beweis:

- Seien mind. 15 Punkte zwischen q und r

Zwischen den beiden roten Punkten liegen 15 andere Punkte



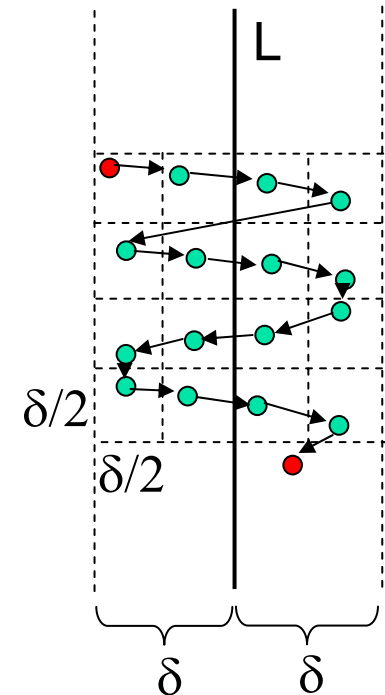
Nächste Paare

Lemma 1.2

Liegen in der Sortierung mindestens 15 Punkte zwischen q und r , dann ist $d(q,r) > \delta$.

Beweis:

- Seien mind. 15 Punkte zwischen q und r
- Dann sind mindestens 3 Reihen Quadrate zwischen q,r , weil in jedem Quadrat höchstens ein Punkt ist



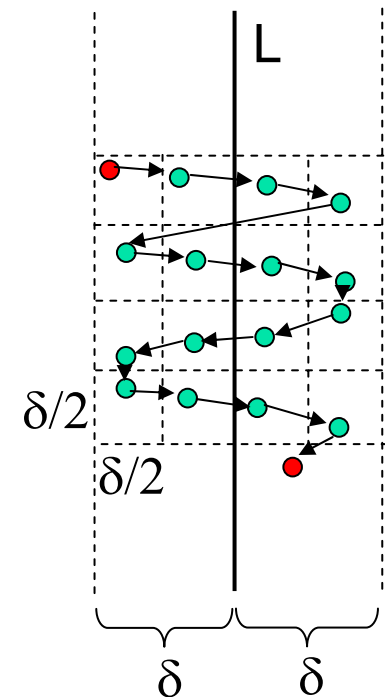
Nächste Paare

Lemma 1.2

Liegen in der Sortierung mindestens 15 Punkte zwischen q und r , dann ist $d(q,r) > \delta$.

Beweis:

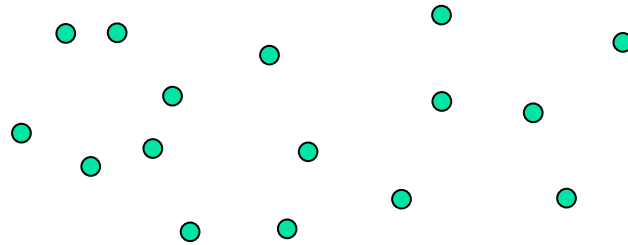
- Seien mind. 15 Punkte zwischen q und r
- Dann sind mindestens 3 Reihen Quadrate zwischen q,r , weil in jedem Quadrat höchstens ein Punkt ist
- Dann muss $d(q,r)$ mindestens $3/2 \cdot \delta > \delta$ sein



Nächste Paare

Der Algorithmus im Überblick:

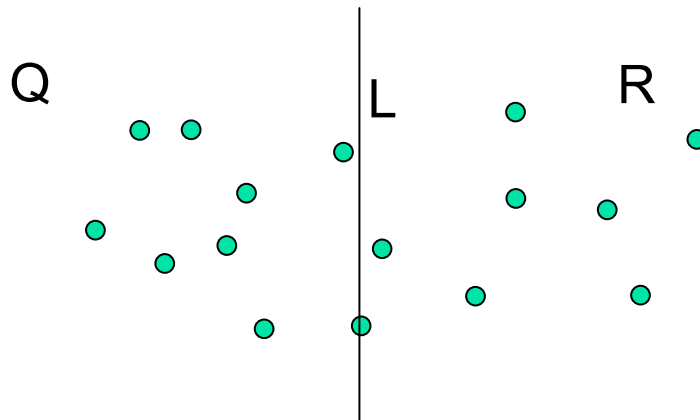
- Teile Punktmenge an Linie L in Q und R auf



Nächste Paare

Der Algorithmus im Überblick:

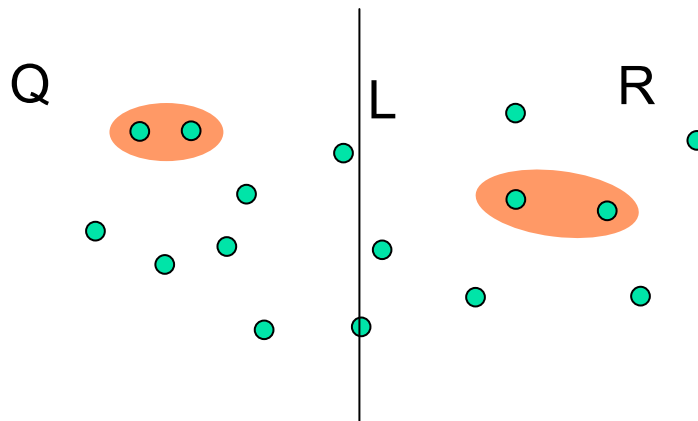
- Teile Punktmenge an Linie L in Q und R auf



Nächste Paare

Der Algorithmus im Überblick:

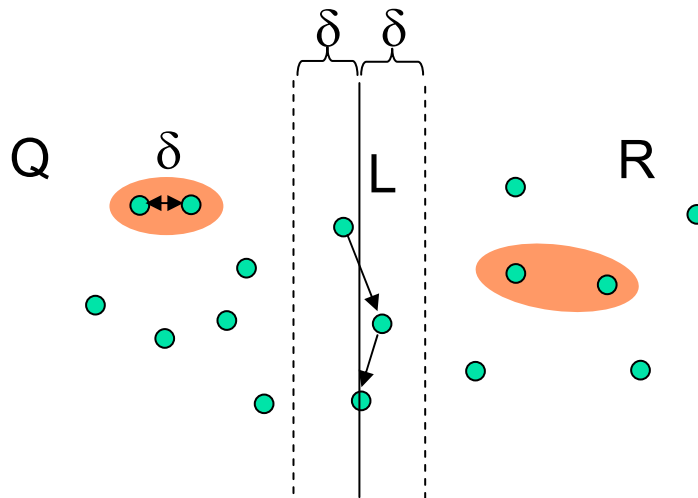
- Teile Punktmenge an Linie L in Q und R auf
- Löse Problem rekursiv auf Q und R



Nächste Paare

Der Algorithmus im Überblick:

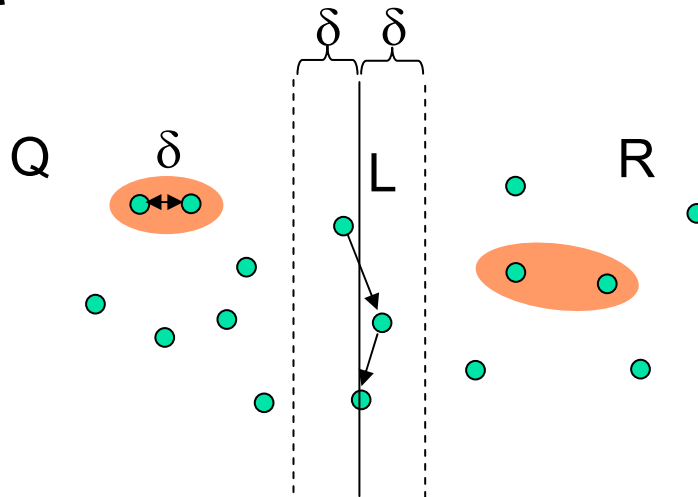
- Teile Punktmenge an Linie L in Q und R auf
- Löse Problem rekursiv auf Q und R
- Sortiere Punkte nach y-Koord.



Nächste Paare

Der Algorithmus im Überblick:

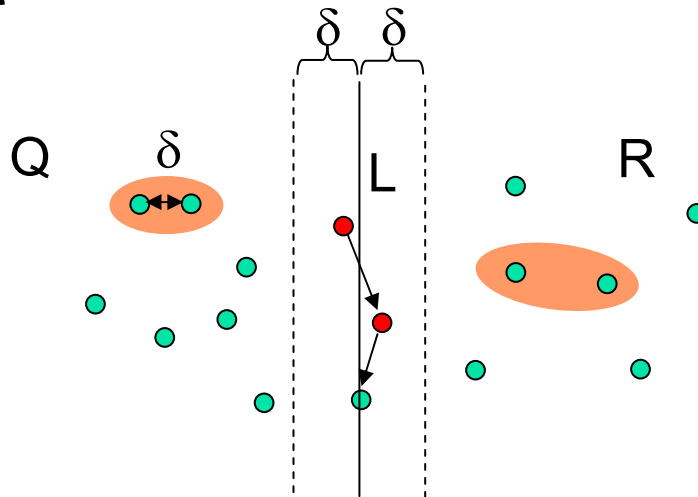
- Teile Punktmenge an Linie L in Q und R auf
- Löse Problem rekursiv auf Q und R
- Sortiere Punkte nach y-Koord.
- Teste alle Paare, deren Abstand in der Sortierung kleiner als 16 ist



Nächste Paare

Der Algorithmus im Überblick:

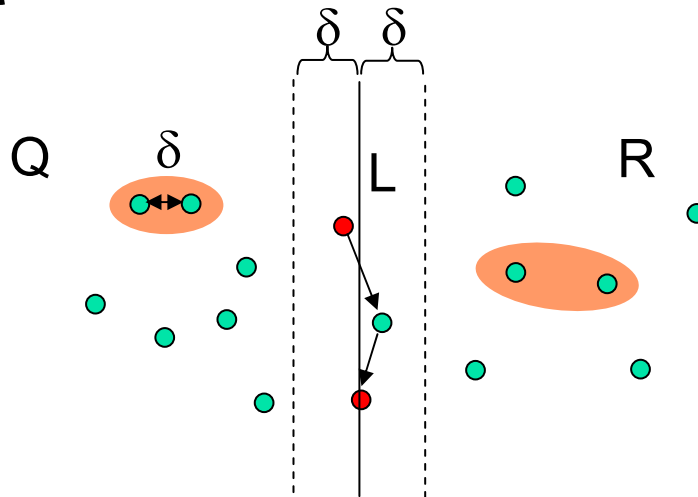
- Teile Punktmenge an Linie L in Q und R auf
- Löse Problem rekursiv auf Q und R
- Sortiere Punkte nach y-Koord.
- Teste alle Paare, deren Abstand in der Sortierung kleiner als 16 ist



Nächste Paare

Der Algorithmus im Überblick:

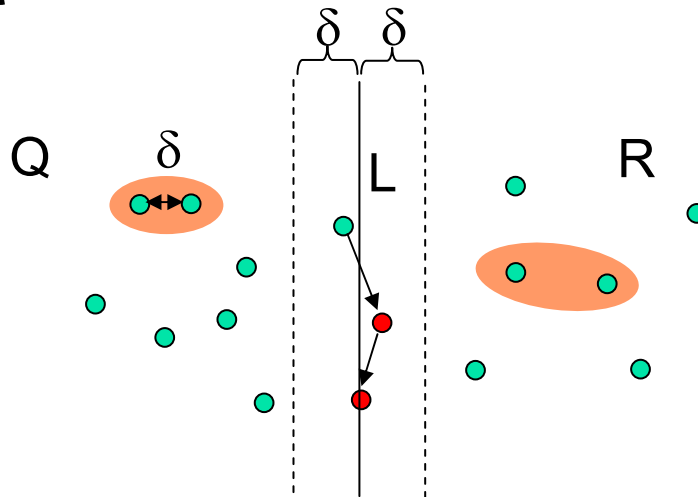
- Teile Punktmenge an Linie L in Q und R auf
- Löse Problem rekursiv auf Q und R
- Sortiere Punkte nach y-Koord.
- Teste alle Paare, deren Abstand in der Sortierung kleiner als 16 ist



Nächste Paare

Der Algorithmus im Überblick:

- Teile Punktmenge an Linie L in Q und R auf
- Löse Problem rekursiv auf Q und R
- Sortiere Punkte nach y-Koord.
- Teste alle Paare, deren Abstand in der Sortierung kleiner als 16 ist



Nächste Paare

NächstesPaar(P)

1. MergeSort(P, a, length[P]) ➤ Sortiere nach x-Koordinate
2. **return** NächstesPaarRec(P, 1, length[P])

Nächste Paare

NächstesPaar(P)

1. MergeSort(P, a, length[P]) $\Theta(n \log n)$
2. **return** NächstesPaarRec(P, 1, length[P])

Nächste Paare

NächstesPaar(P)

1. MergeSort(P, a, length[P])

2. **return** NächstesPaarRec(P, 1, length[P])

n=length[P]

$\Theta(n \log n)$

$T(n) + \Theta(1)$

Nächste Paare

NächstesPaar(P)

1. MergeSort(P, a, length[P])
2. **return** NächstesPaarRec(P, 1, length[P])

$\Theta(n \log n)$

$T(n) + \Theta(1)$

$T(n) + \Theta(n \log n)$

Nächste Paare

NächstesPaarRec(P, a, c)

1. **if** $c-a=1$ **then return** $(P[a], P[a+1])$
2. **if** $c-a=2$ **then return** nächstes Paar aus
 $(P[a], P[a+1]), (P[a], P[a+2]), (P[a+1], P[a+2])$
4. $b \leftarrow \lfloor (a+c)/2 \rfloor$
5. $(q_1^*, q_2^*) \leftarrow$ NächstesPaar(P, a, b)
6. $(r_1^*, r_2^*) \leftarrow$ NächstesPaar(P, b+1, c)
7. ➤ **Zusammensetzen**

Nächste Paare

Rekursionsabbruch
bei 2 oder 3 Punkten. Sortiere
die Punkte anschließend nach
den y-Koordinaten

NächstesPaarRec(P, a, c)

1. **if** $c-a=1$ **then return** $(P[a], P[a+1])$
2. **if** $c-a=2$ **then return** nächstes Paar aus
 $(P[a], P[a+1]), (P[a], P[a+2]), (P[a+1], P[a+2])$
4. $b \leftarrow \lfloor (a+c)/2 \rfloor$
5. $(q_1^*, q_2^*) \leftarrow$ NächstesPaar(P, a, b)
6. $(r_1^*, r_2^*) \leftarrow$ NächstesPaar(P, b+1, c)
7. ➤ **Zusammensetzen**

Nächste Paare

NächstesPaarRec(P, a, c)

1. **if** $c-a=1$ **then return** $(P[a], P[a+1])$
2. **if** $c-a=2$ **then return** $(P[a], P[a+1]), (P[a+1], P[a+2])$
Berechnung des mittleren Elements (Median)
4. $b \leftarrow \lfloor (a+c)/2 \rfloor$
5. $(q_1^*, q_2^*) \leftarrow$ NächstesPaar(P, a, b)
6. $(r_1^*, r_2^*) \leftarrow$ NächstesPaar(P, b+1, c)
7. ➤ **Zusammensetzen**

Nächste Paare

NächstesPaarRec(P,a,c)

1. **if** $c-a=1$ **then return** $(P[a],P[a+1])$

2. **if** $c-a=2$ **then return** nächstes Paar aus

$(P[a],P[a+1]), (P[a],P[a+2]), (P[a+1],P[a+2])$

Rekursiver Aufruf für
linke Hälfte

4. $b \leftarrow \lfloor (a+c)/2 \rfloor$


5. $(q_1^*, q_2^*) \leftarrow$ NächstesPaar(P, a, b)

6. $(r_1^*, r_2^*) \leftarrow$ NächstesPaar(P, b+1, c)

7. ➤ **Zusammensetzen**

Nächste Paare

NächstesPaarRec(P,a,c)

1. **if** $c-a=1$ **then return** $(P[a],P[a+1])$
2. **if** $c-a=2$ **then return** nächstes Paar aus
 $(P[a],P[a+1]), (P[a],P[a+2]), (P[a+1], P[a+2])$
4. $b \leftarrow \lfloor (a+c)/2 \rfloor$
5. $(q_1^*, q_2^*) \leftarrow$ NächstesPaar(P, a, 
6. $(r_1^*, r_2^*) \leftarrow$ NächstesPaar(P, b+1, c)
7. ➤ **Zusammensetzen**

Nächste Paare

NächstesPaarRec(P, a, c)

1. **if** $c-a=1$ **then return** $(P[a], P[a+1])$
2. **if** $c-a=2$ **then return** nächstes Paar aus
 $(P[a], P[a+1]), (P[a], P[a+2]), (P[a+1], P[a+2])$
4. $b \leftarrow \lfloor (a+c)/2 \rfloor$
5. $(q_1^*, q_2^*) \leftarrow$ NächstesPaar(P, a, b)
6. $(r_1^*, r_2^*) \leftarrow$ NächstesPaar(P, b+1, c)
7. ➤ **Zusammensetzen**

Pseudocode für das
Zusammensetzen der
beiden Hälften

Nächste Paare

NächstesPaarRec(P, a, c)

1. **if** $c-a=1$ **then return** $(P[a], P[a+1])$
2. **if** $c-a=2$ **then return** nächstes Paar aus
 $(P[a], P[a+1]), (P[a], P[a+2]), (P[a+1], P[a+2])$
4. $b \leftarrow \lfloor (a+c)/2 \rfloor$
5. $(q_1^*, q_2^*) \leftarrow$ NächstesPaar(P, a, b)
6. $(r_1^*, r_2^*) \leftarrow$ NächstesPaar($P, b+1, c$)
7. ➤ **Zusammensetzen**

Nächste Paare

Zusammensetzen

1. Merge(P,a,b,c) ➤ Sortiere nach y-Koordinate
2. $j \leftarrow 1$
3. **for** $i \leftarrow a$ **to** c **do**
4. **if** $d(P[i],L) < \delta$ **then**
5. $P'[j] \leftarrow P[i]; j \leftarrow j+1$
6. $(q,r) \leftarrow (P'[1], P'[\text{length}(P')])$
7. **for** $i \leftarrow 1$ **to** $\text{length}(P')$ **do**
8. **for** $j \leftarrow 1$ **to** 15 **do**
9. **if** $i+j \leq \text{length}(P')$ **then**
10. **if** $d(P'[i],P'[i+j]) < d(q,r)$ **then** $q \leftarrow P'[i]; r \leftarrow P'[i+j]$
11. **return** nächstes Paar aus (q,r) , (q_1^*,q_2^*) , und (r_1^*,r_2^*)

Nächste Paare

Zusammensetzen

1. **Merge(P,a,b,c)** ➤ Sortiere nach y-Koordinate
2. $j \leftarrow 1$
3. **for** $i \leftarrow a$ **to** c **do**
4. **if** $d(P[i],L) < \delta$ **then**
5. $P'[j] \leftarrow P[i]; j \leftarrow j+1$
6. $(q,r) \leftarrow (P'[1], P'[\text{length}(P')])$
7. **for** $i \leftarrow 1$ **to** $\text{length}(P')$ **do**
8. **for** $j \leftarrow 1$ **to** 15 **do**
9. **if** $i+j \leq \text{length}(P')$ **then**
10. **if** $d(P'[i], P'[i+j]) < d(q,r)$ **then** $q \leftarrow P'[i]; r \leftarrow P'[i+j]$
11. **return** nächstes Paar aus (q,r) , (q_1^*, q_2^*) , und (r_1^*, r_2^*)

Nächste Paare

Zusammensetzen

1. Merge(P,a,b,c) ➤ Sortieren
2. $j \leftarrow 1$
3. **for** $i \leftarrow a$ **to** c **do**
4. **if** $d(P[i],L) < \delta$ **then**
5. $P'[j] \leftarrow P[i]; j \leftarrow j+1$
6. $(q,r) \leftarrow (P'[1], P'[\text{length}(P')])$
7. **for** $i \leftarrow 1$ **to** $\text{length}(P')$ **do**
8. **for** $j \leftarrow 1$ **to** 15 **do**
9. **if** $i+j \leq \text{length}(P')$ **then**
10. **if** $d(P'[i],P'[i+j]) < d(q,r)$ **then** $q \leftarrow P'[i]; r \leftarrow P'[i+j]$
11. **return** nächstes Paar aus (q,r) , (q_1^*,q_2^*) , und (r_1^*,r_2^*)

In P' werden alle Punkte abgelegt, die nah an L sind

Nächste Paare

Zusammensetzen

1. Merge(P,a,b,c) ➤ Sortiere nach y-Koordinate
2. $j \leftarrow 1$
3. **for** $i \leftarrow a$ **to** c **do**
4. **if** $d(P[i],L) < \delta$ **then**
5. $P'[j] \leftarrow P[i]; j \leftarrow j+1$
6. $(q,r) \leftarrow (P'[1], P'[\text{length}(P')])$
7. **for** $i \leftarrow 1$ **to** $\text{length}(P')$ **do**
8. **for** $j \leftarrow 1$ **to** 15 **do**
9. **if** $i+j \leq \text{length}(P')$ **then**
10. **if** $d(P'[i], P'[i+j]) < d(q,r)$ **then** $q \leftarrow P'[i]; r \leftarrow P'[i+j]$
11. **return** nächstes Paar aus (q,r) , (q_1^*, q_2^*) , und (r_1^*, r_2^*)

(q,r) wird mit initialisiert (hier kann man jedes beliebige Paar verwenden)

Nächste Paare

Zusammensetzen

1. Merge(P,a,b,c) ➤ Sortiere nach y-Koordinate
2. $j \leftarrow 1$
3. **for** $i \leftarrow a$ **to** c **do**
4. **if** $d(P[i],L) < \delta$ **then**
5. $P'[j] \leftarrow P[i]; j \leftarrow j+1$
6. $(q,r) \leftarrow (P'[1], P'[\text{length}(P')])$
7. **for** $i \leftarrow 1$ **to** $\text{length}(P')$ **do**
8. **for** $j \leftarrow 1$ **to** 15 **do**
9. **if** $i+j \leq \text{length}(P')$ **then**
10. **if** $d(P'[i], P'[i+j]) < d(q,r)$ **then** $q \leftarrow P'[i]; r \leftarrow P'[i+j]$
11. **return** nächstes Paar aus (q,r) , (q_1^*, q_2^*) , und (r_1^*, r_2^*)

Für alle Punkte aus P teste, ob sie zusammen mit einem ihrer 15 Nachfolger ein besseres Paar bilden als das bislang gemerkte

Nächste Paare

Zusammensetzen

1. Merge(P,a,b,c) ➤ Sortiere nach y-Koordinate
2. $j \leftarrow 1$
3. **for** $i \leftarrow a$ **to** c **do**
4. **if** $d(P[i],L) < \delta$ **then**
5. $P'[j] \leftarrow P[i]; j \leftarrow j+1$
6. $(q,r) \leftarrow (P'[1], P'[\text{length}(P')])$
7. **for** $i \leftarrow 1$ **to** $\text{length}(P')$ **do**
8. **for** $j \leftarrow 1$ **to** 15 **do**
9. **if** $i+j \leq \text{length}(P')$ **then**
10. **if** $d(P'[i],P'[i+j]) < d(q,r)$ **then** $q \leftarrow P'[i]; r \leftarrow P'[i+j]$
11. **return** nächstes Paar aus (q,r) , (q_1^*,q_2^*) , und (r_1^*,r_2^*)

Gib das beste
gefundene Paar
zurück

Nächste Paare

Zusammensetzen

1. Merge(P,a,b,c) ➤ Sortiere nach y-Koordinate
2. $j \leftarrow 1$
3. **for** $i \leftarrow a$ **to** c **do**
4. **if** $d(P[i],L) < \delta$ **then**
5. $P'[j] \leftarrow P[i]; j \leftarrow j+1$
6. $(q,r) \leftarrow (P'[1], P'[\text{length}(P')])$
7. **for** $i \leftarrow 1$ **to** $\text{length}(P')$ **do**
8. **for** $j \leftarrow 1$ **to** 15 **do**
9. **if** $i+j \leq \text{length}(P')$ **then**
10. **if** $d(P'[i], P'[i+j]) < d(q,r)$ **then** $q \leftarrow P'[i]; r \leftarrow P'[i+j]$
11. **return** nächstes Paar aus (q,r) , (q_1^*, q_2^*) , und (r_1^*, r_2^*)

Rate durch wiederholtes Einsetzen

$$T(n) = \begin{cases} 2T(n/2) + an & n > 3 \\ a & n \leq 3 \end{cases}$$

$$T(n) =$$

Verifiziere durch Induktion

$$T(n) \leq an \log n$$

$$T(n) = \begin{cases} 2T(n/2) + an & n > 3 \\ a & n \leq 3 \end{cases}$$

$$n = 2^i$$

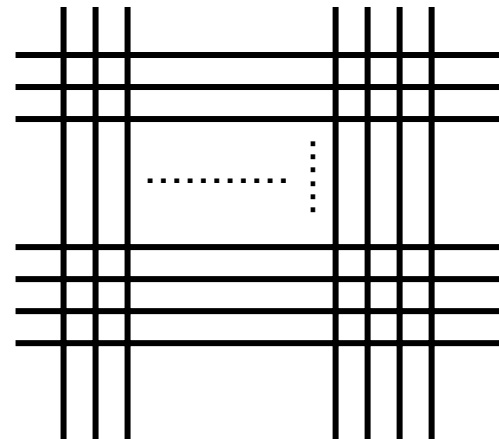
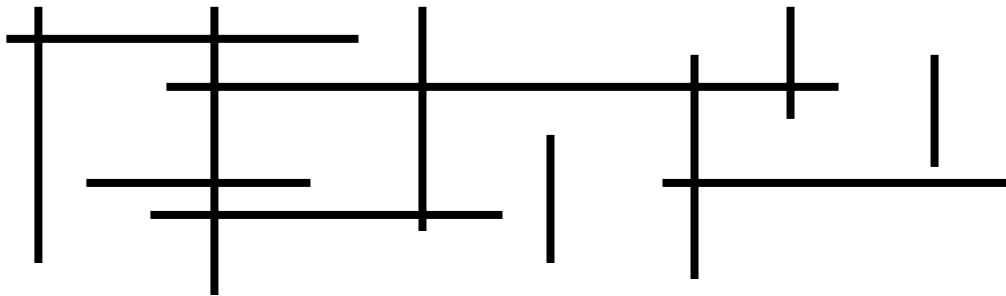
$$i = 1: \text{ ok}$$

$$i > 1$$

$$T(2^i) =$$

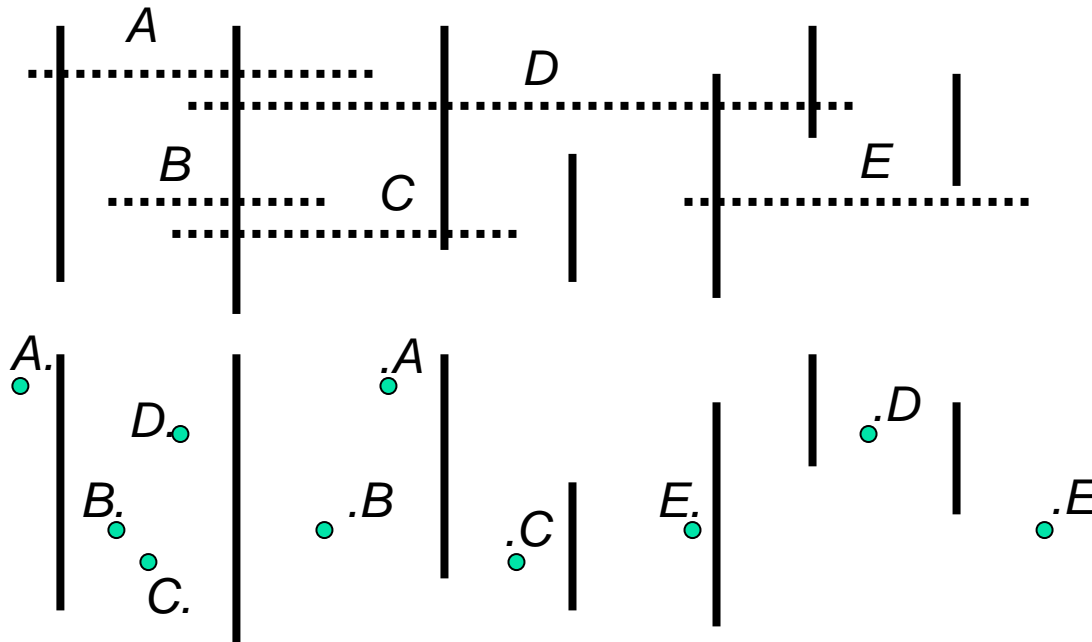
Segmentschnittproblem

Bestimme alle Paare sich schneidender Segmente



Segmentschnittproblem

Bestimme alle Paare sich schneidender Segmente



Die getrennte Repräsentation der Segmente erlaubt eine Aufteilung

Input: Menge S bestehend aus vertikalen Segmenten und Endpunkten von horizontalen Segmenten.

Output: Alle Schnittpunkte von vertikalen Segmenten mit horizontalen Segmenten, von denen mindestens ein Endpunkt in S ist.

1. Divide

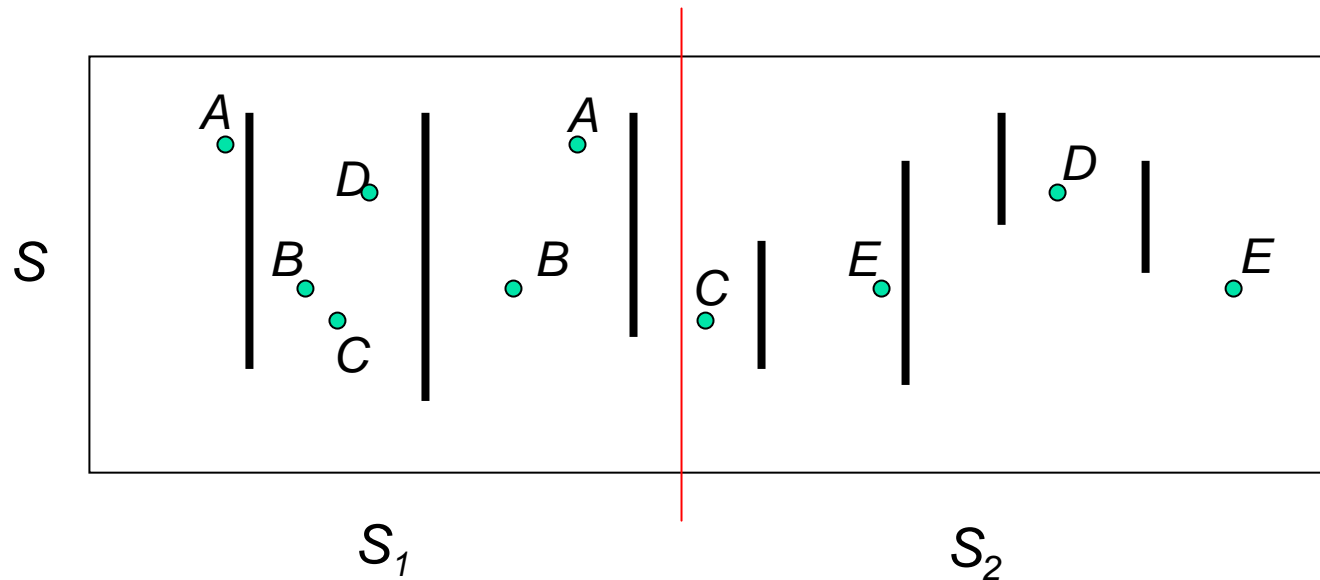
if $|S| > 1$

then teile S mittels einer vertikalen Geraden G in zwei gleichgroße Mengen S_1 (links von G) und S_2 (rechts von G)

else S enthält keine Schnitte

ReportCuts

1. Divide-Schritt



2. Conquer

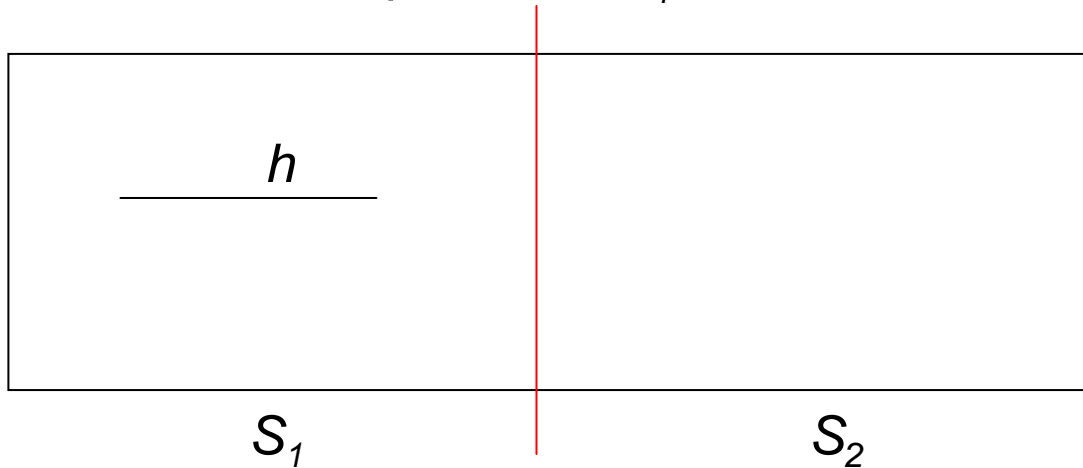
$\text{ReportCuts}(S_1); \text{ReportCuts}(S_2)$

ReportCuts

3. Merge: ???

Mögliche Schnitte für ein horizontales Segment in S_1

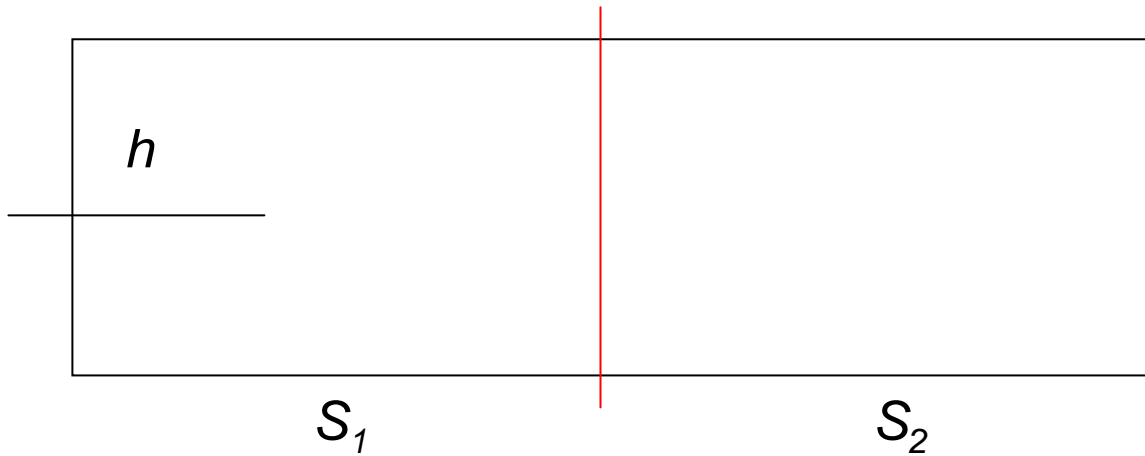
Fall 1: beide Endpunkte in S_1



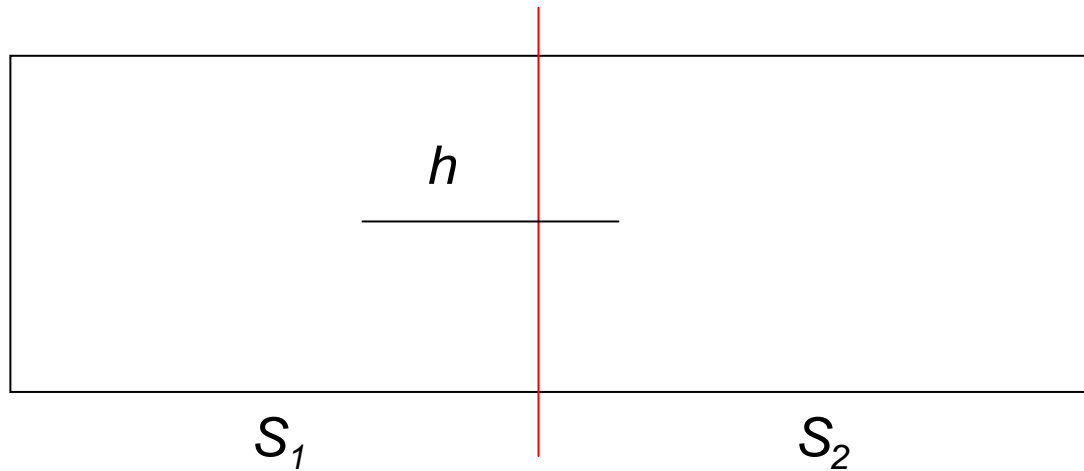
ReportCuts

Fall 2: nur ein Endpunkt von h in S_1

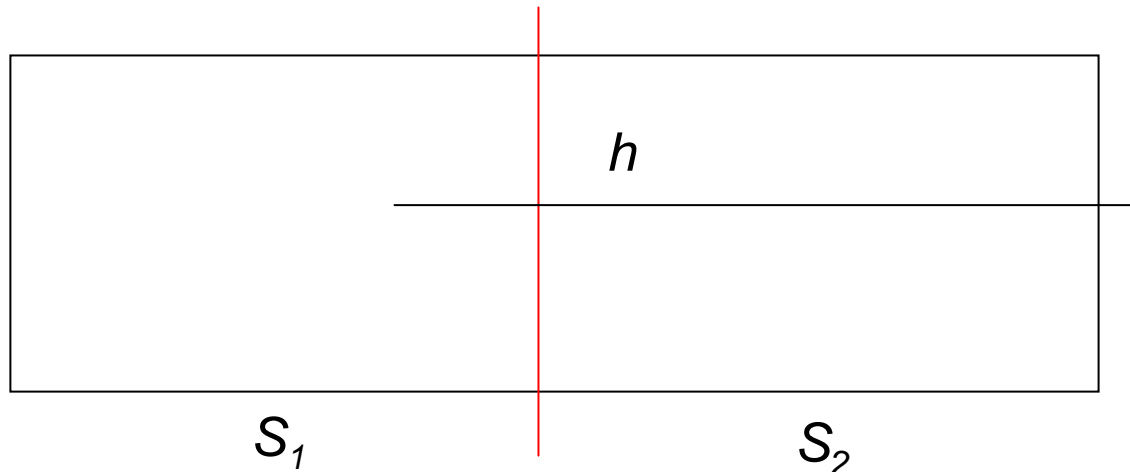
2 a) rechter Endpunkt in S_1



2 b) linker Endpunkt von h in S_1



rechter Endpunkt
in S_2

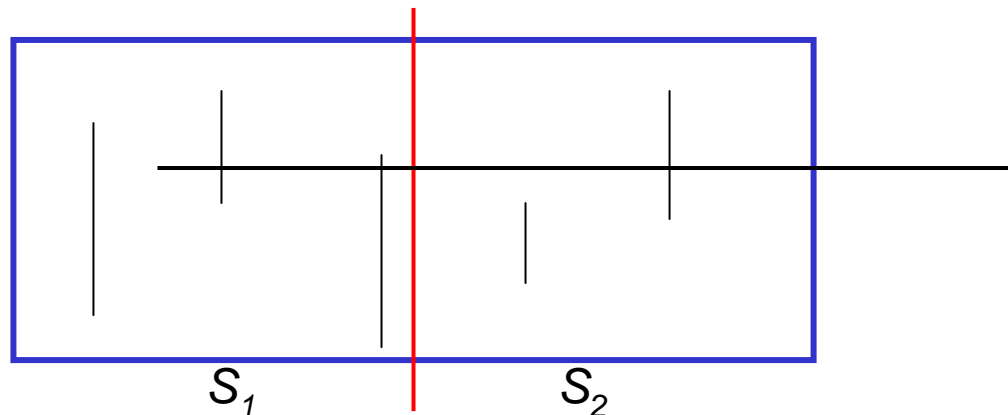


rechter Endpunkt
nicht in S_2

Verfahren: ReportCuts(S)

3. Merge:

Gib Schnitte aus zwischen vertikalen Segmenten in S_2 und horizontalen Segmenten in S_1 , bei denen linker Endpunkt in S_1 und rechter Endpunkt weder in S_1 noch S_2
 Analog für S_1



Implementierung

Menge S

$L(S)$: y -Koordinaten aller linken Endpunkte in S , deren rechter Partner nicht in S

$R(S)$: y -Koordinaten aller rechten Endpunkte in S , deren linker Partner nicht in S

$V(S)$: y -Intervalle der vertikalen Segmente in S

Basisfälle

S enthält nur ein Element s

Fall 1: $s = (x, y)$ ist ein linker Endpunkt

$$L(S) = \{y\} \quad R(S) = \emptyset \quad V(S) = \emptyset$$

Fall 2: $s = (x, y)$ ist ein rechter Endpunkt

$$L(S) = \emptyset \quad R(S) = \{y\} \quad V(S) = \emptyset$$

Fall 3: $s = (x, y_1, y_2)$ ist ein vertikales Segment

$$L(S) = \emptyset \quad R(S) = \emptyset \quad V(S) = \{[y_1, y_2]\}$$

Merge-Schritt

$L(S_i), R(S_i), V(S_i)$ $i=1,2$ seien berechnet

$$S = S_1 \cup S_2$$

$$L(S) =$$

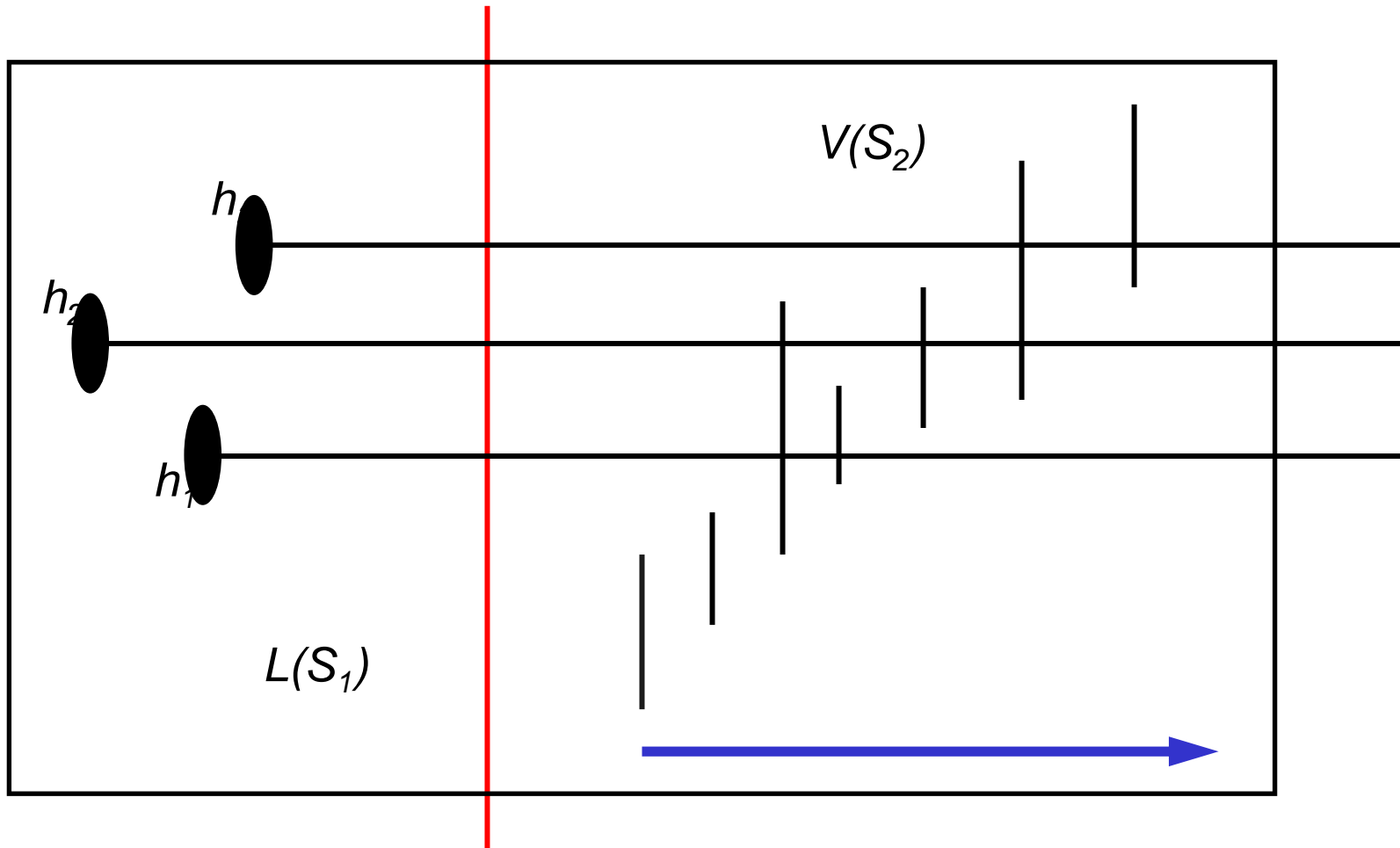
$$R(S) =$$

$$V(S) =$$

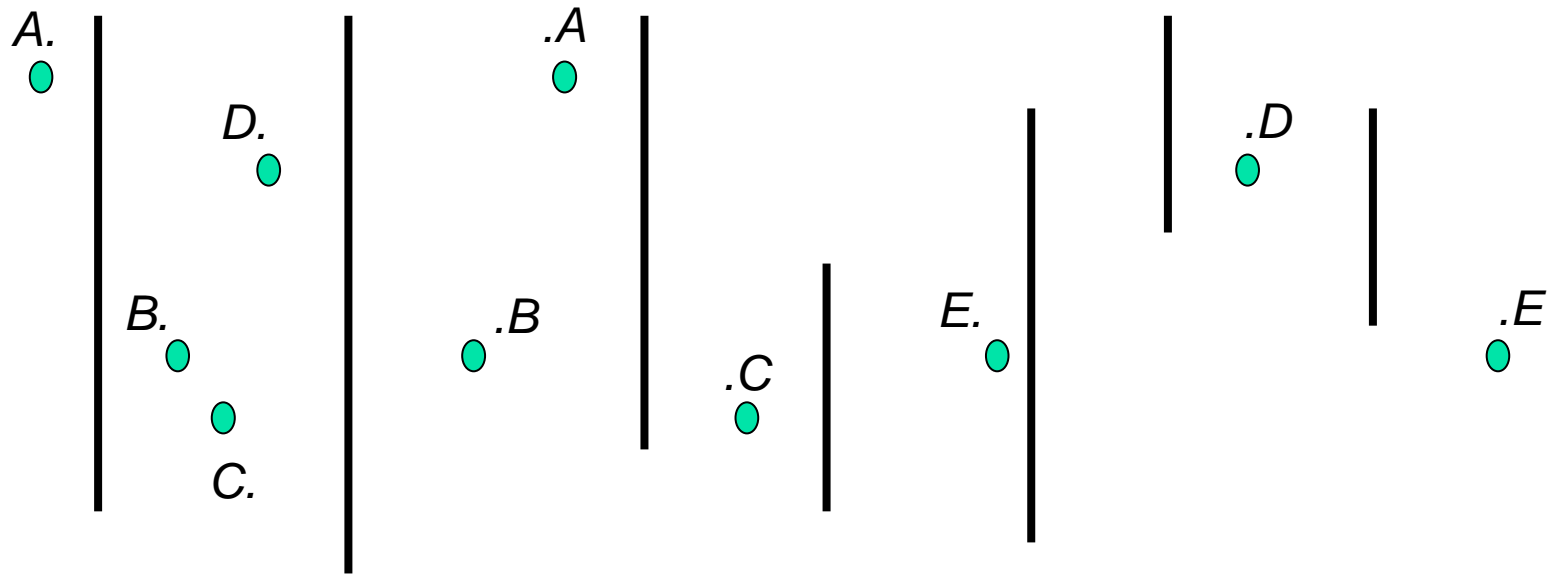
L, R : sortiert nach steigenden y -Koordinaten
verkettete Listen

V : sortiert nach steigenden unteren Endpunkten
verkettete Liste

Ausgabe der Schnittpunkte



Segmentschnittproblem



Eingabe (vertikale Seg., linke/rechte Endpunkte horizontaler Seg.)
wird anfangs einmal **sortiert; abgespeichert in Array.**

Divide-and-Conquer:

$$T(n) = 2T(n/2) + an + \text{Größe der Ausgabe}$$

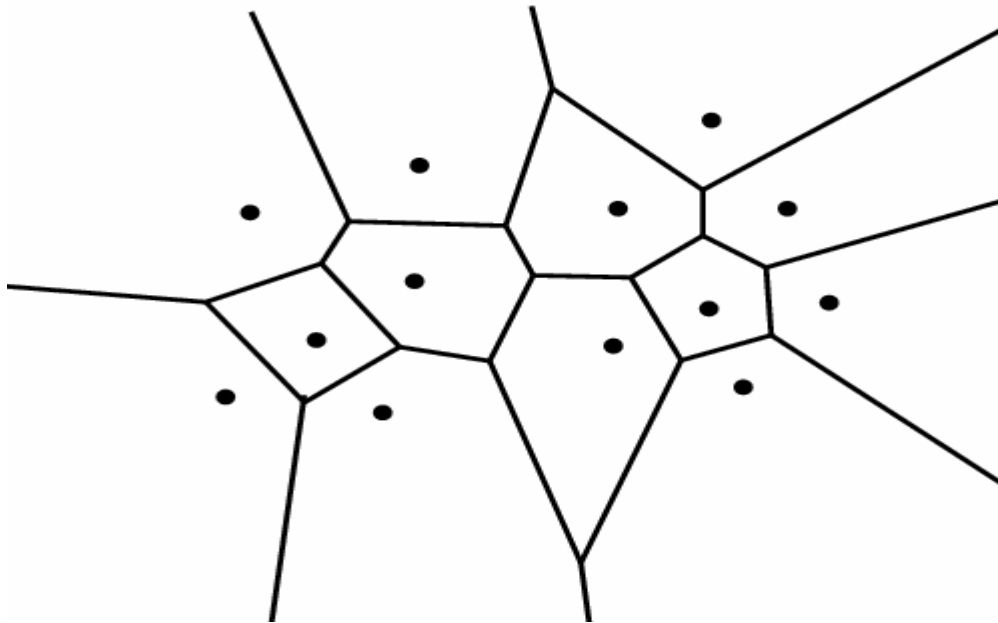
$$T(1) = O(1)$$

$$O(n \log n + k) \quad k = \#\text{Schnittpunkte}$$

Berechnung des Voronoi-Diagramms

Gegeben: Eine Menge von Orten (sites)

Gesucht: Eine Unterteilung der Ebene in Regionen gleicher nächster Nachbarn



Definition von Voronoi-Diagrammen

P : Menge von Orten

$$H(p \mid p') = \{x \mid x \text{ liegt näher an } p \text{ als an } p'\}$$

Voronoi-Region von p

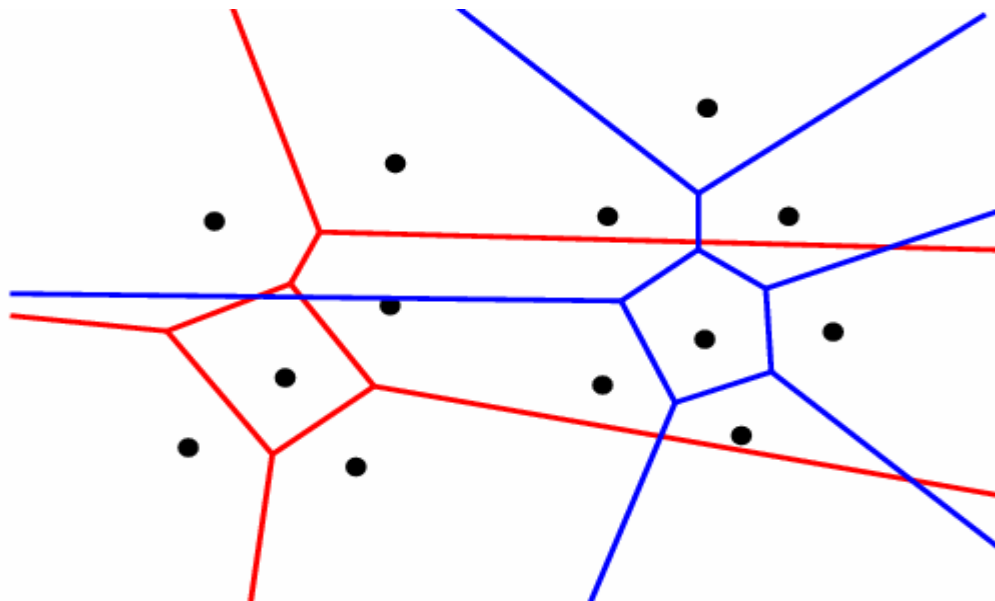
$$VR(p) = \bigcap_{p' \in P \setminus \{p\}} H(p \mid p')$$

Berechnung des Voronoi-Diagramms

Divide : Einteilung der Menge der Orte in zwei Hälften

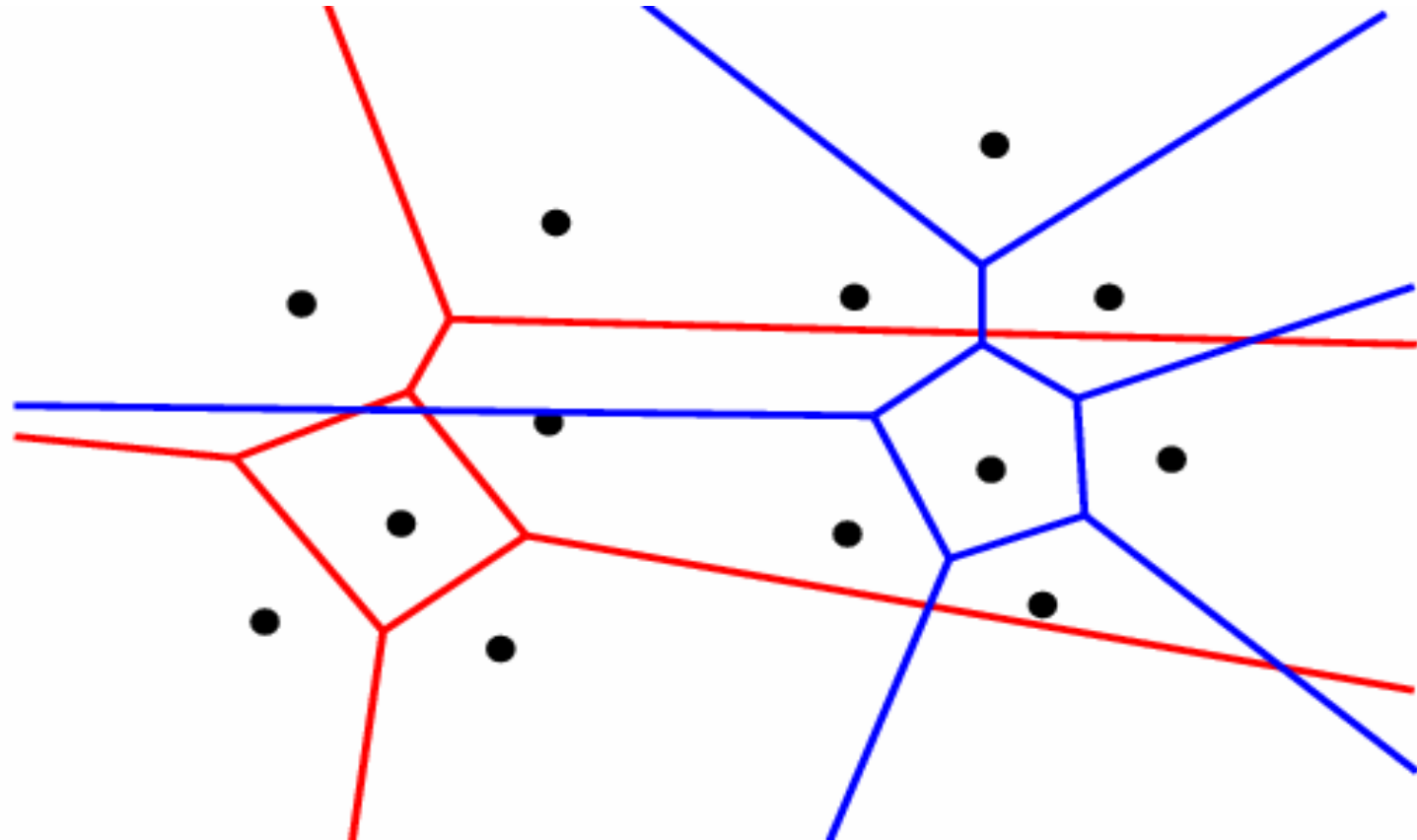
Conquer: Rekursive Berechnung der beiden kleineren Voronoi-Diagramme

Abbruchbedingung: Voronoi-Diagramm eines einzelnen Ortes ist die gesamte Ebene



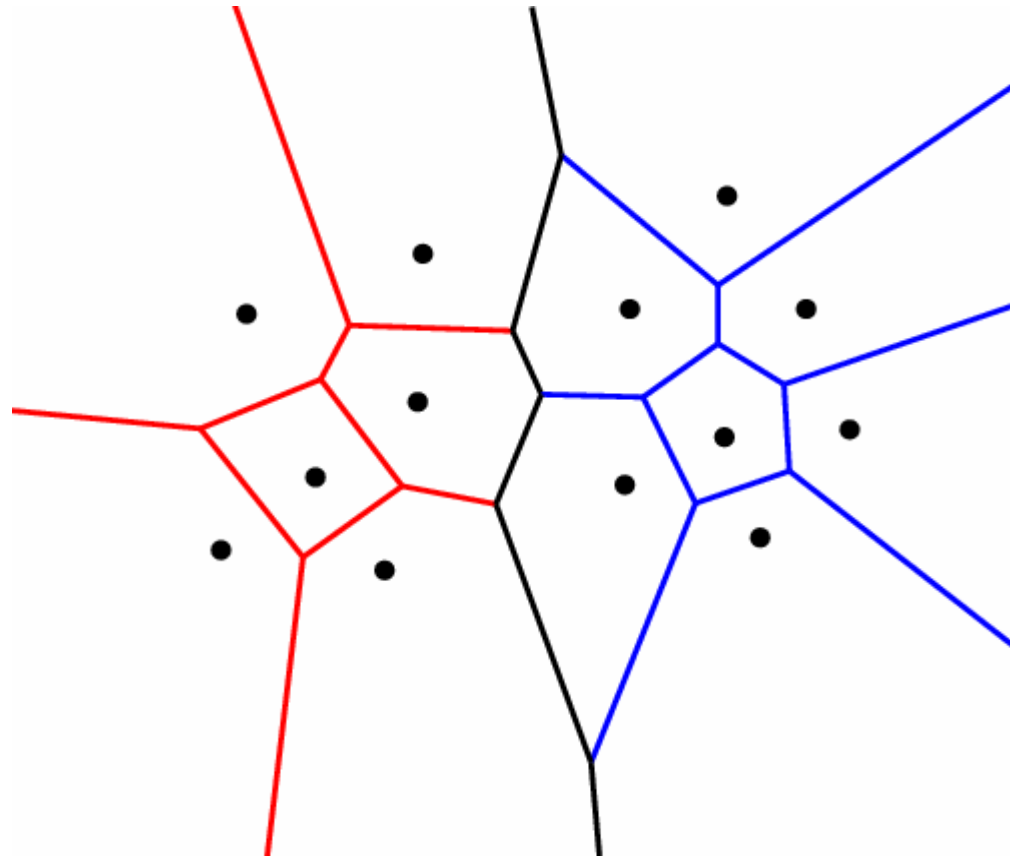
Merge: Verbindung der Diagramme durch einen Kantenzug

Berechnung des Voronoi-Diagramms



Berechnung des Voronoi-Diagramms

Ergebnis: Das fertige Voronoi-Diagramm



Laufzeit: Bei n gegebenen Orten $O(n \log n)$