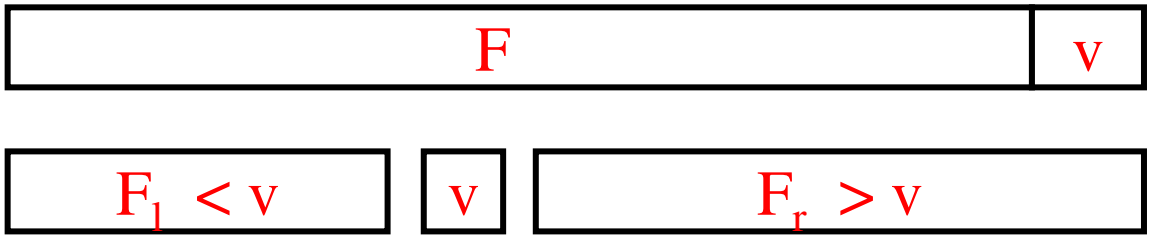
Divide and Conquer

Das Divide - and - Conquer Prinzip

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

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 $n_1,...,n_k$ 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

Analyse

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

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

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

Divide- und Mergeaufwand : *DM(n)*

$T(1) = a$

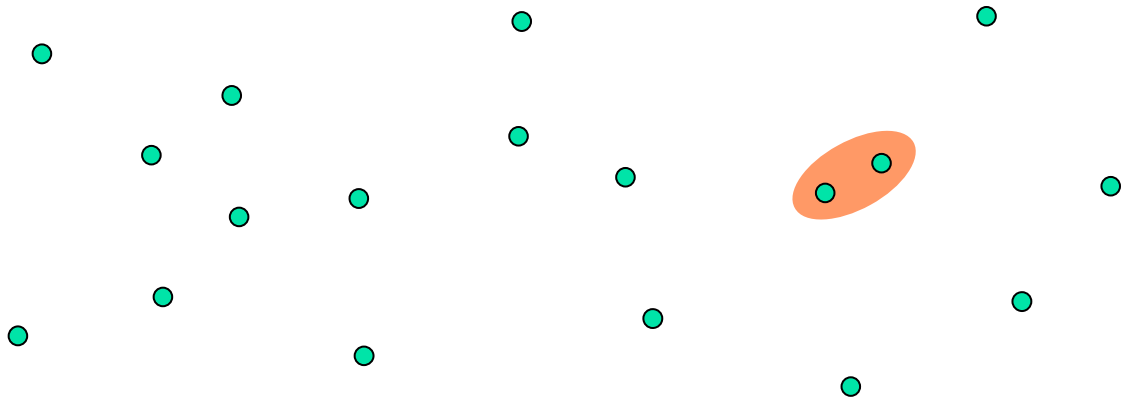
$T(n) = 2T(n/2) + DM(n)$

Nächste Paare

Geometrisches Problem:

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

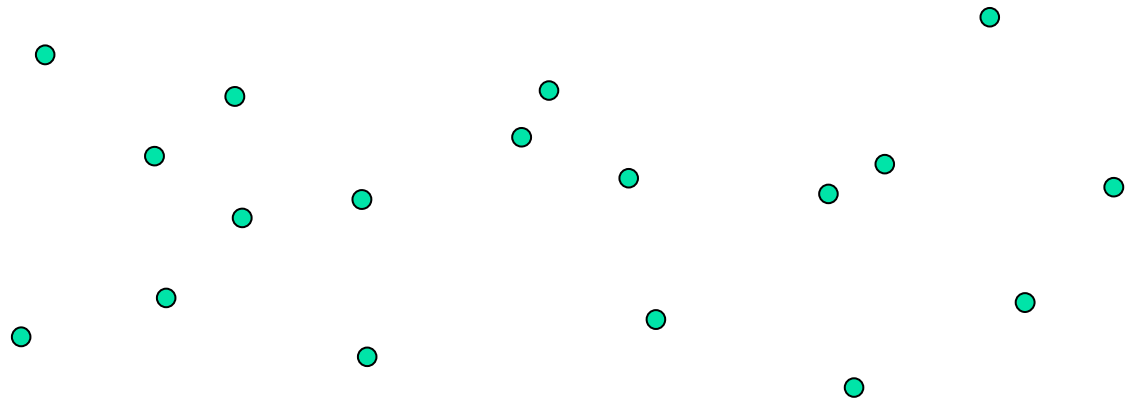
Beispiel:



Nächste Paare

Notation:

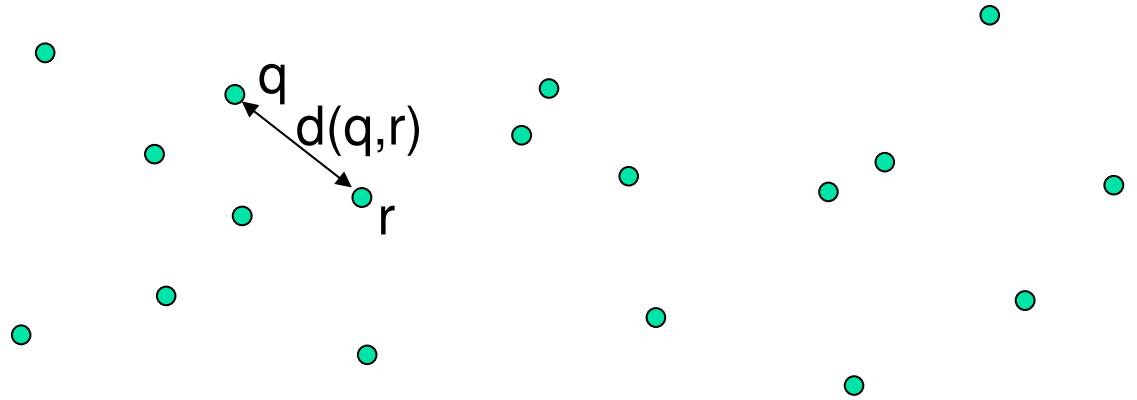
- d(q,r) bezeichnet Abstand zwischen q und r



Nächste Paare

Notation:

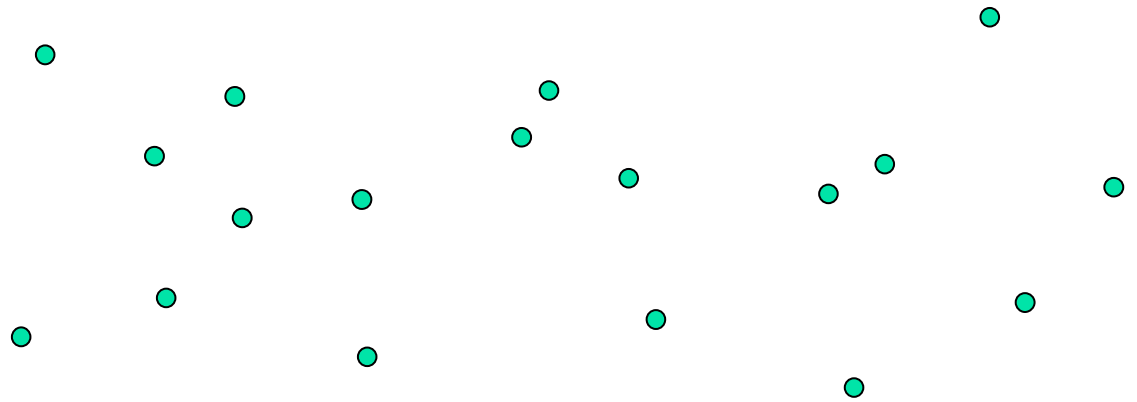
- d(q,r) bezeichnet Abstand zwischen q und r



Nächste Paare

Plan für 2D:

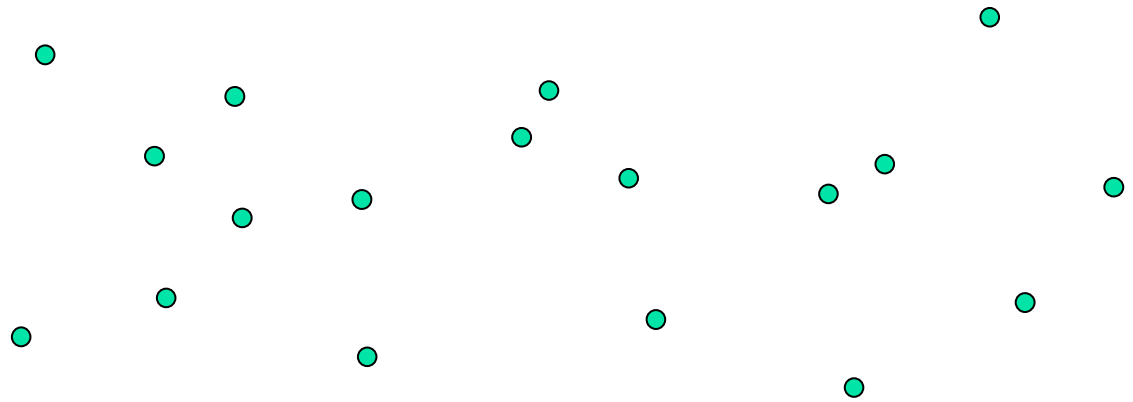
- Wie MergeSort nur im 2D



Nächste Paare

Plan:

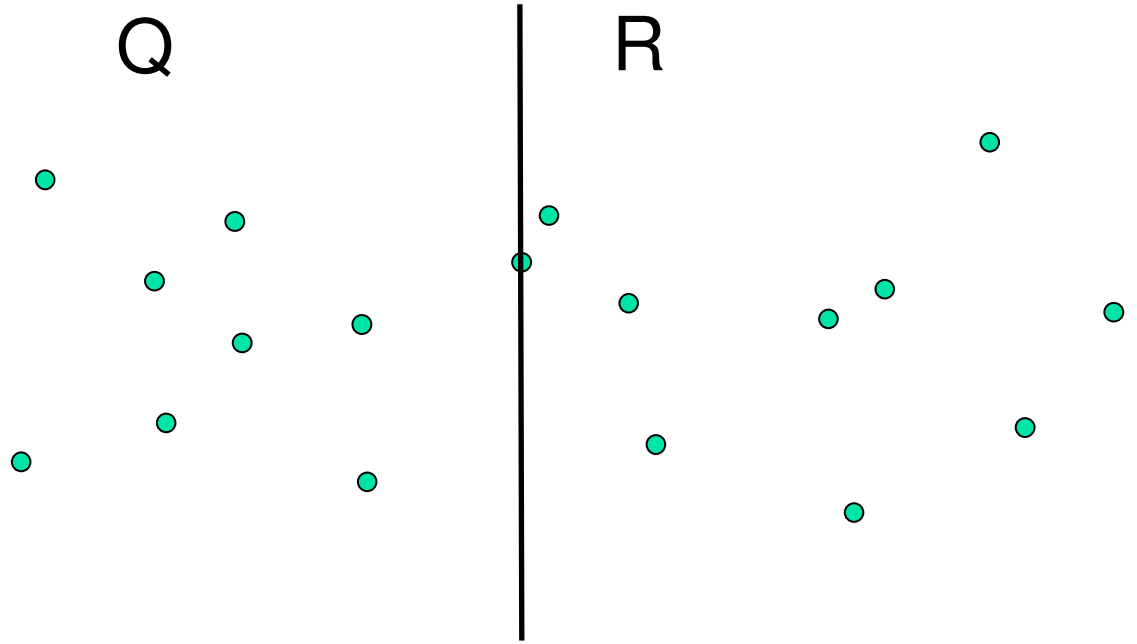
- Wie MergeSort nur im 2D
- Sortiere Punktmenge nach x-Koordinate



Nächste Paare

Plan:

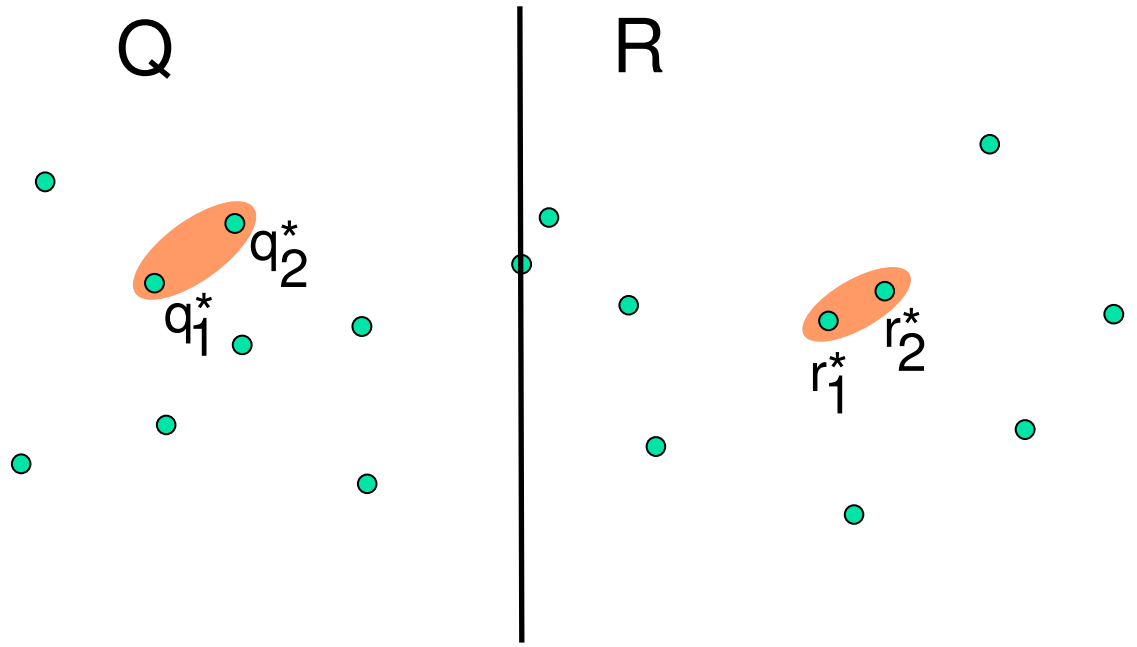
- Wie MergeSort nur im 2D
- Sortiere Punktmenge nach x-Koordinate
- Teile in der Mitte



Nächste Paare

Plan:

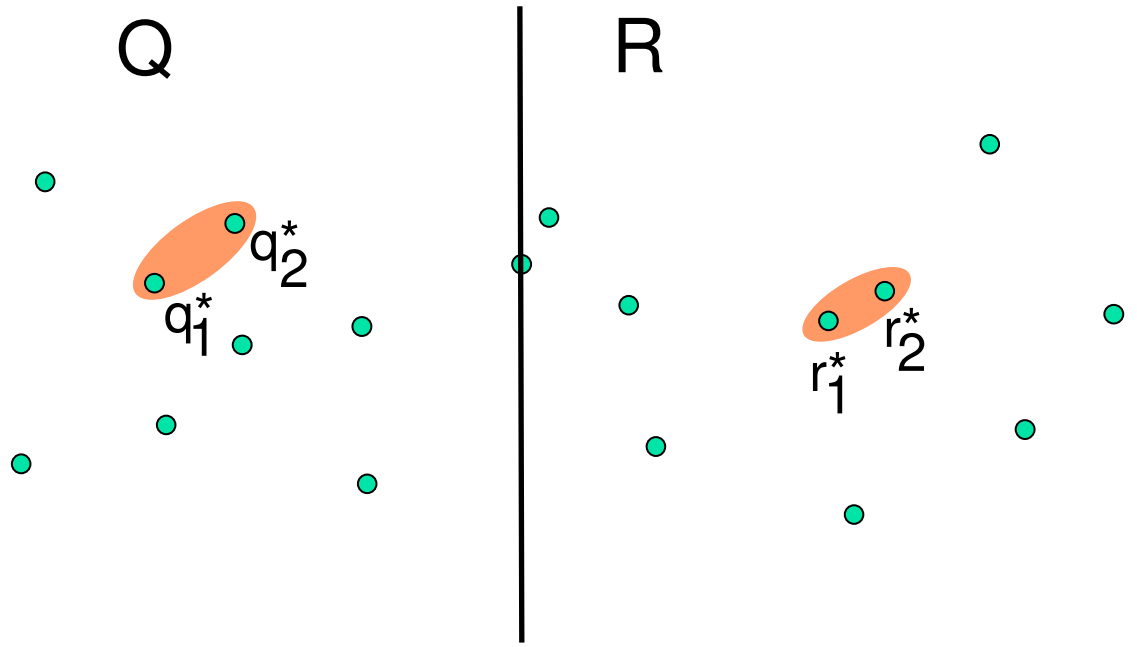
- Wie MergeSort nur im 2D
- Sortiere Punktmenge nach x-Koordinate
- Teile in der Mitte
- Löse rekursiv



Nächste Paare

Plan:

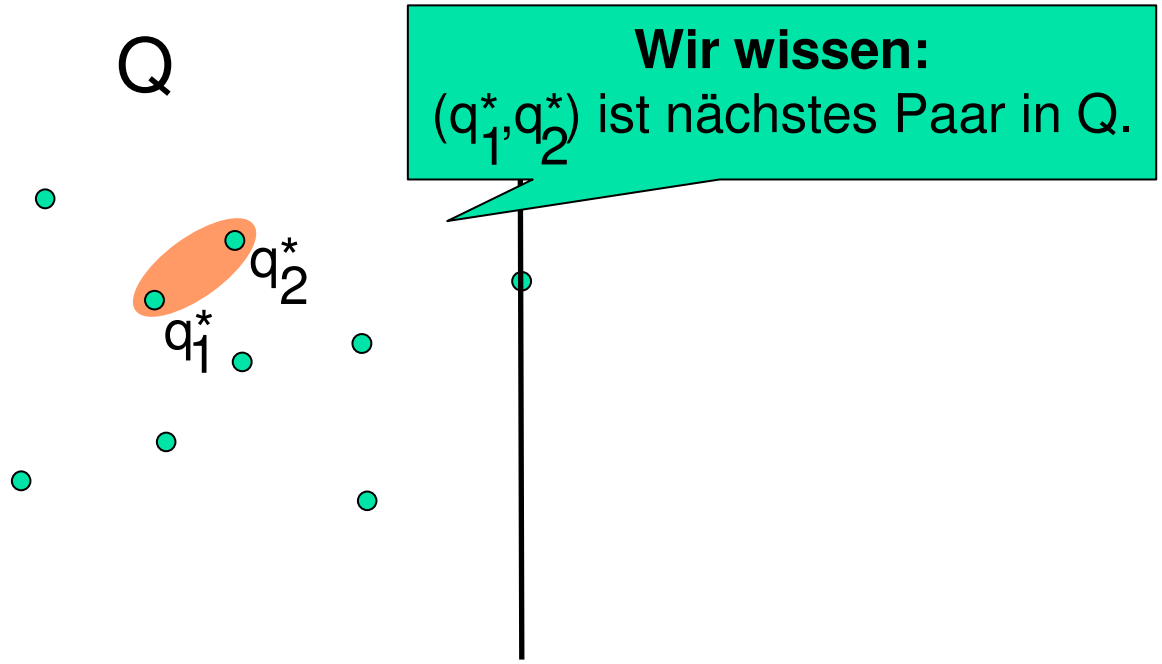
- Wie MergeSort nur im 2D
- Sortiere Punktmenge nach x-Koordinate
- Teile in der Mitte
- Löse rekursiv
- Füge zusammen



Nächste Paare

Plan:

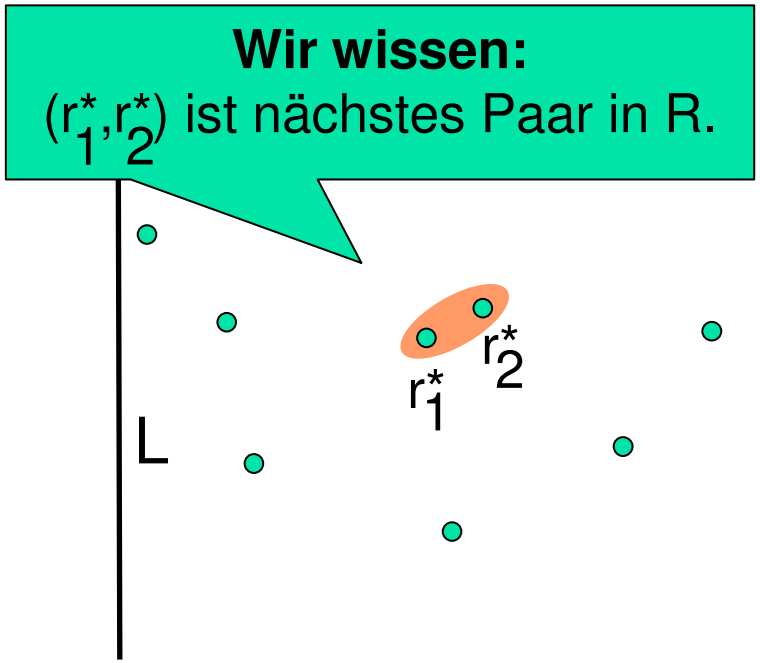
- Wie MergeSort nur im 2D
- Sortiere Punktmenge nach x-Koordinate
- Teile in der Mitte
- Löse rekursiv
- Füge zusammen



Nächste Paare

Plan:

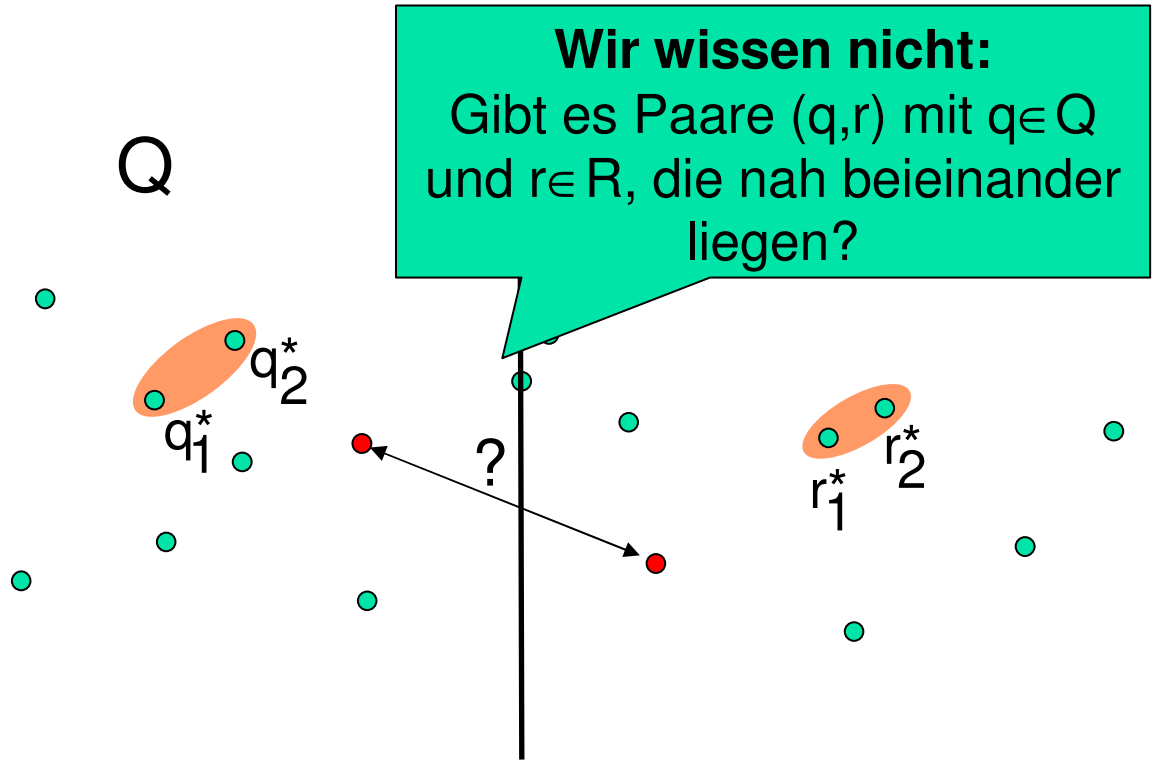
- Wie MergeSort nur im 2D
- Sortiere Punktmenge nach x-Koordinate
- Teile in der Mitte
- Löse rekursiv
- Füge zusammen



Nächste Paare

Plan:

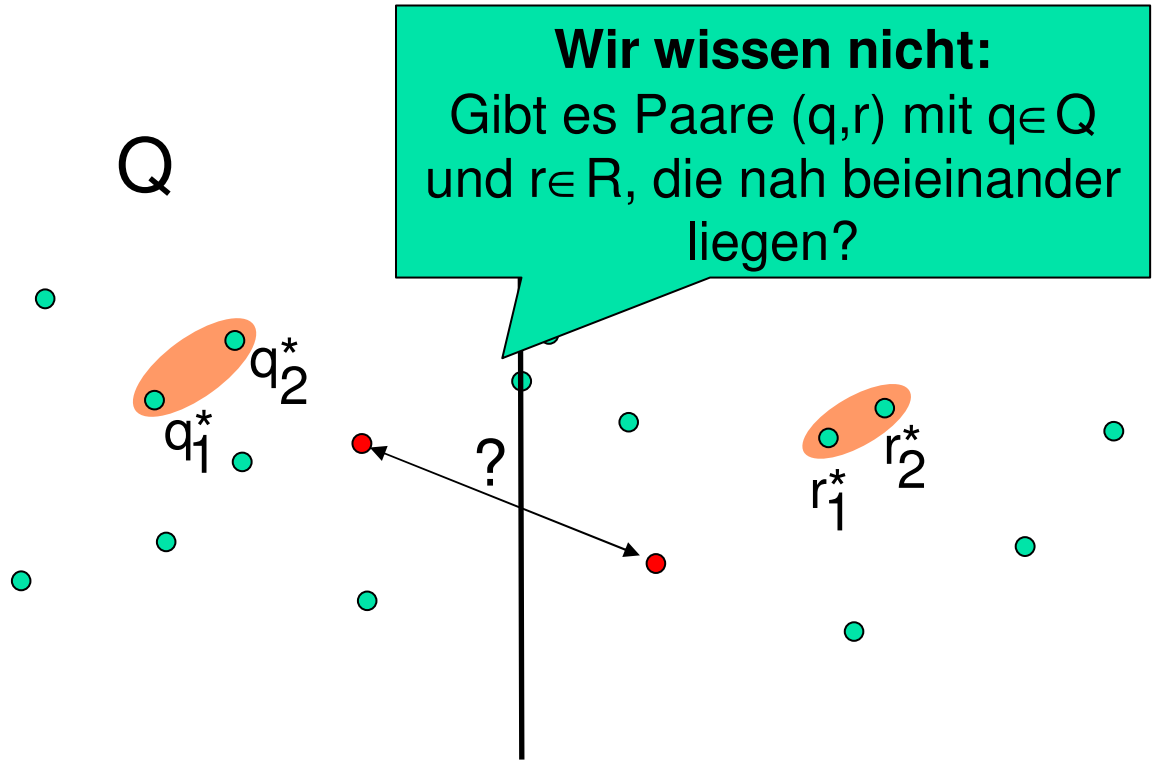
- Wie MergeSort nur im 2D
- Sortiere Punktmenge nach x-Koordinate
- Teile in der Mitte
- Löse rekursiv
- Füge zusammen



Nächste Paare

Plan:

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

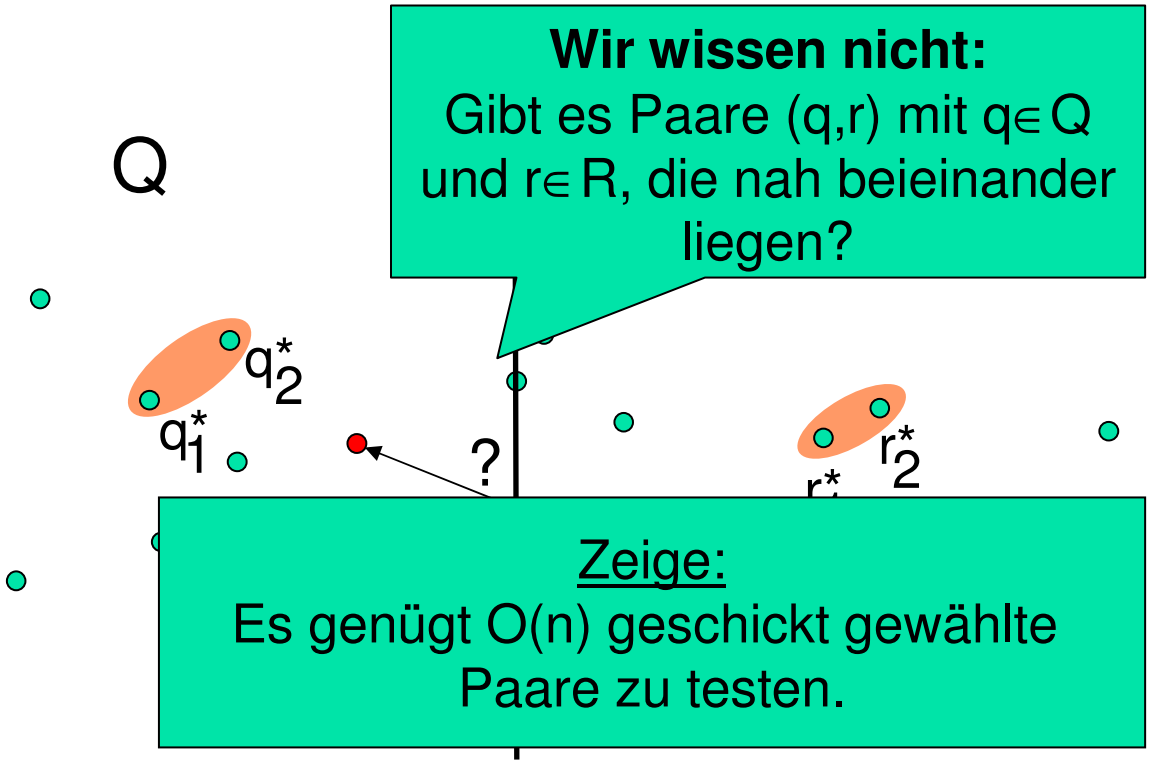


Problem:
Es gibt sehr viele solche Paare. ($\Theta(n^2)$ viele).

Nächste Paare

Plan:

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



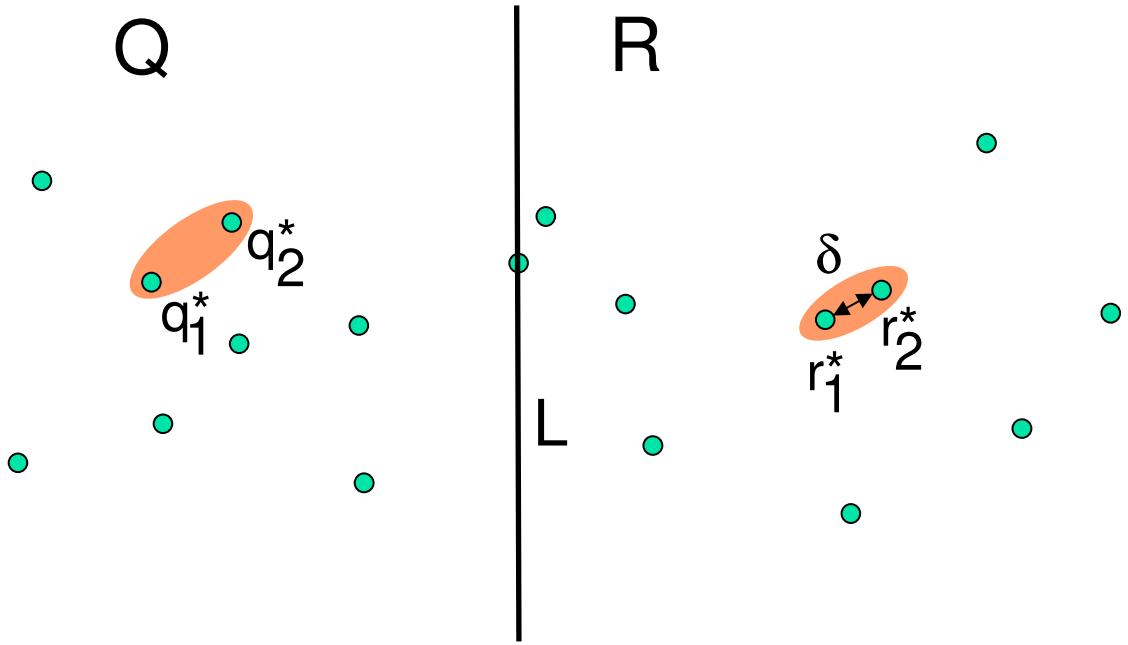
Problem:
Es gibt sehr viele solche Paare. ($\Theta(n^2)$ viele).

Zeige:
Es genügt O(n) geschickt gewählte Paare zu testen.

Nächste Paare

Definition:

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

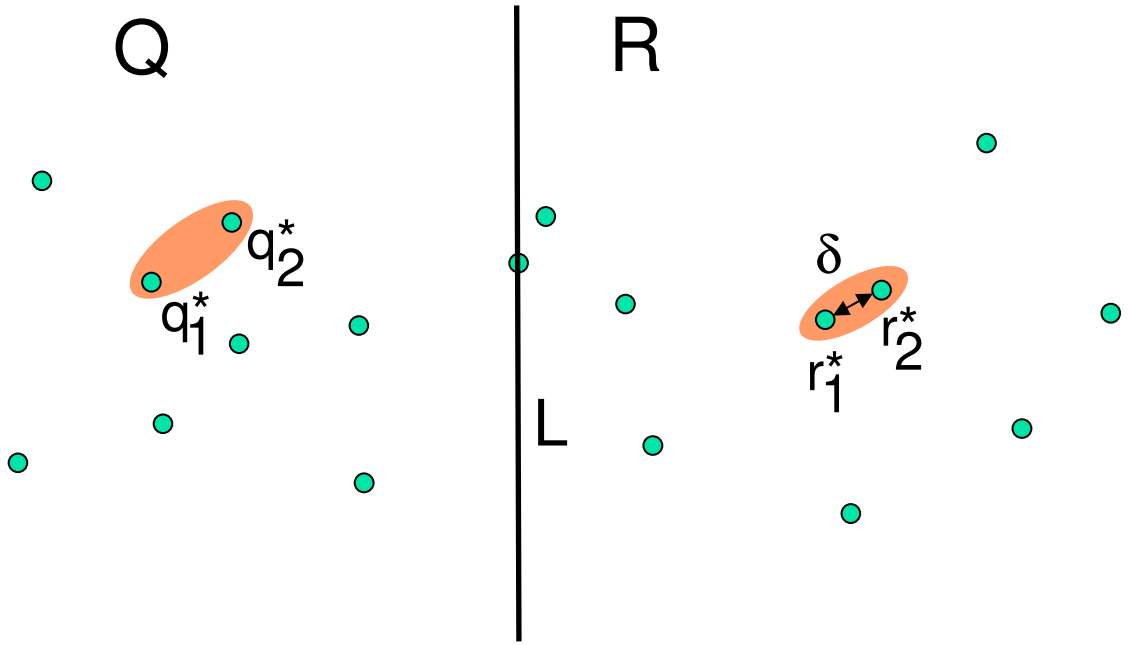


Nächste Paare

Wichtige Beobachtung:

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

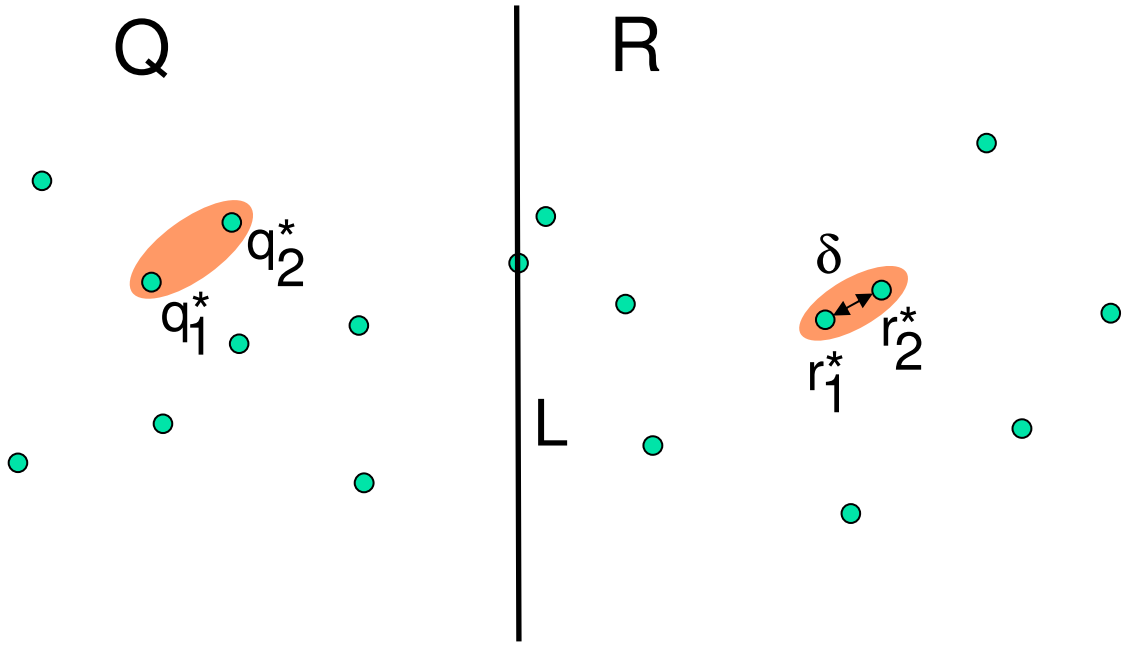
Sind $q,r \in R$, dann gild $d(p,r) \geq \delta$.



Nächste Paare

Lemma 1.1

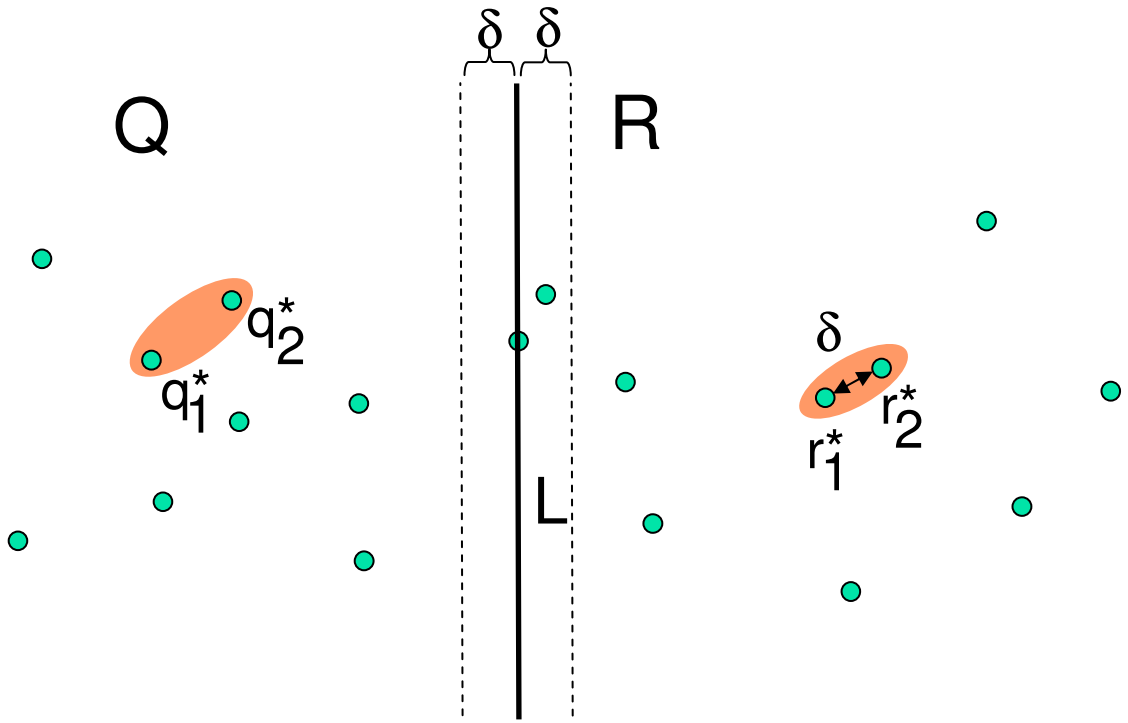
Gibt es $q \in Q$ und $r \in R$ mit $d(q,r) < \delta$, dann sind sowohl q als auch r höchstens δ von L entfernt.



Nächste Paare

Lemma 1.1

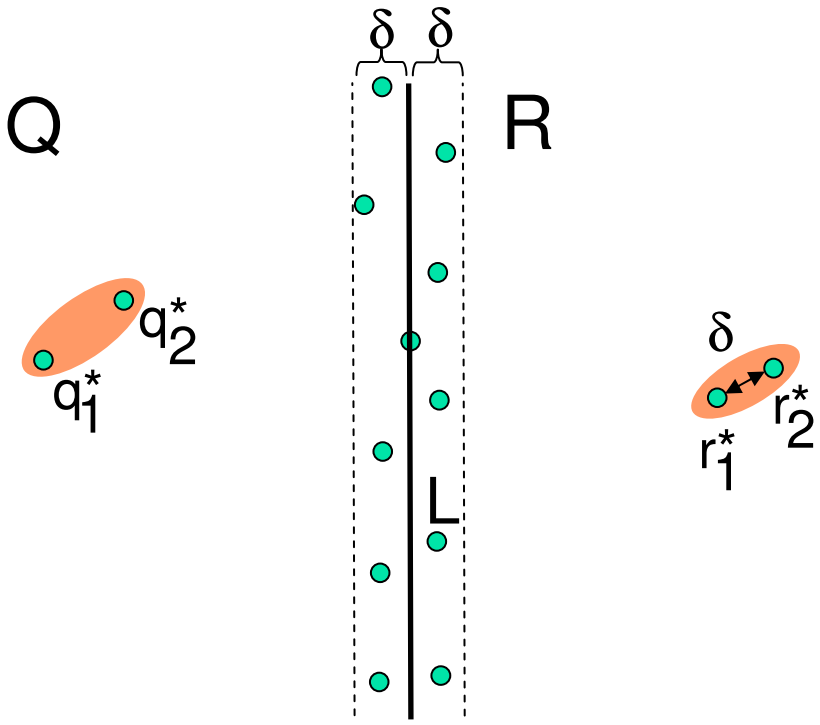
Gibt es $q \in Q$ und $r \in R$ mit $d(q,r) < \delta$, dann sind sowohl q als auch r höchstens δ von L entfernt.



Nächste Paare

Was gewinnen wir?

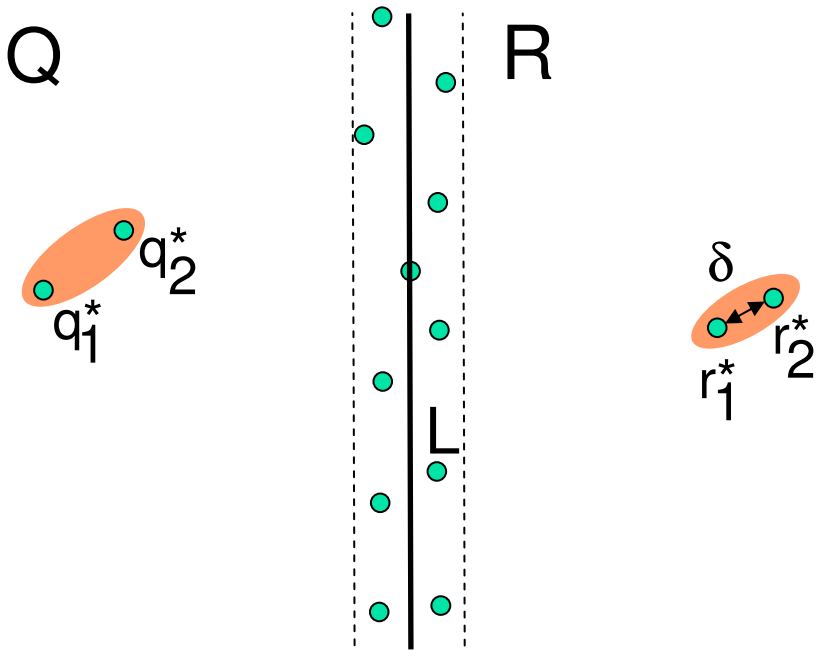
- Alle Punkte können nahe an L liegen



Nächste Paare

Abhilfe:

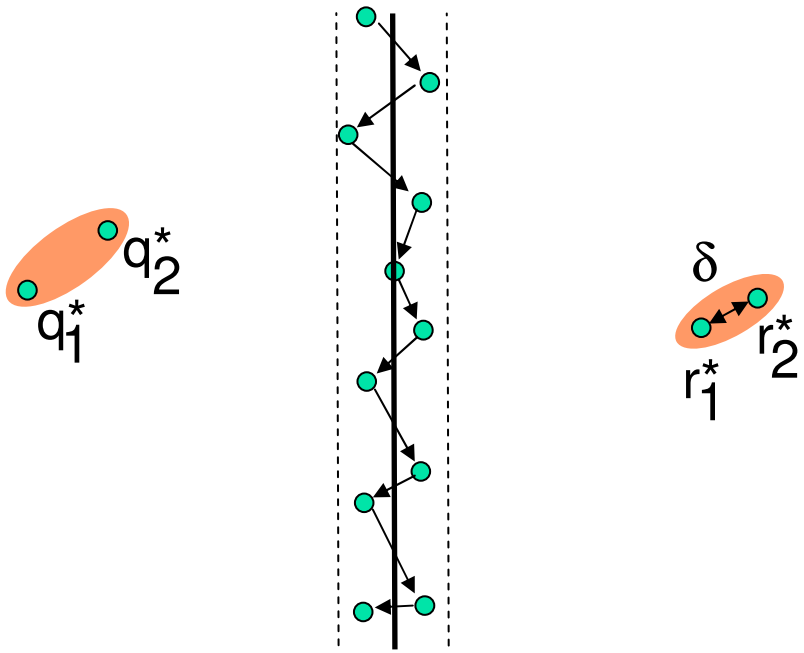
- Wir wollen nur Paare (q,r) testen, wenn
  (a) q und r nah an L liegen und
  (b) q und r kleinen Abstand in y-Richtung haben



Nächste Paare

Idee:

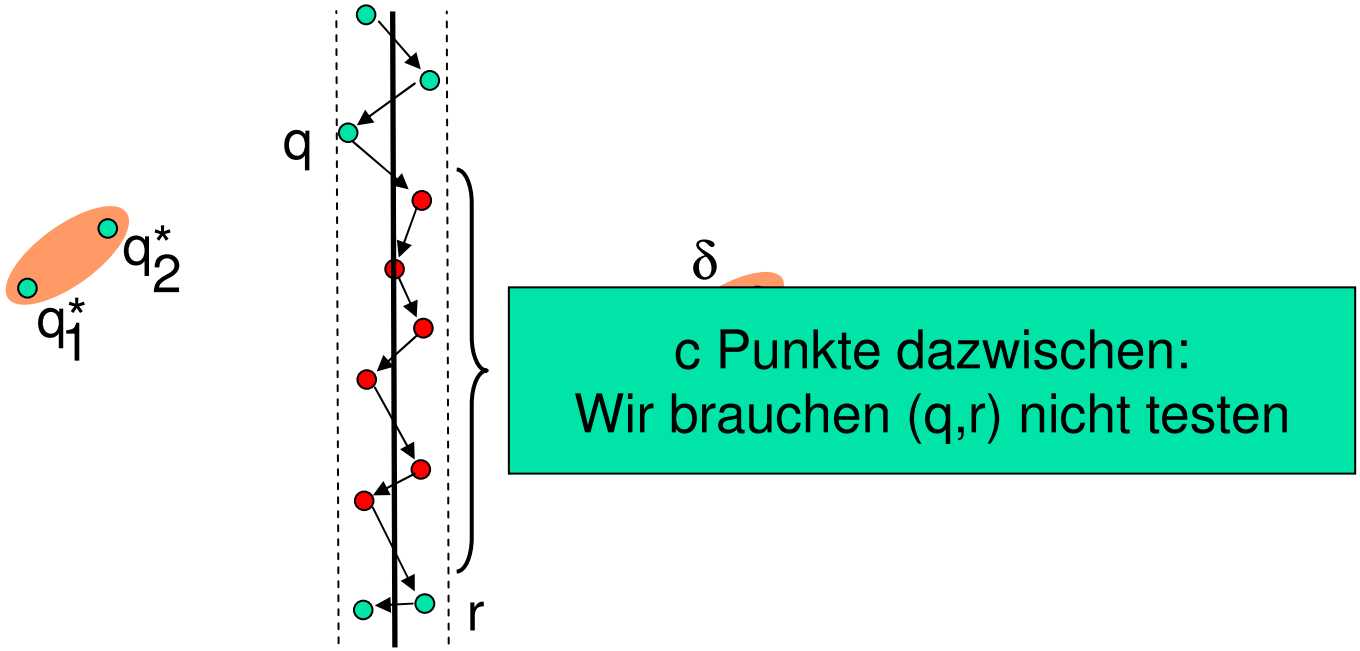
- Sortiere nahe Punkte nach y-Koordinate
- Zeige: Liegen in dieser Sortierung mehr als c Punkte zwischen q und r, dann kann (q,r) nicht nächstes Paar sein



Nächste Paare

Idee:

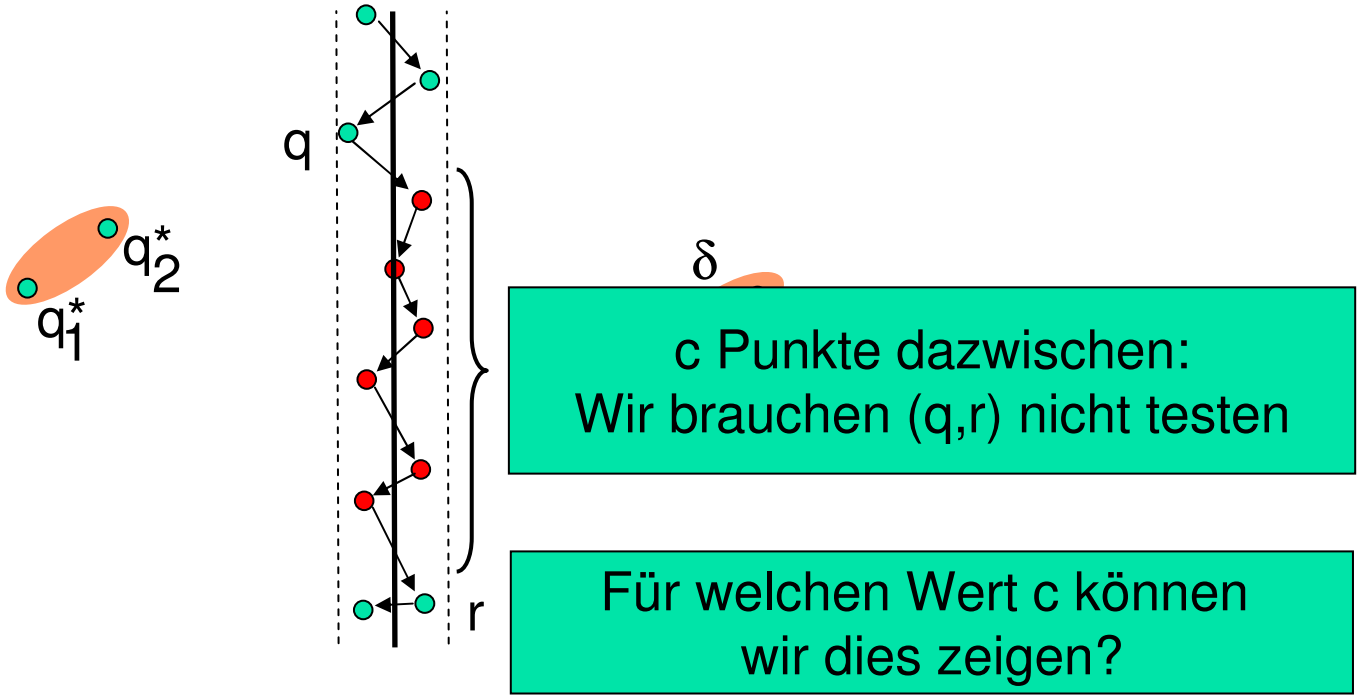
- Sortiere nahe Punkte nach y-Koordinate
- Zeige: Liegen in dieser Sortierung mehr als c Punkte zwischen q und r, dann kann (q,r) nicht nächstes Paar sein



Nächste Paare

Idee:

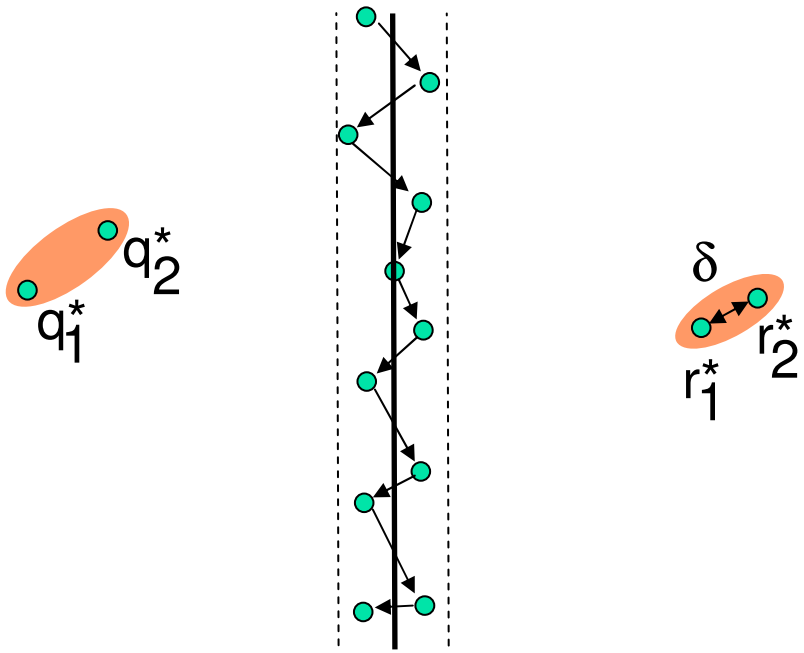
- Sortiere nahe Punkte nach y-Koordinate
- Zeige: Liegen in dieser Sortierung mehr als c Punkte zwischen q und r, dann kann (q,r) nicht nächstes Paar sein



Nächste Paare

Lemma 1.2

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



Nächste Paare

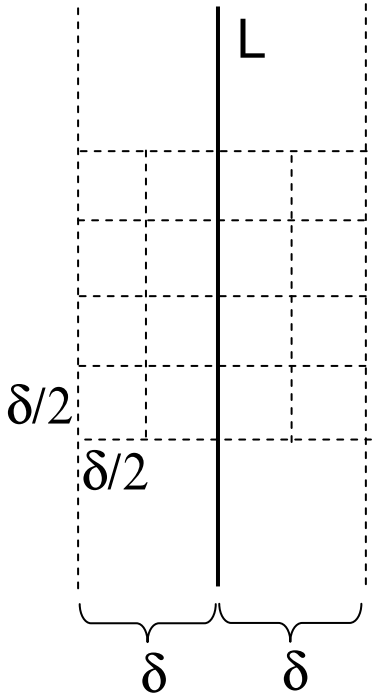
Lemma 1.2

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

Beweis:

- Teile Nahbereich in Quadrate der Seitenlänge δ/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) >δ.

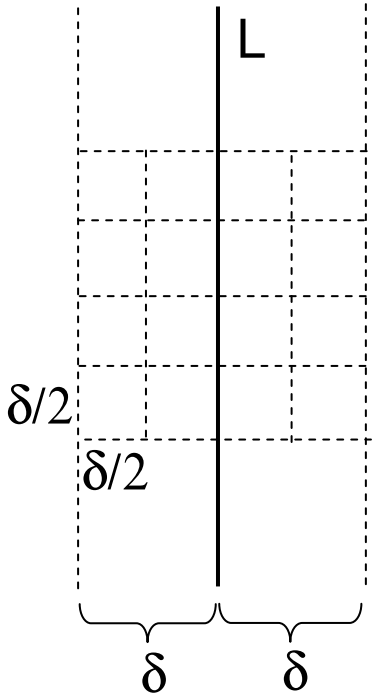
Beweis:

- Teile Nahbereich in Quadrate der Seitenlänge δ/2 auf

Beh.: In jedem Quadrat ist nur ein Punkt.

Beweis:

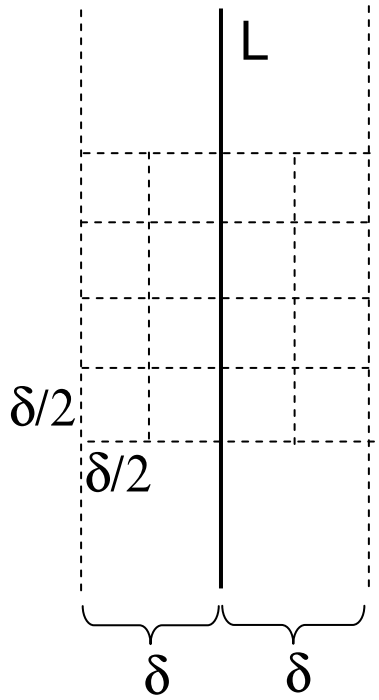
- Annahme: Zwei Punkte q,r in einem Quadrat



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 $d(q,r) \leq \delta / 2^{1/2} < \delta$

Nächste Paare

Lemma 1.2

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

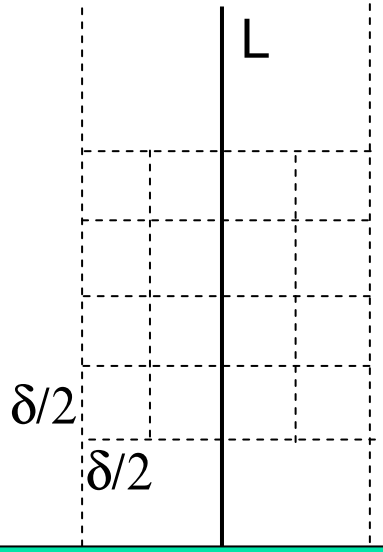
Beweis:

- Teile Nahbereich in Quadrate der Seitenlänge δ/2 auf

Beh.: In jedem Quadrat ist nur ein Punkt.

Beweis:

- Annahme: Zwei Punkte q,r in einem Quadrat
- Dann ist $d(q,r) \leq \delta / 2^{1/2} < \delta$



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) ≥ δ.

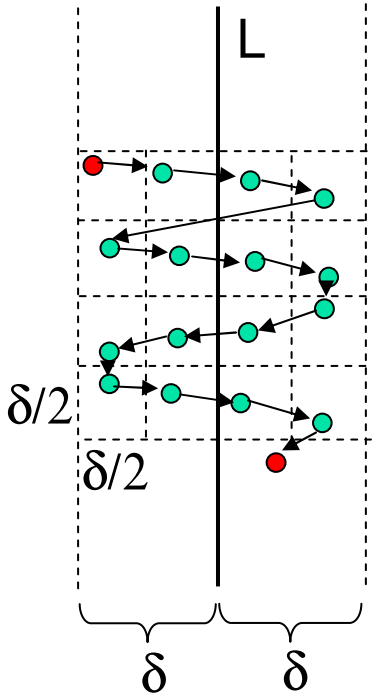
Nächste Paare

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

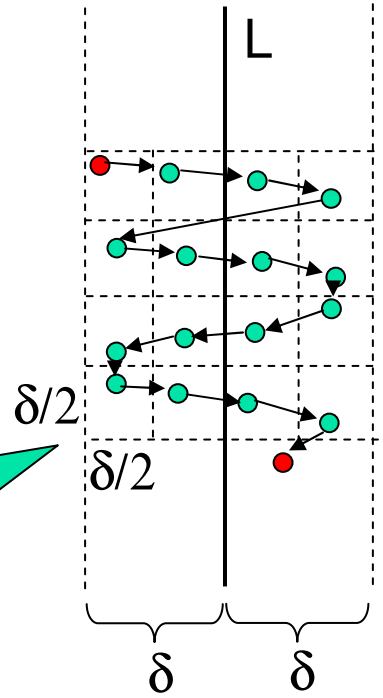
Lemma 1.2

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



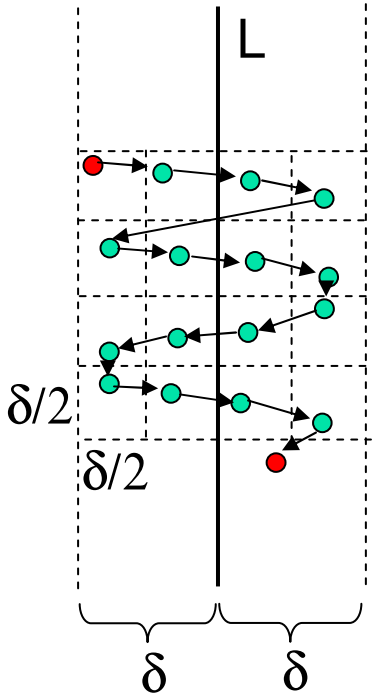
Nächste Paare

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



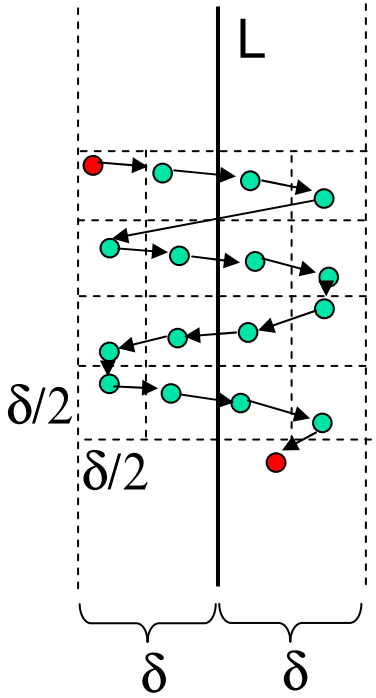
Nächste Paare

Lemma 1.2

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

Beweis:

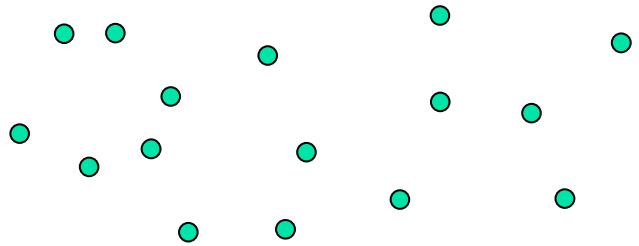
- Seien mind. 15 Punkte zwischen q und r
- Dann sind mindestens 3 Reihen Quadrate zwischen q,r, weil in jedem Quadrat höchstens ein Punkt ist
- Dann muss d(q,r) mindestens $3/2 \cdot \delta > \delta$ sein



Nächste Paare

Der Algorithmus im Überblick:

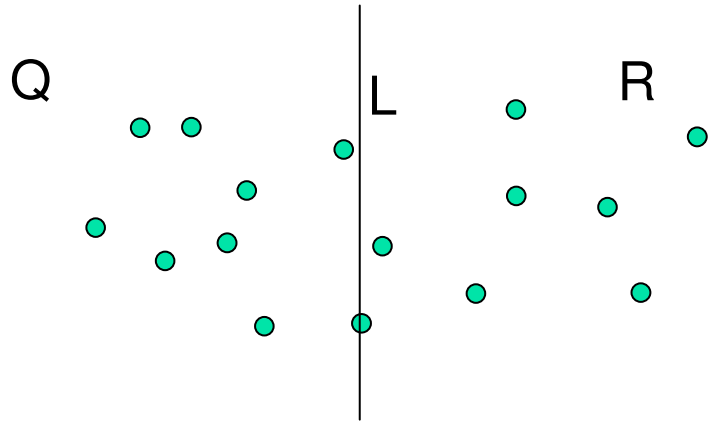
- Teile Punktmenge an Linie L in Q und R auf



Nächste Paare

Der Algorithmus im Überblick:

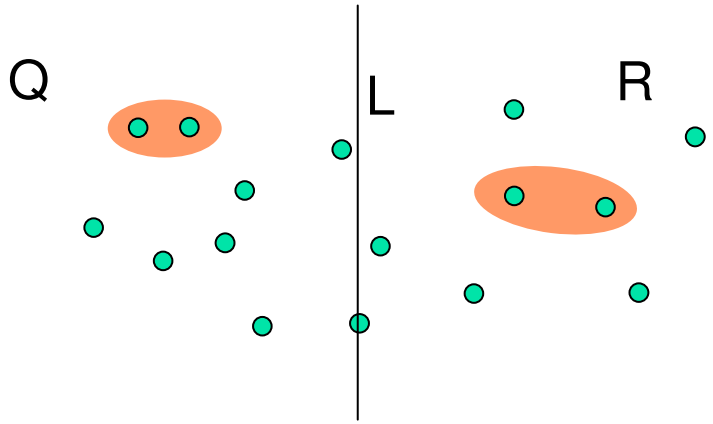
- Teile Punktmenge an Linie L in Q und R auf



Nächste Paare

Der Algorithmus im Überblick:

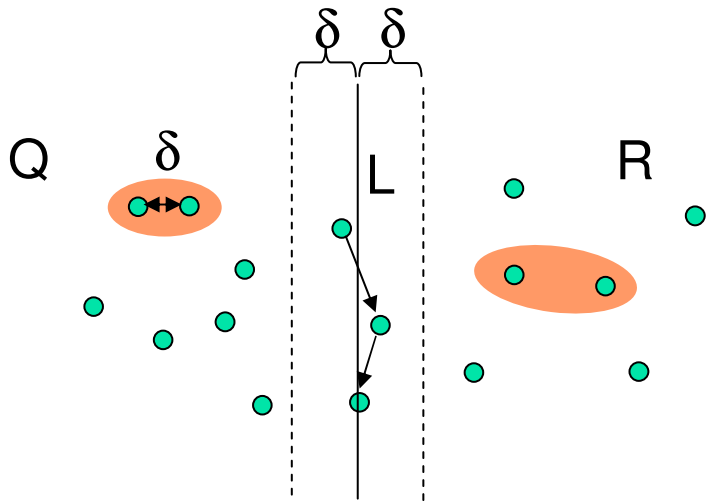
- Teile Punktmenge an Linie L in Q und R auf
- Löse Problem rekursiv auf Q und R



Nächste Paare

Der Algorithmus im Überblick:

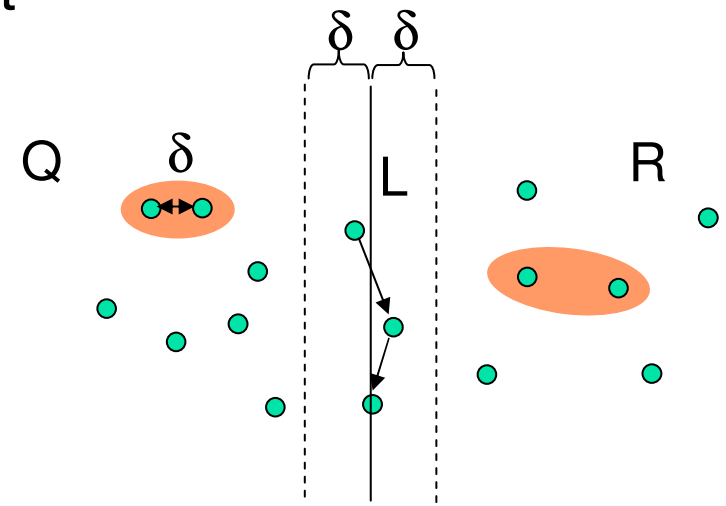
- Teile Punktmenge an Linie L in Q und R auf
- Löse Problem rekursiv auf Q und R
- Sortiere Punkte nach y-Koord.



Nächste Paare

Der Algorithmus im Überblick:

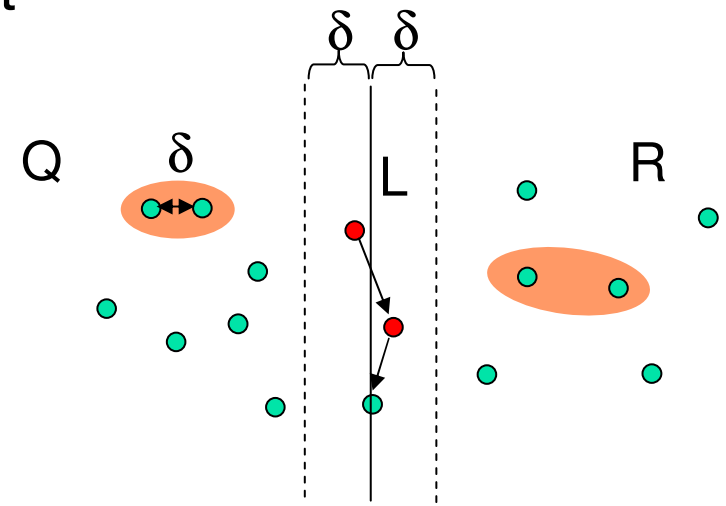
- Teile Punktmenge an Linie L in Q und R auf
- Löse Problem rekursiv auf Q und R
- Sortiere Punkte nach y-Koord.
- Teste alle Paare, deren Abstand in der Sortierung kleiner als 16 ist



Nächste Paare

Der Algorithmus im Überblick:

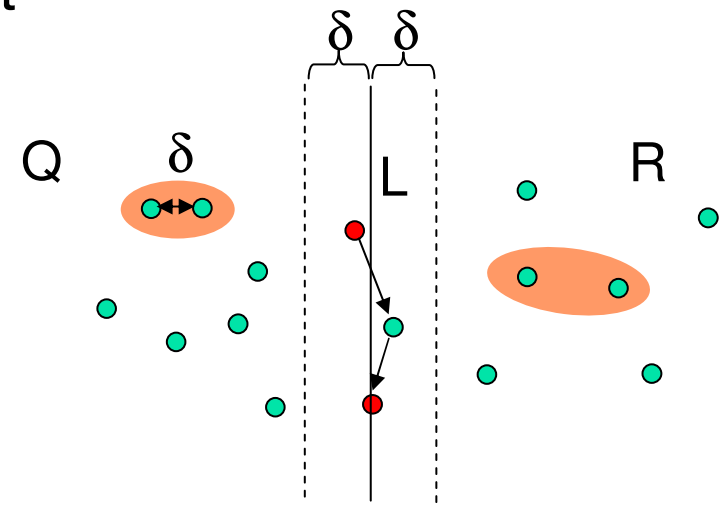
- Teile Punktmenge an Linie L in Q und R auf
- Löse Problem rekursiv auf Q und R
- Sortiere Punkte nach y-Koord.
- Teste alle Paare, deren Abstand in der Sortierung kleiner als 16 ist



Nächste Paare

Der Algorithmus im Überblick:

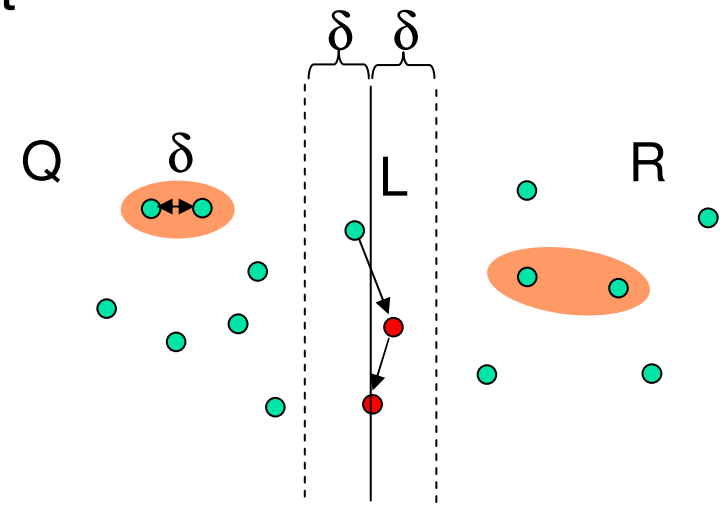
- Teile Punktmenge an Linie L in Q und R auf
- Löse Problem rekursiv auf Q und R
- Sortiere Punkte nach y-Koord.
- Teste alle Paare, deren Abstand in der Sortierung kleiner als 16 ist



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

Nächste Paare

NächstesPaar(P)

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

Nächste Paare

NächstesPaar(P)

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

n=length[P]

Nächste Paare

NächstesPaar(P)

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

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

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. $(q_1^*, q_2^*) \leftarrow$ NächstesPaarRec(P, a, b)
6. $(r_1^*, r_2^*) \leftarrow$NächstesPaarRec(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 ←⌊(a+c)/2⌋
5. (q_1^*, q_2^*) ← NächstesPaarRec(P, a, b)
6. (r_1^*, r_2^*) ←NächstesPaarRec(P, b+1, c)
7. ➢ **Zusammensetzen**

Nächste Paare

NächstesPaarRec(P,a,c)

1. **if** c-a=1 **then retu**
2. **if** c-a=2 **then retu** s
   (P[a],P[a+1]),], P[a+2])
4. b ←⌊(a+c)/2⌋
5. (q_1^*, q_2^*) ← NächstesPaarRec(P, a, b)
6. (r_1^*, r_2^*) ←NächstesPaarRec(P, b+1, c)
7. ➢ **Zusammensetzen**

Berechnung des mittleren Elements (Median)

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-
4. b ←⌊(a+c)/2⌋
5. $(q_1^*, q_2^*) \leftarrow$ NächstesPaarRec(P, a, b)
6. $(r_1^*, r_2^*) \leftarrow$NächstesPaarRec(P, b+1, c)
7. ➢ **Zusammensetzen**

Rekursiver Aufruf für linke Hälfte

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 ←⌊(a+c)/2⌋
5. (q_1^*, q_2^*) ← NächstesPaarRec(P
6. (r_1^*, r_2^*) ←NächstesPaarRec(P, b+1, c)
7. ➢ **Zusammensetzen**

Rekursiver Aufruf für rechte Hälfte

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 ←⌊(a+c)/2⌋
5. (q_1^*, q_2^*) ← NächstesPaarRec(P, a, b)
6. (r_1^*, r_2^*) ←NächstesPaarRec(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 ←⌊(a+c)/2⌋
5. (q_1^*, q_2^*) ← NächstesPaarRec(P, a, b)
6. (r_1^*, r_2^*) ←NächstesPaarRec(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. $(q,r) \leftarrow (P'[1], P'[length(P')])$
7. **for** $i \leftarrow 1$ **to** length(P') **do**
8. **for** $j \leftarrow 1$ **to** 15 **do**
9. **if** $i+j \leq$ 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 ← 1
3. **for** i ← a **to** c **do**
4. **if** d(P[i],L) < δ **then**
5. P‘[j] ← P[i]; j ← j+1
6. (q,r)← (P‘[1], P‘[length(P‘)])
7. **for** i ← 1 **to** length(P‘) **do**
8. **for** j ←1 **to** 15 **do**
9. **if** i+j ≤ length(P‘) **then**
10. **if** d(P‘[i],P‘[i+j])<d(q,r) **then** q ← P‘[i]; r ← P‘[i+j]
11. **return** nächstes Paar aus (q,r), (q_1*,q_2*), und (r_1*,r_2*)

Nächste Paare

Zusammensetzen

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

```
1.   Merge(P,a,b,c)    ➢ Sortiere
2.   j ← 1
3.   for i ← a to c do
4.       if d(P[i],L) < δ then
5.           P‘[j] ← P[i]; j ← j+1
6.   (q,r)← (P‘[1], P‘[length(P‘)])
7.   for i ← 1 to length(P‘) do
8.     for j ←1 to 15 do
9.       if i+j ≤ length(P‘) then
10.        if d(P‘[i],P‘[i+j])<d(q,r) then q ← P‘[i]; r ← P‘[i+j]
11. return nächstes Paar aus (q,r), (q1*,q2*), und (r1*,r2*)
```

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. $(q,r) \leftarrow (P'[1], P'[length(P')])$
7. **for** $i \leftarrow 1$ **to** length(P') **do**
8. **for** $j \leftarrow 1$ **to** 15 **do**
9. **if** $i+j \leq$ 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. $(q,r) \leftarrow (P'[1], P'[length(P')])$
7. **for** $i \leftarrow 1$ **to** length(P') **do**
8. **for** $j \leftarrow 1$ **to** 15 **do**
9. **if** $i+j \leq$ 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 ← 1
3. **for** i ← a **to** c **do**
4. **if** d(P[i],L) < δ **then**
5. P‘[j] ← P[i]; j ← j+1
6. (q,r)← (P‘[1], P‘[length(P‘)])
7. **for** i ← 1 **to** length(P‘) **do**
8. **for** j ←1 **to** 15 **do**
9. **if** i+j ≤ length(P‘) **then**
10. **if** d(P‘[i],P‘[i+j])<d(q,r) **then** q ← P‘[i]; r ← 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. $(q,r) \leftarrow (P'[1], P'[length(P')])$
7. **for** $i \leftarrow 1$ **to** length(P') **do**
8. **for** $j \leftarrow 1$ **to** 15 **do**
9. **if** $i+j \leq$ 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

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

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

$n = 2^i$

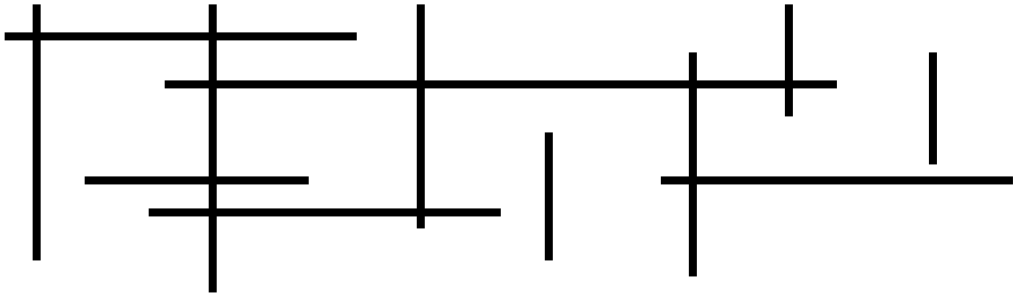
$i = 1$: ok

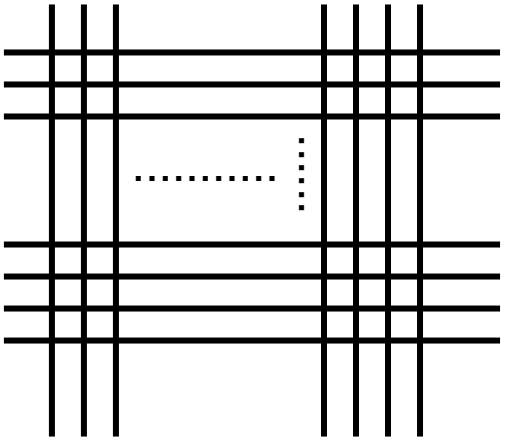
$i > 1$

$T(2^i) \; =$

Segmentschnittproblem

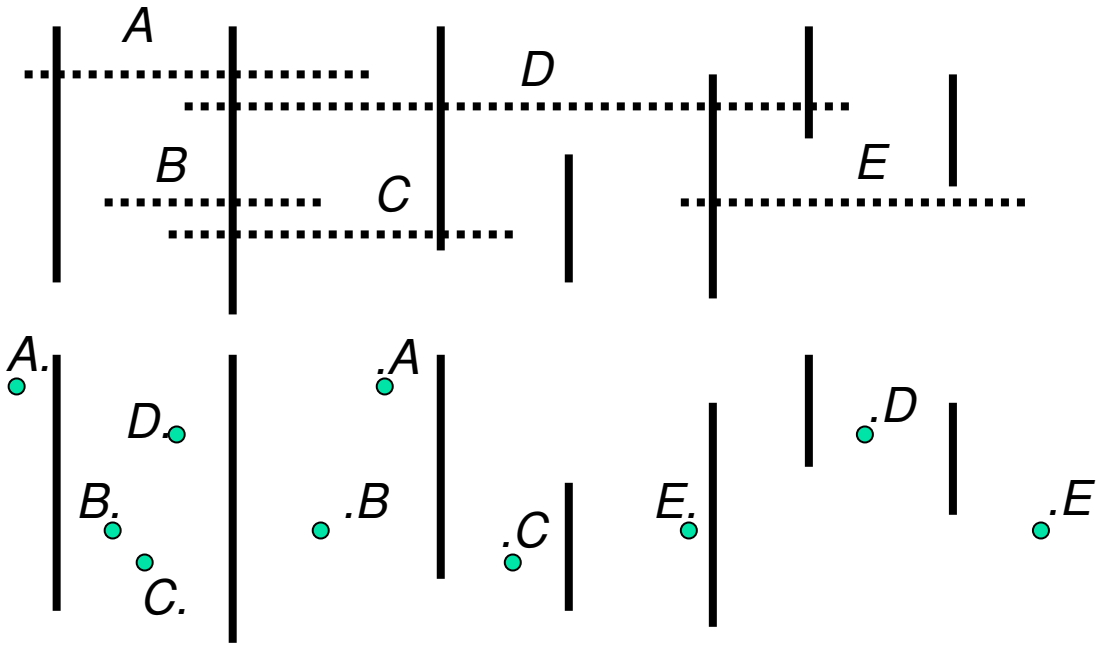
Bestimme alle Paare sich schneidender Segmente





Segmentschnittproblem

Bestimme alle Paare sich schneidender Segmente



Die getrennte Repräsentation der Segmente erlaubt eine Aufteilung

ReportCuts

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

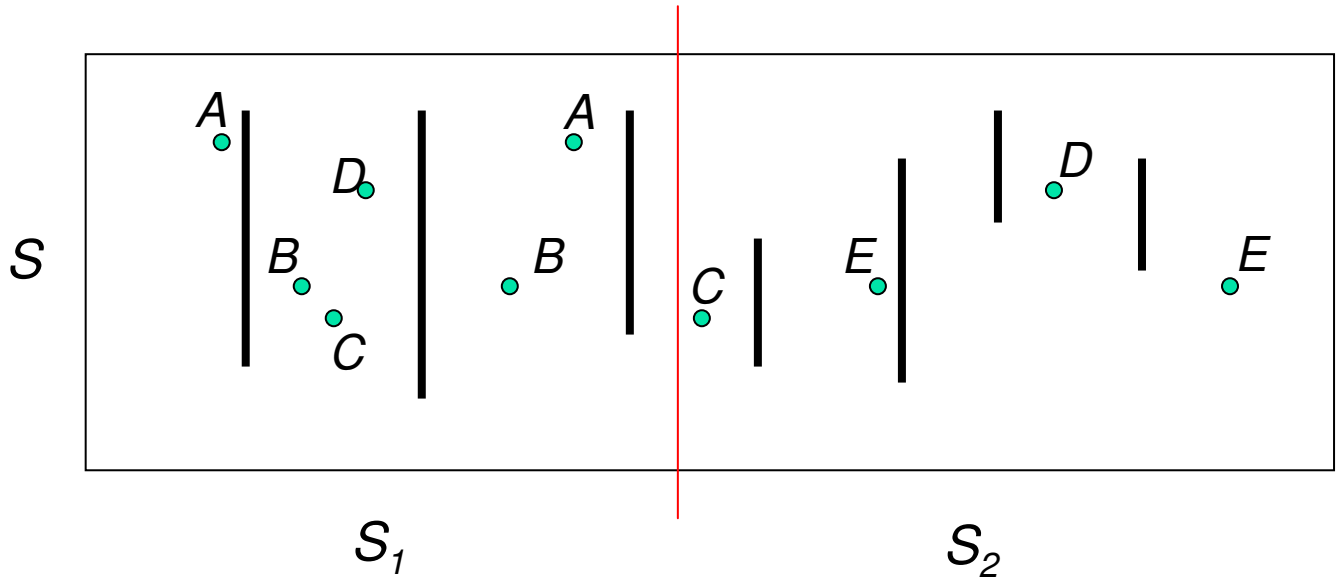
if $|S| > 1$

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

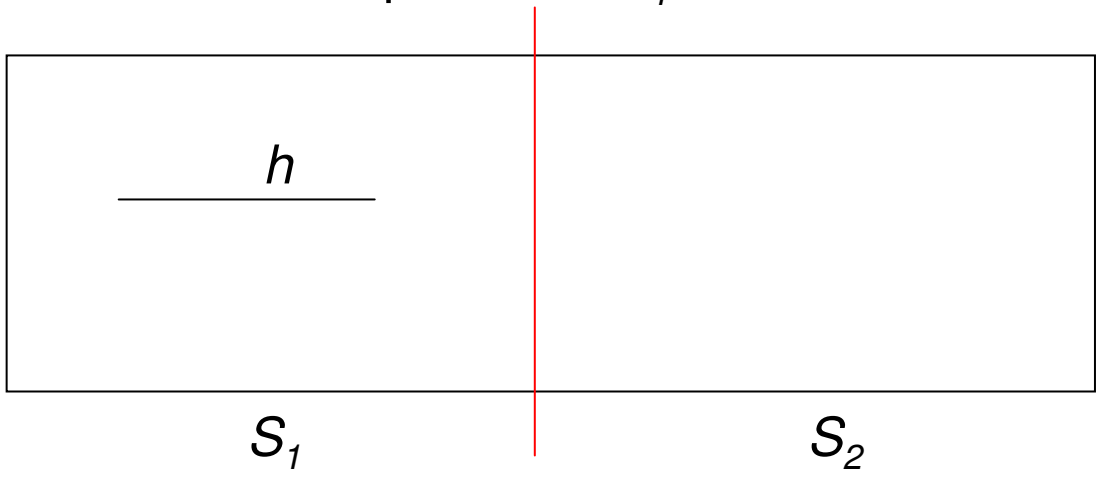
ReportCuts(S_1); ReportCuts(S_2)

ReportCuts

3. Merge: ???

Mögliche Schnitte für ein horizontales Segment in S_1

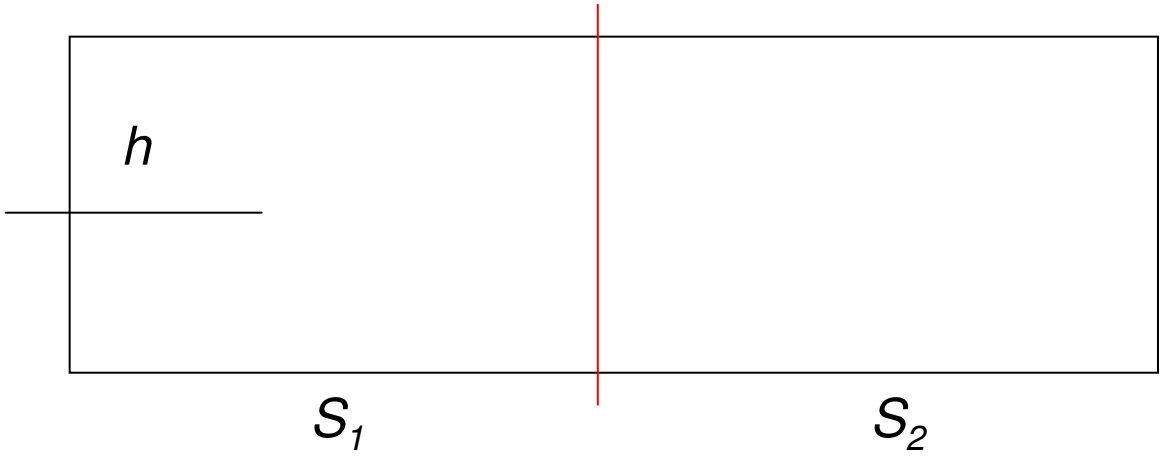
Fall 1: beide Endpunkte in S_1



ReportCuts

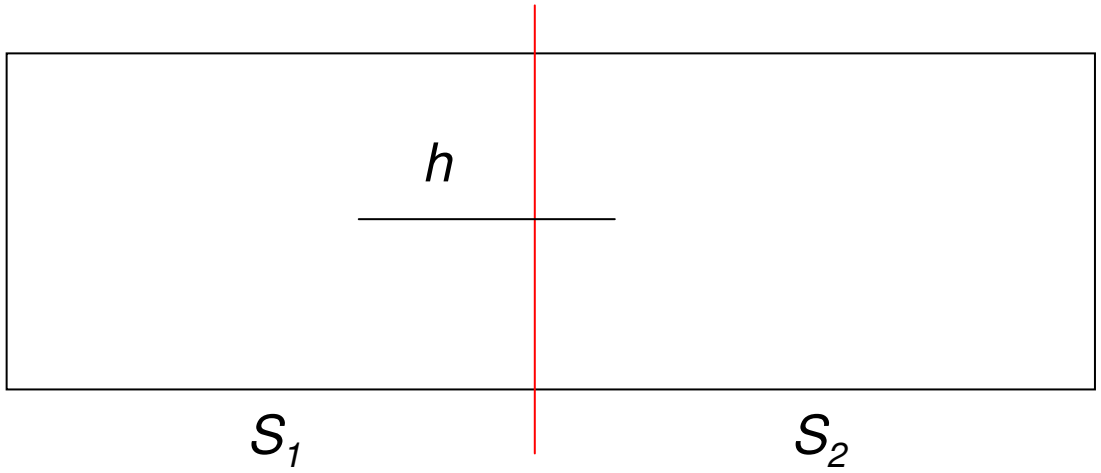
Fall 2: nur ein Endpunkt von *h* in S_1

2 a) rechter Endpunkt in S_1

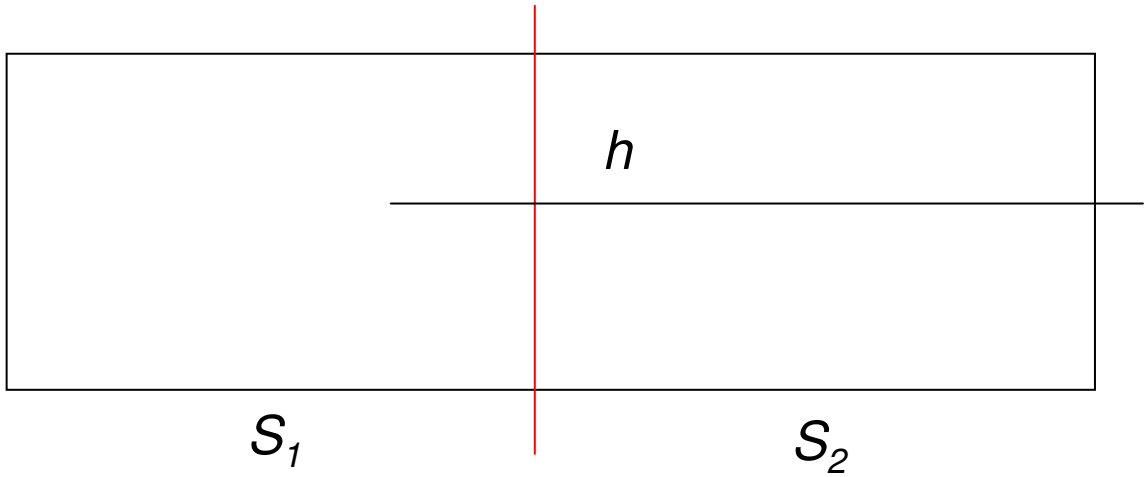


ReportCuts

2 b) linker Endpunkt von h in S_1



rechter Endpunkt
in S_2

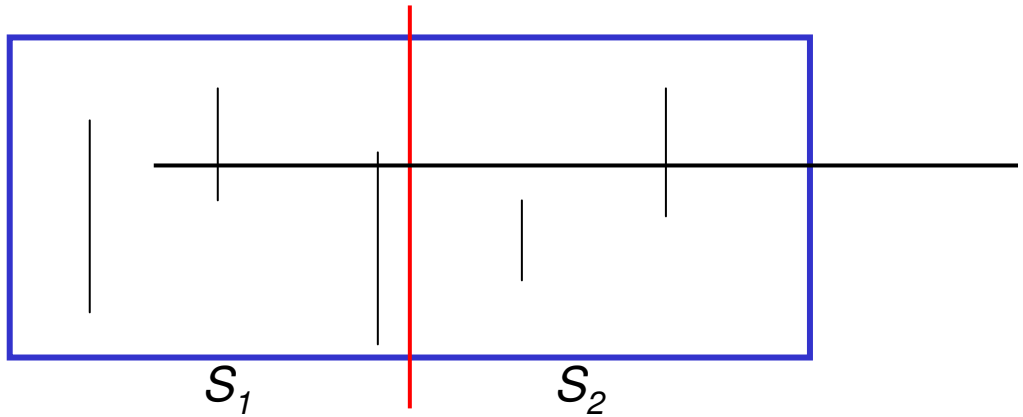


rechter Endpunkt
nicht in S_2

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

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

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

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

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

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

Merge-Schritt

$L(S_i)$, $R(S_i)$, $V(S_i)$ $i=1,2$ seien berechnet $S = S_1 \cup S_2$

$L(S)$ =

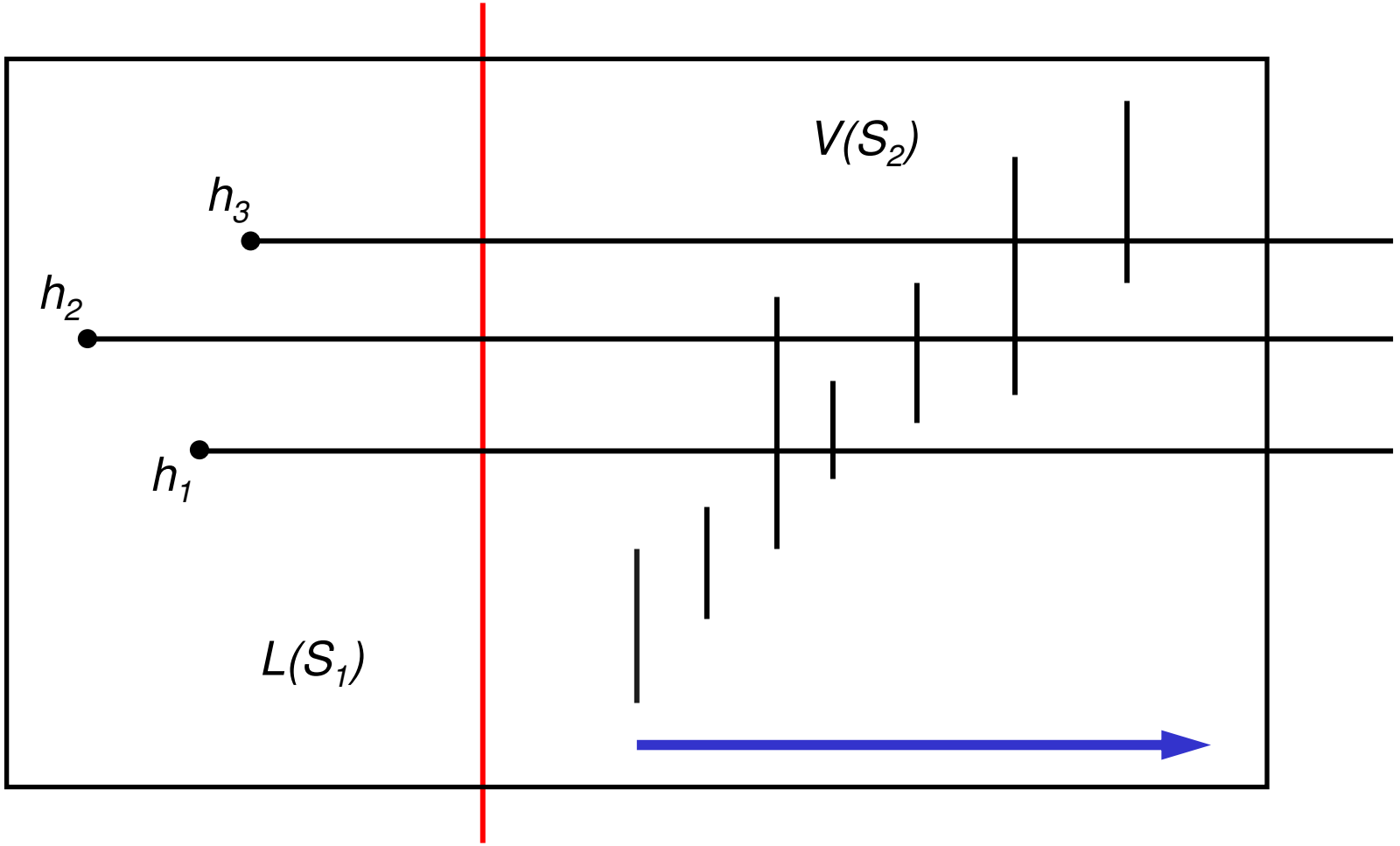
$R(S)$ =

$V(S)$ =

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

V: sortiert nach steigenden unteren Endpunkten
verkettete Liste

Ausgabe der Schnittpunkte



Laufzeit

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

Divide-and-Conquer:

$T(n) = 2T(n/2) + an$ + Größe der Ausgabe
$T(1) = O(1)$

$O(n \log n + k)$ k = #Schnittpunkte