Algorithmentheorie

01 - Einleitung

Prof. Dr. S. Albers

Organisatorisches

Vorlesung: Mo 14.00 – 16.00 Uhr, HS 00-026, Geb.101
Do 14.00 – 16.00 Uhr, HS 00-036, Geb.101
alle 14 Tage

Übungen: alle 14 Tage
Anmeldung bis 28. Oktober 2005, 12 Uhr
- Mo 11-13, SR 00-006, Geb. 051
- Mo 11-13, SR 00-034, Geb. 051
- Mo 16-18, SR 03-026, Geb. 051
- Mi 9-11, SR 00-006, Geb. 051
- Fr 11-13, SR 00-006, Geb. 051
- Fr 11-13, SR 03-036, Geb. 051

http://www.informatik.uni-freiburg.de/~ipr/
→ Teaching → Algorithmentheorie

Organisatorisches

Klausur: 14. März 2006

Teilnahmevoraussetzung:
- 50% der Übungen bearbeitet
- 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

Orginalliteratur

Genauere Angaben zum Thema werden jeweils im Web veröffentlicht

Algorithmen und Datenstrukturen

Entwurfsprinzipien und Analysetechniken für Algorithmen

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

Algorithmen und Datenstrukturen

Problem-/Anwendungsbereiche

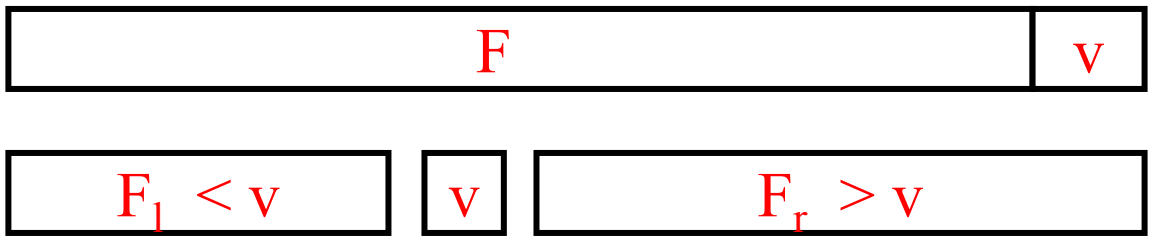
- Geometrische Algorithmen
- Algebraische Algorithmen
- Graphenalgorithmen
- Datenstrukturen
- Internet-Algorithmen
- Optimierungsverfahren
- Zeichenkettenverarbeitung

Divide and Conquer

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 $n_1,...,n_k$ 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**

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

Spezialfall: $k = 2$, $n_1 = n_2 = n/2$

Divide- und Mergeaufwand : *DM(n)*

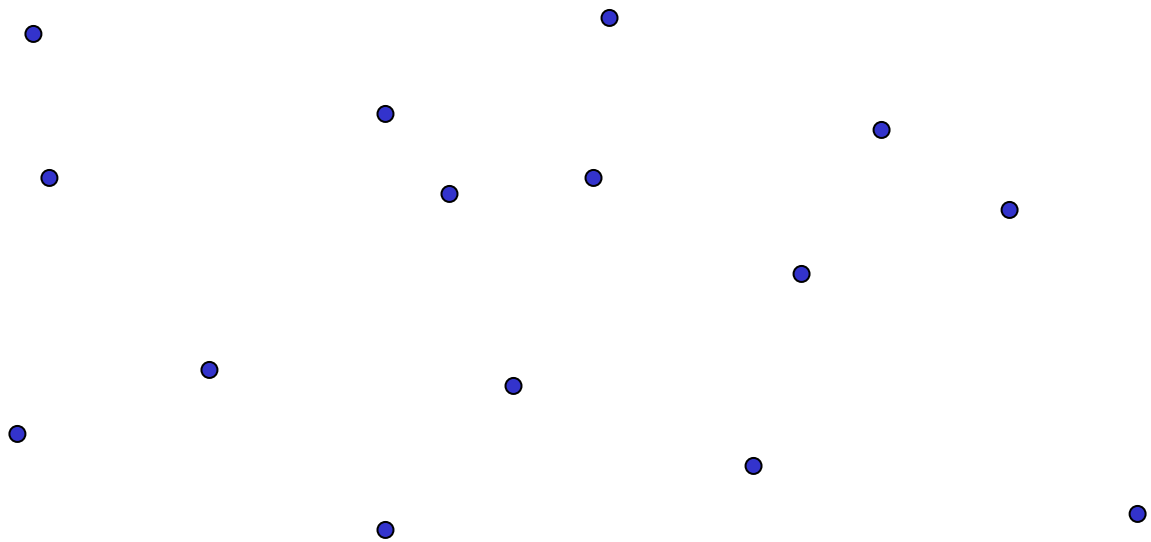
$T(1) = a$

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

Geometrisches Divide - and - Conquer

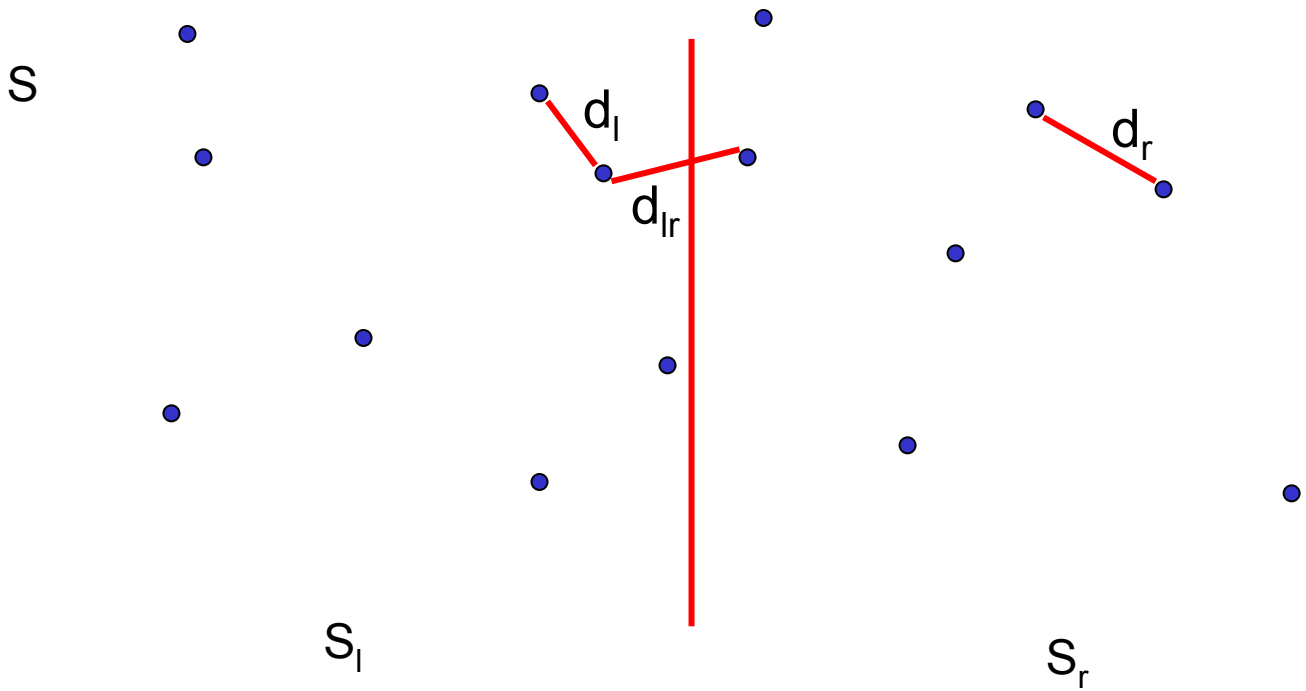
Closest Pair Problem:

Bestimme für eine Menge *S* von *n* Punkten ein Paar mit minimaler Distanz



Divide-and-Conquer Ansatz

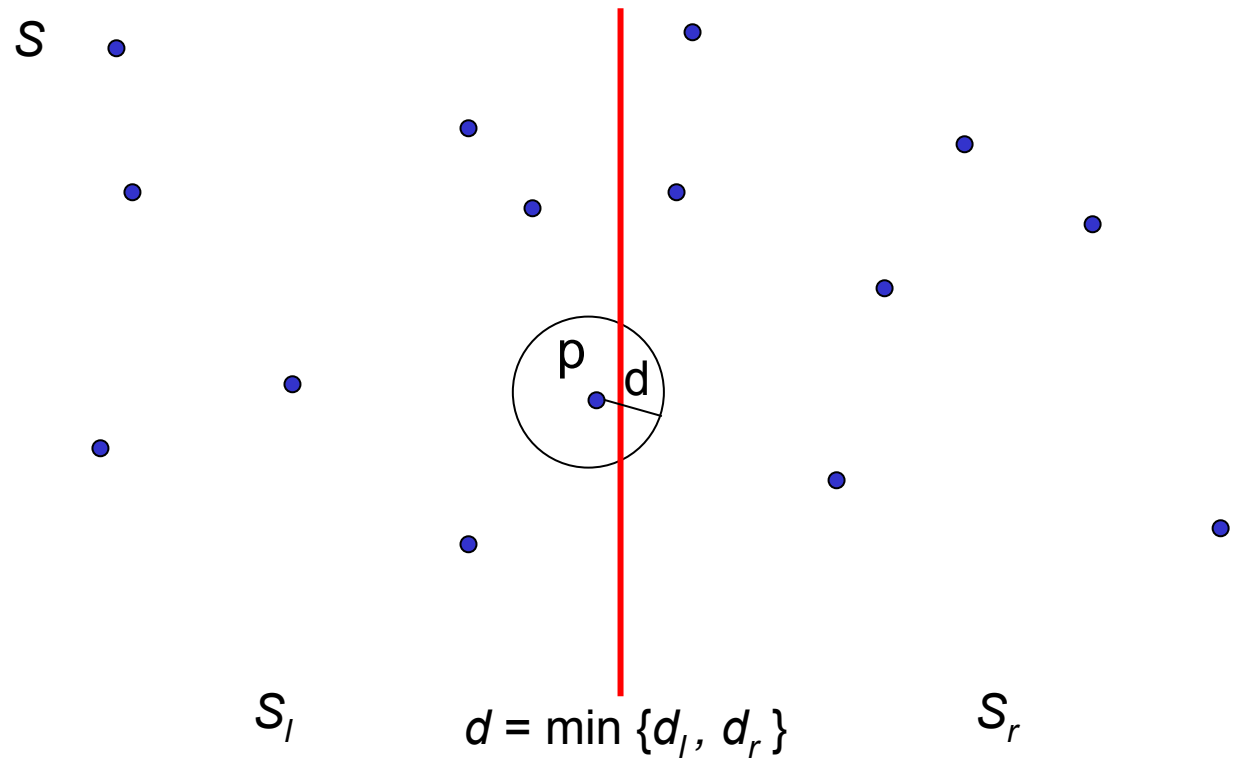
1. **Divide :** Teile S in zwei gleichgroße Mengen S_l und S_r
2. **Conquer:** $d_l = mindist(S_l)$ $\quad d_r = mindist(S_r)$
3. **Merge:** $d_{lr} = \min\{d(p_l, p_r) \mid p_l \in S_l, p_r \in S_r\}$
   return min $\{d_l, d_r, d_{lr}\}$



Divide-and-Conquer Ansatz

1. **Divide :** Teile S in zwei gleichgroße Mengen S_l und S_r
2. **Conquer:** $d_l = mindist(S_l)$ $\quad d_r = mindist(S_r)$
3. **Merge:** $d_{lr} = \min\{d(p_l, p_r) \mid p_l \in S_l, p_r \in S_r\}$
   return min $\{d_l, d_r, d_{lr}\}$

Berechnung von d_{lr}:

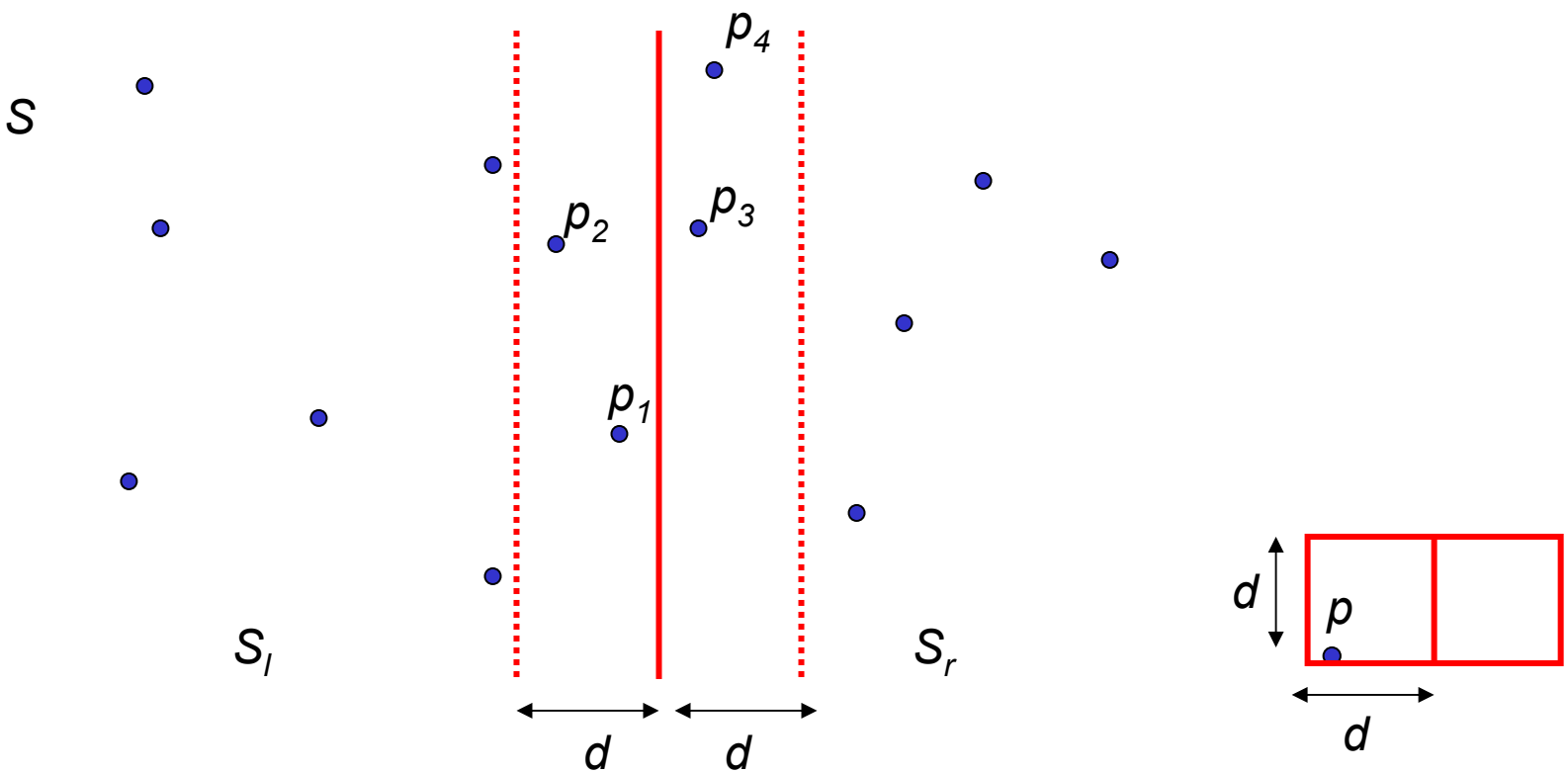


Merge - Schritt

1. Betrachte nur Punkte im Abstand *d* von der Mittellinie, gemäß aufsteigenden y-Koordinaten

2. Betrachte zu jedem Punkt *p* alle Punkte *q* mit y-Abstand höchstens *d*; es gibt maximal 7 solche Punkte

Merge - Schritt



$d = \min \{ d_l, d_r \}$

Implementierung

- Sortiere Eingabepunkte anfangs einmal gemäß steigenden x-Koordinaten $O(n \log n)$

- Sind Teilprobleme S_l , S_r gelöst, so erzeuge Liste der Punkte in S gemäß steigenden y-Koordinaten (Mergesort)

Laufzeit (Divide-and-Conquer)

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

- Rate Lösung durch wiederholtes Einsetzen
- Verifiziere durch Induktion

Lösung: *O(n log n)*

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

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

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

$n = 2^i$

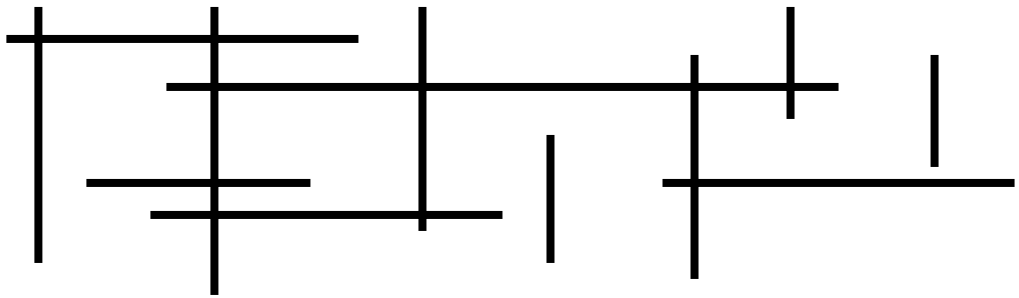
$i = 1$: ok

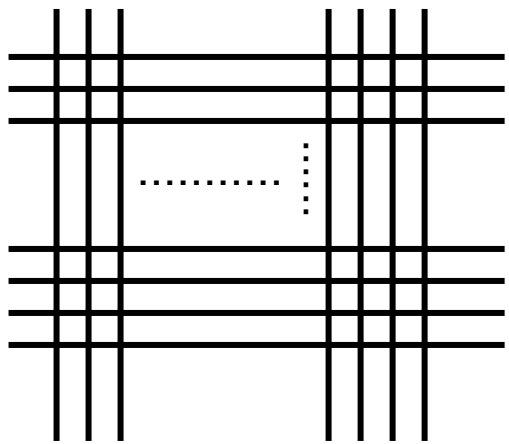
$i > 1$

$T(2^i) \; =$

Segmentschnittproblem

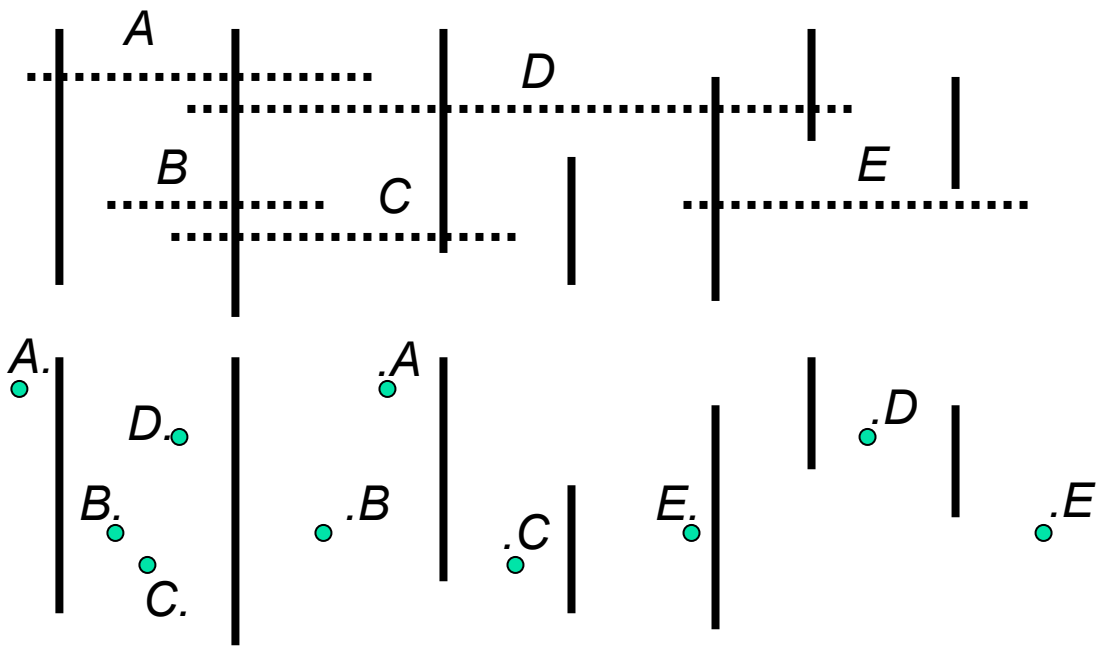
Bestimme alle Paare sich schneidender Segmente





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

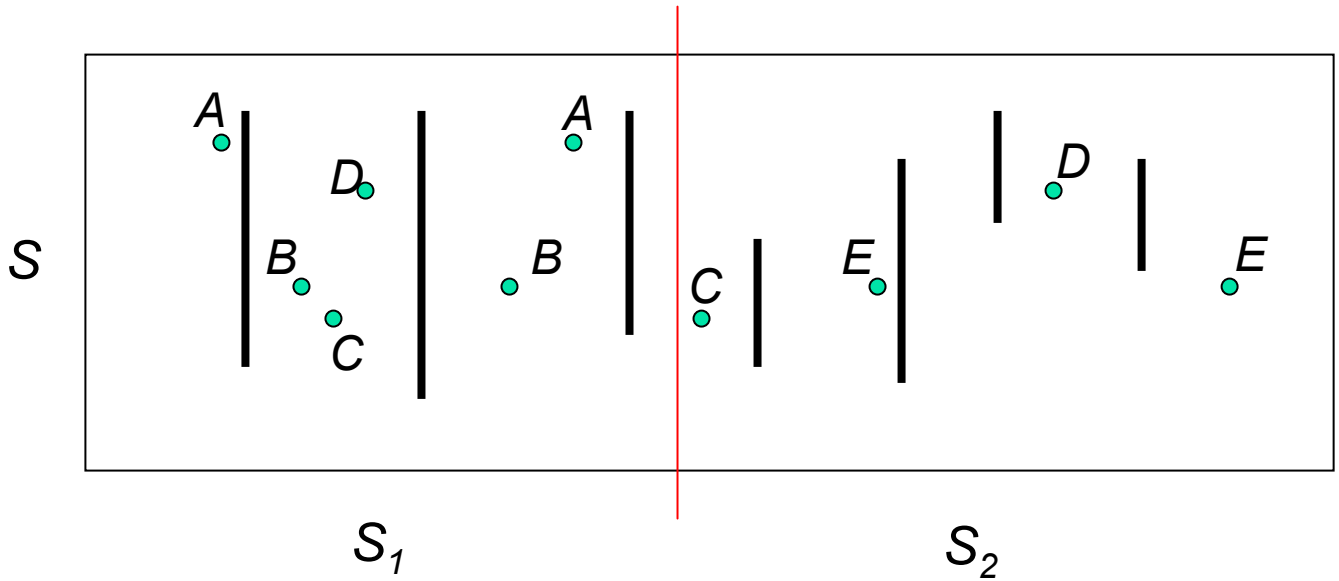
if $|S| > 1$

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



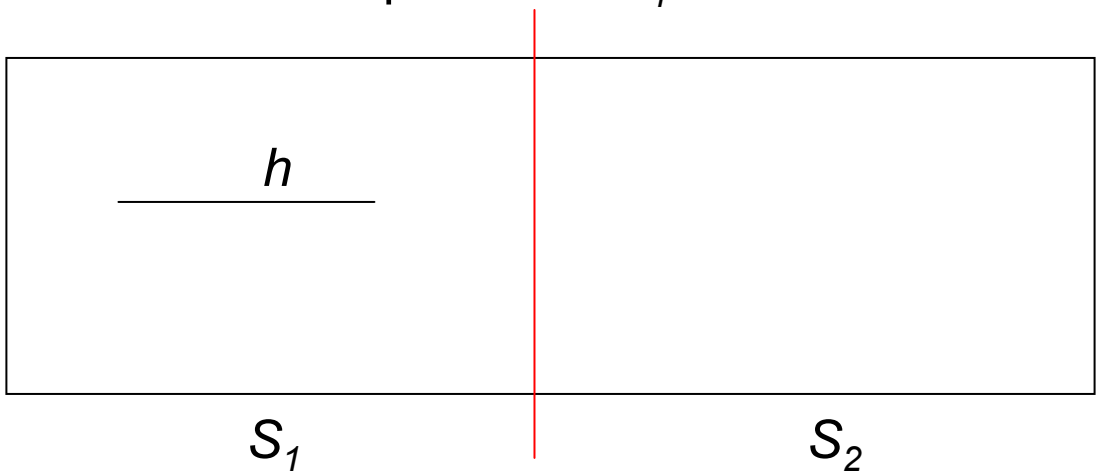
2. Conquer

ReportCuts(S_1); ReportCuts(S_2)

3. Merge: ???

Mögliche Schnitte für ein horizontales Segment in S_1

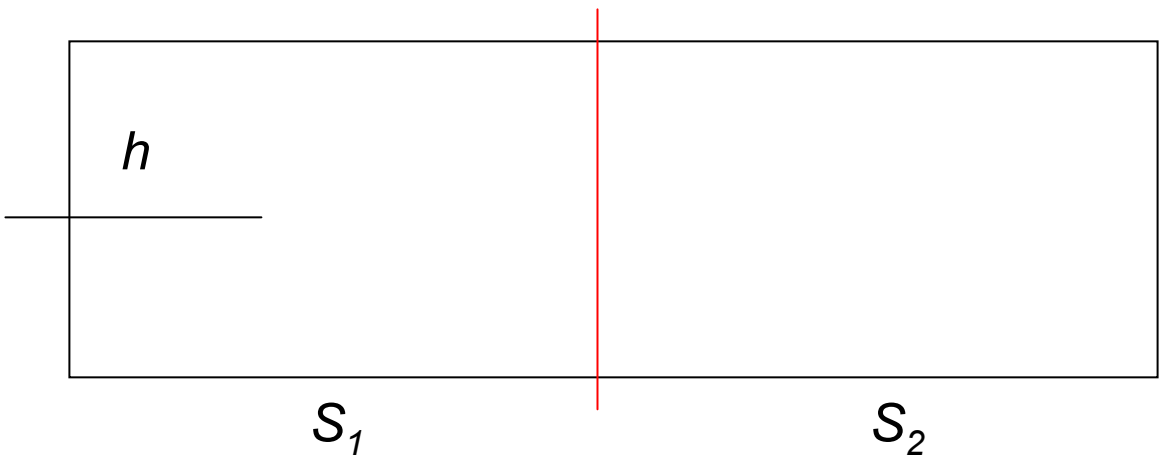
Fall 1: beide Endpunkte in S_1



ReportCuts

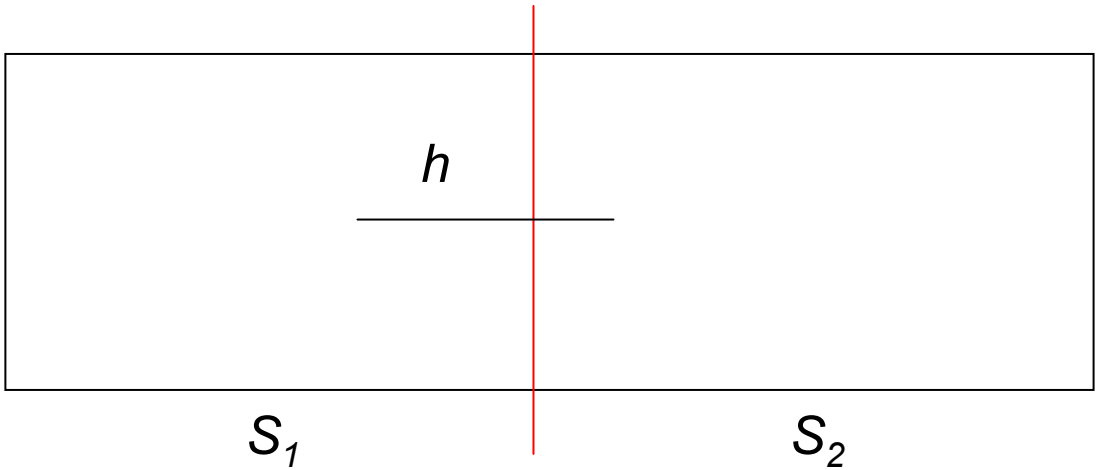
Fall 2: nur ein Endpunkt von *h* in S_1

2 a) rechter Endpunkt in S_1

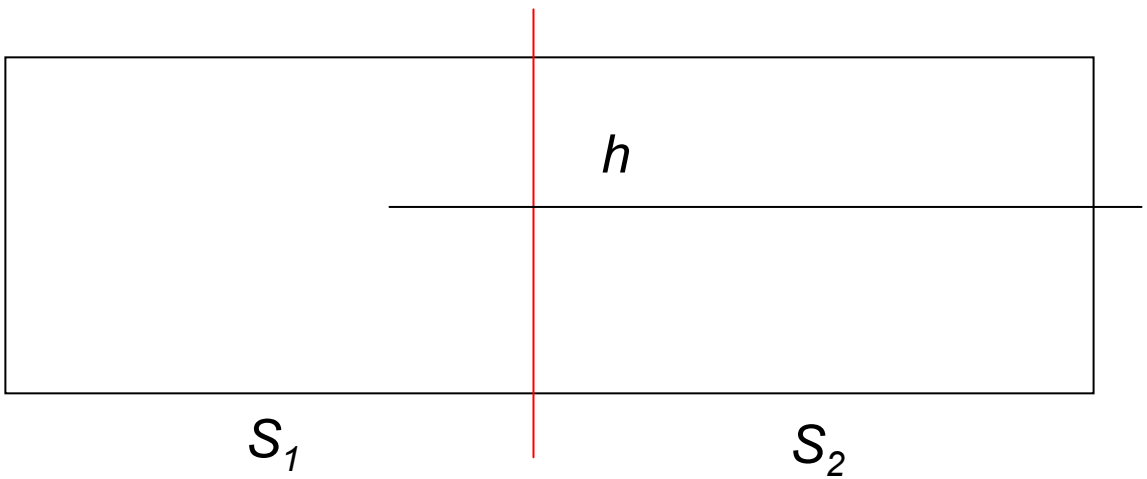


ReportCuts

2 b) linker Endpunkt von *h* in S_1



rechter Endpunkt in S_2

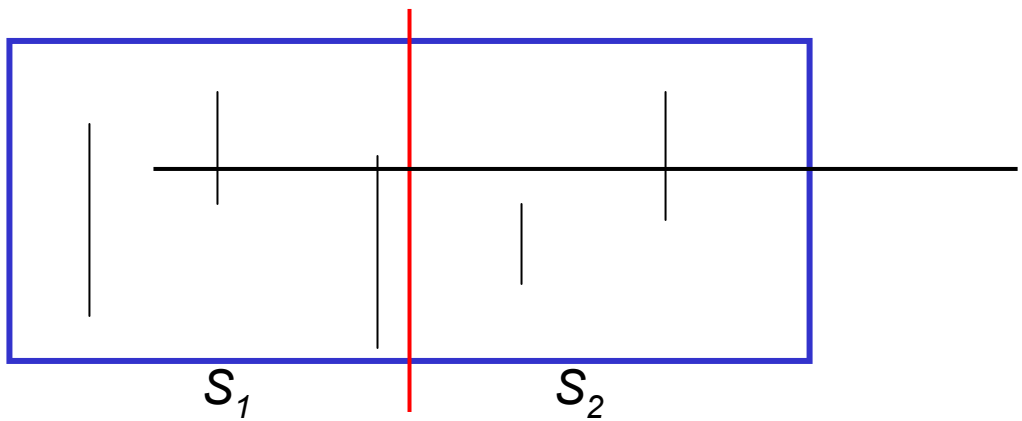


rechter Endpunkt nicht in S_2

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

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

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

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

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

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

Merge-Schritt

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

$L(S)$ =

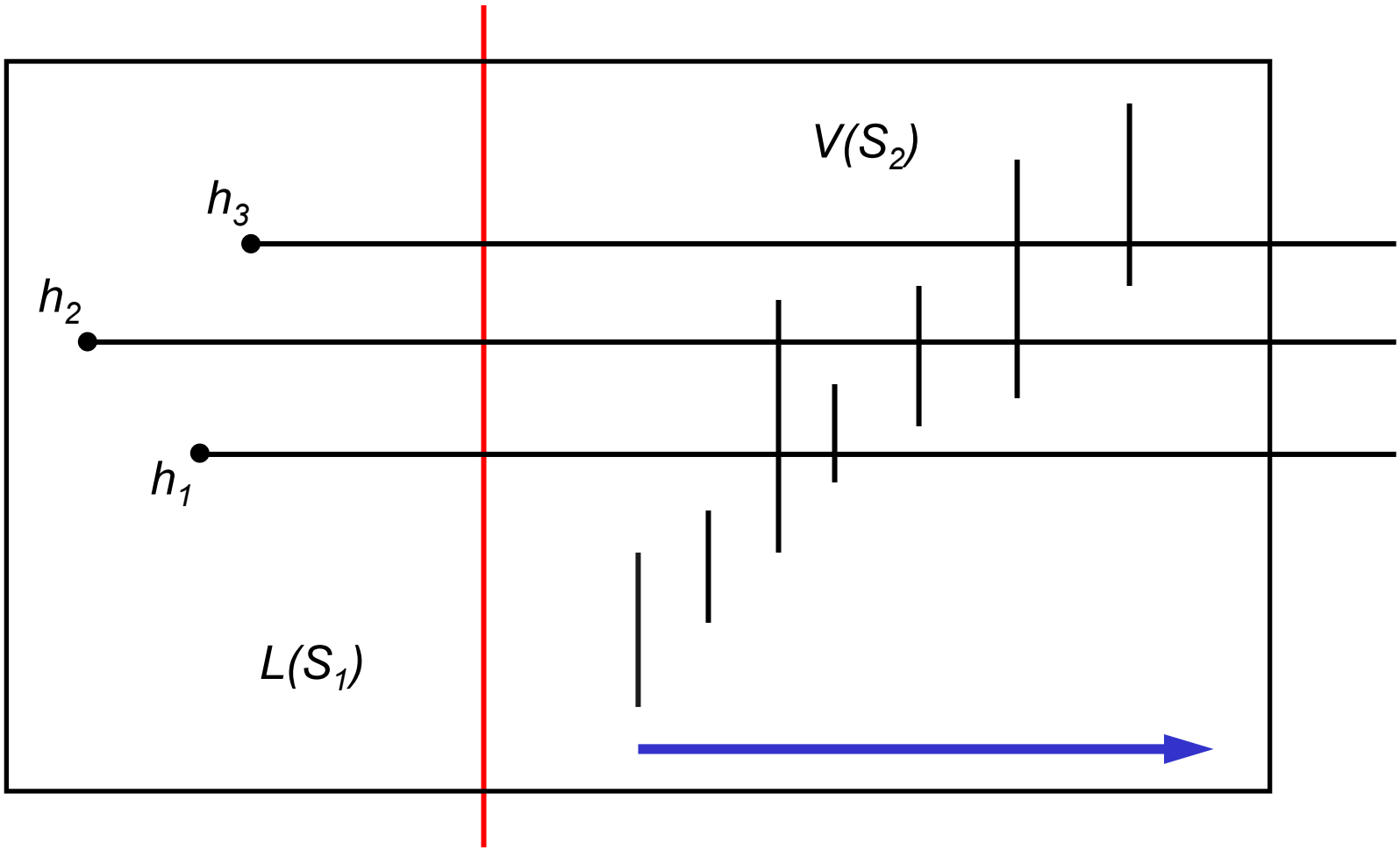
$R(S)$ =

$V(S)$ =

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:

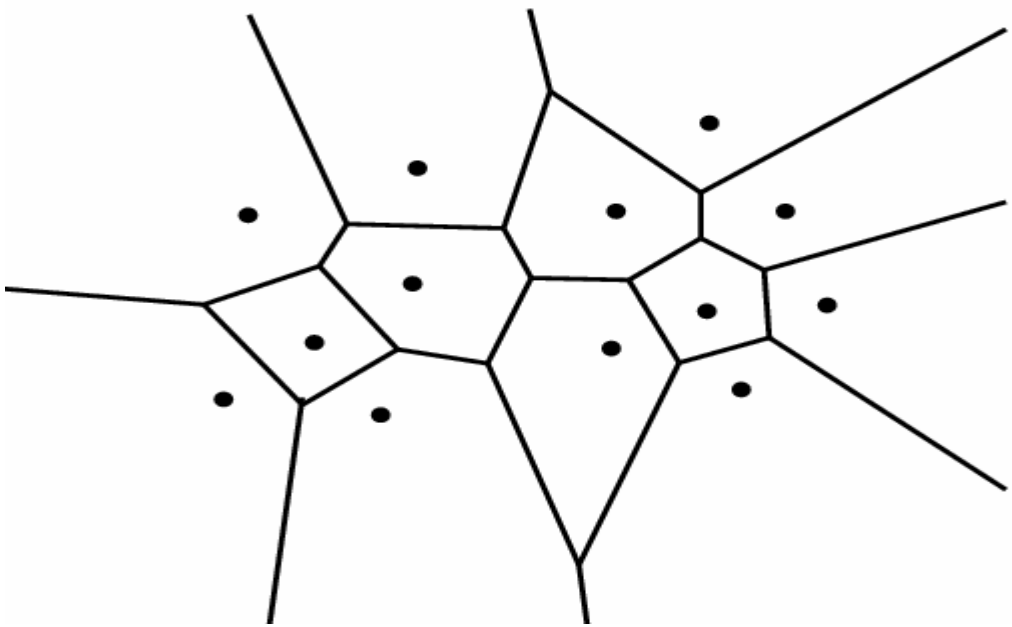
$T(n) = 2T(n/2) + an$ + Größe der Ausgabe
$T(1) = O(1)$

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

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

Voronoi-Region von *p*

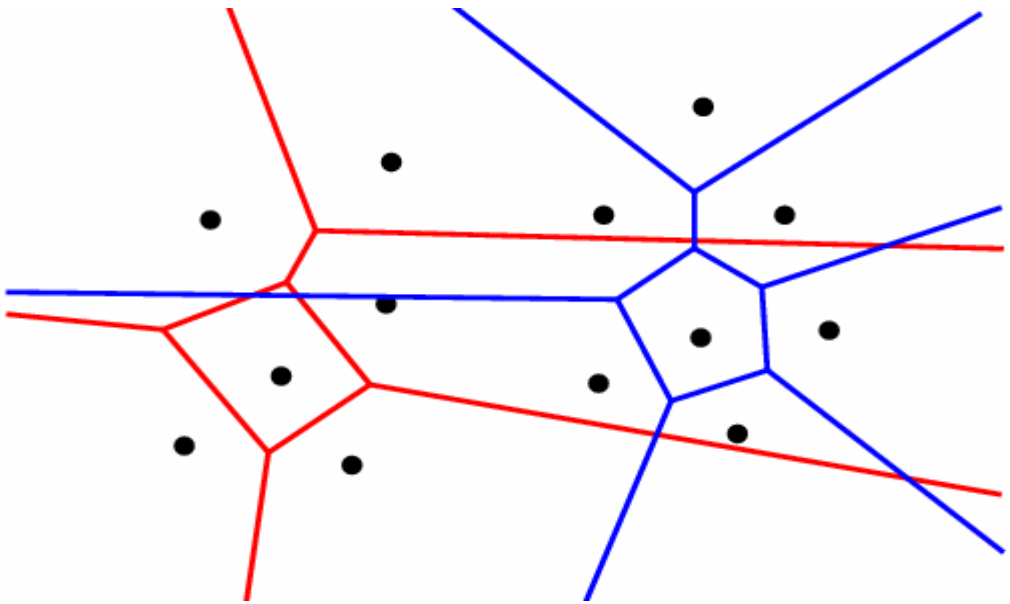
$$VR(p) = \bigcap_{p' \in P \setminus \{p\}} H(p \mid p')$$

Berechnung des Voronoi-Diagramms

Divide : Einteilung der Menge der Orte in zwei Hälften

Conquer: Rekursive Berechnung der beiden kleineren Voronoi-Diagramme

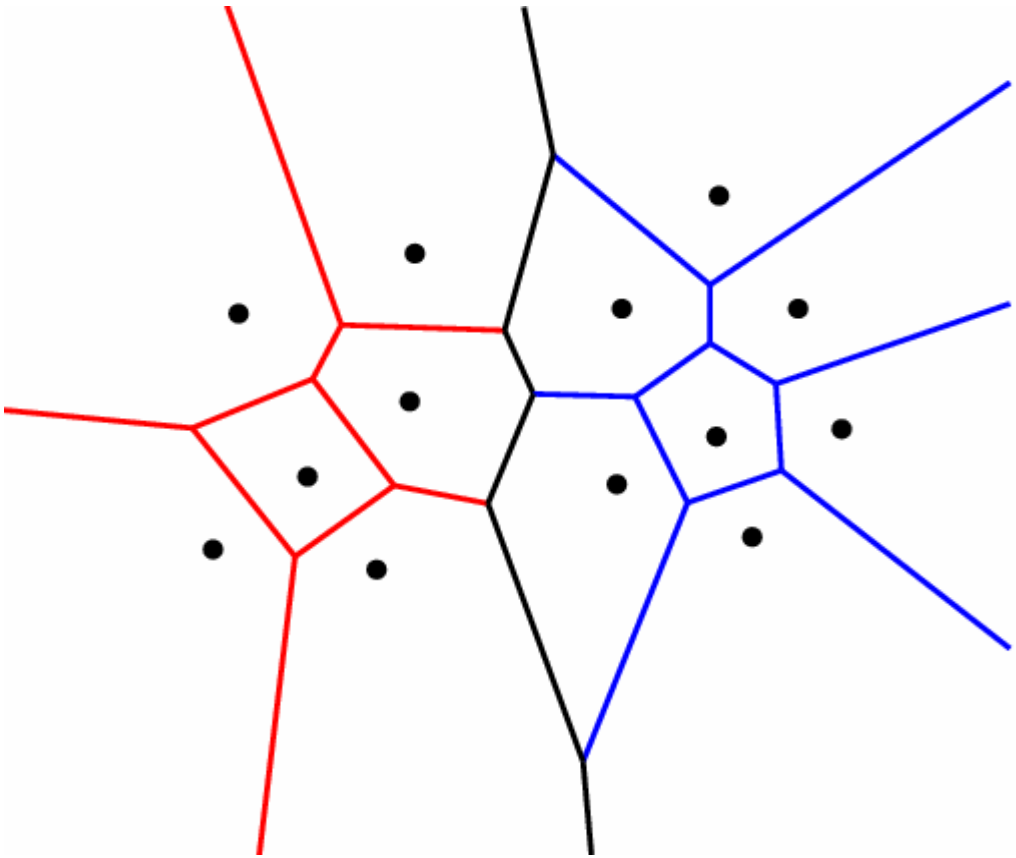
Abbruchbedingung: Voronoi-Diagramm eines einzelnen Ortes ist die gesamte Ebene



Merge: Verbindung der Diagramme durch einen Kantenzug

Berechnung des Voronoi-Diagramms

Ergebnis: Das fertige Voronoi-Diagramm



Laufzeit: Bei *n* gegebenen Orten *O(n log n)*