

14.1 Vorbemerkungen

14.2 Das Scan-line Prinzip

14.3 Geometrisches Divide-and-Conquer

14.4 Punkt-in-Polygon-Problem

14.5 Konstruktion von Polygonen

14.6 Konvexe Hülle

14.7 Ballung und nächstes Paar



Innensicht

Algorithmenentwurf und Algorithmen (incl. Aufwände)

- Suche und Sortieren (Reihen, Folgen, Bäume)
- Graphen
- Dyn. Programmieren
- Geometr. Algorithmen

Prozesse

- Prozesse
- Prozesskommunikation
- Parallelität/Synchronisation
- Java: Thread-Programmierung

Außensicht

Skalierbarkeit und Persistenz

- Mengenorientierung
- Assoziativer Zugriff, Schlüsselbegriff
- Aufwände
- Polymorphie
- Schema
- Deskr. Sprachen (SQL)

Autonome Dienste

- Enge/lose Kopplung
- Protokolle / Automaten
- Verteilung



Einige einfache Definitionen

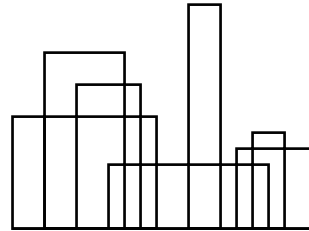
- Punkt: n -Tupel im n -dimensionalen Raum (n Koordinaten)
 - In diesem Kapitel i.d.R. ganzzahlige Koordinaten und zweidimensionaler Raum
- Gerade: Linie durch zwei beliebige Punkte
- Strecke: Linie durch ihre beiden Endpunkte
- Streckenzug: Folge von Strecken (Kanten), wobei der Endpunkt (Knoten) einer Strecke der Anfangspunkt der nächsten Strecke ist
- Polygon: (oder geschlossener Streckenzug) wenn Anfangspunkt der ersten Strecke gleich Endpunkt der letzten Strecke.
- innen: Ein Polygon, dessen Kanten sich nicht kreuzen, umrahmt eine Region, deren Punkte innen sind
- konvex: Polygon ist konvex, wenn jede Strecke zwischen zwei Punkten in seinem Inneren komplett im Inneren liegt
- Schreibweise: Punkt (x,y) , Strecke $(u,v)-(x,y)$



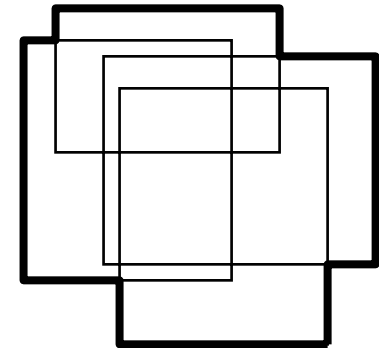
Schnitte vieler horizontaler und vertikaler Strecken

Motivation

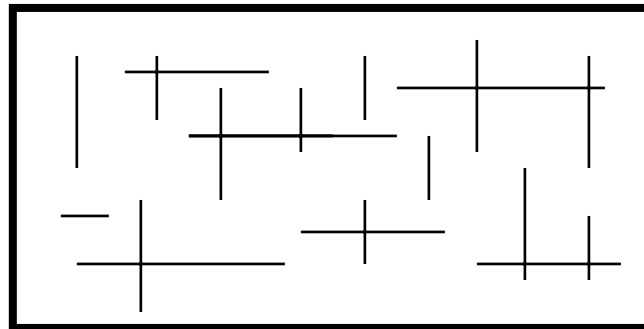
- Berechnen einer Skyline



- Berechnen der Form überlagerter Rechtecke



- VLSI-Entwurf



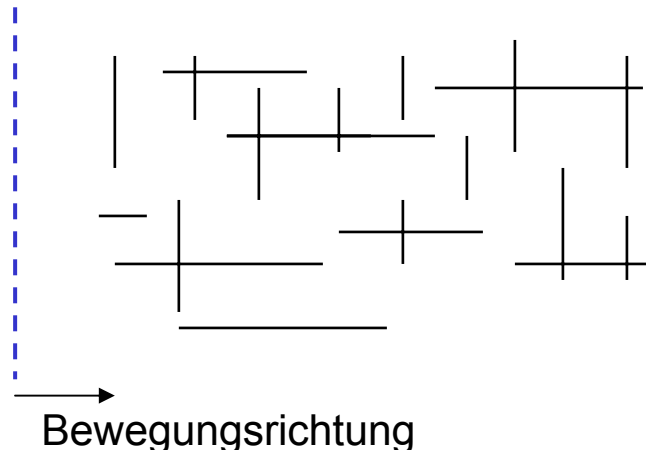
→ Wird auch als Plane-Sweep bezeichnet mit Sweepline



Gegeben: Menge achsenparalleler Objekte in der Ebene
 Idee: Schwenke vertikale Linie von links nach rechts
 über die Menge
 (Alternativ: horizontale Linie von oben nach unten).

- Statisches zweidimensionales geometrisches Problem wird in dynamische Folge eindimensionaler Probleme zerlegt.
 - Allgemeiner: statisches n -dimensionales Problem zerlegt in dynamische Folge $(n-1)$ -dimensionaler Probleme

Scan-line



Scan-line teilt zu jedem Zeitpunkt die Menge von Objekten in disjunkte Teilmengen auf

- „tote“ Objekte, die bereits vollständig von der Scan-line überstrichen wurden
- „aktive“ Objekte, die gegenwärtig von der Scan-line geschnitten werden
- „inaktive“ Objekte, die erst künftig von der Scan-line geschnitten werden

Scan-line definiert lokale Ordnung auf aktiven Objekten

Kein kontinuierliches Schwenken sondern diskrete Schritte

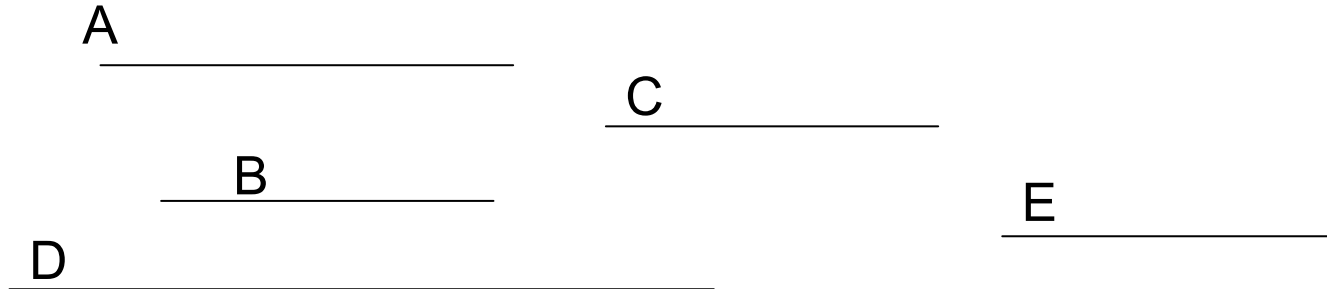
- Q: Aufsteigend sortierte Menge aller x-Werte der Objekte
- Zwischen zwei aufeinander folgenden Punkten in Q ändert sich die Menge der aktiven Objekte und deren relative Anordnung nicht.
 - Änderungen sind nur an den Punkten in Q möglich
- Q stellt Menge der Haltepunkte für die Scan-line dar



Problem bei der Kompaktierung höchstintegrierter Schaltkreise

- Zur Kompaktierung in y-Richtung müssen Abstandsbedingungen zwischen relevanten Paaren von Elementen eingehalten werden.
- Es genügt, die Menge aller Paare zu bestimmen, die sich sehen können.
 - Erfüllt, falls es eine vertikale Gerade gibt, die zwei Strecken s und s' schneidet und keine weitere Strecke zwischen diesen

Beispiel



- Menge der sichtbaren Paare: (A,B), (A,D), (B,D), (C,D)

Paarweiser Vergleich aller N Strecken

- Aufwand $\Omega(N^2)$

Aber

- Es kann nur höchstens linear viele gegenseitig sichtbare Paare geben.
- Auffassen der Relation „ist gegenseitig sichtbar“ als planarer Graph.
 - Knoten: Strecken
 - Kanten: verbinden sichtbare Strecken

Planarer Graph (plättbarer Graph)

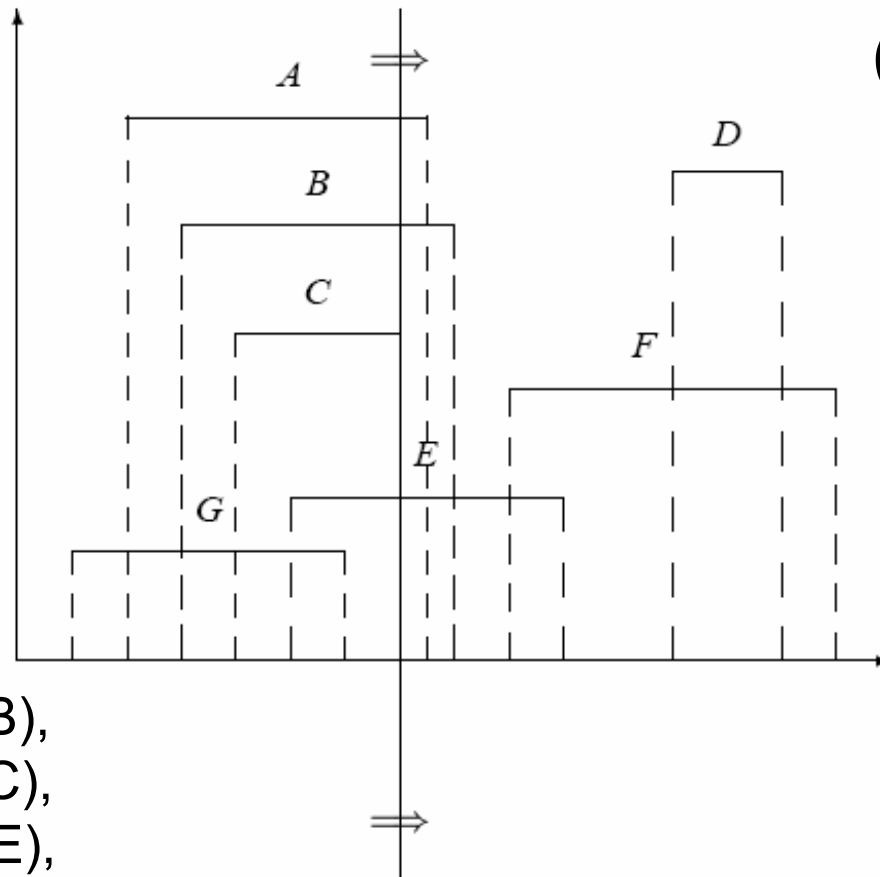
- Graph, der auf einer Ebene mit Punkten für die Knoten und Linien für die Kanten dargestellt werden kann, so dass sich die Kanten nur in den Knoten schneiden.
 - Planarer Graph kann höchstens $3N-6$ Kanten haben



Mit Scan-line Prinzip

- Vereinfachende Annahmen
 - x-Werte von Anfangs- und Endpunkten sämtlicher Strecken sind paarweise verschieden
- Menge L enthält die am jeweiligen Haltepunkt sichtbaren Strecken geordnet nach deren y-Wert
 - Implementierung von L als balancierter Suchbaum
 - $O(\log N)$ für Einfügen/Entfernen/Bestimmen Vorgänger, Nachfolger
 - N: maximale Anzahl der Elemente in L
- Aufwand
 - Bei jedem Schritt maximal vier der oben bestimmten Operationen durchzuführen
 - $O(N \log N)$





Haltepunkte (in
aufsteigender
x-Reihenfolge)

Ausgabe: (A,G),(A,B),
(B,G),(B,C),
(C,G),(C,E),
(E,G),(B,E)

G A A A A A A
G B B B B B
G C C C E
G E E
G

Menge L am Haltepunkt
(in aufsteigender y-Reihenfolge)



```
Q := Folge der 2N Anfangs- und Endpunkte von Elementen in S in aufsteigender
    x-Reihenfolge;
L := leer; //Menge der jeweils aktiven Liniensegmente in aufsteigender y-Reihenfolge

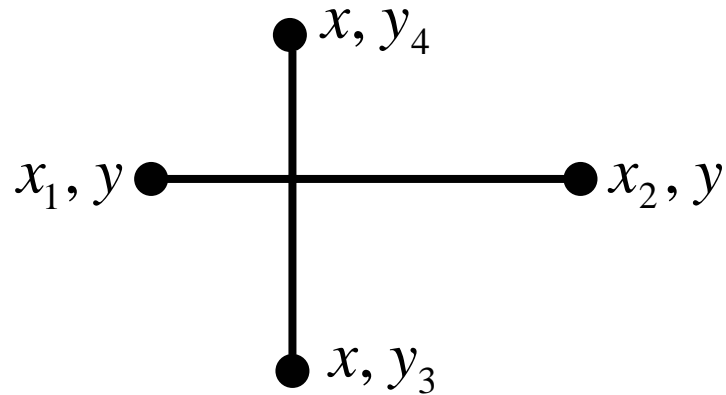
while Q ist nicht leer do
begin
    p := nächster (Halte)-Punkt von Q;
    if p ist linker Endpunkt einer Strecke s then
        begin
            füge s in L ein;
            bestimme die Nachbarn s' und s'' von s in L und gib
            (s, s') und (s, s'') als Paare sichtbarer Elemente aus
        end
    else //p ist rechter Endpunkt einer Strecke s
        begin
            bestimme die Nachbarn s' und s'' von s in L;
            entferne s aus L;
            gib (s', s'') als Paar sichtbarer Elemente aus
        end
    end while
```



Annahme

- keine Überschneidung von Strecken gleicher Ausrichtung.

Prüfe $x_1 \leq x \leq x_2$ oder $y_3 \leq y \leq y_4$



Aufwandsabschätzung

- Seien n horizontale und m vertikale Strecken gegeben.
- Es können maximal $m*n$ Schnittpunkte vorhanden sein.
- Es gibt keinen Algorithmus, der für alle Fälle garantiert weniger Aufwand hat als $O(m*n)$.
- Aufwand ist aber auch höchstens $O(mn)$, weil je Streckenpaar nur eine konstante Zeit für die Schnittpunktprüfung benötigt wird.



Mit Scan-line Prinzip

■ Vereinfachende Annahme

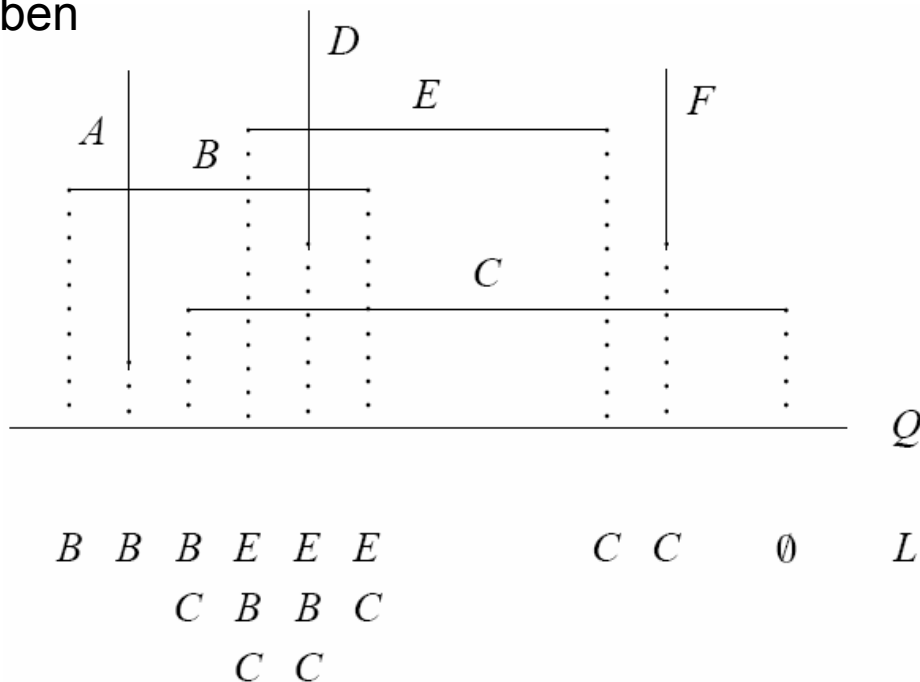
- Anfangs- und Endpunkte horizontaler Strecken und alle vertikalen Segmente haben paarweise verschiedene x-Koordinaten

■ Beobachtung

- Man merke sich die beim Schwenken jeweils aktiven Objekte in L
- Trifft man auf eine vertikale Strecke, so kann diese lediglich Schnittpunkte mit den Objekten in L haben

Beispiel

Ausgabe: (A,B),(D,E),
(D,B)



Q := Menge der x-Koordinaten der Anfangs- und Endpunkte horizontaler Strecken und von vertikalen Strecken in aufsteigender x-Reihenfolge;

L := leer //Menge der jeweils aktiven horizontalen Strecken in aufsteigender //y-Reihenfolge

while Q nicht leer do

begin

 p := nächster (Halte)-Punkt von Q;

 if p ist linker Endpunkt einer horizontalen Strecke s then

 füge s in L ein

 else

 if p ist rechter Endpunkt einer horizontalen Strecke s then

 entferne s aus L

 else

 //p ist x-Wert einer vertikalen Strecke s mit unterem

 //Endpunkt $(p; y_u)$ und oberem Endpunkt $(p; y_o)$

 bestimme alle horizontalen Strecken t aus L, deren y-Koordinate $y(t)$ im

 Bereich $y_u \leq y(t) \leq y_o$ liegt und gib (s, t) als Paar sich schneidender Strecken aus

end while

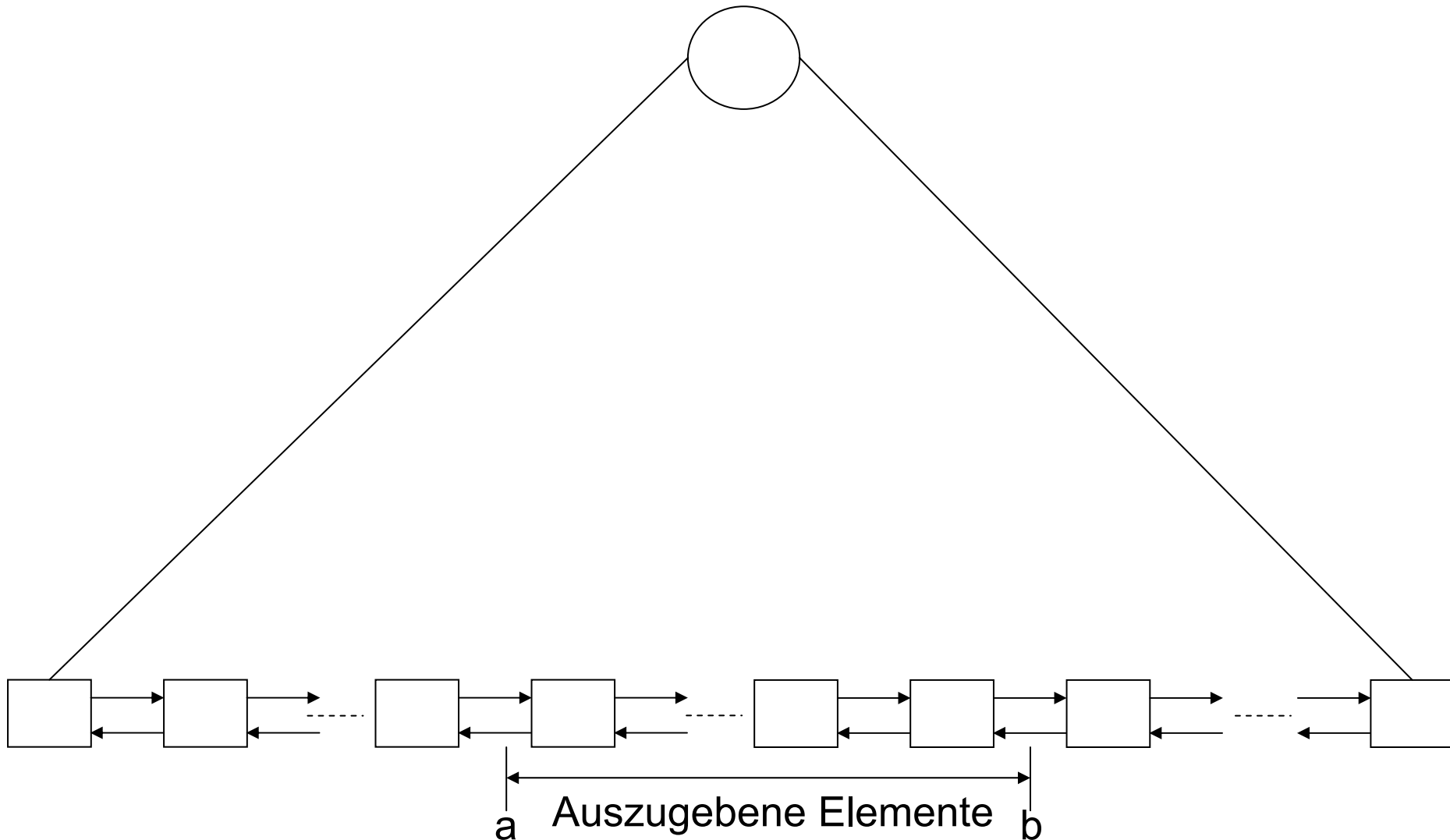


Mit Scan-line Prinzip

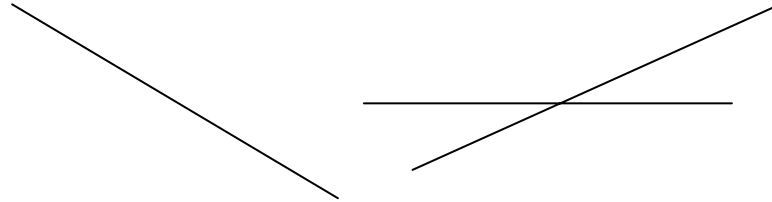
- Aufwand
 - Für die Menge L erforderlich
 - Einfügen, Entfernen, Bereichsanfrage im Bereich $[a, b]$
 - Implementierung von L als balancierter Suchbaum mit doppelter Verkettung benachbarter Blätter
 - Einfügen/Löschen in $O(\log N)$
 - Bereichsanfrage (*range query*): Zunächst durch Suchoperationen Blatt mit kleinstem Wert größer a und Blatt mit größtem Wert kleiner b bestimmen, dann Baum in diesem Bereich ablaufen und Elemente ausgeben.
 - Sei r die Anzahl der Objekte im Bereich $[a, b]$, dann $O(\log N + r)$
 - Insgesamt $O(N \log N + k)$
 - N : Anzahl der Segmente
 - k : Anzahl der Paare sich schneidender horizontaler und vertikaler Strecken
 - Für k kann gelten $k = N^2$



Skizze zu Verfahren und Struktur



Jetzt: Beliebige orientierte Strecken in der Ebene



Zwei Probleme

- Schnittpunkttest: gibt es in der Menge der Strecken wenigstens ein paar sich schneidender Strecken
 - Schnittpunktaufzählung: Bestimme für Menge von N Strecken alle Paare sich schneidender Strecken
 - Beides in $O(N^2)$ lösbar
- Lösung mit Scan-line

Annahmen zur Vereinfachung

- Keine Strecke ist vertikal
- In jedem Punkt schneiden sich höchstens 2 Strecken
- Alle Anfangs- und Endpunkte von Strecken haben paarweise verschiedene x-Koordinaten

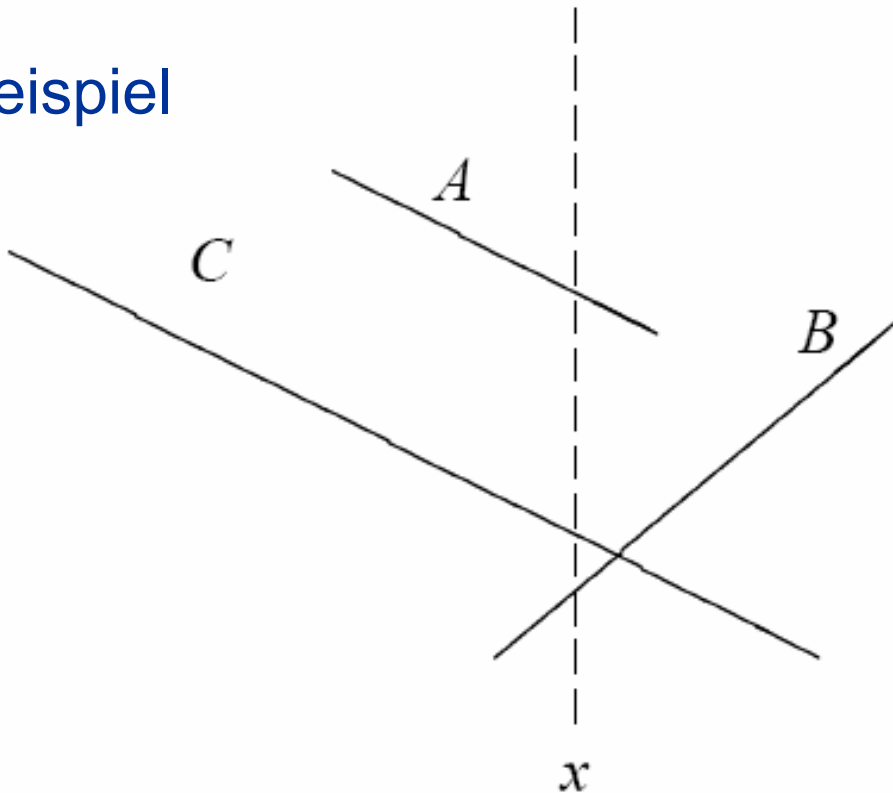


„ist oberhalb von“

- Seien A und B zwei Strecken.

Dann heißt *A x-oberhalb von B*, $A \uparrow_x B$, wenn die vertikale Gerade durch x sowohl A als auch B schneidet und der Schnittpunkt von x und A oberhalb des Schnittpunktes von x und B liegt.

Beispiel



$C \uparrow_x B,$
 $A \uparrow_x C$
 $A \uparrow_x B.$



Scan-line Prinzip

- Von links nach rechts
- Strecken sind an jeder Stelle x durch \uparrow_x vollständig geordnet
 - Änderungen sind möglich am linken oder rechten Ende einer Strecke
 - Änderungen sind möglich an Schnittpunkten

Für zwei Strecken a und b gilt

- Schneiden sich a und b so existiert vor dem Schnittpunkt eine Stelle, an der a und b in der Ordnung \uparrow_x direkt aufeinander folgen
- Es genügt also, jeweils benachbarte Strecken auf einen Schnittpunkt zu überprüfen.



Q := Folge der $2N$ Anfangs- und Endpunkte von Elementen in S in aufsteigender x -Reihenfolge;

L := leer; //Menge der jeweils aktiven Strecken in \uparrow_x -Ordnung

gefunden := false;

while (Q ist nicht leer) and not gefunden do

begin

p := nächster Haltepunkt von Q ; // p habe x -Koordinate $p:x$

 if p ist linker Endpunkt einer Strecke s then

 begin

 füge s entsprechend der an der Stelle p gültigen Ordnung $\uparrow_{p:x}$ in L ein;

 bestimme den Nachfolger s' und den Vorgänger s'' von s in L bzgl. $\uparrow_{p:x}$;

 if $(s \cap s' \neq \text{leer})$ or $(s \cap s'' \neq \text{leer})$ then

 gefunden := true

 end



else //p ist rechter Endpunkt einer Strecke s

begin

bestimme den Nachfolger s' und den Vorgänger s'' von s bzgl. der an der Stelle p gültigen Ordnung $\uparrow_{p:x}$;

entferne s aus L;

if $s' \cap s'' \neq \text{leer}$ then

gefunden := true

end

end while

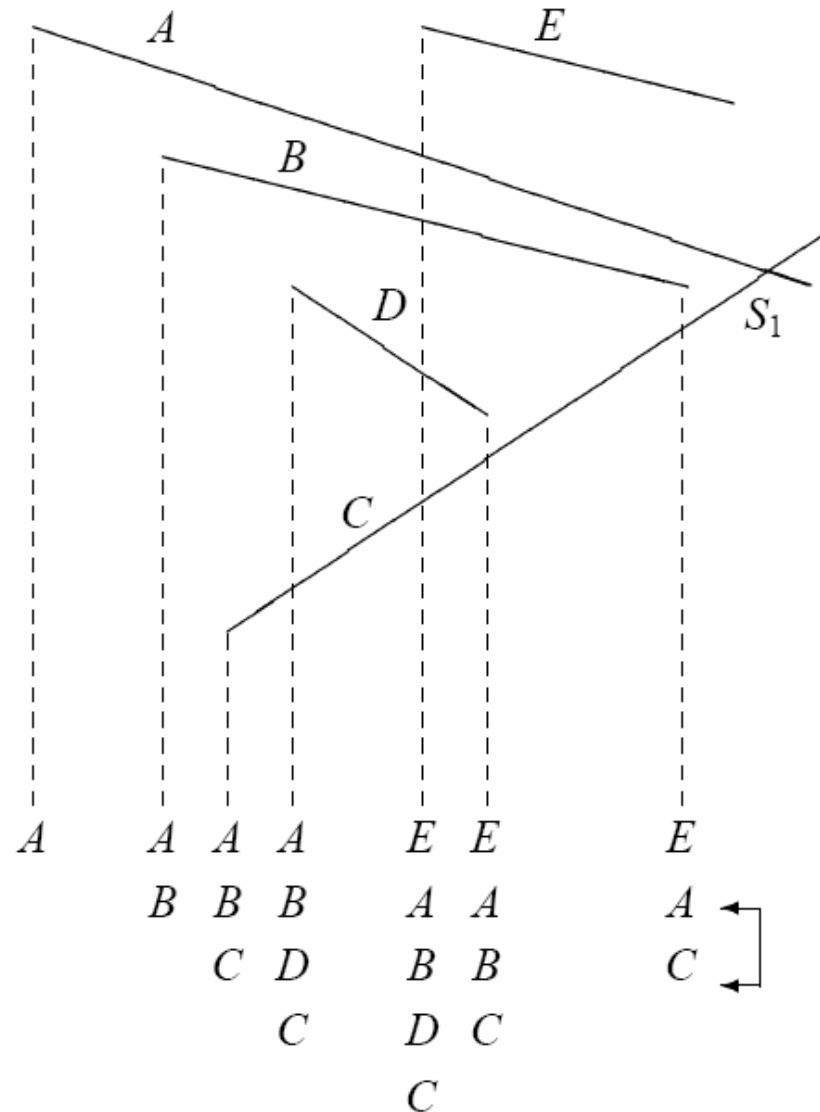
if gefunden then

write('ja')

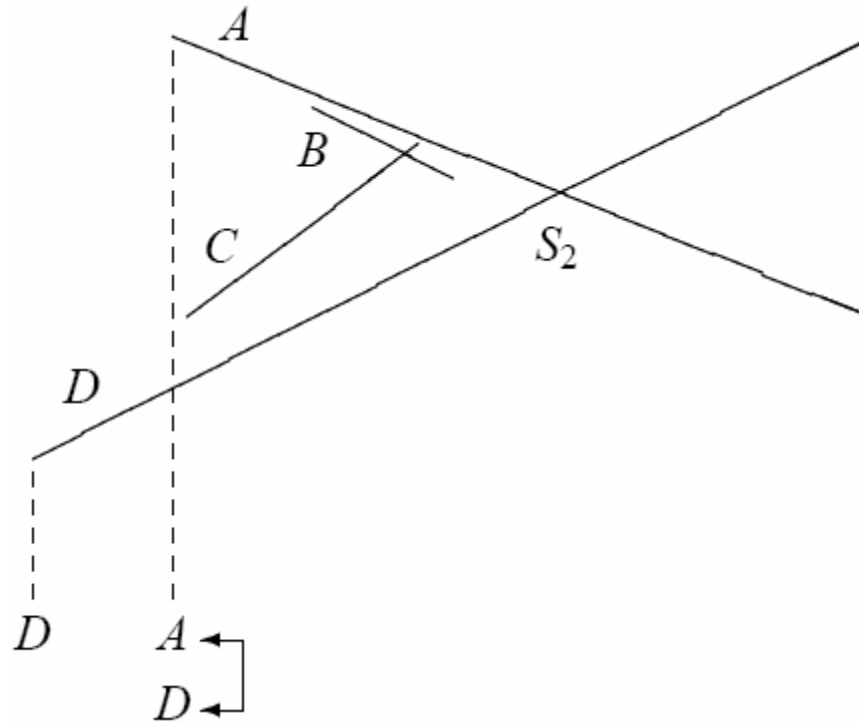
else write('nein')



Verfahren findet Schnittpunkt



Verfahren findet nicht unbedingt den am weitesten links angesiedelten Schnittpunkt



Sortieren der Endpunkte

- $O(N \log N)$

L als balancierten Baum implementieren

- Operationen haben $O(\log N)$

Feststellung ob N Strecken mindestens einen Schnittpunkt haben

- $O(N \log N)$



Vorheriger Algorithmus läuft weiter

- L muss nach dem Durchlaufen eines Schnittpunkts angepasst werden
- Schnittpunkte werden in die sortierte Liste der Haltepunkte eingeführt

Weitere vereinfachende Annahme

- Schnittpunkte haben nicht die gleiche x-Koordinate

Problem

- Schnittpunkt kann mehrfach gefunden werden
- Vor Einfügen in Q ist damit eine Suche in Q erforderlich
- Q wird als balancierter Binärbaum organisiert

Aufwand

- $O((N+k) \log N)$
 - k sei die Zahl vorhandener Schnittpunkte





Frage

- Wie soll geteilt werden?

Idee

- Teilende Gerade durch die Objektmenge

Problem

- Aufteilung kann Objekte durchschneiden.
- Drei Teilmengen: „Links“, „rechts“, „durchschnitten“

→Prinzip der getrennten Repräsentation geometrischer Objekte

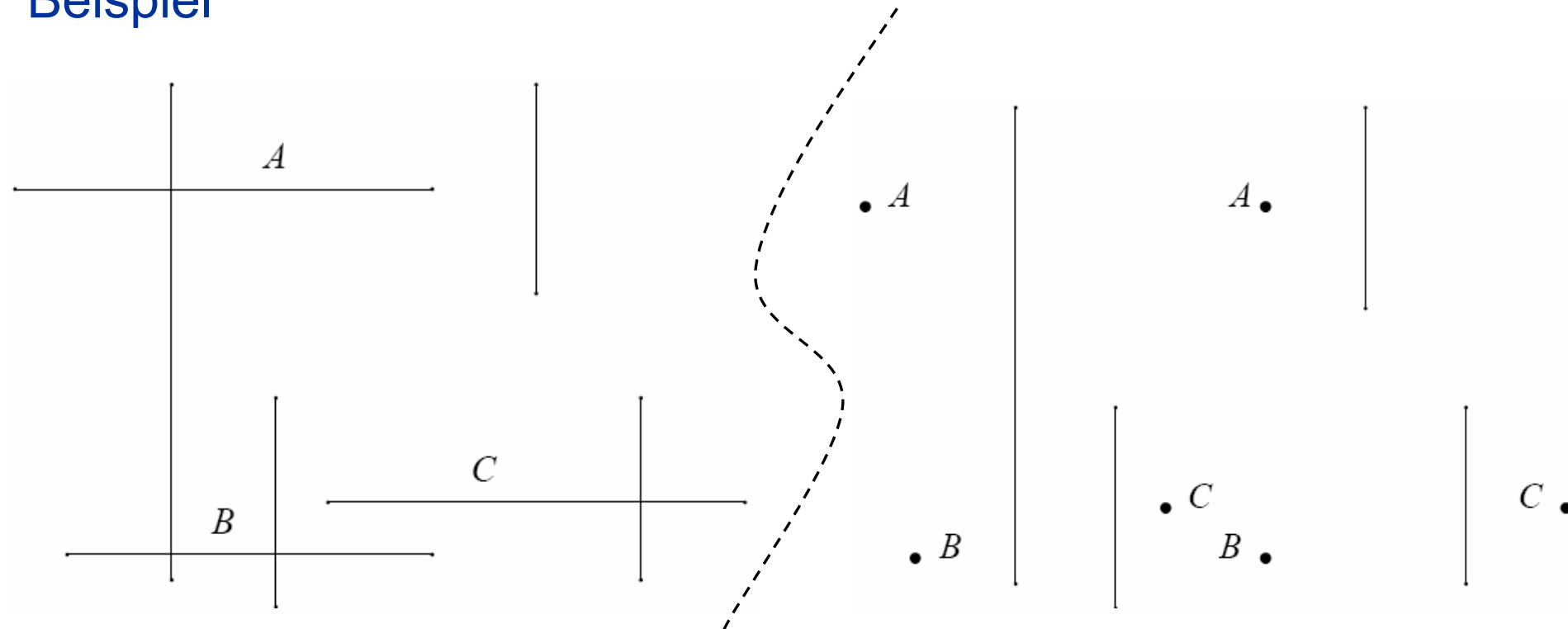
- Objekt, dessen Ausdehnung in x-Richtung größer Null ist, wird durch linken und rechten Endpunkt repräsentiert
- Endpunkte werden als unabhängige Einheiten behandelt



Mit Divide-and-Conquer

- Vereinfachende Annahme: nur horizontale und vertikale Strecken
- Horizontale Strecke h wird durch Paar der Endpunkte repräsentiert
 - Linker Endpunkt: $.h$ und rechter Endpunkt: $h.$
- Operation auf Menge vertikaler Strecken und Punkte

Beispiel



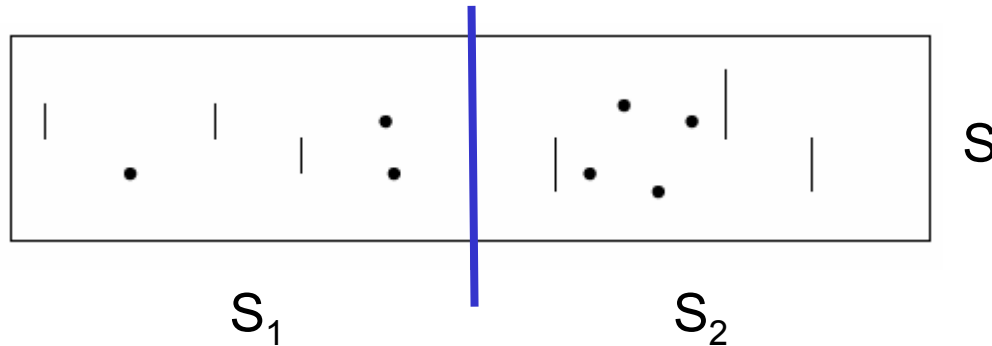
Vereinfachende Annahmen

- Keine zwei vertikalen Strecken und Anfangs- oder Endpunkte horizontaler Strecken haben die gleichen x-Koordinaten.

Algorithmus: ReportCuts(S)

- //liefert alle Paare von sich schneidenden vertikalen Strecken in S und //horizontalen Strecken mit linkem oder rechtem Endpunkt in S

1. Divide: Teile S (durch eine vertikale Gerade) in eine linke Hälfte S_1 und eine rechte Hälfte S_2 , falls S mehr als ein Element enthält; sonst enthält S kein sich schneidendes Paar:

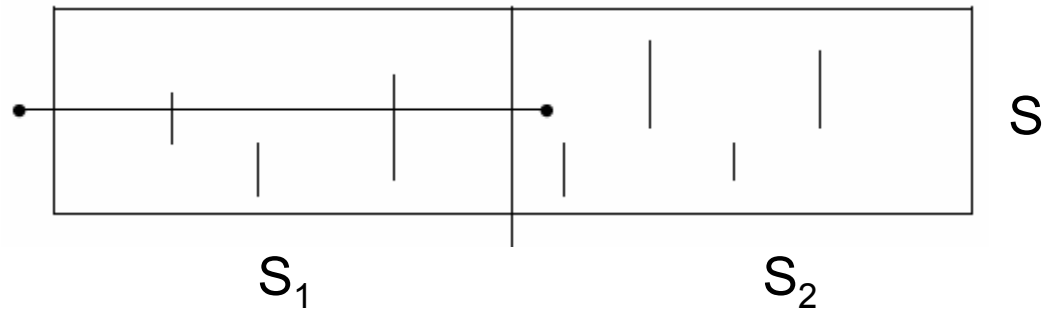


2. Conquer: ReportCuts(S_1); ReportCuts(S_2);

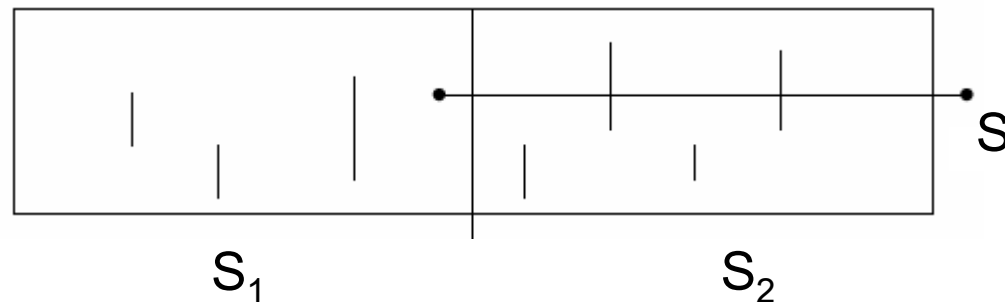
//alle Schnitte in S_1 oder S_2 zwischen Paaren von Strecken,
 //die wenigstens einmal repräsentiert sind, sind bereits berichtet



3. Merge: Berichte alle Schnitte zwischen vertikalen Strecken in S_1 und horizontalen Strecken mit rechtem Endpunkt in S_2 , deren linker Endpunkt nicht in S_1 oder S_2 vorkommt:



- Berichte alle Schnitte zwischen vertikalen Segmenten in S_2 und horizontalen Segmenten mit linkem Endpunkt in S_1 , deren rechter Endpunkt nicht in S_1 oder S_2 vorkommt:



Rekursionsinvariante

- Nach Beendigung von $\text{ReportCuts}(S_i)$ sind alle Schnitte zwischen den in S_i repräsentierten Strecken ausgegeben. Eine horizontale Strecke ist in S_i repräsentiert, wenn mindestens ein Endpunkt in S_i enthalten ist.

Im folgenden wird gezeigt

- Gilt Rekursionsinvariante für S_1 und S_2 , dann gilt Rekursionsvariante auch für S .

Betrachtet

- Beliebige horizontale Strecke h , deren linker oder rechter Endpunkt in S vorkommt.
- Zu zeigen
 - Nach Beendigung von $\text{ReportCut}(S)$ sind alle Schnitte von h mit vertikalen Strecken berichtet.
 - h kann keine weiteren Schnitte haben.



Fall 1

- Beide Endpunkte von h liegen in S_1 .

Fall 2

- Beide Endpunkte von h liegen in S_2 .

Fall 3

- Nur der rechte Endpunkt von h liegt in S_1 .



- Schnitte von h mit vertikalen Strecken in S_1 sind von $\text{ReportCuts}(S_1)$ berichtet.

Fall 4

- Analog zu Fall 3: linker Eckpunkt von h liegt in S_2 .



Fall 5

- Linker Eckpunkt von h liegt in S_1 , rechter Eckpunkt in S_2 .
 - Nach Beendigung $\text{ReportCuts}(S_1)$ und $\text{ReportCuts}(S_2)$ sind alle Schnitte mit vertikalen Strecken berichtet.

Fall 6

- Linker Eckpunkt von h liegt in S_1 , rechter Eckpunkt liegt weder in S_1 noch in S_2 .

Fall 7

- Analog zu Fall 6, aber unter Betrachtung des linken Eckpunkts.



- Bestimmte Schnitte im Merge-Schritt möglichst schnell
- Benötigter Aufwand proportional zur Anzahl der Schnitte

Ordne den Mengen S die Mengen $L(S)$, $R(S)$, $V(S)$ zu

- $L(S) = \{ y(h) \mid h \text{ ist horizontale Strecke mit linkem Endpunkt von } h \in S \text{ aber rechtem Endpunkt von } h \notin S \}$
- $R(S) = \{ y(h) \mid h \text{ ist horizontale Strecke mit linkem Endpunkt von } h \notin S \text{ aber rechtem Endpunkt von } h \in S \}$
- $V(S) =$ Menge der durch die vertikalen Strecken in S definierten y -Intervalle:
 $\{ [y_u(v), y_o(v)] \mid v \text{ ist vertikale Strecke in } S \}$

obere und untere y -Koordinate von v



- Zu Beginn des Merge-Schritts bekannt: $L(S_i)$, $R(S_i)$, $V(S_i)$ mit $i = 1, 2$

Bestimme nun alle Paare (h, v) :

- $y(h) \in R(S_2) \setminus L(S_1)$,
 $[y_u(v), y_o(v)] \in V(S_1)$,
 $y_u(v) \leq y(h) \leq y_o(v)$ oder
- $y(h) \in L(S_1) \setminus R(S_2)$,
 $[y_u(v), y_o(v)] \in V(S_2)$,
 $y_u(v) \leq y(h) \leq y_o(v)$

Aus $L(S_i)$, $R(S_i)$, $V(S_i)$ erhält man $S = S_1 \cup S_2$

$$L(S) := (L(S_1) \setminus R(S_2)) \cup L(S_2)$$

$$R(S) := (R(S_2) \setminus L(S_1)) \cup R(S_1)$$

$$V(S) := V(S_1) \cup V(S_2)$$

Falls S einelementig, initialisiere wie folgt:

Fall 1: $S = \{\text{Rechter Endpunkt von } h\}$, d.h. S enthält nur den linken Endpunkt einer horizontalen Strecke h

$$L(S) := \{y(h)\}, R(S) := \emptyset, V(S) := \emptyset$$



Fall 2: $S = \{ \text{Rechter Endpunkt von } h \}$, d.h. S enthält nur den rechten Endpunkt einer horizontalen Strecke h

$L(S) := \emptyset$, $R(S) := \{ y(h) \}$, $V(S) := \emptyset$

Fall 3: $S = \{ v \}$, d.h. S enthält nur die vertikale Strecke v

$L(S) := 0$, $R(S) := 0$, $V(S) := \{ [y_u(v), y_o(v)] \}$

- Speichere nun Menge S von vertikalen Strecken und linken und rechten Endpunkten horizontaler Strecken in einem nach aufsteigenden x -Werten sortierten Array.
- Das Teilen im Divide-Schritt wird in konstanter Zeit ausgeführt.
- Implementiere Mengen $L(S)$ und $R(S)$ als nach aufsteigenden y -Werten sortierte, verkettete lineare Listen.
- $V(S)$ ebenfalls als nach unteren Endpunkten, also nach y_u -Werten sortierte, verkettete lineare Liste implementieren



Aufwand von ReportCuts(S)

- Menge S habe N Elemente
- $T(N)$ bezeichne Anzahl der notwendigen Schritte, um Verfahren durchzuführen
- $T(1)=1$
- Rekurrenzgleichung

$$T(N) = \underbrace{O(1)}_{\text{Divide}} + \underbrace{2T\left(\frac{N}{2}\right)}_{\text{Conquer}} + \underbrace{O(N)}_{\text{Merge}}$$

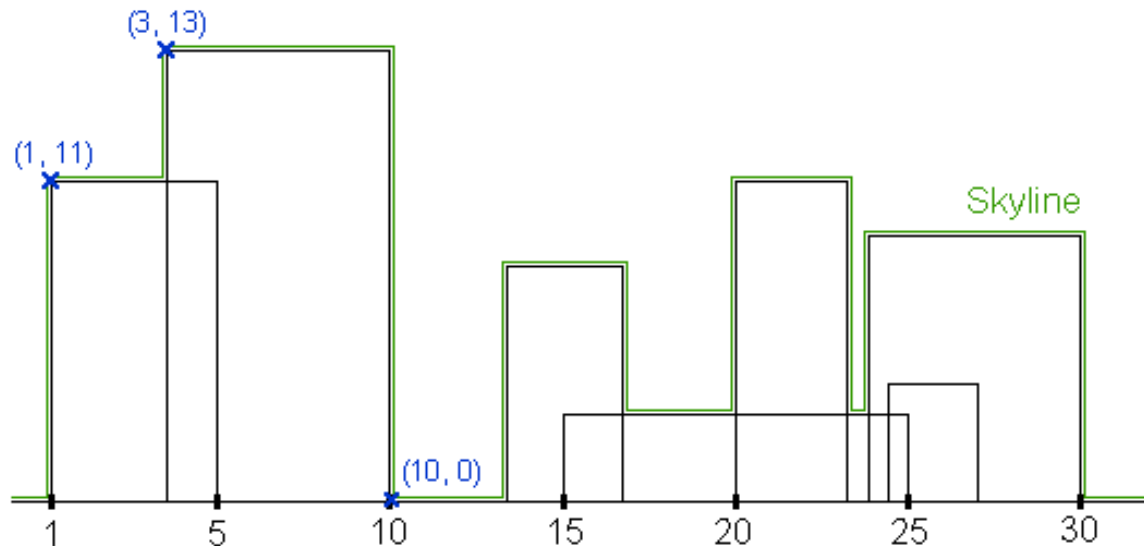
Diese Gleichung hat die Lösung $O(N \log N)$.



Skyline-Problem

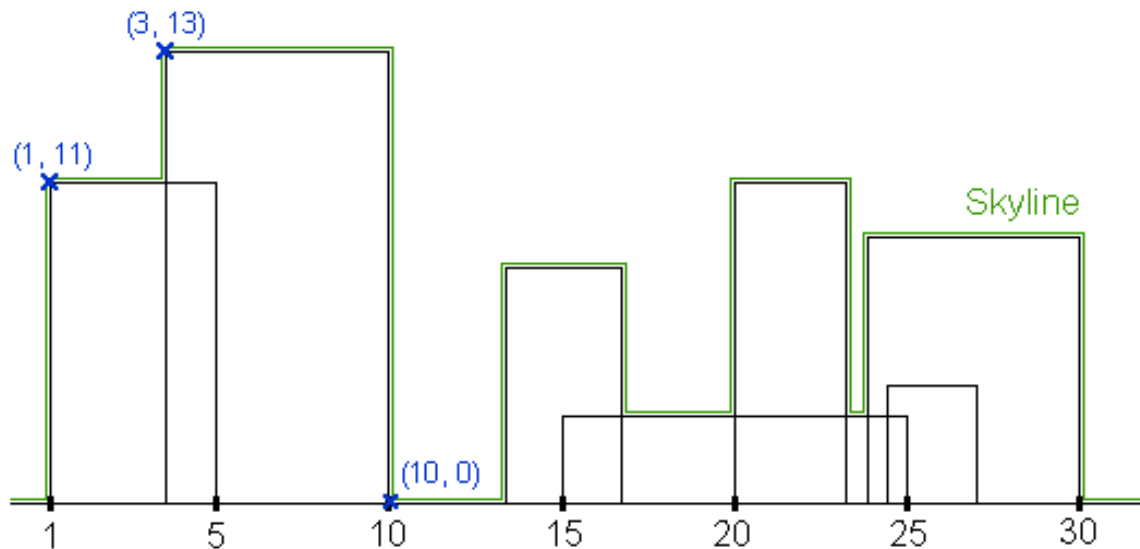
Gegeben: Exakte Positionen und Umrissse von viereckigen Gebäuden in einer Stadt

- Gebäude $G_i = (L_i, H_i, R_i)$ mit
 - L_i = linke x-Koordinate, R_i = rechte x-Koordinate, H_i = Höhe
 - z.B. $[(1, 11, 5), (3, 13, 10), (10, 0, 13), (13, 8, 17), (15, 4, 25), (20, 11, 23), \dots]$



Gesucht: Skyline der Gebäude (zweidimensional)

- Die Skyline $[v_i]$ ist eine Folge von Punkten, an denen sich die Höhe ändert,
z. B. $v_1 = (1, 11)$; $v_2 = (3, 13)$; $v_3 = (10, 0)$; ...



... bei n Gebäuden:

```
for {i= 1; i<= n; i++}
```

füge G_i zur aktuellen Skyline hinzu, in dem man von links nach rechts die bestehende Skyline scannt und geeignete v_j zufügt

Zeitbedarf:

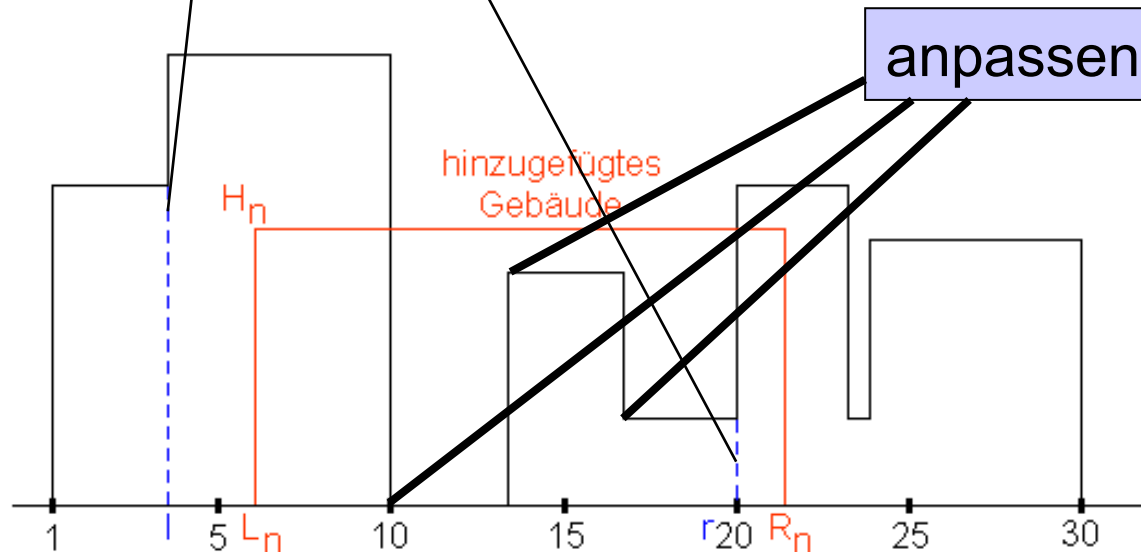
- Schleifenrumpf in Zeit $O(i)$ realisierbar, aber nicht schneller
- $O(\sum\{i \mid i=1,\dots,n\}) = O(n^2)$



Einfügen eines Gebäudes:

- Es dauert lineare Zeit, um ein Gebäude G_n in eine Skyline einzufügen
- Suche maximale l, r , mit $x(v_l) \leq L_n$, $x(v_r) \leq R_n$
- Falls $y(v_l) < H_n \Rightarrow (L_n, H_n)$ zur Skyline hinzufügen
- Falls $y(v_r) < H_n \Rightarrow (R_n, H_n)$ zur Skyline hinzufügen

Für alle i mit $l < i \leq r$: Anpassen oder Löschen von v_i .

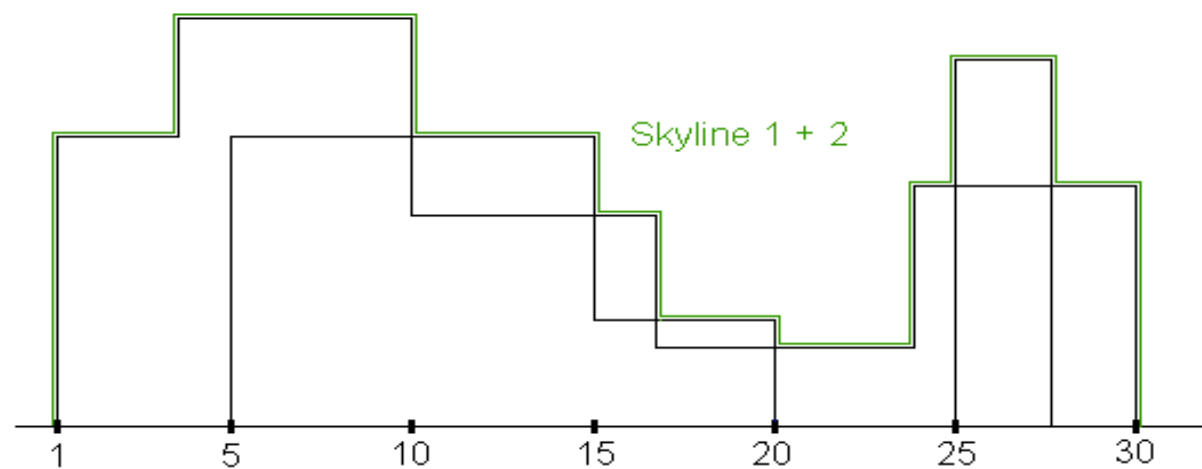
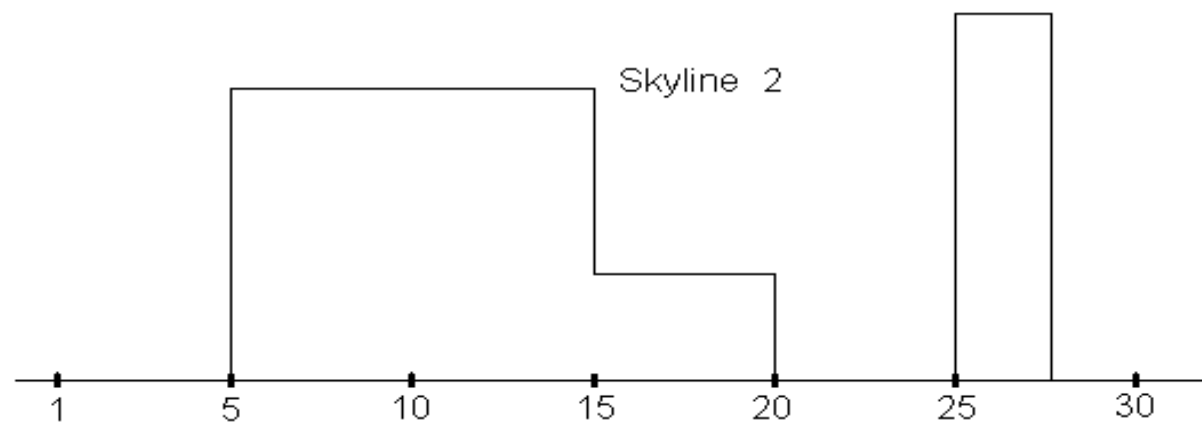
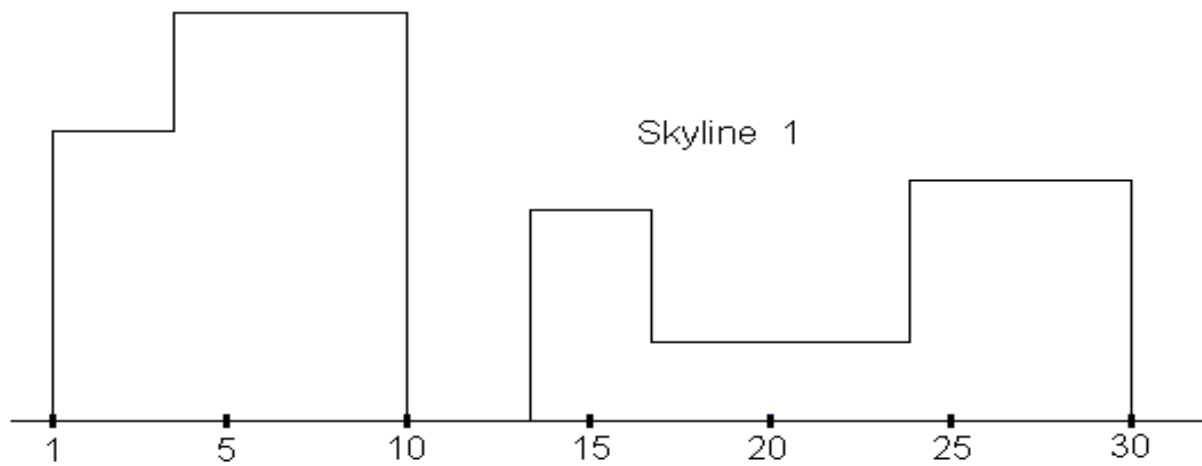


Idee: Es dauert lineare Zeit, um zwei Skylines zu verschmelzen.

Algorithmus:

```
// Typ Skyline ist eine Liste von Punkten
// erzeugt Skyline fuer die Gebaeude  $G_1, \dots, G_r$ 
Skyline computeSkyline(int l, r) {
    if (l > r) return [];
    if (l = r) return [( $L_1$ ,  $H_1$ ), ( $R_1$ , 0)];
    // teile und herrsche
    M = (r + l) / 2;
    Skyline skyLine1 := computeSkyline(l, M);
    Skyline skyLine2 := computeSkyline(M + 1, r);
    return Zusammenbau(skyLine1, skyLine2);
}
```





Prozedur Zusammenbau:

- Füge zwei Skylines der Größe $n/2$ zu einer Skyline der Größe n zusammen
- Reißverschlussverfahren

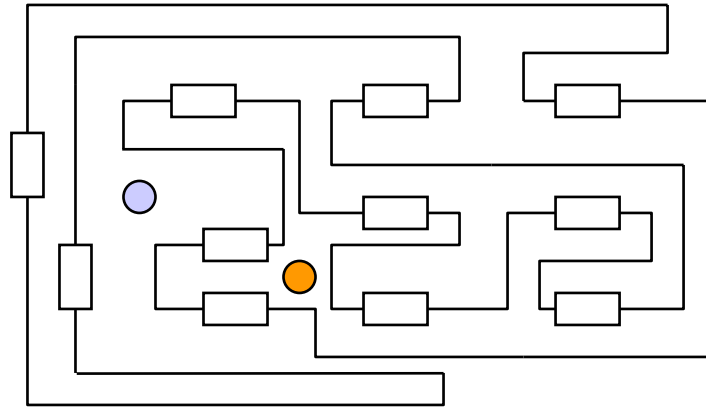
Zeitbedarf:

- Zusammenbau von zwei Skylines der Größe $n/2$ ist in $O(n)$ realisierbar.
- Somit:
$$T(n) = 2T(n/2) + c_n$$
$$T(1) = d$$
- $T(n) = O(n \log n)$



Beispiel: Schaltplan

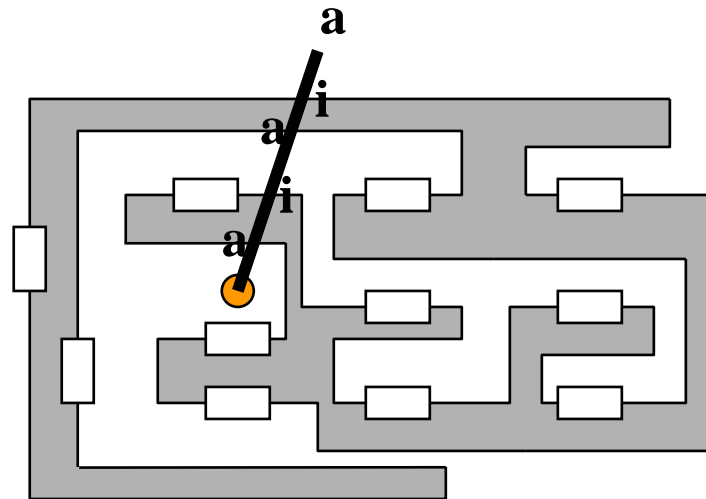
Gegeben: Schaltplan, die Leitungen bilden ein Polygon



- Frage: Kann man auf der Platine noch eine Leiterbahn von ○ nach ● unterbringen?
- Antwort: Ja, wenn beide Punkte innerhalb oder beide außerhalb des Polygons sind.



Punkt-in-Polygon-Problem



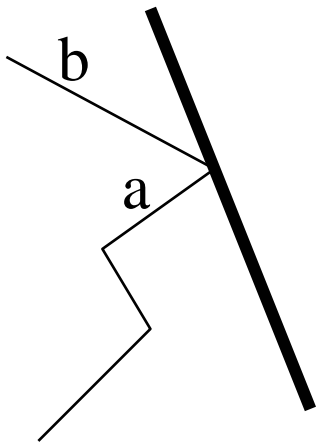
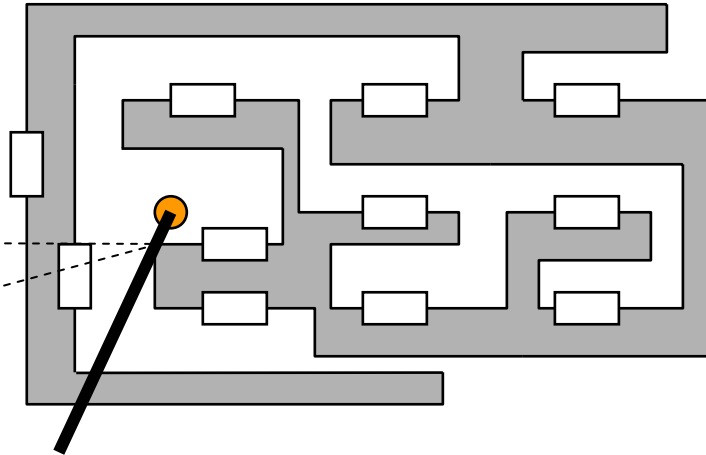
Algorithmus:

- wähle Punkt „unendlich“ weit außen und verbinde ihn mit ● gehe auf der Verbindung in Richtung ●
- wenn eine Polygonkante gekreuzt wird und wir außen sind, gehen wir nach innen
- wenn eine Polygonkante gekreuzt wird und wir innen sind, gehen wir nach außen

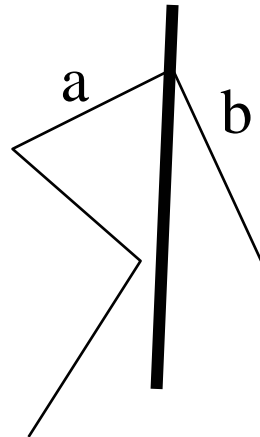


Ein Sonderfall:

Was ist, wenn die Halbgerade einen Eckpunkt des Polygons schneidet?



beide angrenzenden Kanten auf der selben Seite:
innen/außen bleibt gleich



verschiedene Seiten:
innen/außen wechselt

Punkt-in-Polygon-Algorithmus

Ist (u,v) innerhalb des Polygons $P = (x_1, y_1), \dots, (x_n, y_n)$?

Definiere eine Halbgerade H ausgehend von (u,v)

setze $Ort=1$ // 0=„innen“, 1=„außen“

für alle Kanten $k = (x_i, y_i) - (x_{i+1}, y_{i+1})$:

falls H und k sich schneiden, aber nicht in (x_i, y_i)

dann setze $Ort=1-Ort$

falls H durch (x_i, y_i) geht

falls $(x_{i-1}, y_{i-1}) - (x_i, y_i)$ und k auf gleicher Seite von H

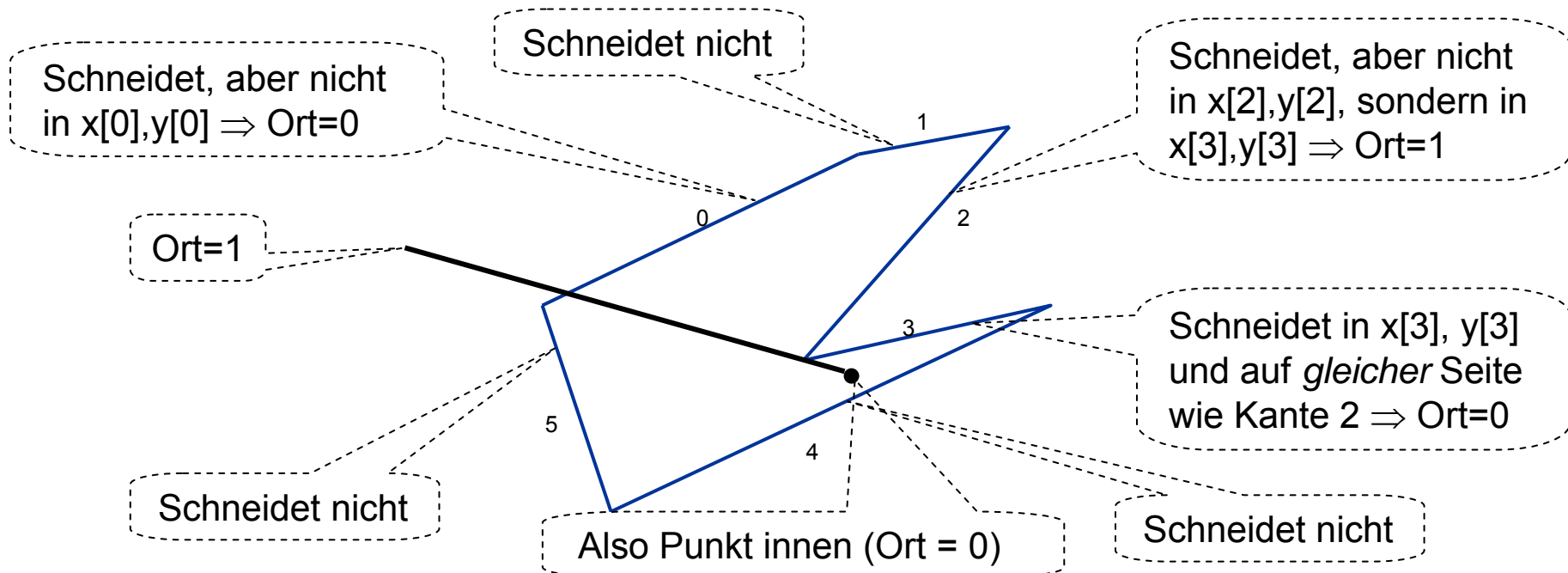
dann setze $Ort=1-Ort$

Aufwandsabschätzung: Je Kante ein konstanter Aufwand; also
bei n Kanten: $O(n)$.

Beachte: Aussage nur bezüglich dieses Polygons!

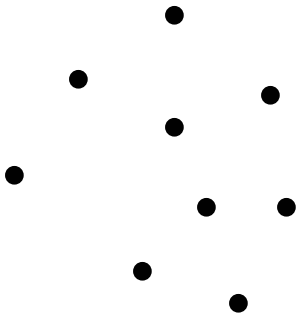


Definiere eine Halbgerade H ausgehend von (u,v)
 setze $Ort=1$ // 0=„innen“, 1=„außen“
 für alle Kanten $k = (x_i, y_i) - (x_{i+1}, y_{i+1})$:
 falls H und k sich schneiden, aber nicht in (x_i, y_i)
 dann setze $Ort=1-Ort$
 falls H durch (x_i, y_i) geht
 falls $(x_{i-1}, y_{i-1}) - (x_i, y_i)$ und k auf gleicher Seite von H
 dann setze $Ort=1-Ort$

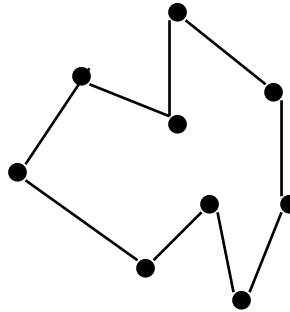


14.5 Konstruktion von Polygonen

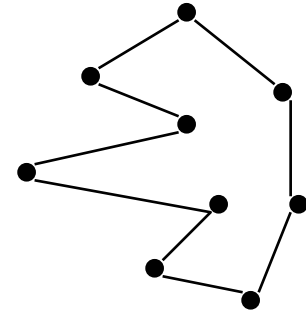
Eine Punktmenge, die die Eckpunkte eines Polygons darstellen soll, kann für viele Polygone stehen:



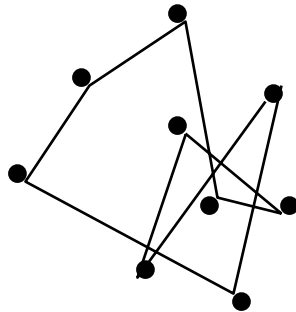
z.B:



oder:



aber auch
möglich:

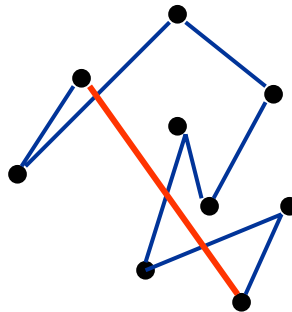


Frage: Wie bekommt man
garantiert ein Polygon mit
überschneidungsfreien Kanten?

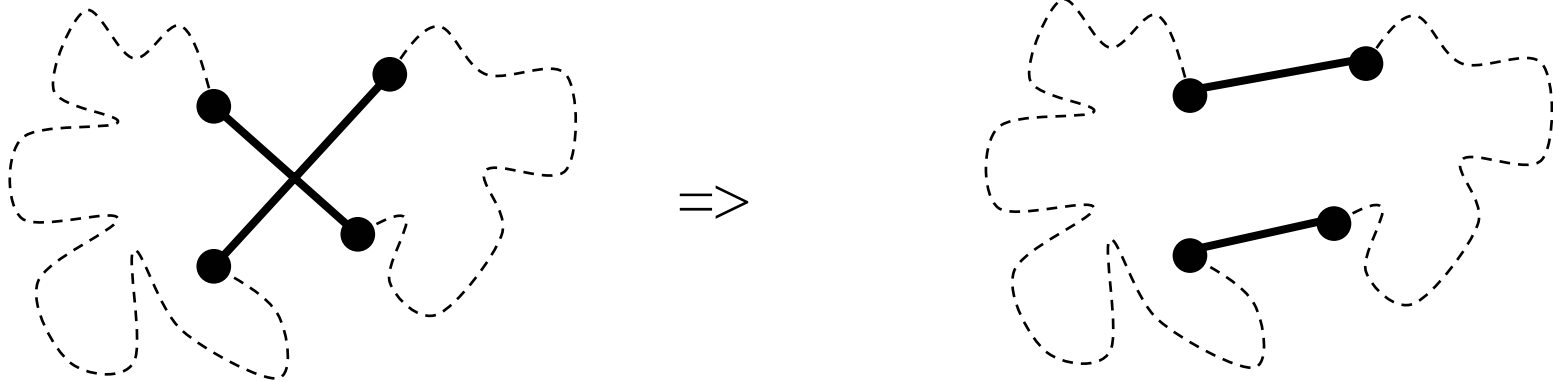


Ein „Greedy“ Algorithmus

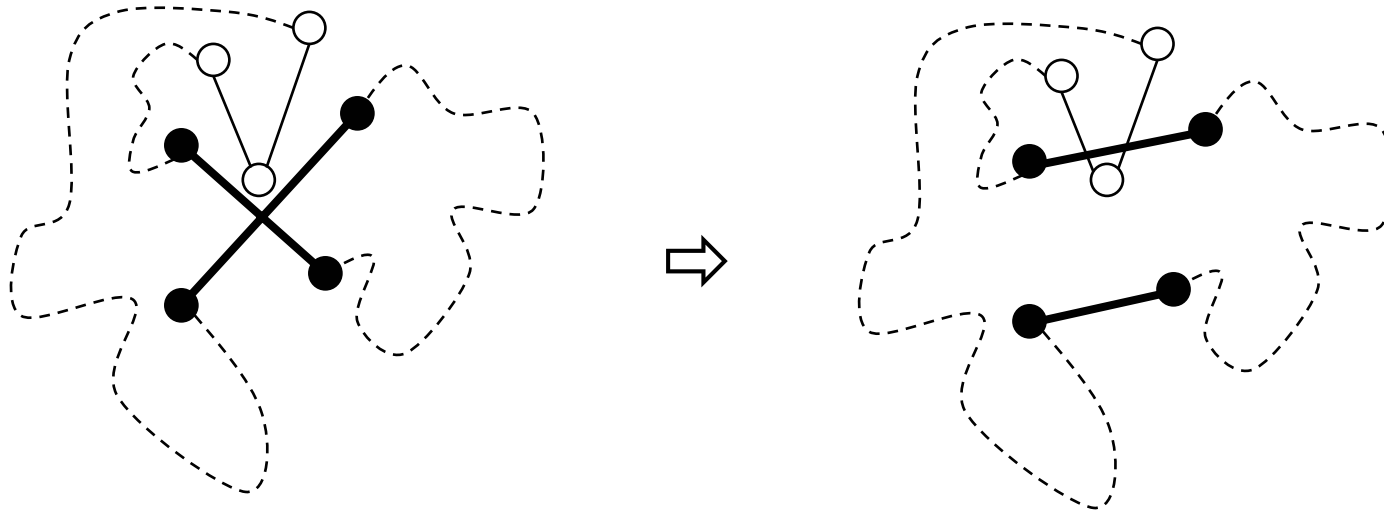
fange mit beliebiger Kante an
solange Polygon nicht fertig
 füge beliebige Kante hinzu,
 die keine vorhandene Kante schneidet und
 nicht vorzeitig das Polygon schließt



Beliebig verbinden, dann Kreuzungen auflösen:



Auflösen von Kreuzungen kann neue Kreuzungen erzeugen

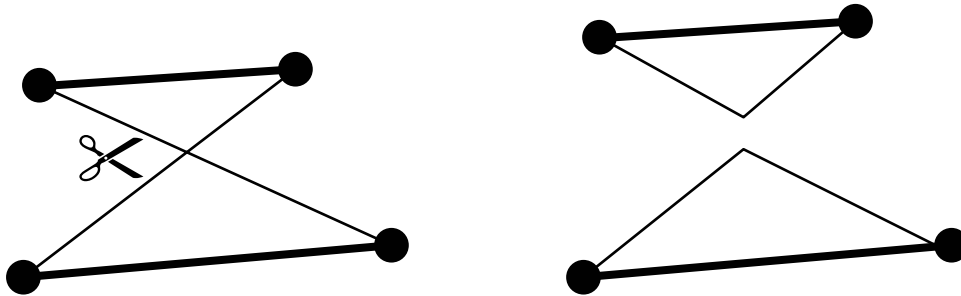


Frage: Klappt es trotzdem?

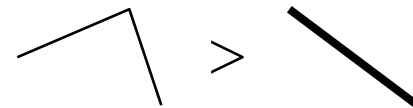
Konstruktion durch Auflösen von Kreuzungen funktioniert!

Beweis:

- In jedem Schritt wird die Gesamtlänge aller Kanten um mindestens einen positiven Wert d verringert.



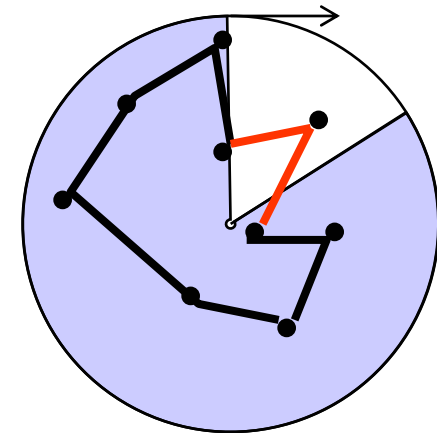
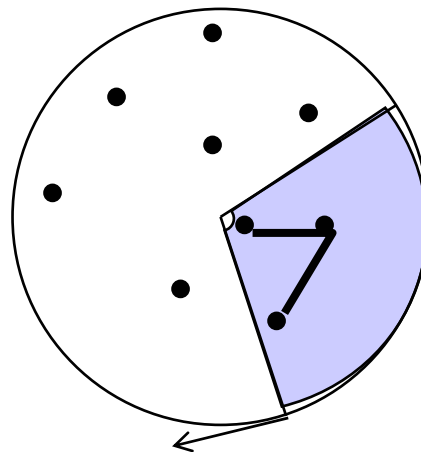
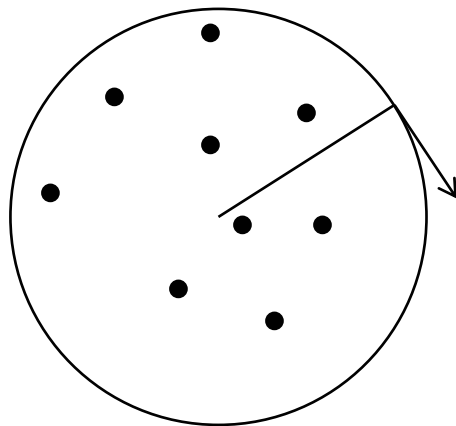
Dreiecksungleichung:



- Würde man beliebig oft Kreuzungen entfernen können, würde die Gesamtlänge negativ werden.
- Also terminiert der Algorithmus und es gibt irgendwann keine Kreuzungen mehr.
- Auflösen von Kreuzungen ist aufwändig!
- Geht es auch einfacher?

Verfahren

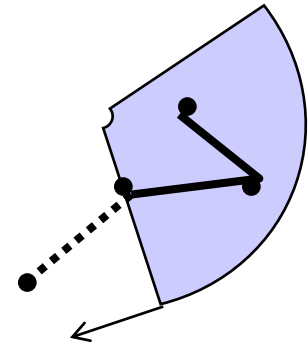
- Radius einer Kreisscheibe über alle Punkte rotieren.
- Füge immer Kante zum neu „überstrichenen“ Punkt hinzu.



Frage: Klappt es immer?

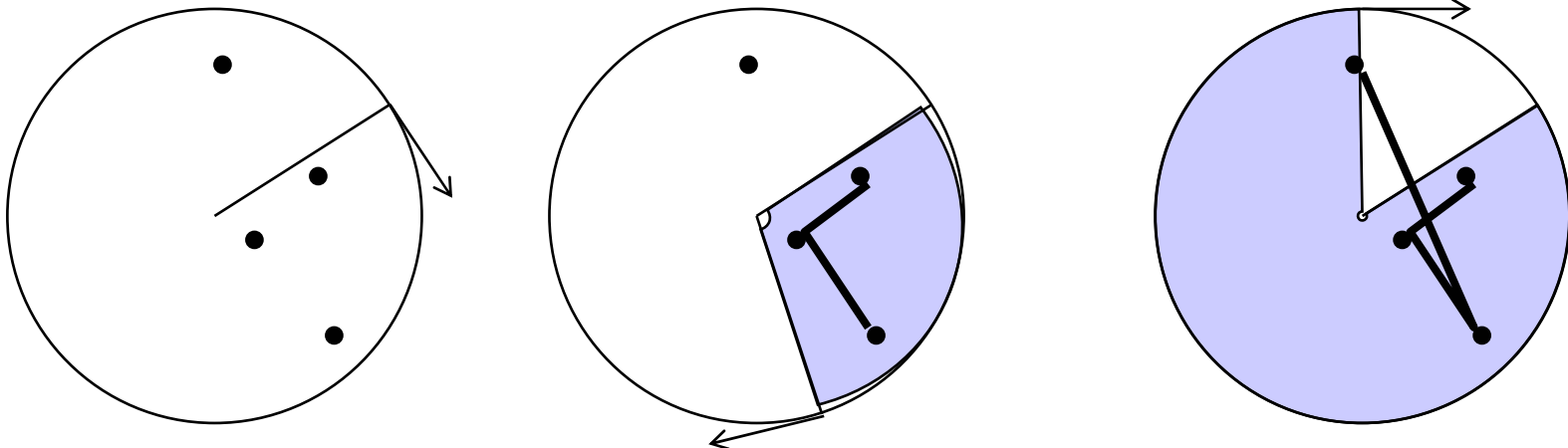
Beweisidee:

- Jede neu hinzugenommene Kante liegt komplett außerhalb des bis zur vorherigen Kante „überstrichenen“ Bereiches.
- Also sind Überschneidungen ausgeschlossen...



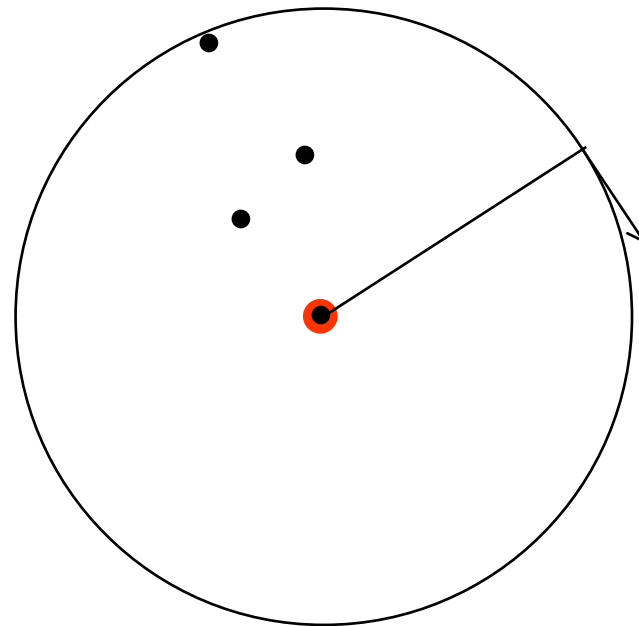
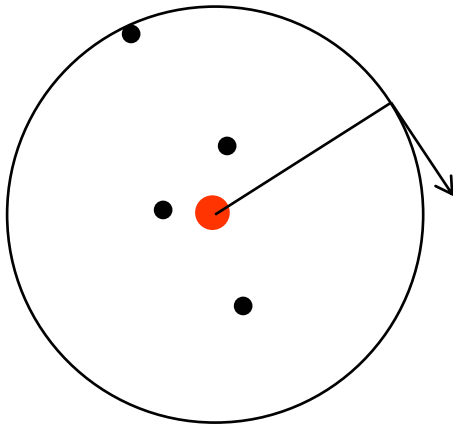
Leider:

- Wenn der Radius 180° zurückgelegt hat, ohne einen Punkt zu überstreichen, kann eine Kreuzung entstehen.



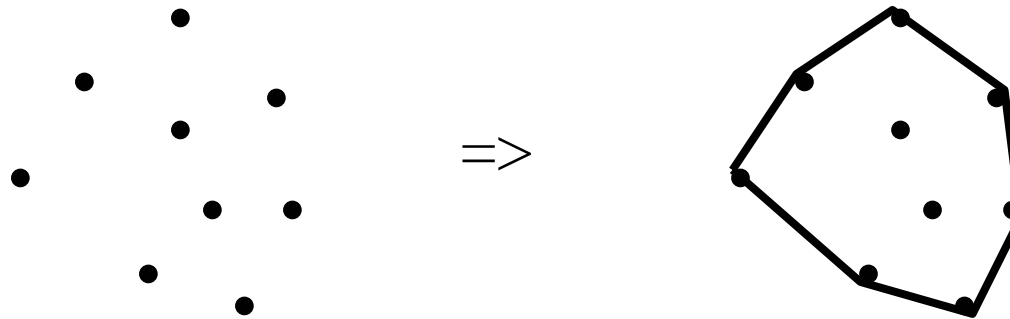
Ergänzung mit Wahl des Kreismittelpunkts:

- Wähle Mittelpunkt der Kreisscheibe innerhalb dreier Punkte der Punktmenge.
- Dreiecksinnenwinkel ist immer $< 180^\circ$.
- Oder: Wähle einen Randpunkt der Punktmenge als Mittelpunkt (z.B. äußerst rechts, bei mehreren dann den obersten).
- Der gesamte zu überstreichende Winkel ist nun unter 180° .



Definition:

- Die konvexe Hülle einer Punktmenge ist das kleinste konvexe Polygon, das alle Punkte in seinem Innern hat.

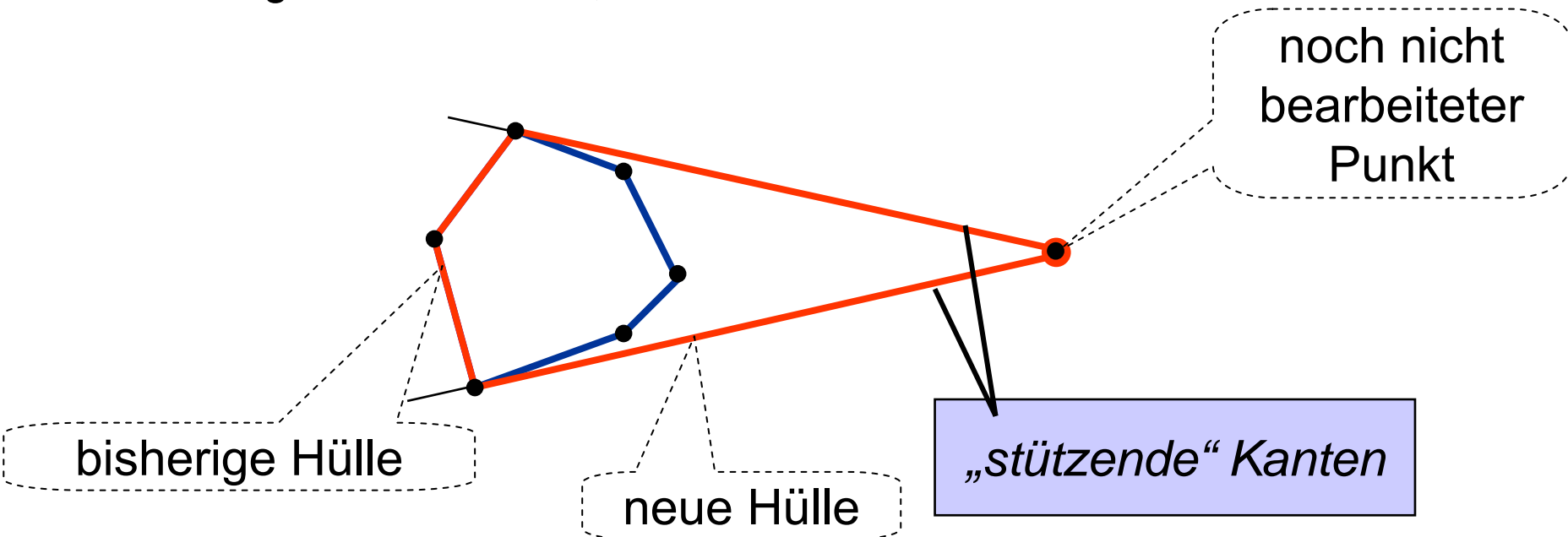


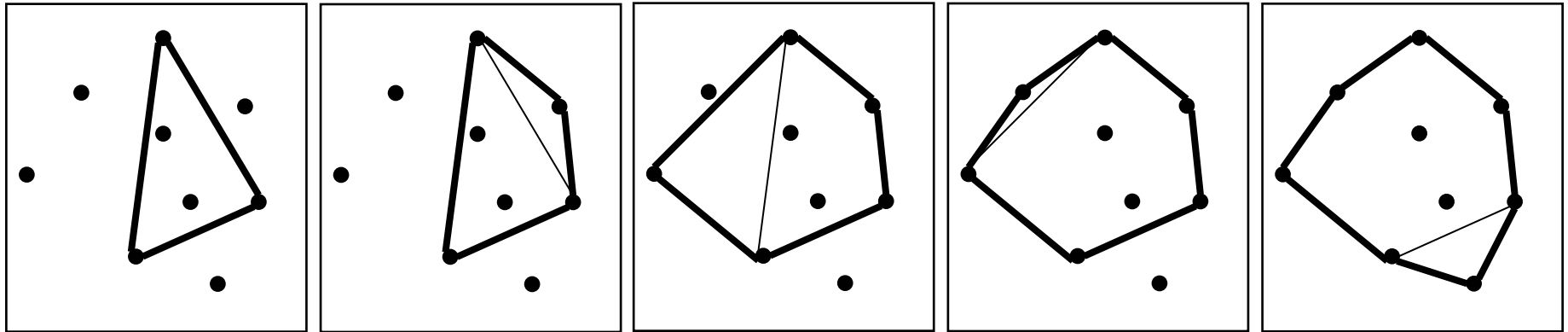
Anfang:

- Beginne mit Polygon aus beliebigen drei Punkten der gegebenen Punktmenge.

Schritt:

- Wähle beliebigen noch nicht bearbeiteten Punkt.
- Liegt er außerhalb des bisher erzeugten Polygons, dann modifiziere die bisher gefundene Hülle, so dass sie den neuen Punkt enthält.





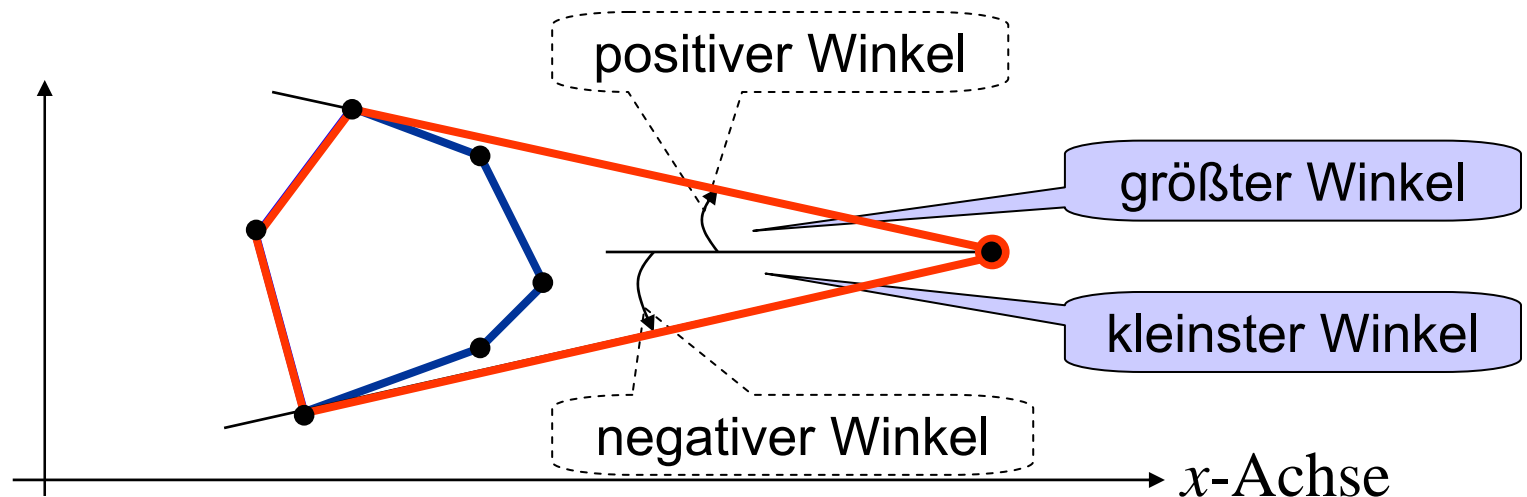
beginne mit beliebigem Dreieck

für alle Punkte P

wenn P nicht innerhalb der aktuellen Hülle,

dann füge Kante zum bisherigen Hüllpunkt mit dem kleinsten Winkel zu P und zum Punkt mit dem größten Winkel zu P ein;

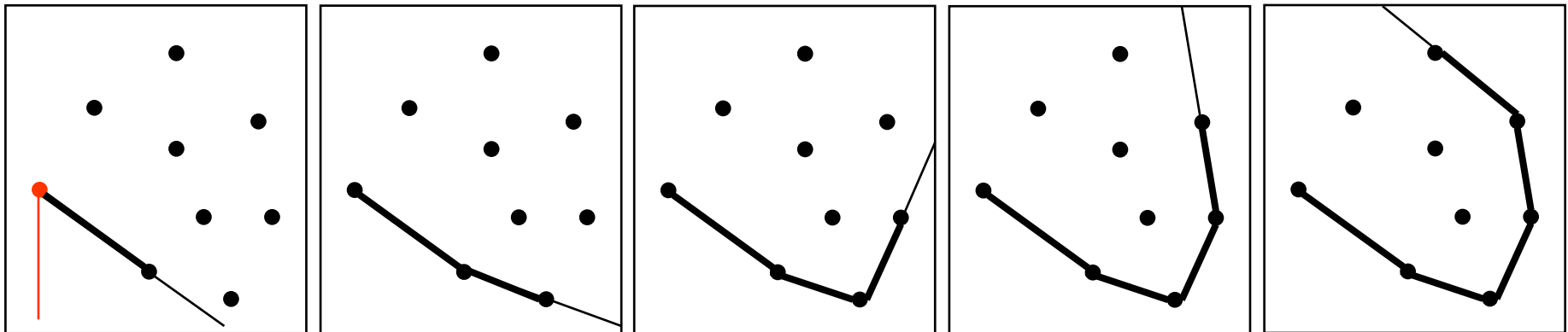
entferne alle alten Kanten im Inneren der neuen Hülle



Aufwand: Für alle Punkte alle Winkel zu anderen Punkten, also $O(n^2)$.

Anfang: Beginne mit einem äußersten Punkt (z.B. äußerst links) und Schnur nach unten.

Schritt: Verbinde immer zu dem Punkt, den bei Drehung gegen den Uhrzeigersinn die Einwickelschnur als nächstes erreicht (stützende Kante).



Aufwand: Wie beim Ausdehnungsalgorithmus $O(n^2)$
Winkelberechnungen zum Finden der stützenden Kanten.

beginne mit einem äußersten Punkt P_1 (z.B. äußerst rechts unten)
 berechne die Winkel der Verbindungen aller Punkte zu P_1 mit der x-Achse
 sortiere alle Punkte entsprechend dieser Winkel in eine Liste

für alle Punkte P_i in der Liste

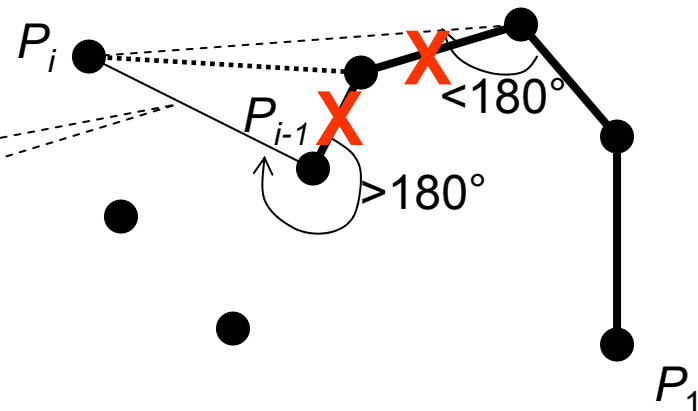
falls P_i von P_{i-1} über Innenwinkel $\leq 180^\circ$ erreichbar

füge neue Kante von P_{i-1} zu P_i hinzu

sonst

entferne P_{i-1} aus der Hülle, ebenso alle P_{i-k}

von denen P_i über Innenwinkel $> 180^\circ$ erreichbar



von P_i nach P_{i-1} entstünde ein Innenwinkel $> 180^\circ$, genauso von P_i nach P_{i-2} aber nicht zu P_{i-3} .

Aufwand für Durchsuchen nach Graham:

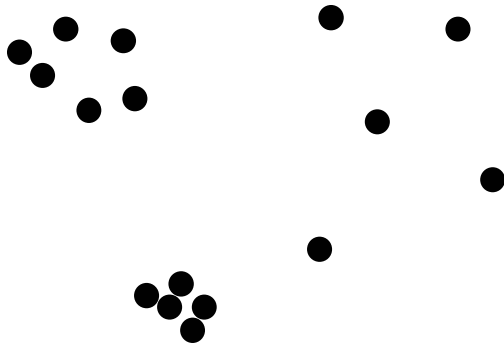
- Jeder Punkt wird höchstens einmal zur Hülle hinzugefügt und höchstens einmal wieder entfernt $\Rightarrow O(n)$.
- Das Sortieren der Winkel kostet $O(n \log n)$.
- Gesamt also $O(n \log n)$.



Ballung (clustering)

Gegeben: Punktmenge

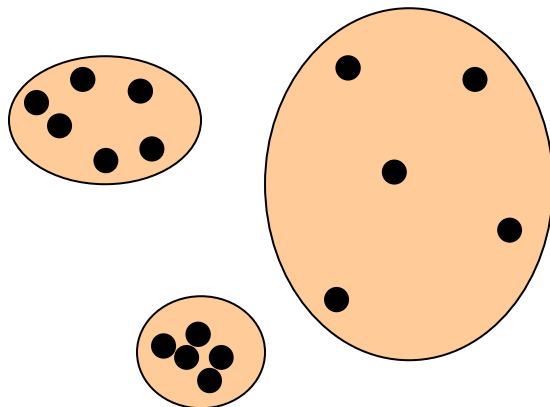
Gesucht: Einteilung in „homogene“ Klassen



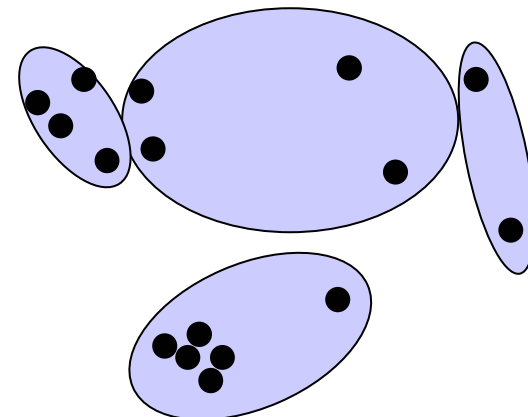
In wie viele Klassen würden Sie diese Punkte einteilen?

Warum?

besser so



oder so



Zweck

- Ohne a-priori-Wissen bestimmen, was zusammen gehört bzw. ähnlich ist.

Scharfe Klassifikation

- Punkte werden zu Klassen zusammengefasst, so dass jeder Punkt genau zu einer Klasse gehört.

Der Heterogenitätsindex einer Klasse gibt an, wie gut die Punkte einer Klasse zusammenpassen

- z.B. durchschnittlicher Abstand zwischen allen Punkten der Klasse

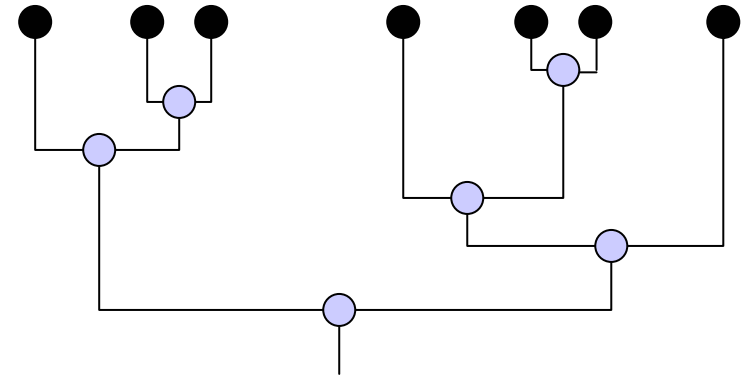
Der Güteindex einer Klassifikation bewertet eine Klassifikation

- z.B. Summe über alle Heterogenitätsindizes der enthaltenen Klassen



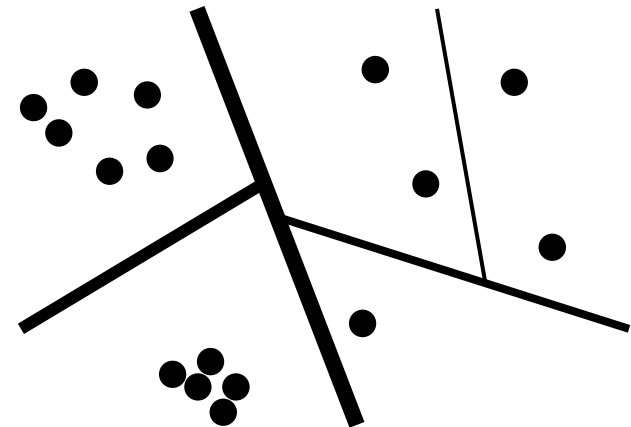
agglomerativ

- Zu Beginn enthält jede Klasse genau einen Punkt.
- Füge in jedem Schritt die zwei am wenigsten entfernten Klassen zusammen.



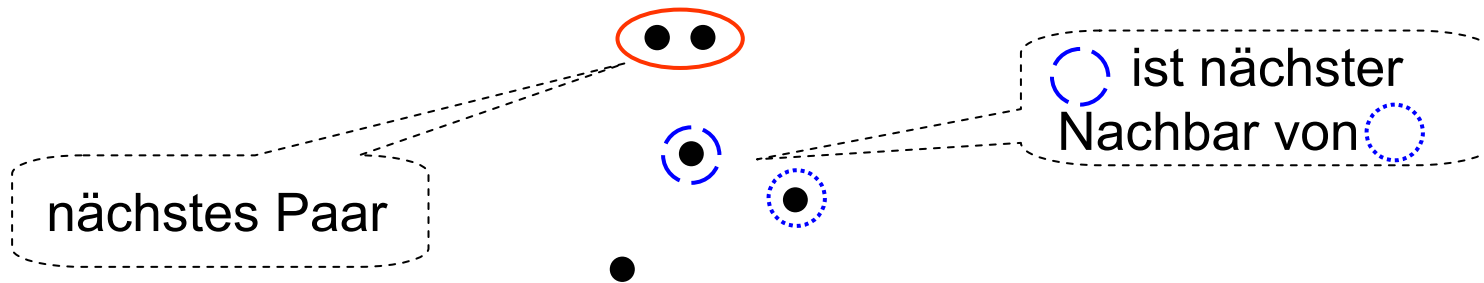
divisiv

- Beginne mit einer Klasse, die alle Punkte enthält.
- Teile in jedem Schritt eine Klasse in zwei Unterklassen.
- Für jede pro Schritt entstehende Klassifikation wird ein Güteindex berechnet.
- Wähle die Klassifikation, bei der sich im nächsten Schritt der Güteindex nur noch gering verbessern würde.



Beispiel agglomerative Ballung

- Finde Nächstes Paar: Diejenigen zwei Punkte einer Punktmenge mit dem geringsten gegenseitigen Abstand.
- Beruht auf Nächster Nachbar: Punkt, der aufgrund eines Abstandmaßes am nächsten bei einem gegebenen Punkt liegt.



Ein erster, einfacher Algorithmus:

Berechne alle Distanzen
wähle das Paar mit der kleinsten Distanz

Aufwand: Berechnen von $n*(n-1)/2$ Distanzen, also $O(n^2)$.



Ein teile-und-herrsche-Algorithmus:

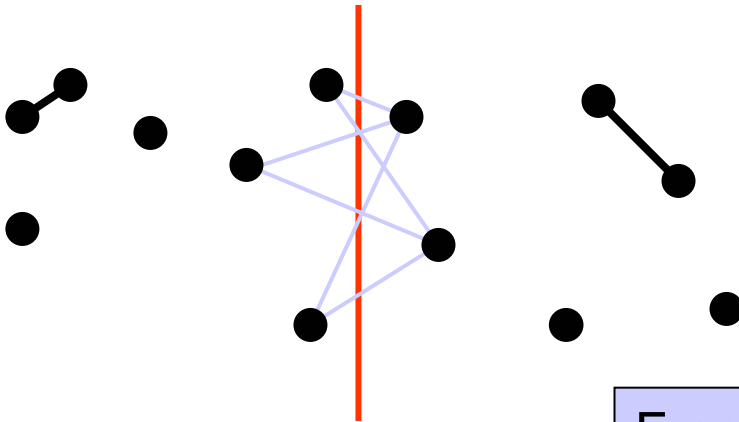
- teile den Raum in zwei Teile ein
- berechne für jede Hälfte das nächste Paar
- überprüfe alle in Frage kommenden „grenzüberschreitenden“ Paare
- wähle von allen Paaren das mit der kleinsten Distanz

- Jede Trennebene wird immer senkrecht zur x -Achse gewählt.
⇒ leichtes Feststellen, auf welcher Seite ein Punkt liegt.
- Einmaliges Sortieren aller x -Koordinaten genügt, um immer eine Trennebene wählen zu können, so dass auf beiden Seiten gleich viele (± 1) Punkte liegen.



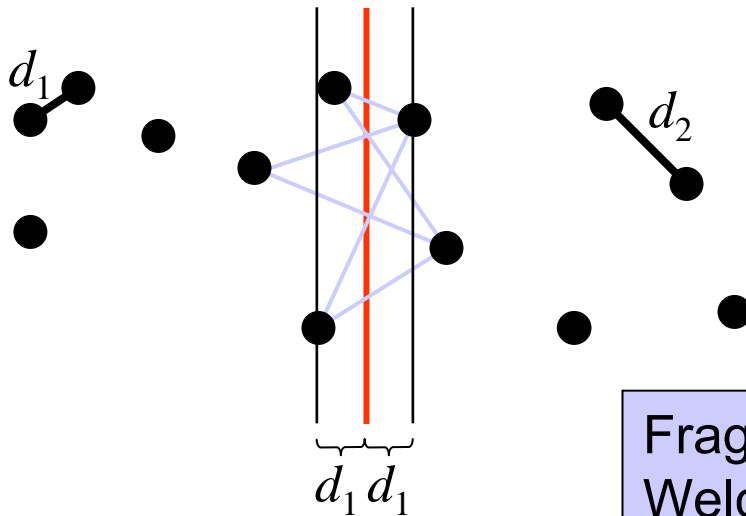
Ein teile-und-herrsche-Algorithmus

- teile den Raum in zwei Teile ein
- berechne für jede Hälfte das nächste Paar
- überprüfe alle in Frage kommenden „grenzüberschreitenden“ Paare
- wähle von allen Paaren das mit der kleinsten Distanz



Frage:
Welche „grenzüberschreitenden“ Paare
kommen in Frage?

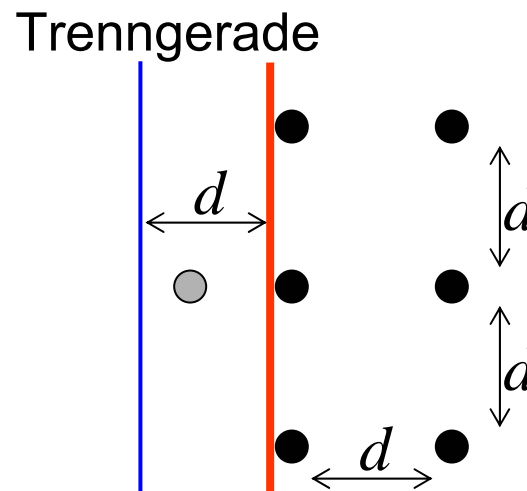
Nur solche, deren Abstand zur Grenze kleiner ist, als der kleinste bisher gefundene Abstand der nicht grenzüberschreitenden Paare



Frage:
Welche „grenzüberschreitenden“ Paare kommen in Frage?

Aufwandsabschätzung

- Sei d die minimale Distanz aus beiden Hälften.
- Es genügt, für jeden Punkt mit Abstand kleiner d von der Trennebene *maximal* 6 Punkte aus der anderen Hälfte zu betrachten.



Aufwandsabschätzung

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

Aufwand für jede Hälfte

$$T(2) = 1$$

Nur 2 Punkte,
dann trivial.

Sortieren der Punkte nach der y -Koordinate für
den Abstandstest grenzüberschreitender Paare

Etwas Mathematik liefert $T(n) = O(n \log^2 n)$

