

# Übersicht

- 1 Einführung
- 2 Suchen und Sortieren
- 3 Graphalgorithmen
- 4 Algorithmische Geometrie**
- 5 Textalgorithmen
- 6 Paradigmen

# Algorithmische Geometrie

Probleme der Algorithmischen Geometrie haben üblicherweise diese Eigenschaften:

- Die Eingabe besteht aus Punkten, Segmenten, Kreisbögen usw. in der euklidischen Ebene.
- Die Fragestellung ist relativ einfach.
- Sehr große Eingaben müssen bewältigt werden.

Anwendungen beispielsweise im VLSI-Design.

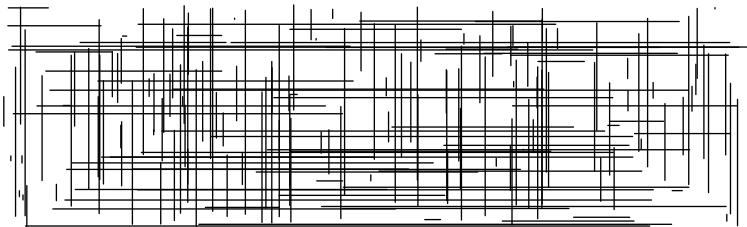
# Übersicht

- 4 Algorithmische Geometrie
  - Segmentschnitte
    - Die Technik der Sweepline
    - Nächste Nachbarn
    - Konvexe Hülle

# Schnitte von Segmenten

Eingabe: Horizontale und vertikale Segmente

Ausgabe: Paare von Segmenten, die sich schneiden



Naiver Algorithmus:

Teste alle Paare, ob sie sich schneiden.

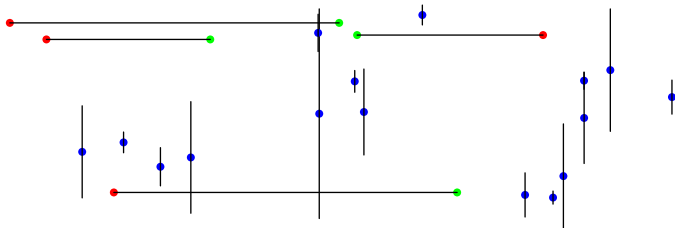
Laufzeit:  $\Theta(n^2)$

Variante: Finde heraus, **ob** es einen Schnitt gibt.

# Übersicht

- 4 Algorithmische Geometrie
  - Segmentschnitte
  - Die Technik der Sweepline
  - Nächste Nachbarn
  - Konvexe Hülle

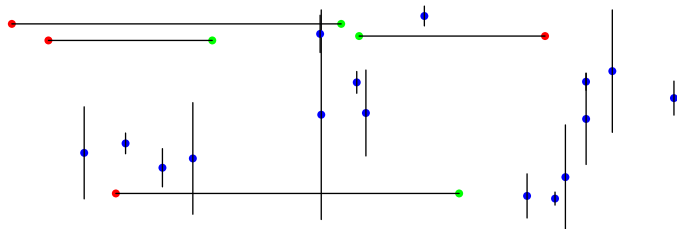
# Sweepline-Algorithmen



- Interessante Punkte: Nach  $x$ -Koordinate sortieren
- Es gibt eine aktive Menge von Segmenten
- Eine imaginäre vertikale Linie bewegt sich von links nach rechts
- Roter Punkt: Segment in aktive Menge aufnehmen
- Grüner Punkt: Segment aus aktiver Menge entfernen
- Blauer Punkt: Segment mit aktiver Menge vergleichen

Suche nur Schnitte zwischen horizontalem und vertikalem Segment.

# Sweepline-Algorithmen

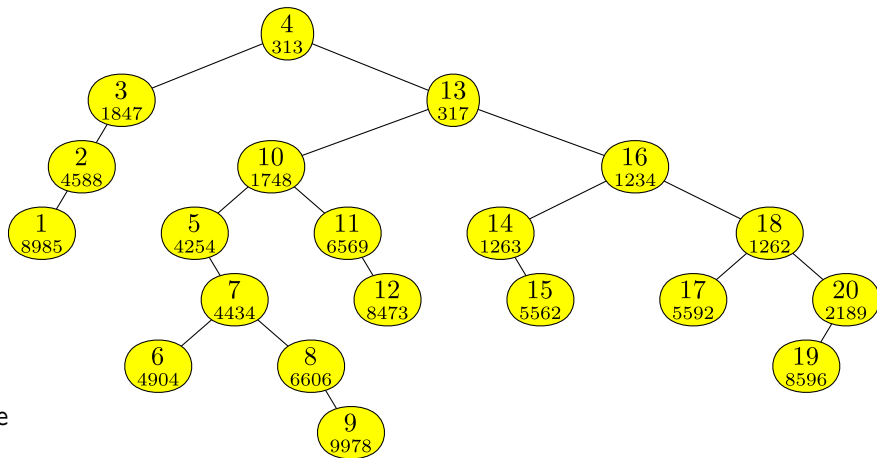


- Wie finden wir bei einem vertikalen Segment die geschnittenen horizontalen Segmente?
- Welche Datenstruktur für die aktive Menge?

Lösung: Speichere  $y$ -Koordinaten  $Y$  der aktiven Menge in balanziertem Suchbaum.  
Gibt es einen Schnitt?  $\rightarrow O(\log |Y|)$  Schritte

Alle Schnitte  $S$  ausgeben:  $O(\log |Y| + |S|)$  Schritte

In aktive Menge einfügen oder löschen:  $O(\log |Y|)$  Schritte



Aufgabe: Finde  
y-Koordinaten

mit  $a \leq y \leq b$ . Laufzeit:  $O(\log |Y| + |S|)$

- ① Finde größtes Element, daß kleiner oder gleich  $a$  ist.
- ② Gehe von dort aus Elemente aufsteigend durch
- ③ Beende, wenn  $b$  überschritten



# Range-Search

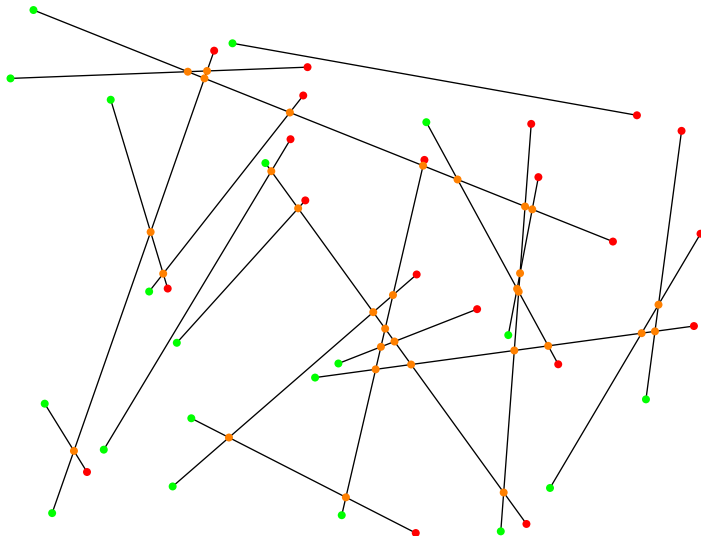
Operationen:

- 1 Einfügen einer Zahl  $x$
- 2 Löschen einer Zahl  $x$
- 3 Ausgabe aller gespeicherter Zahlen in  $[a, b]$

Welche Datenstrukturen sind geeignet?

- Arrays?
- Listen?
- AVL-Bäume?
- Splay-Bäume?
- (2, 3)-Bäume?
- Treaps?
- Skiplists?
- Hashtabellen?

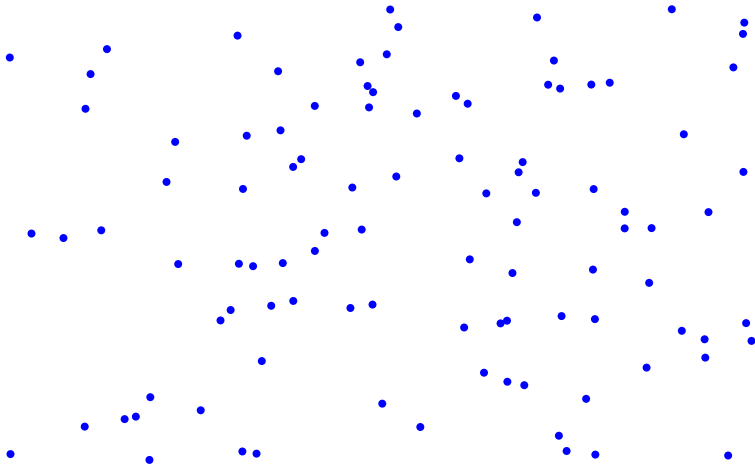
# Das allgemeine Segmentschnittproblem



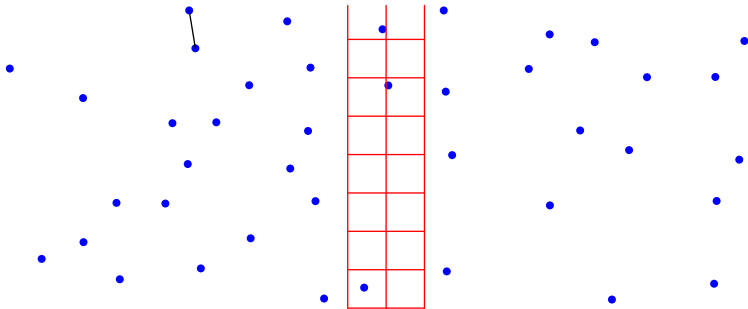
# Übersicht

- 4 Algorithmische Geometrie
  - Segmentschnitte
  - Die Technik der Sweepline
  - **Nächste Nachbarn**
  - Konvexe Hülle

# Nächste Nachbarn



# Nächste Nachbarn – Divide-and-Conquer



- 1 Sortiere nach  $x$ -Koordinate
- 2 Teile in linke und rechte Hälfte
- 3 Finde rekursiv nächste Nachbarn links und rechts (Abstand  $d$ )
- 4 Finde nächste Nachbarn in einem Streifen der Breite  $2d$  um die Trennlinie

## Nächste Nachbarn – Analyse

Es sei  $T(n)$  die Zeit, um  $n$  Punkte zu bearbeiten, nachdem sie nach  $x$ -Koordinate sortiert wurden.

Nehmen wir an,  $n$  sei eine Zweierpotenz.

Wir müssen zwei Probleme der Größe  $n/2$  lösen und anschließend die Punkte im Mittelstreifen behandeln.

Es ergibt sich diese Rekursionsgleichung:

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

Dies ist die selbe Rekursionsgleichung wie für Mergesort.

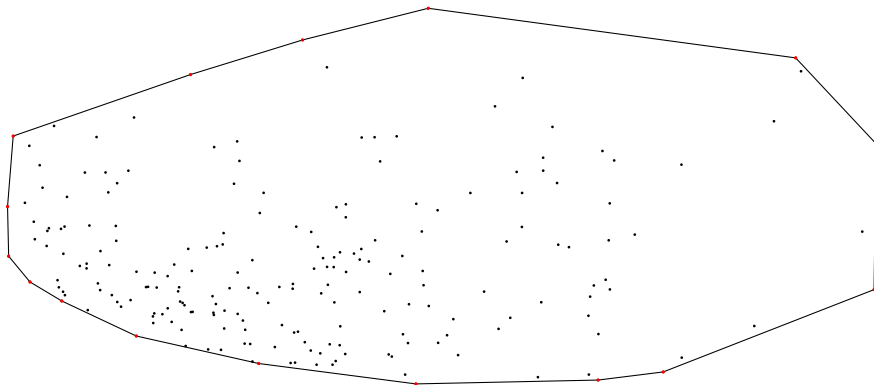
Laufzeit:  $O(n \log n)$

# Übersicht

## 4 Algorithmische Geometrie

- Segmentschnitte
- Die Technik der Sweepline
- Nächste Nachbarn
- Konvexe Hülle

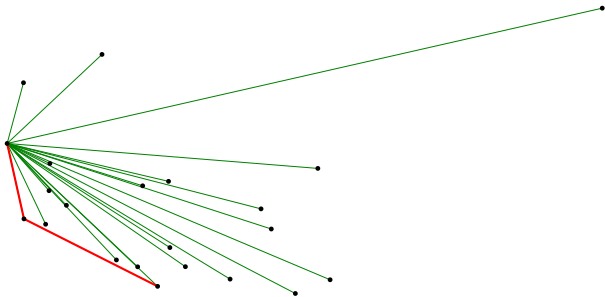
# Konvexe Hülle



Finde das kleinste konvexe Polygon, das alle Punkte enthält.



# Graham Scan-Algorithmus



- Gehe Punkte nach Winkel vom linkesten durch (und Entfernung)
- Bilde ein Polygon aus diesen Punkten
- Lösche wiederholt vorletzten Punkt falls rechts abgebogen

```
public List<Point> hull() {  
    List<Point> hull = new ArrayList<Point>(points);  
    Point z0 = hull.get(0); // left most point  
    for(Point z : hull) if(z.x < z0.x) { z0 = z; }  
    Collections.sort(hull, new AngleComparator(z0));  
    Stack<Point> st = new Stack<Point>(); hull.add(z0);  
    for(Point z : hull) {  
        st.push(z);  
        while(st.size() > 2) {  
            Point c = st.pop(), b = st.pop(), a = st.pop();  
            st.push(a);  
            if(Point.turnLeft(a, b, c)) { st.push(b); st.push(c); break; }  
            else st.push(c);  
        }  
    }  
    st.pop();  
    return new ArrayList<Point>(st);  
}
```