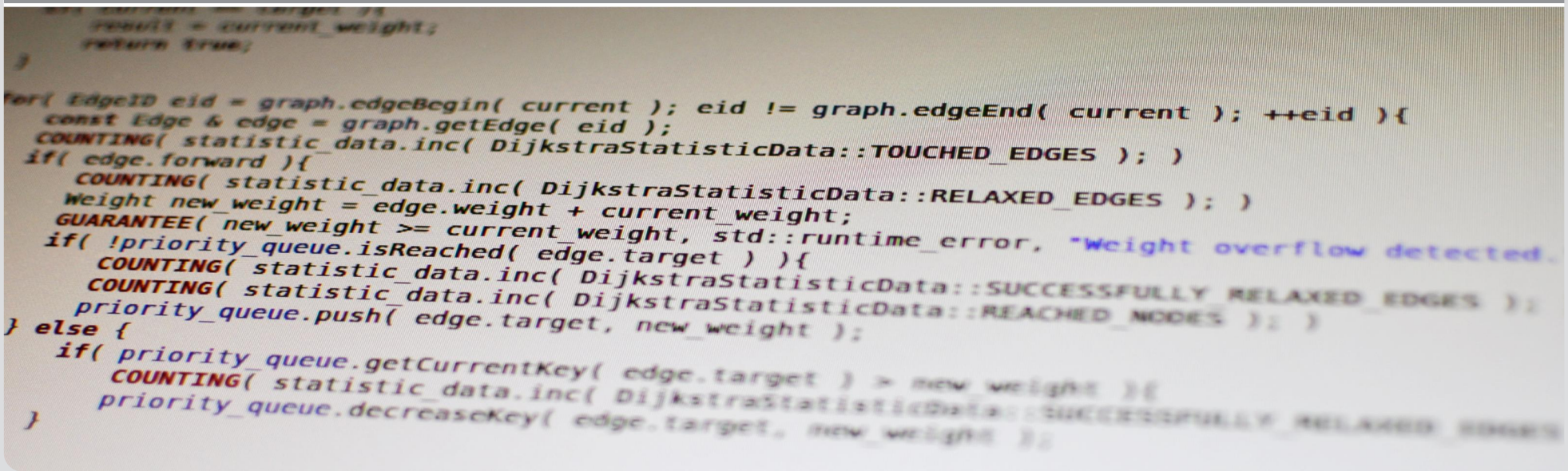
Karlsruher Institut für Technologie

Übung 9 – Algorithmen II

Moritz Laupichler, Hans-Peter Lehmann – *{moritz.laupichler, hans-peter.lehmann}@kit.edu*

`http://algo2.iti.kit.edu/AlgorithmenII_WS22.php`

Institut für Theoretische Informatik - Algorithmik II



Themenübersicht

- Geometrische Algorithmen
  - Interval Tree
  - Skyline-Problem
  - Linienschnitt
  - Graham's Scan
- Besprechung ÜB 8

Geometrische Algorithmen

- geometrische Varianten bekannter Probleme
  - Spezialfälle oft einfacher
  - allgemeines TSP: nicht approximierbar wenn P $\neq$ NP
  - metric TSP: 1.5-Approximation
  - euclidean TSP: ϵ-Approximation
    - Laufzeit in $O\left(n(\log n)^{O(\frac{1}{\epsilon}\sqrt{d})^{d-1}}\right)$

- geometrisch motivierte Probleme
  - Punktlokalisierung
  - Bewegungsplanung Robotik
  - Sichtbarkeitsgraphen/Prüfung
  - Streckenschnitt
  - ...

Geometrische Methoden

Datenstrukturen

- Baumstrukturen
  - Interval Tree — 1-dim
  - Quad Tree — 2-dim
  - k-d-Tree — n-dim
  - Wavelet-Tree — 2-dim
- Facetten
  - Delaunay Triangulierung
  - Voronoi Diagramm

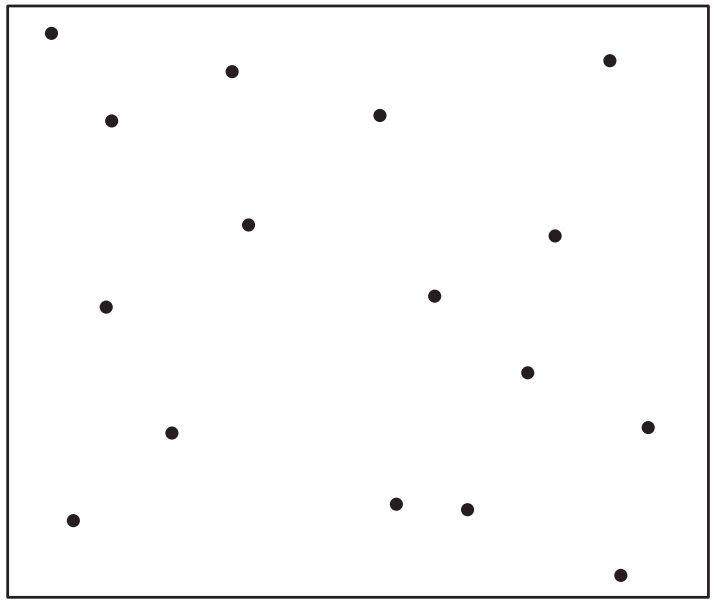
Strukturierter Zugriff

- Sweepline
  - sortiert
  - topologisch sortiert

Geometrische Methoden

Datenstrukturen

- Baumstrukturen
  - Interval Tree — 1-dim
  - Quad Tree — 2-dim
  - k-d-Tree — n-dim
  - Wavelet-Tree — 2-dim
- Facetten
  - Delaunay Triangulierung
  - Voronoi Diagramm



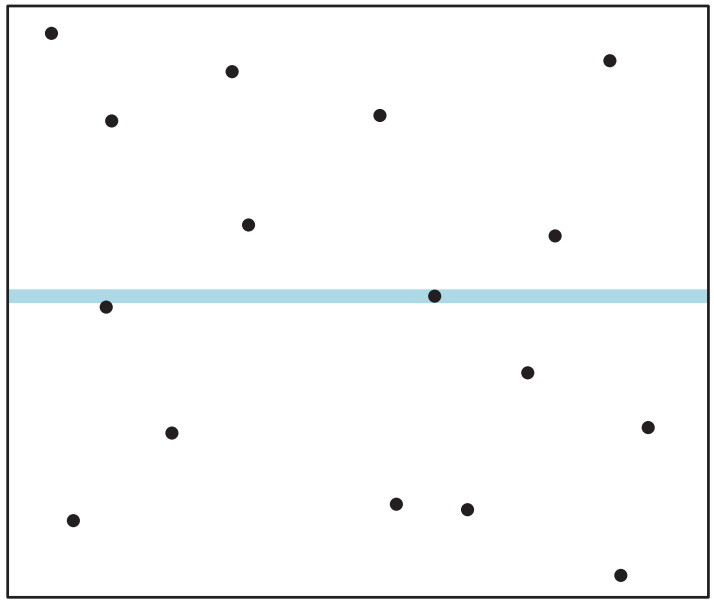
Strukturierter Zugriff

- Sweepline
  - sortiert
  - topologisch sortiert

Geometrische Methoden

Datenstrukturen

- Baumstrukturen
  - Interval Tree — 1-dim
  - Quad Tree — 2-dim
  - k-d-Tree — n-dim
  - Wavelet-Tree — 2-dim
- Facetten
  - Delaunay Triangulierung
  - Voronoi Diagramm



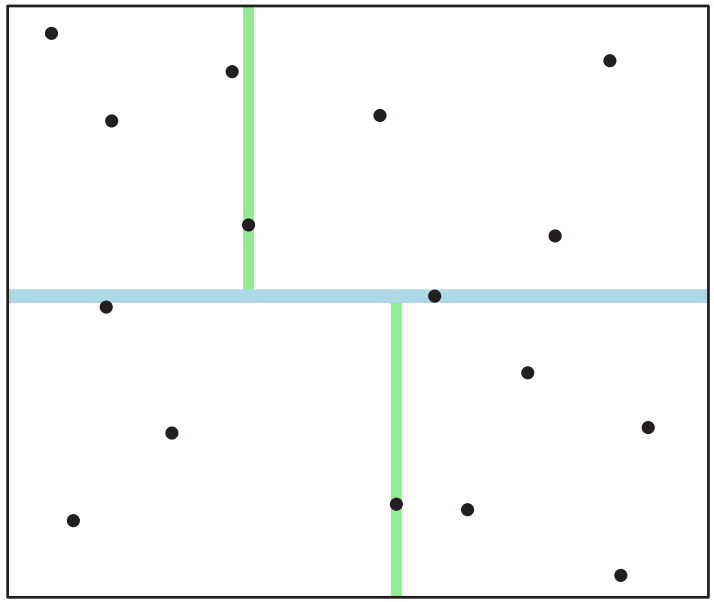
Strukturierter Zugriff

- Sweepline
  - sortiert
  - topologisch sortiert

Geometrische Methoden

Datenstrukturen

- Baumstrukturen
  - Interval Tree — 1-dim
  - Quad Tree — 2-dim
  - k-d-Tree — n-dim
  - Wavelet-Tree — 2-dim
- Facetten
  - Delaunay Triangulierung
  - Voronoi Diagramm



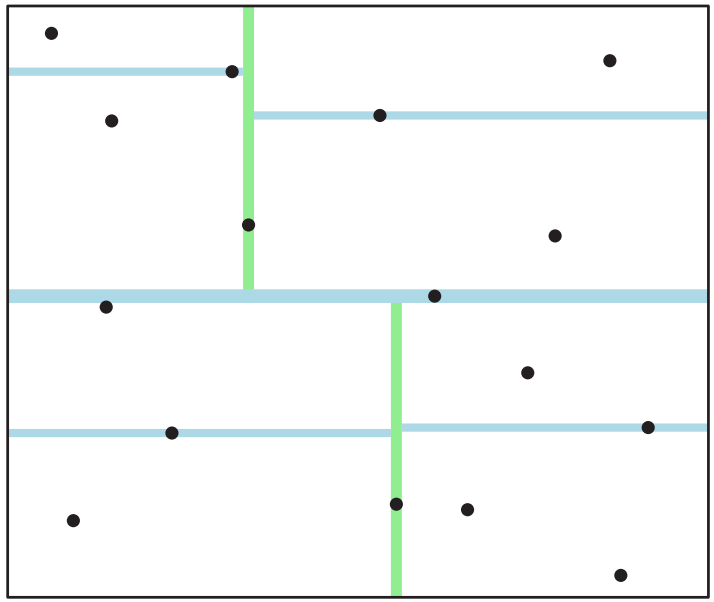
Strukturierter Zugriff

- Sweepline
  - sortiert
  - topologisch sortiert

Geometrische Methoden

Datenstrukturen

- Baumstrukturen
  - Interval Tree — 1-dim
  - Quad Tree — 2-dim
  - k-d-Tree — n-dim
  - Wavelet-Tree — 2-dim
- Facetten
  - Delaunay Triangulierung
  - Voronoi Diagramm



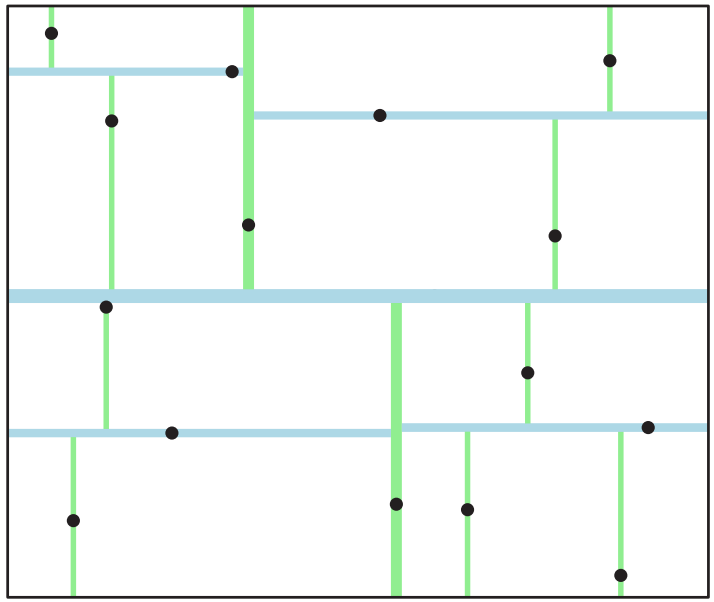
Strukturierter Zugriff

- Sweepline
  - sortiert
  - topologisch sortiert

Geometrische Methoden

Datenstrukturen

- Baumstrukturen
  - Interval Tree — 1-dim
  - Quad Tree — 2-dim
  - k-d-Tree — n-dim
  - Wavelet-Tree — 2-dim
- Facetten
  - Delaunay Triangulierung
  - Voronoi Diagramm



Strukturierter Zugriff

- Sweepline
  - sortiert
  - topologisch sortiert

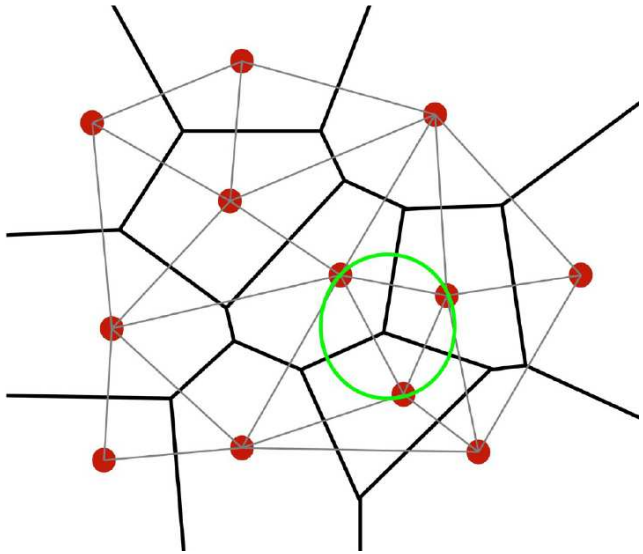
Geometrische Methoden

Datenstrukturen

- Baumstrukturen
  - Interval Tree — 1-dim
  - Quad Tree — 2-dim
  - k-d-Tree — n-dim
  - Wavelet-Tree — 2-dim
- Facetten
  - Delaunay Triangulierung
  - Voronoi Diagramm

Strukturierter Zugriff

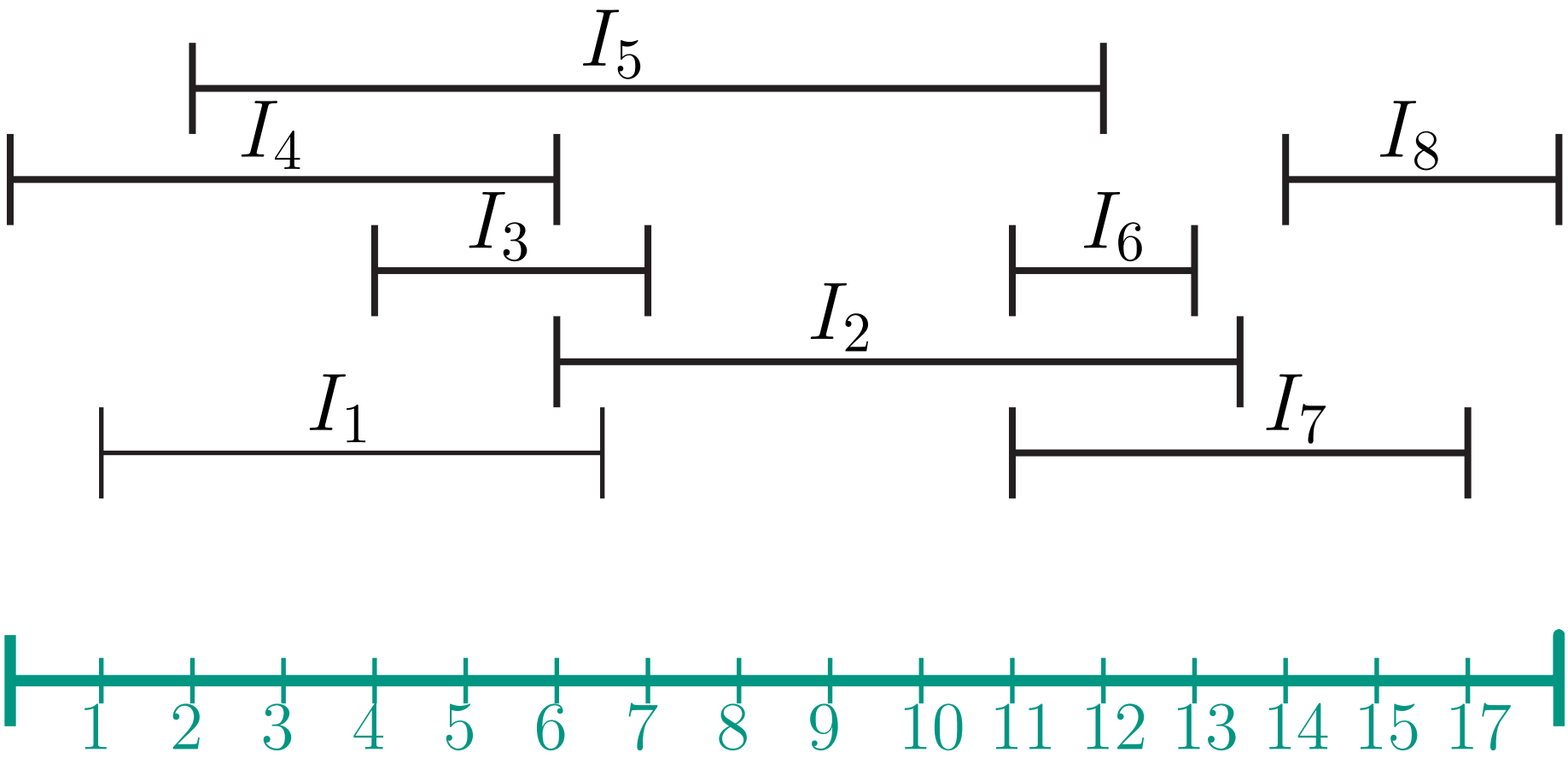
- Sweepline
  - sortiert
  - topologisch sortiert



https://i.stack.imgur.com/01H88.png

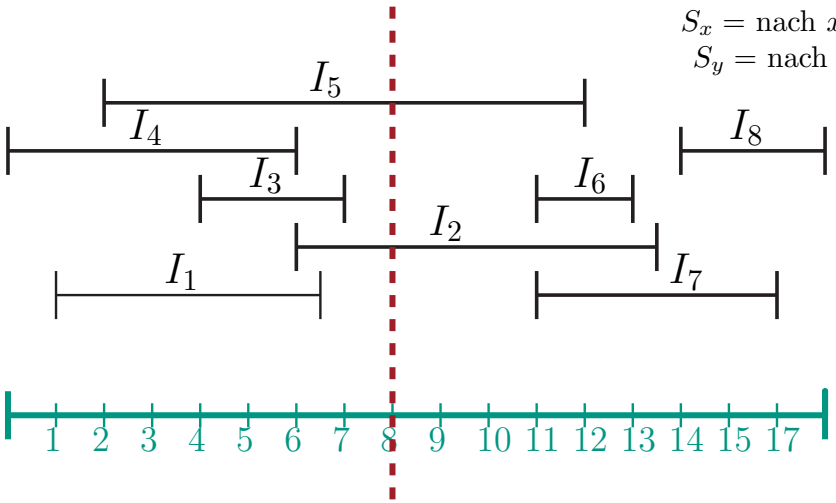
Interval Tree

Konstruktion



Interval Tree

Konstruktion

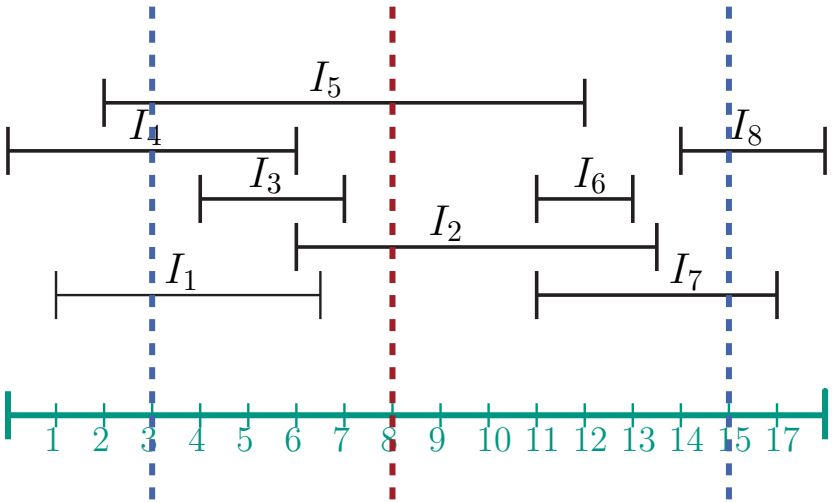


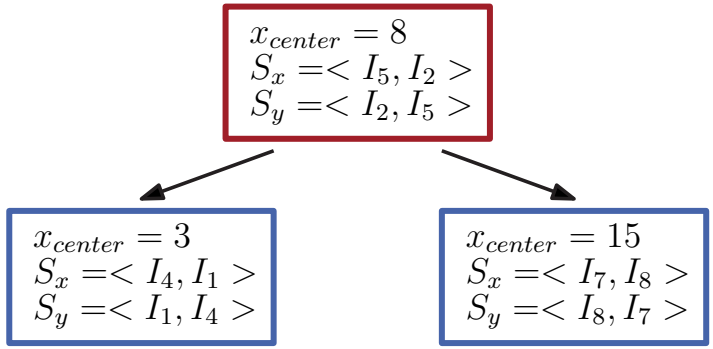
S_x = nach x aufsteigend sortierte Liste
S_y = nach y absteigend sortierte Liste

$x_{center} = 8$
$S_x =< I_5, I_2 >$
$S_y =< I_2, I_5 >$

Interval Tree

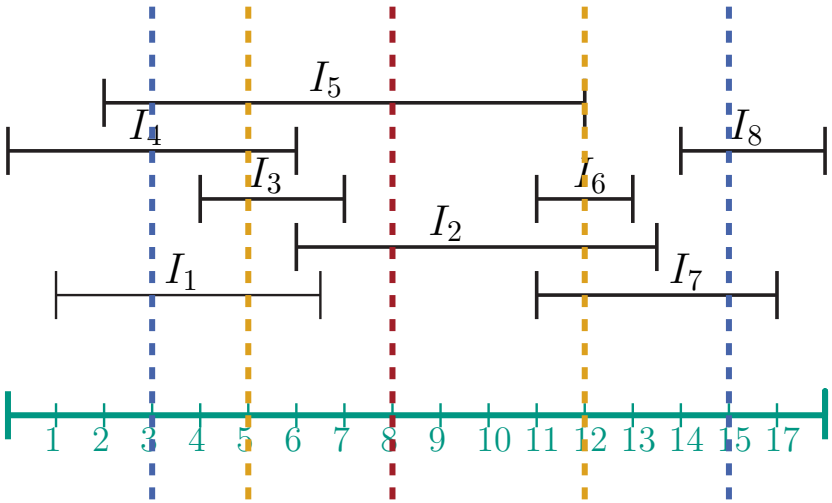
Konstruktion

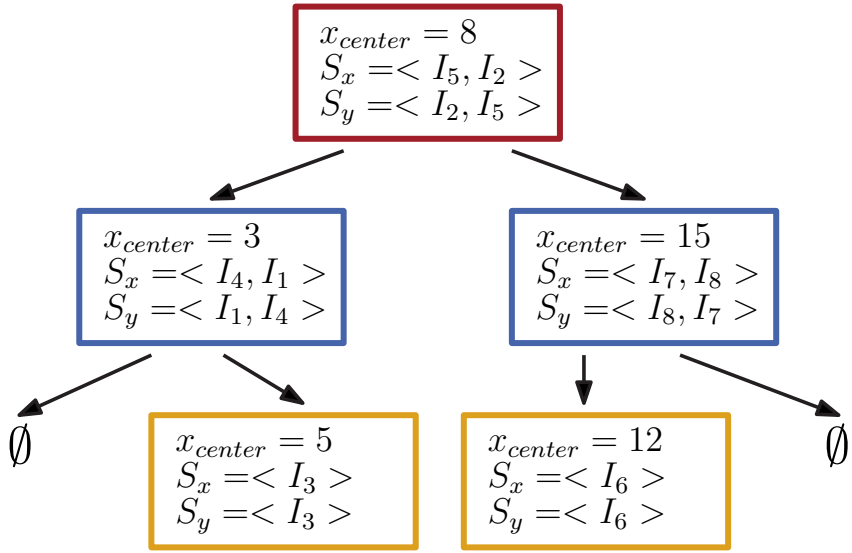




Interval Tree

Konstruktion

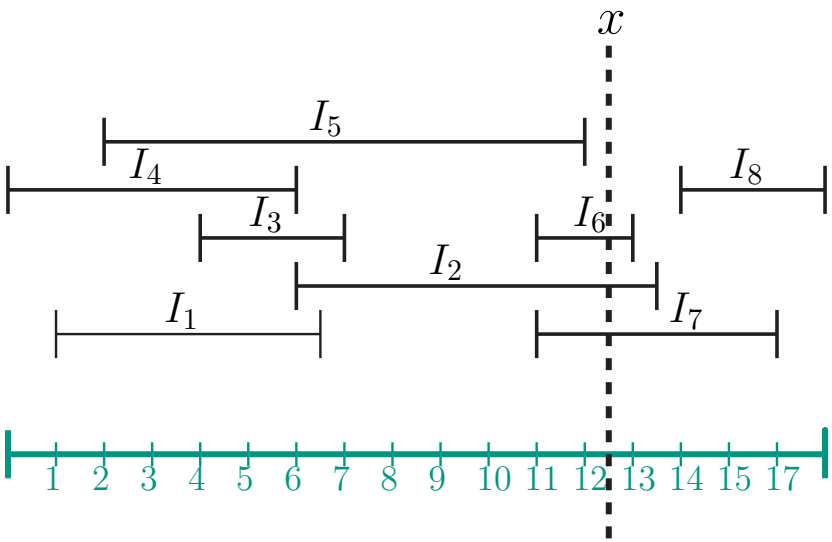


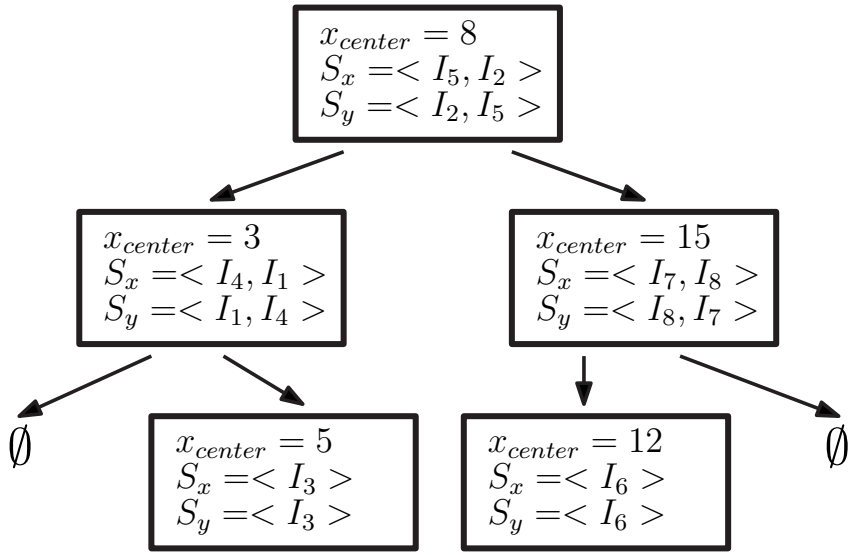


Interval Tree

Schnitt mit Punkt

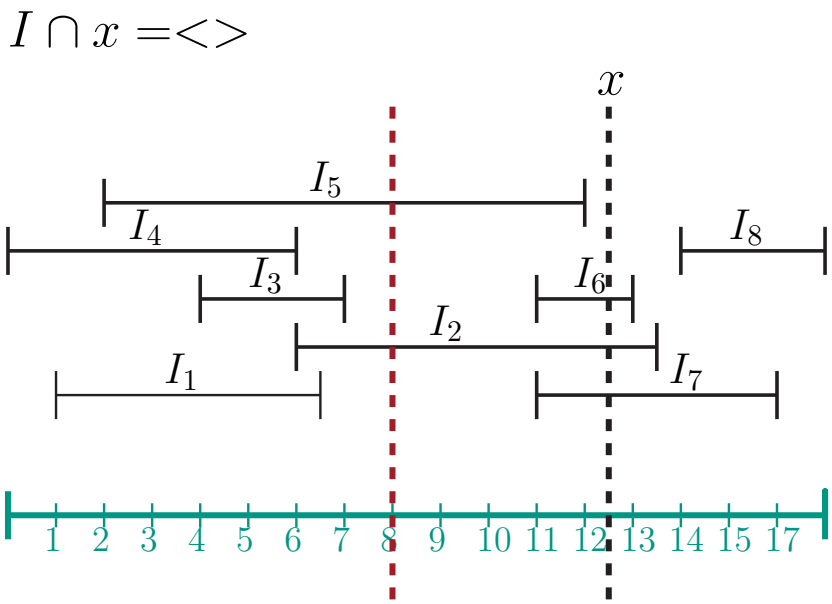
$I \cap x =<>$

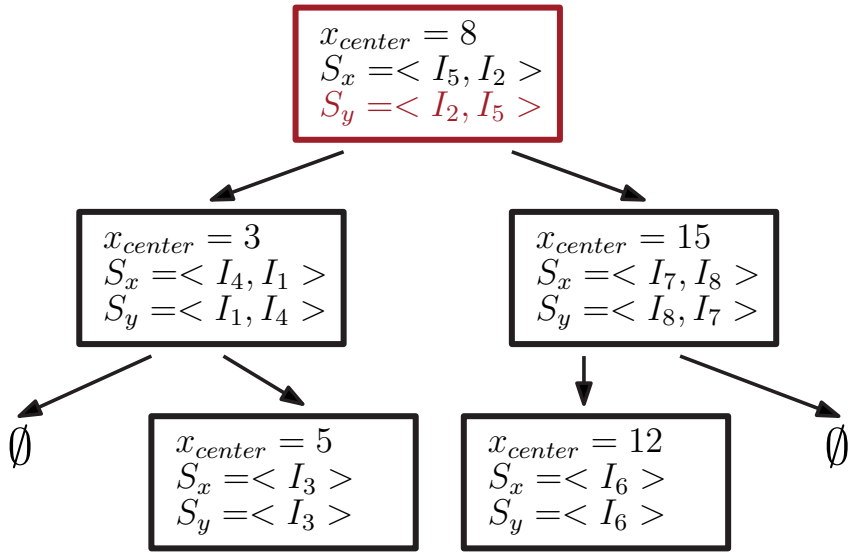




Interval Tree

Schnitt mit Punkt

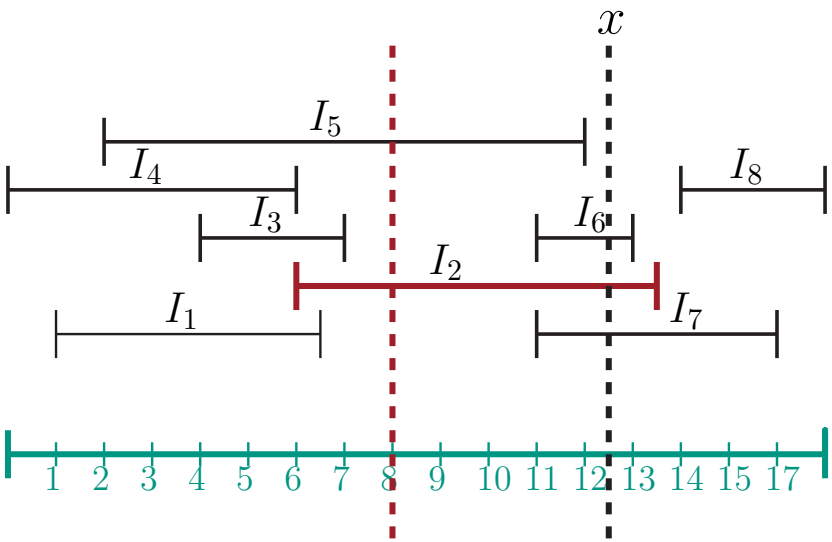


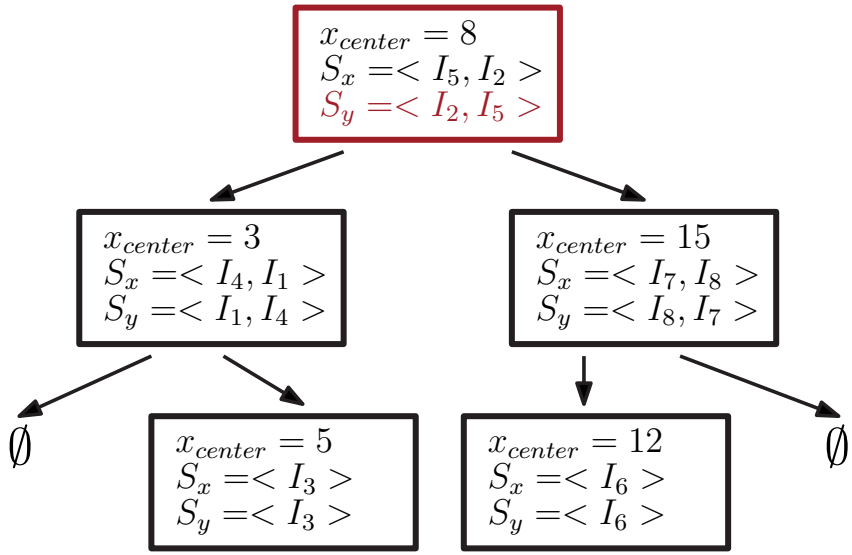


Interval Tree

Schnitt mit Punkt

$I \cap x =< I_2 >$

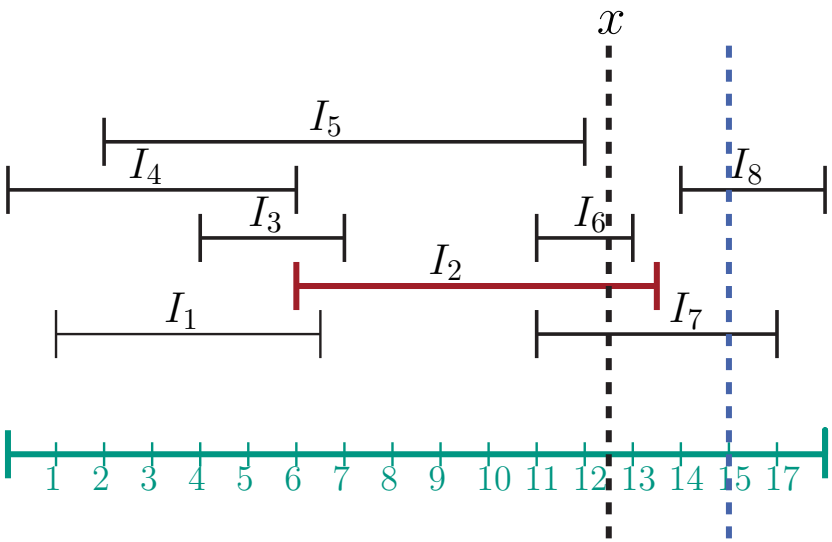


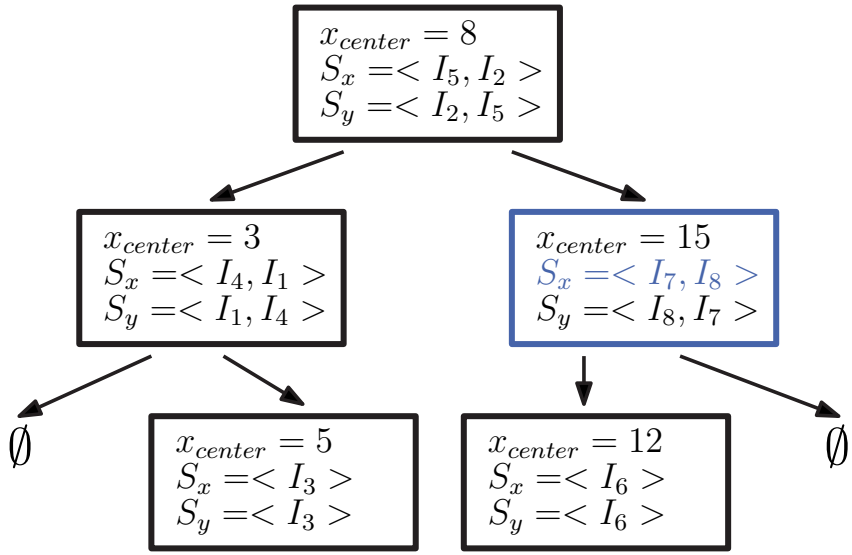


Interval Tree

Schnitt mit Punkt

$I \cap x =< I_2 >$

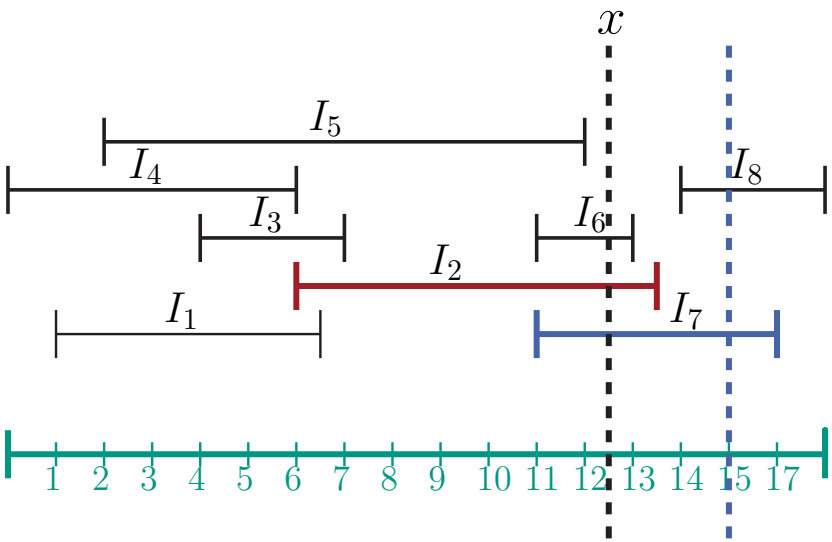


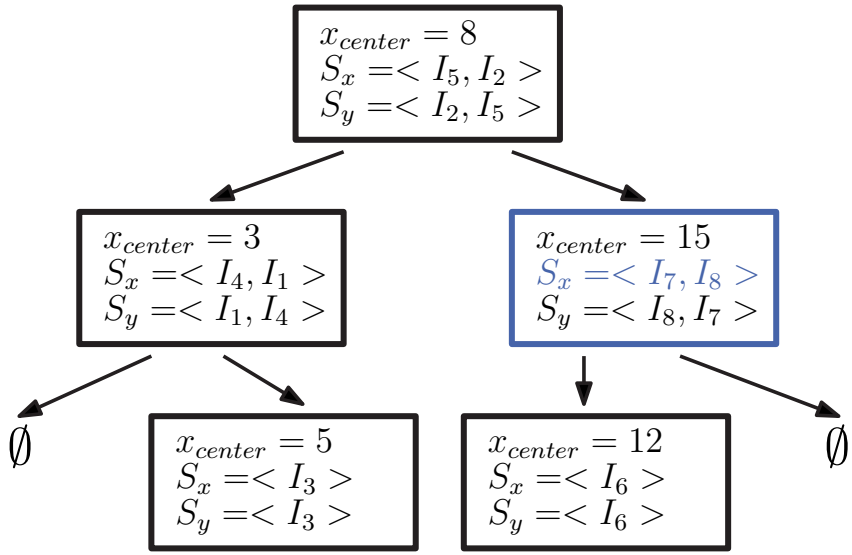


Interval Tree

Schnitt mit Punkt

$I \cap x =< I_2, I_7 >$

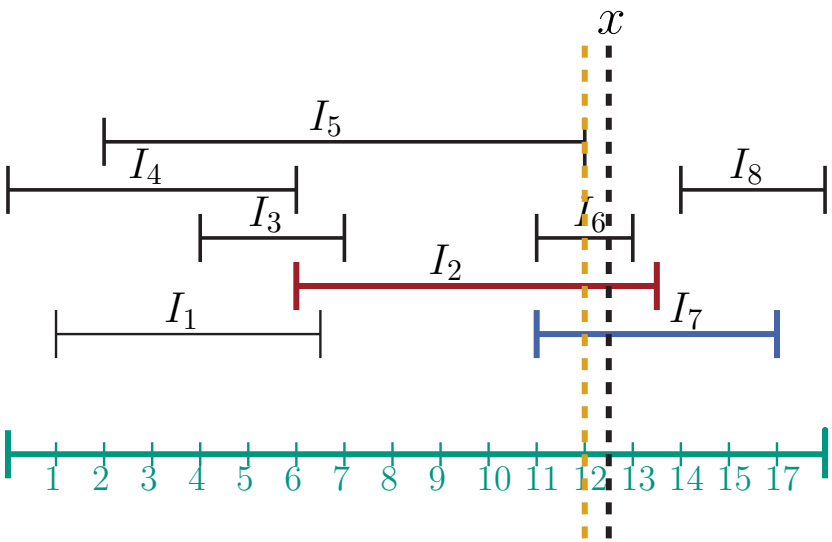


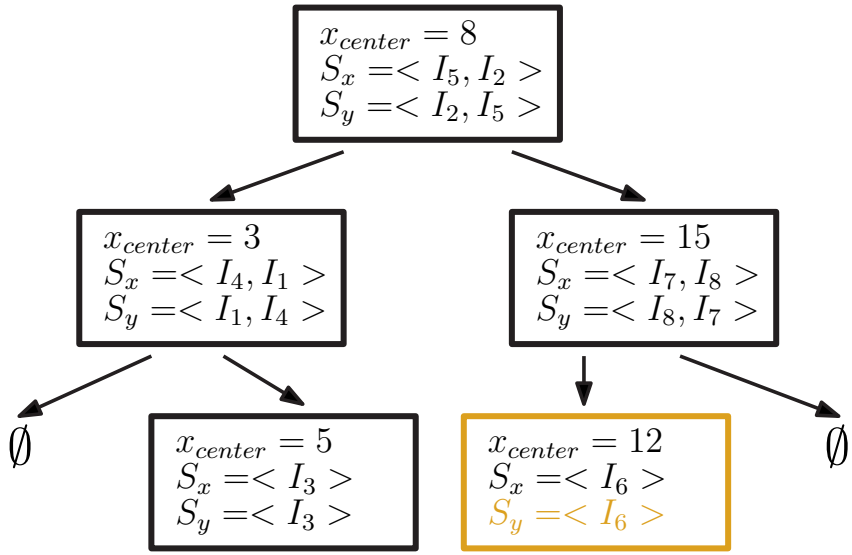


Interval Tree

Schnitt mit Punkt

$I \cap x =< I_2, I_7 >$

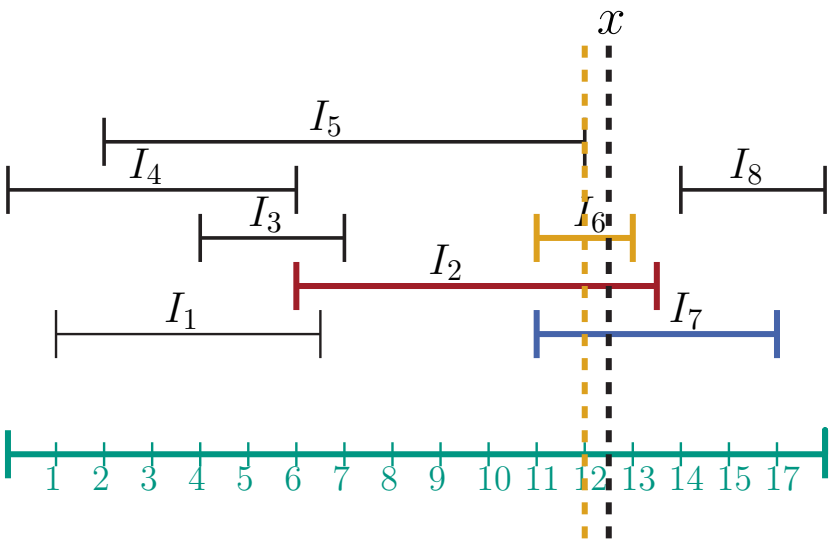


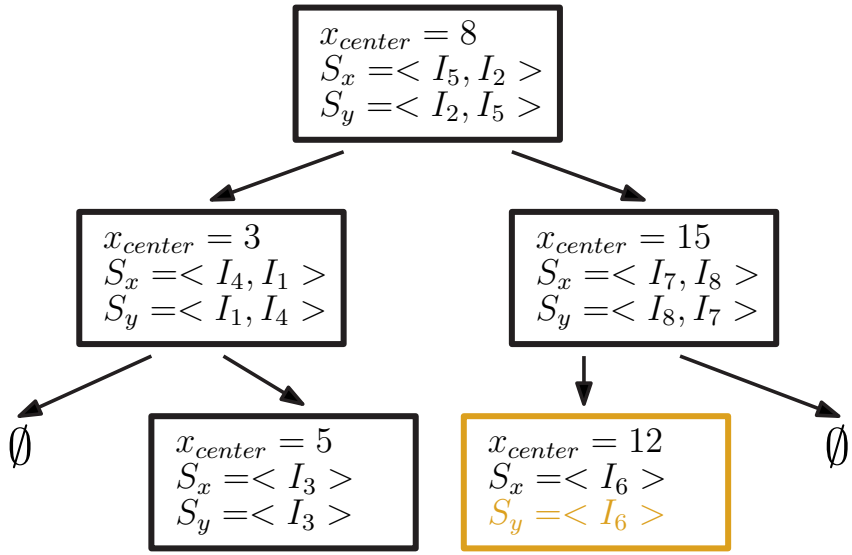


Interval Tree

Schnitt mit Punkt

$I \cap x =< I_2, I_7, I_6 >$

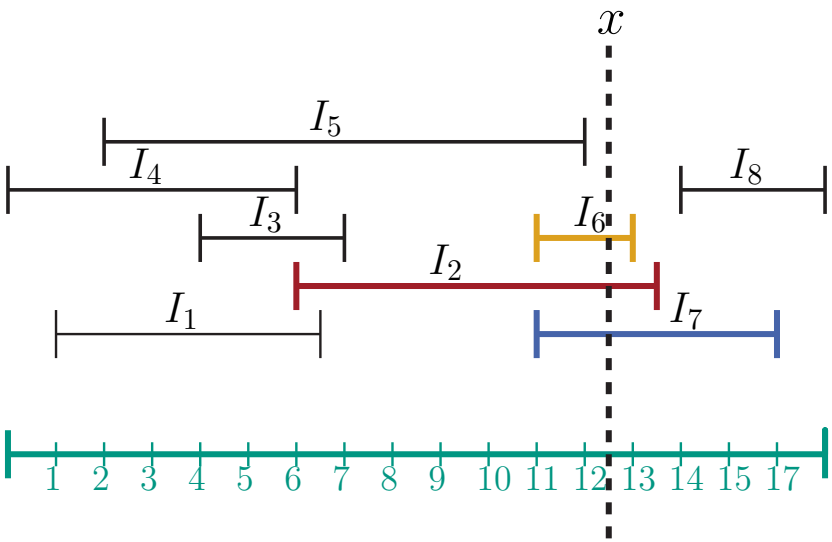


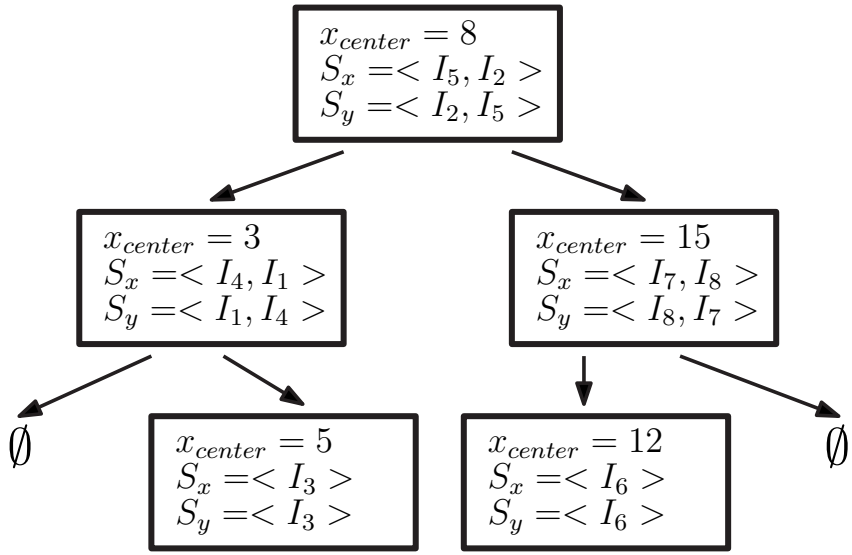


Interval Tree

Schnitt mit Punkt

$I \cap x =< I_2, I_7, I_6 >$





Sweep-Line

Allgemein

Idee

- strukturierte Abarbeitung eines Problems
- nutze Nähe aus
  - geometrisch nahe Objekte beeinflussen sich
  - geometrisch weit entfernte Objekte (nahezu) unabhängig

im Allgemeinen

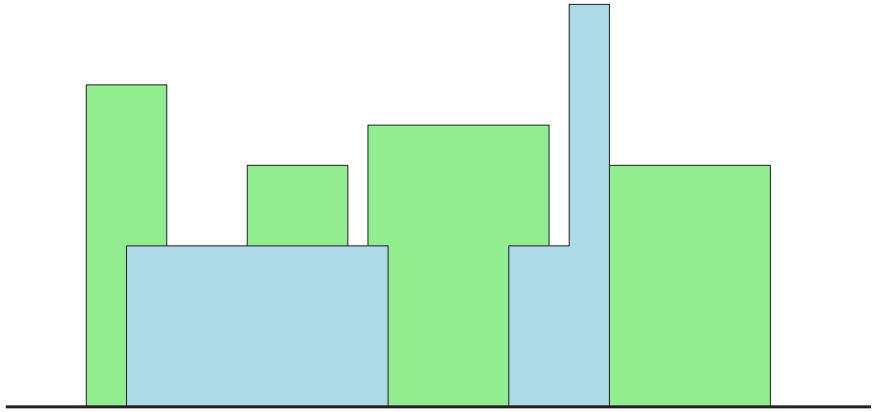
- reduziere n-dim $\rightarrow (n-1)$-dim

Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem
  Maximumsbildung



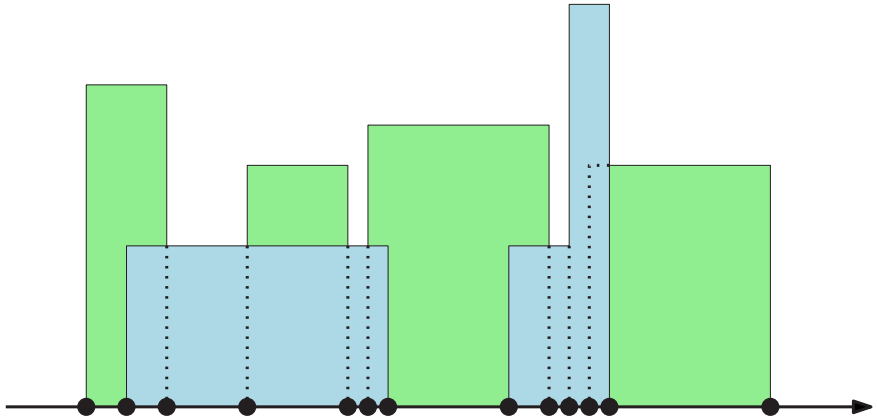
Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem

  Maximumsbildung



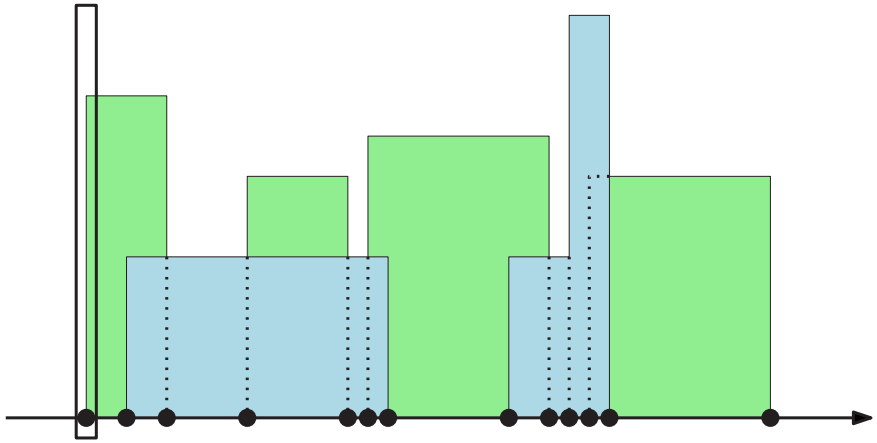
Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem

  Maximumsbildung



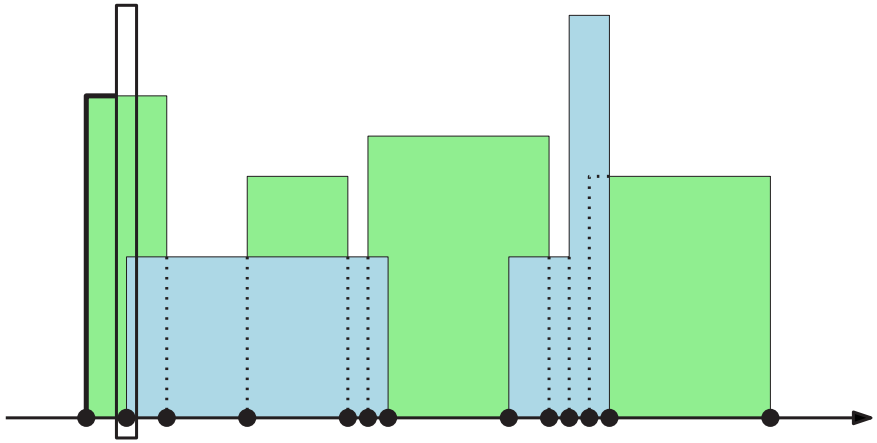
Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem

  Maximumsbildung



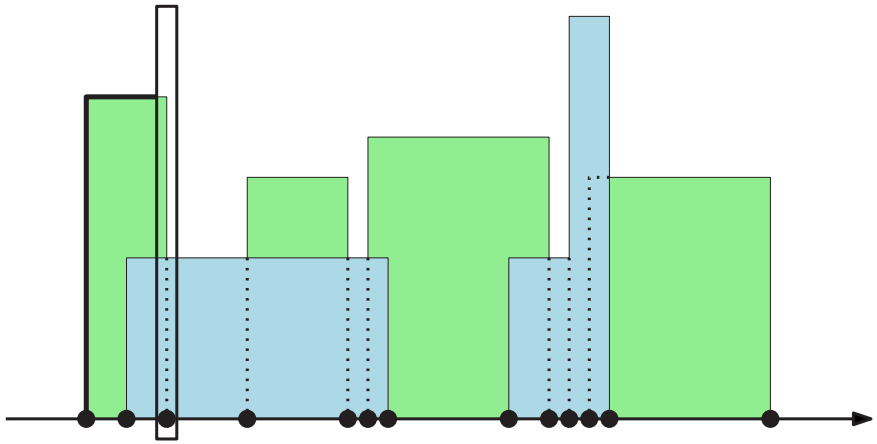
Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem

  Maximumsbildung



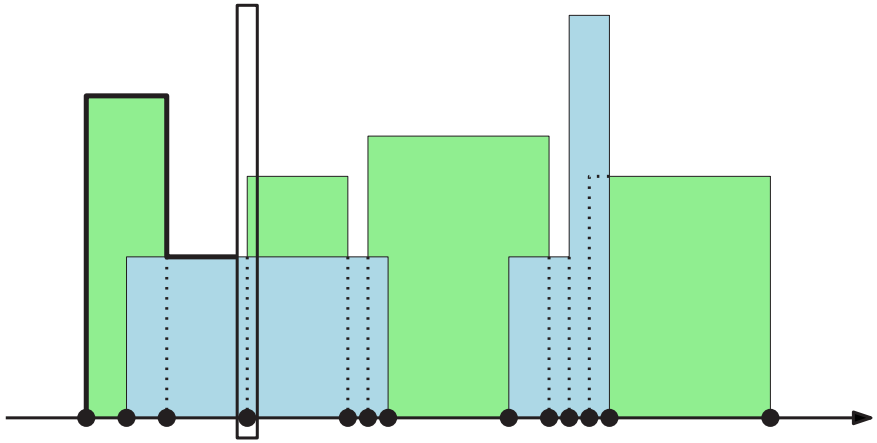
Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem

  Maximumsbildung



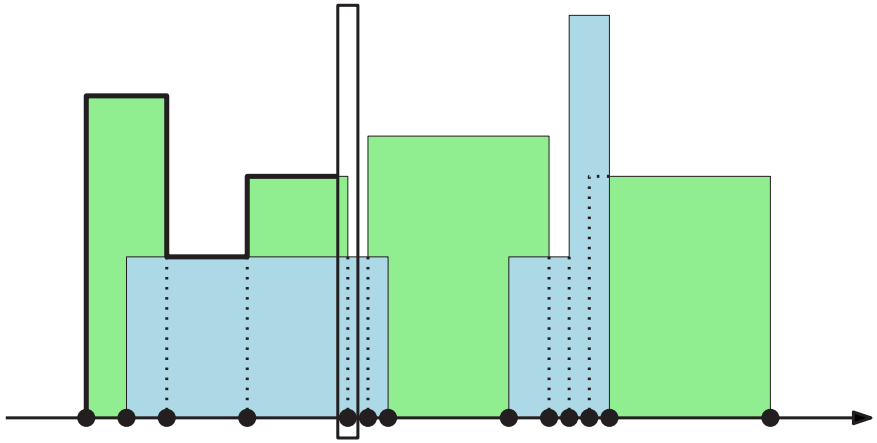
Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem

  Maximumsbildung



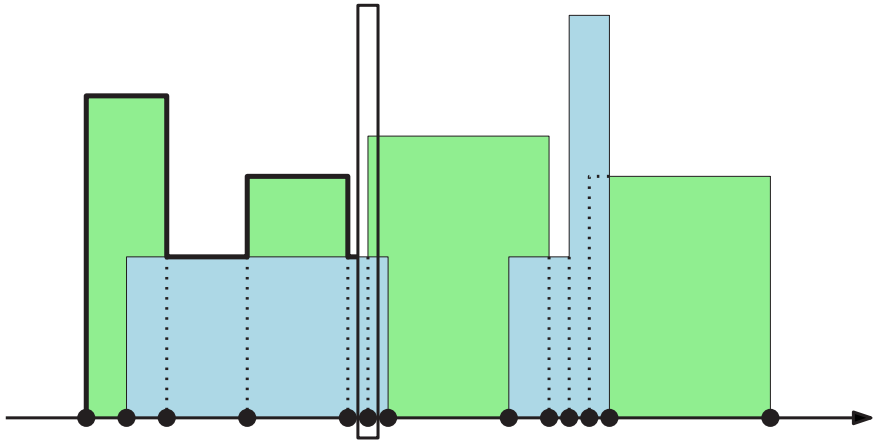
Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem

  Maximumsbildung



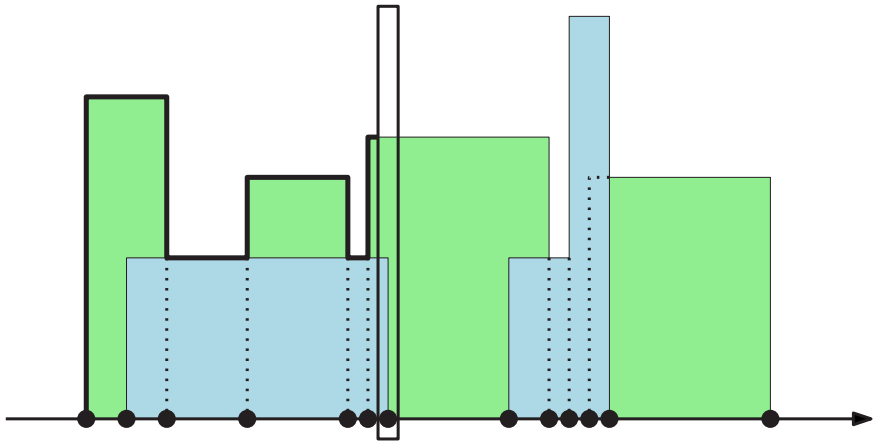
Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem

  Maximumsbildung



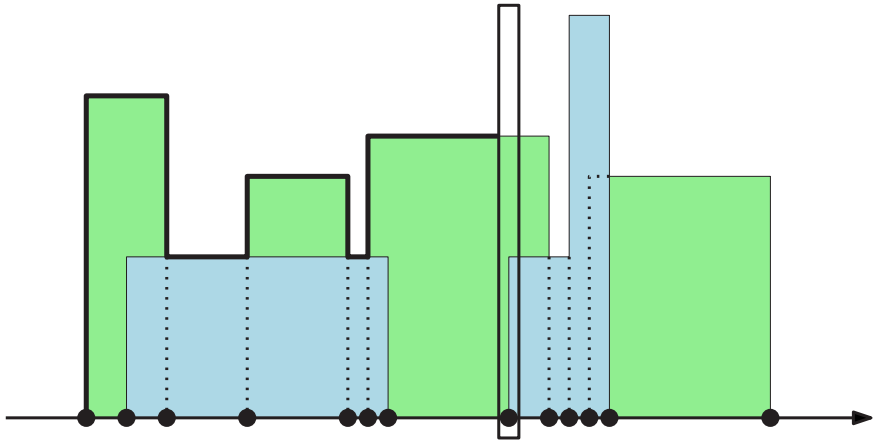
Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem

  Maximumsbildung



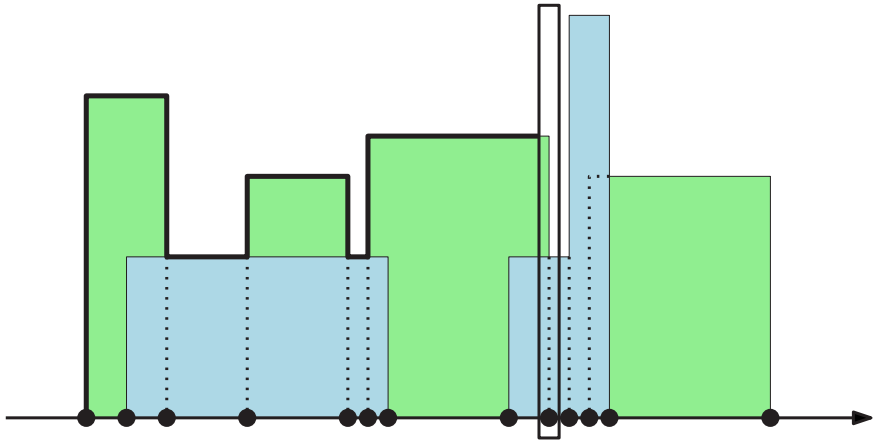
Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem

  Maximumsbildung



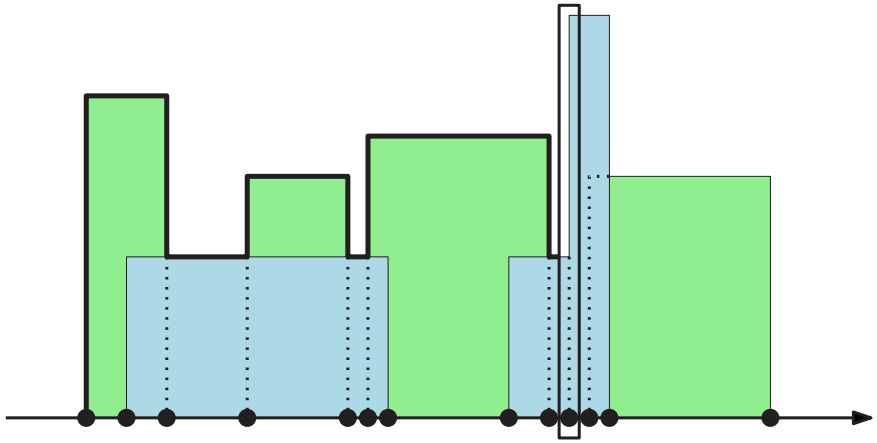
Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem

  Maximumsbildung



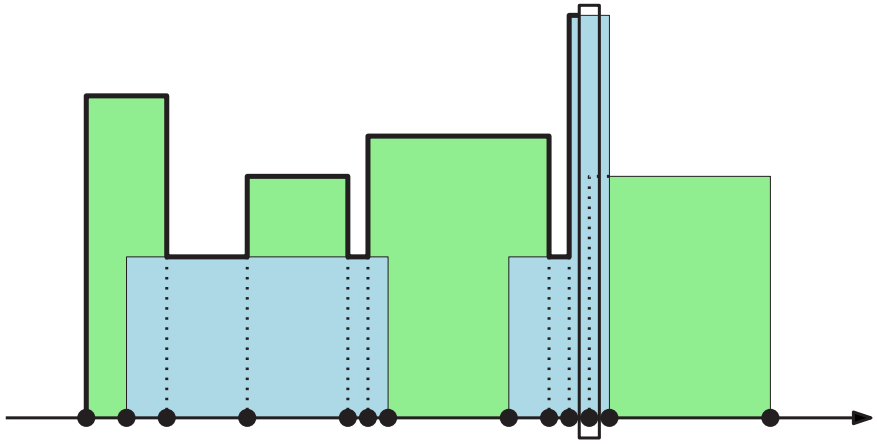
Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem

  Maximumsbildung



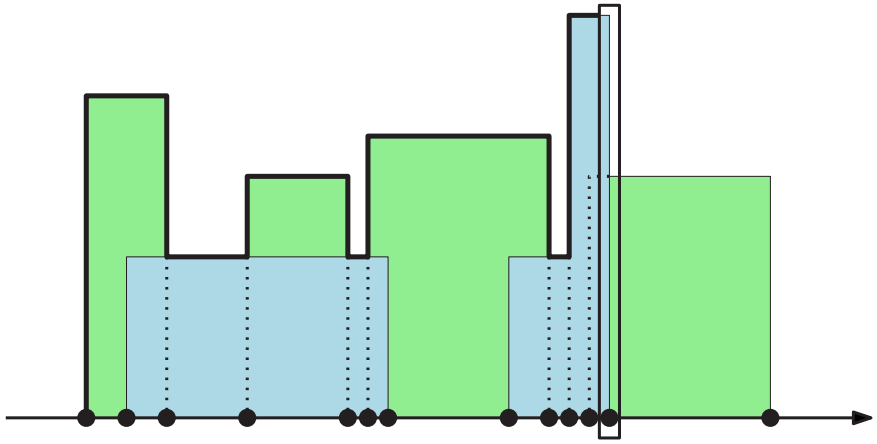
Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem

  Maximumsbildung



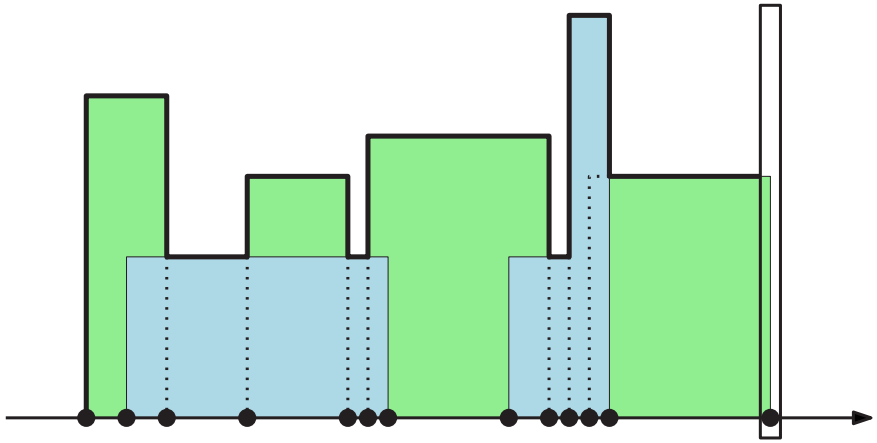
Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem

  Maximumsbildung



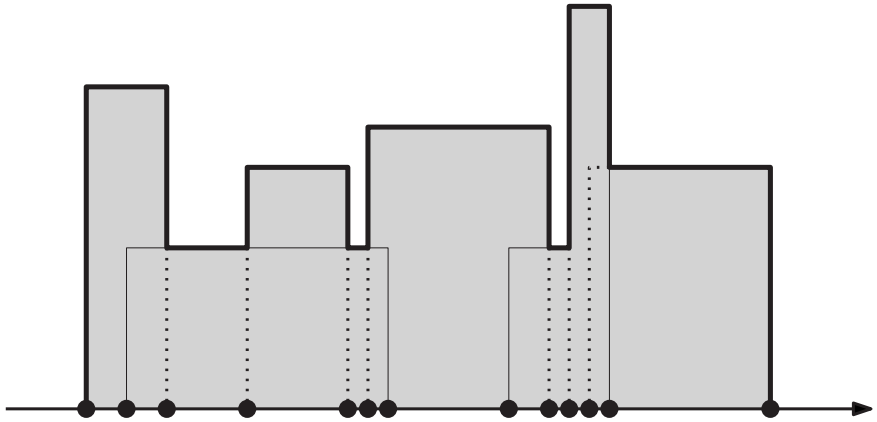
Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem

  Maximumsbildung

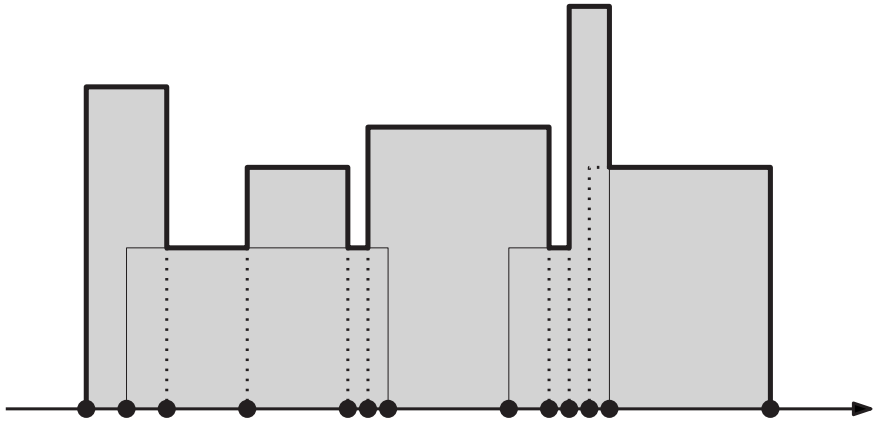


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem
  Maximumsbildung



Problem: effiziente Lösung des 1-dimensionalen Problems

- ineffizient: $\mathcal{O}(n^2)$ vergleiche Linienschnitt
- Ziel hier: Algorithmus mit $O(n \log n)$ Zeit

One-Dimensional Problem

Karlsruher Institut für Technologie

Sortierte Liste

- Array $\rightarrow \mathcal{O}(n)$ für Einfügen/Löschen
- Linked List $\rightarrow \mathcal{O}(n)$ für Positionsbestimmung

One-Dimensional Problem

Karlsruher Institut für Technologie

Sortierte Liste

- Array $\rightarrow \mathcal{O}(n)$ für Einfügen/Löschen
- Linked List $\rightarrow \mathcal{O}(n)$ für Positionsbestimmung

Lösung: Priority Queue

- alle Operationen in maximal $\mathcal{O}(\log n)$ möglich

Pseudocode (1/2)

Data: List of buildings (begin, height, end)
Result: Skyline coordinates (x, height)
$i \leftarrow 0$; $L' \leftarrow \emptyset$;
foreach $(b, h, e) \in L$ **do**
    $L' \leftarrow L' \cup (b, h, \text{“}b\text{“}, i) \cup (e, h, \text{“}e\text{“}, i)$;
    $i \leftarrow i + 1$;
end
sort L' in lexicographical order;
max priority queue $q \leftarrow \emptyset$;
...

Pseudocode (2/2)

```
...
while L' ≠ ∅ do
    actpos ← L'.first().pos;
    while L' ≠ ∅ and actpos = L'.first().pos do
        (pos,height,label,index) ← L'.popfirst();
        if label = "b" then
            q.insert(height,index));
        else
            q.remove(index);
        end
    end
    if q ≠ ∅ then
        print (actpos, q.max().height);
    else
        print actpos,0;
    end
end
```

Linienschnitt

überlappende Liniensegmente

Events

1. *Start*-Event $e := (y, \text{start}, s = \overline{(x, y)(x', y')})$
2. *End*-Event $e := (y', \text{end}, s = \overline{(x, y)(x', y')})$
3. *Intersection*-Event $e := (y, \text{intersection}, (s_i, s_j))$
   - s_i und s_j sind Linien die sich im Intersection-Event schneiden

Initialisierung

- Speichere *Start*- und *Stop*-Events in einer Priority Queue Q (nach y-Wert sortiert) $O(n \log n)$

Linienschnitt

überlappende Liniensegmente

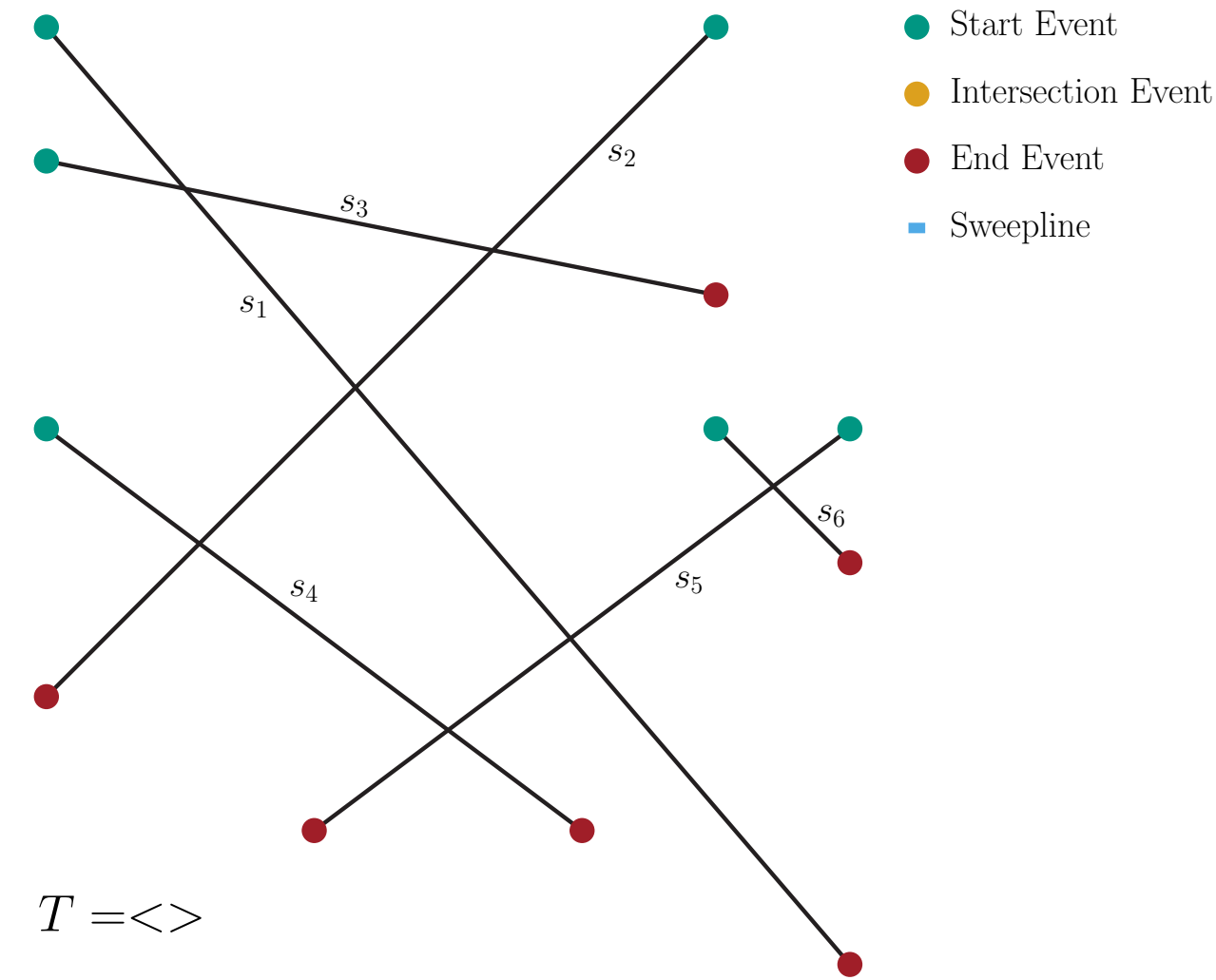
Linearer Scan

1. Entferne nächstes Event aus Priority-Queue Q $O(\log n)$
   - 1.1 *Start*-Event: Füge s in eine sortierte Liste T hinzu (nach x-Koordinate) und teste mit *Nachbarn* in T auf Linienschnitt $O(\log n)$
   - 1.2 *End*-Event: Lösche s aus sortierter Liste T und teste beide vorherigen *Nachbarn* in T auf Linienschnitt $O(\log n)$
   - 1.3 *Intersection*-Event: Vertausche s_i und s_j in sortierter Liste T und teste s_i mit seinem neuen rechten und s_j mit seinem neuen linken *Nachbarn* in T auf Linienschnitt $O(\log n)$
   - 1.4 Falls ein Schnitttest zweier Linien einen Schnittpunkt ergibt, füge *Intersection*-Event zu Q hinzu $O(\log n)$

$\Rightarrow$ Laufzeit $O((n+k)\log n)$ wobei k Anzahl an Schnittpunkten

Linienschnitt

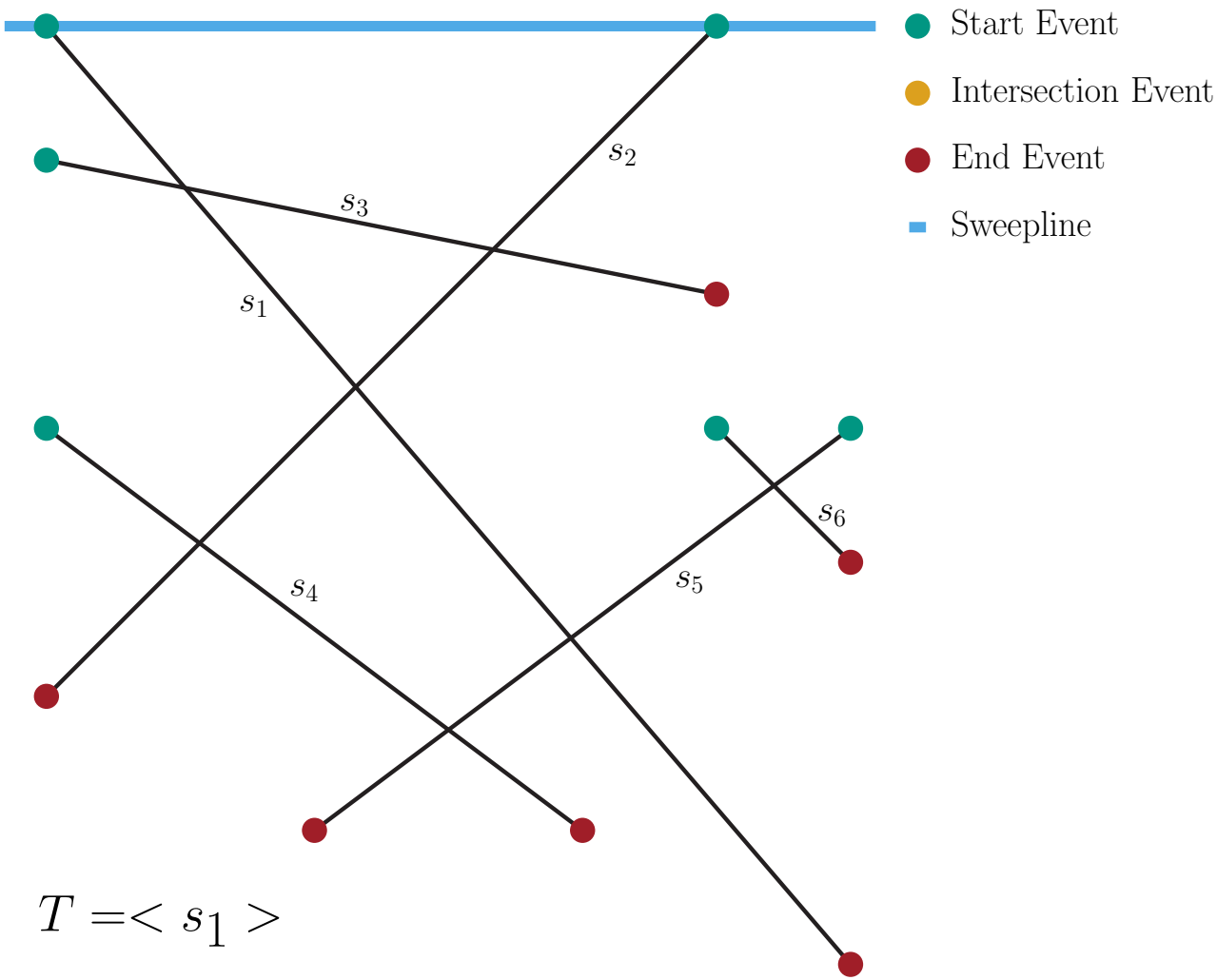
Illustration Sweepline-Algorithmus



- Start Event
- Intersection Event
- End Event
- Sweepline

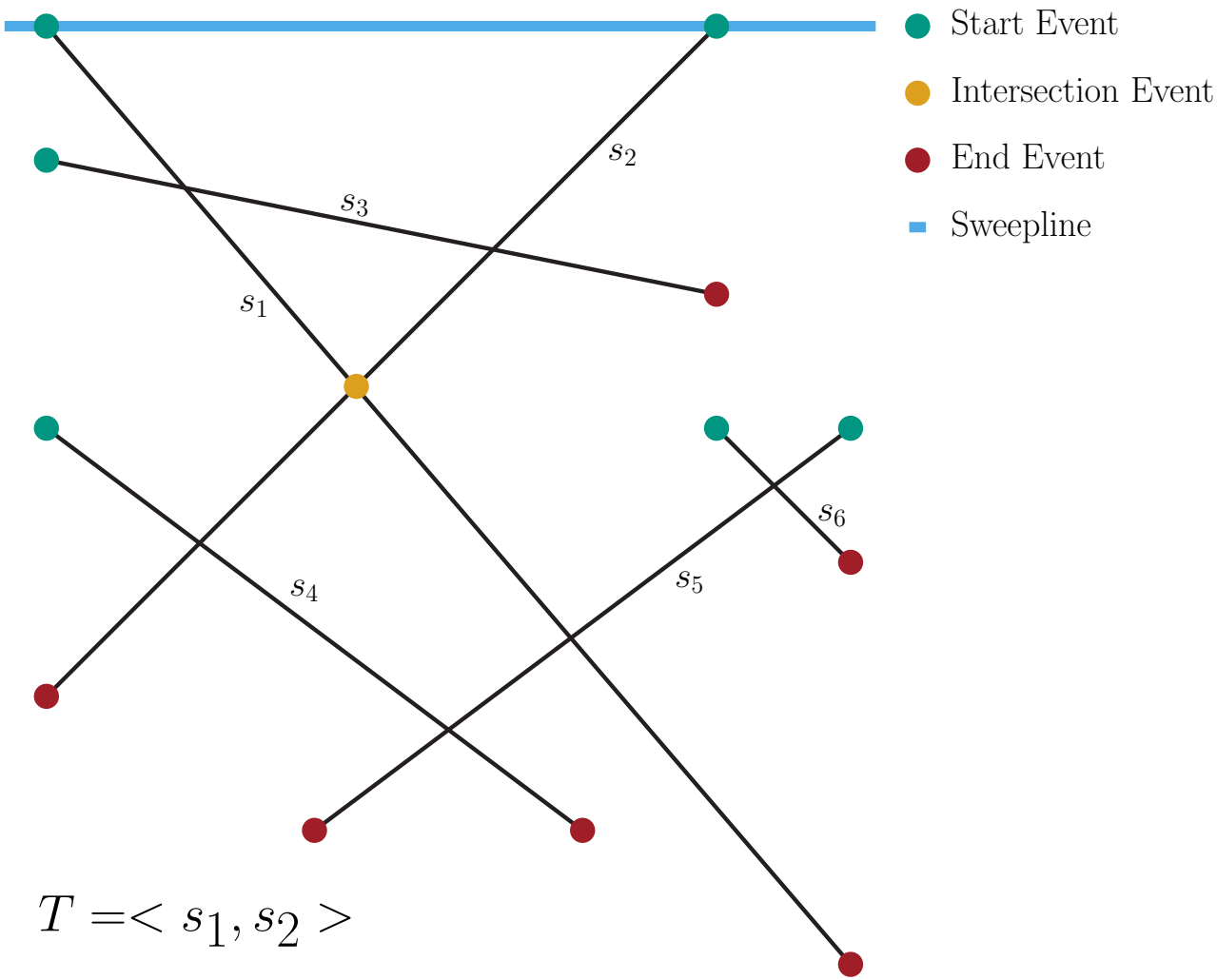
Linienschnitt

Illustration Sweepline-Algorithmus



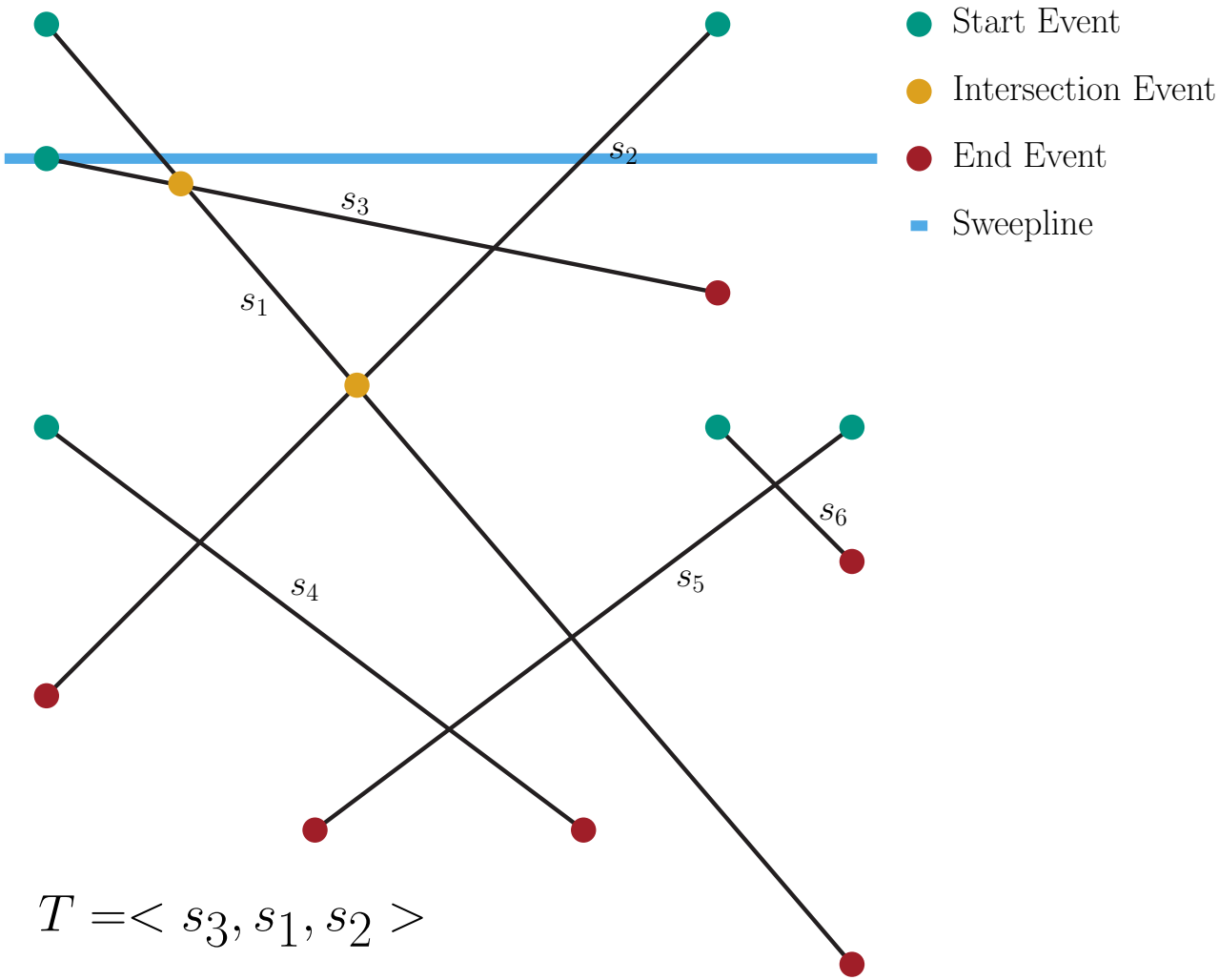
Linienschnitt

Illustration Sweepline-Algorithmus



Linienschnitt

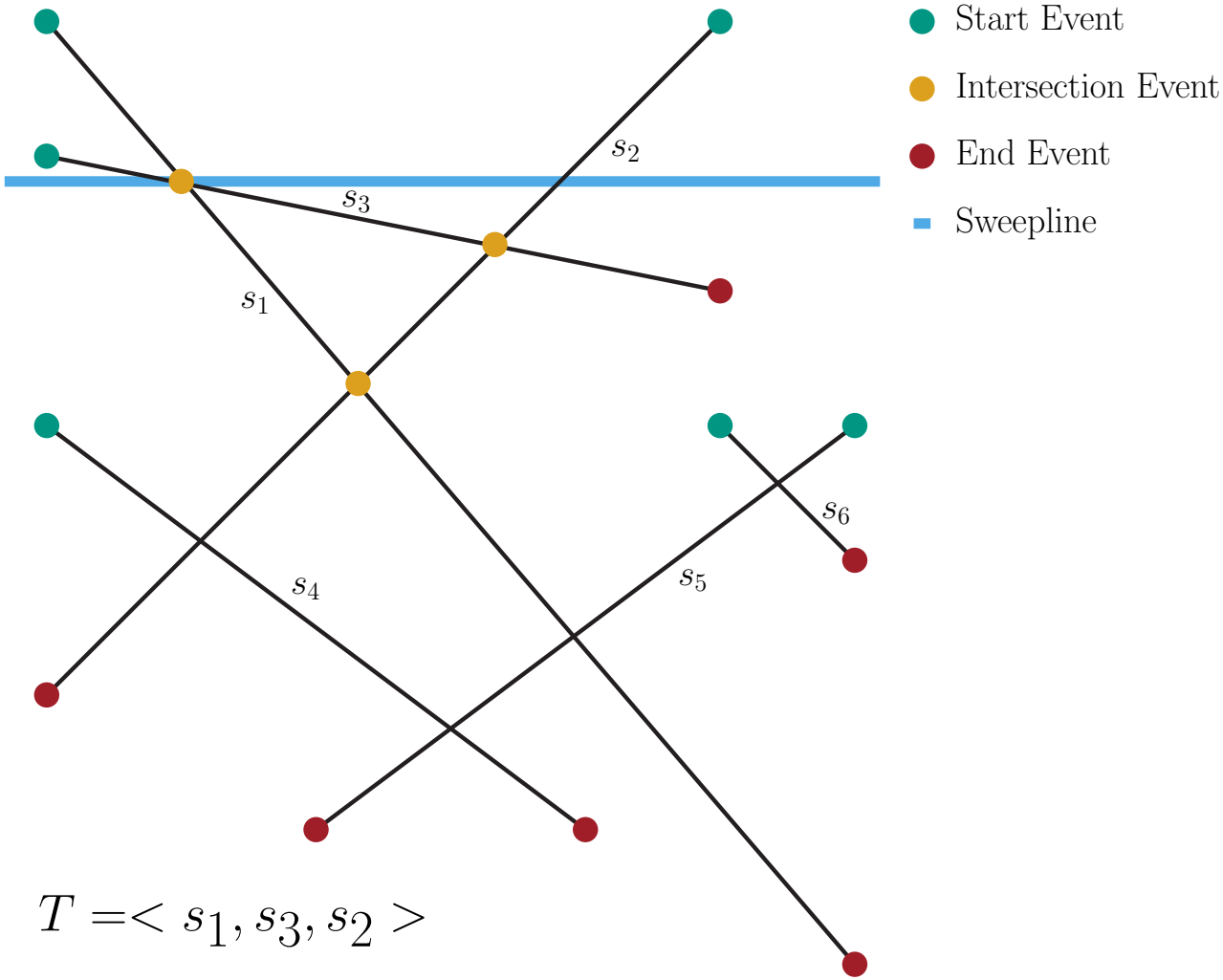
Illustration Sweepline-Algorithmus



$T =< s_3, s_1, s_2 >$

Linienschnitt

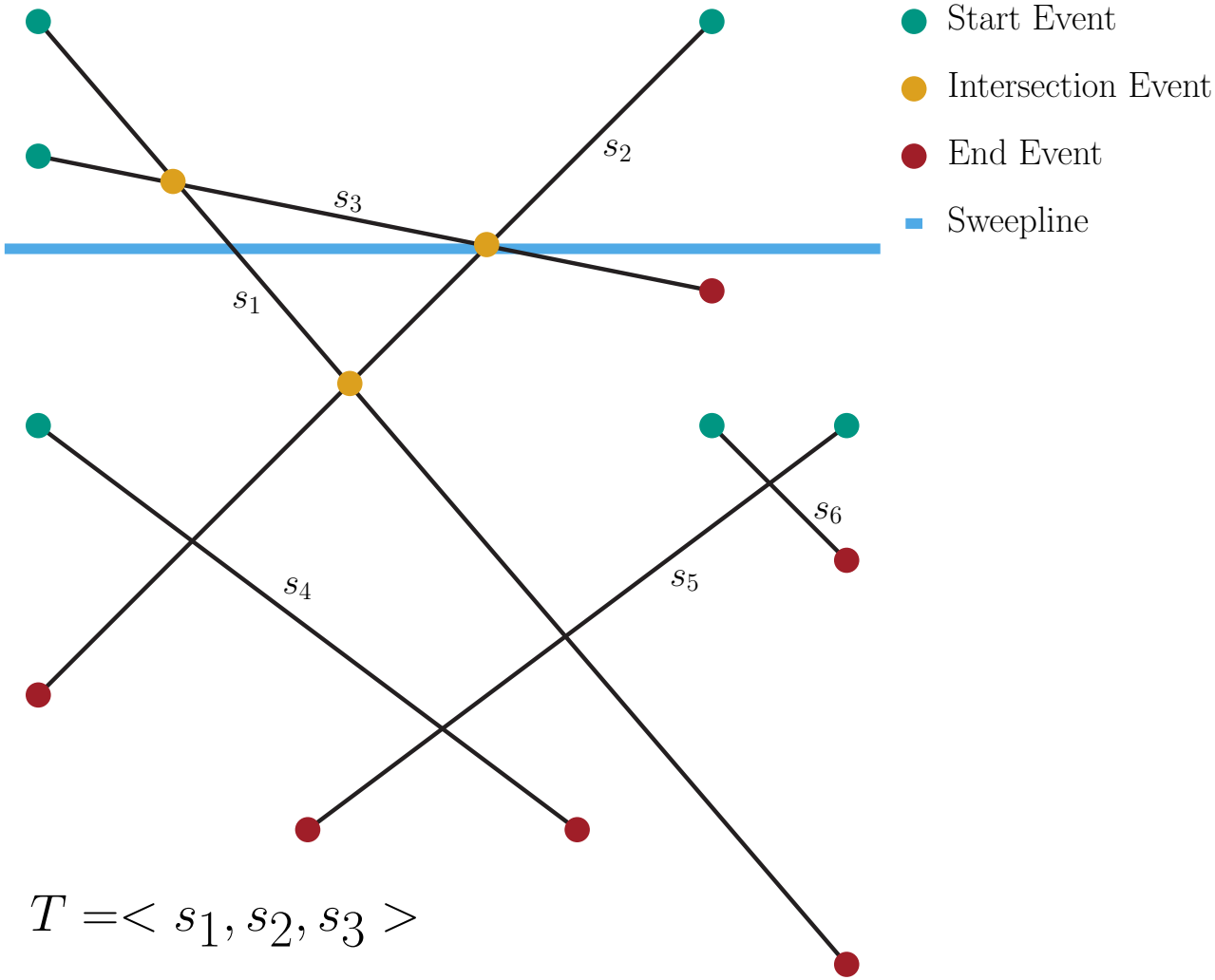
Illustration Sweepline-Algorithmus



$T =< s_1, s_3, s_2 >$

Linienschnitt

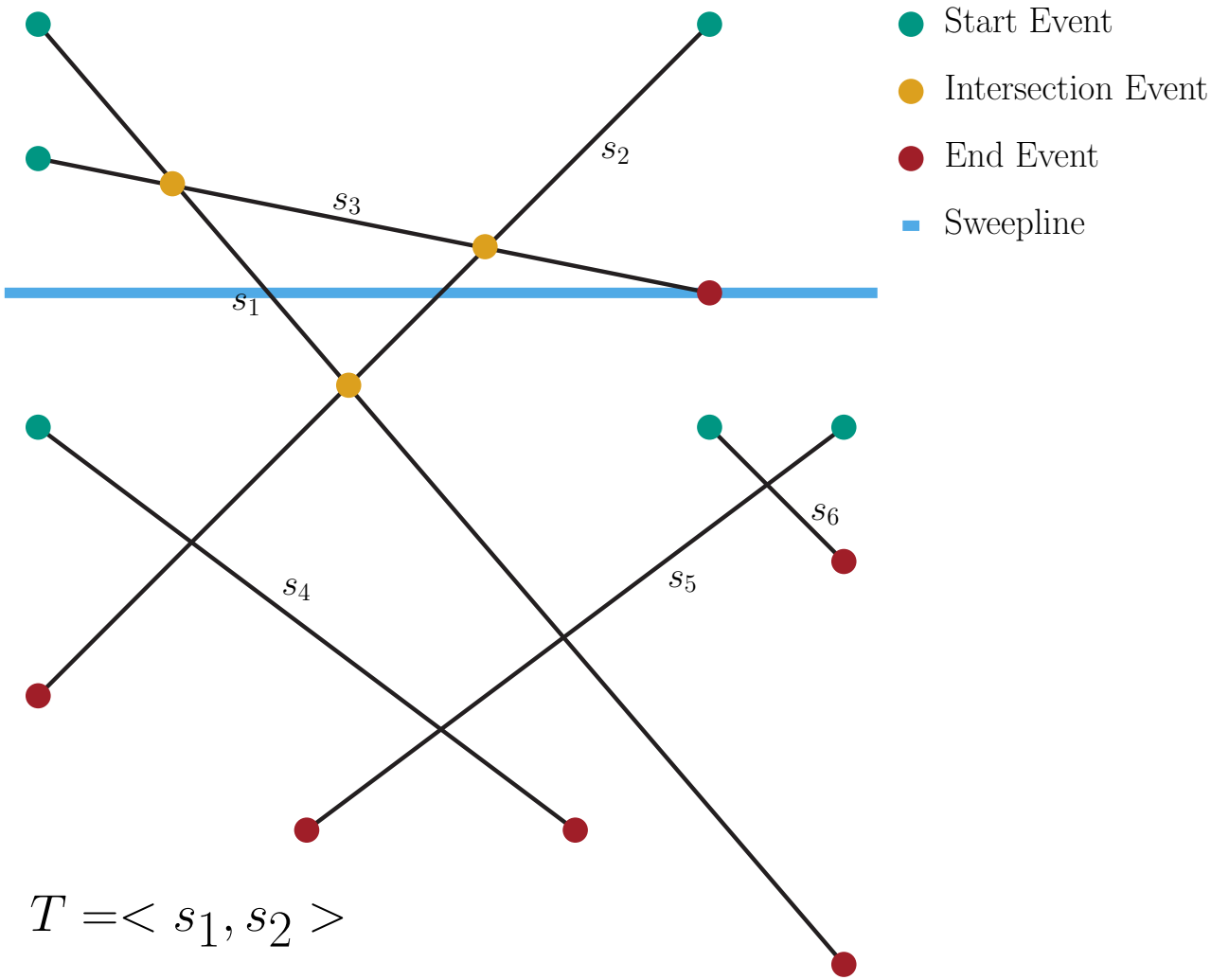
Illustration Sweepline-Algorithmus



$$T =< s_1, s_2, s_3 >$$

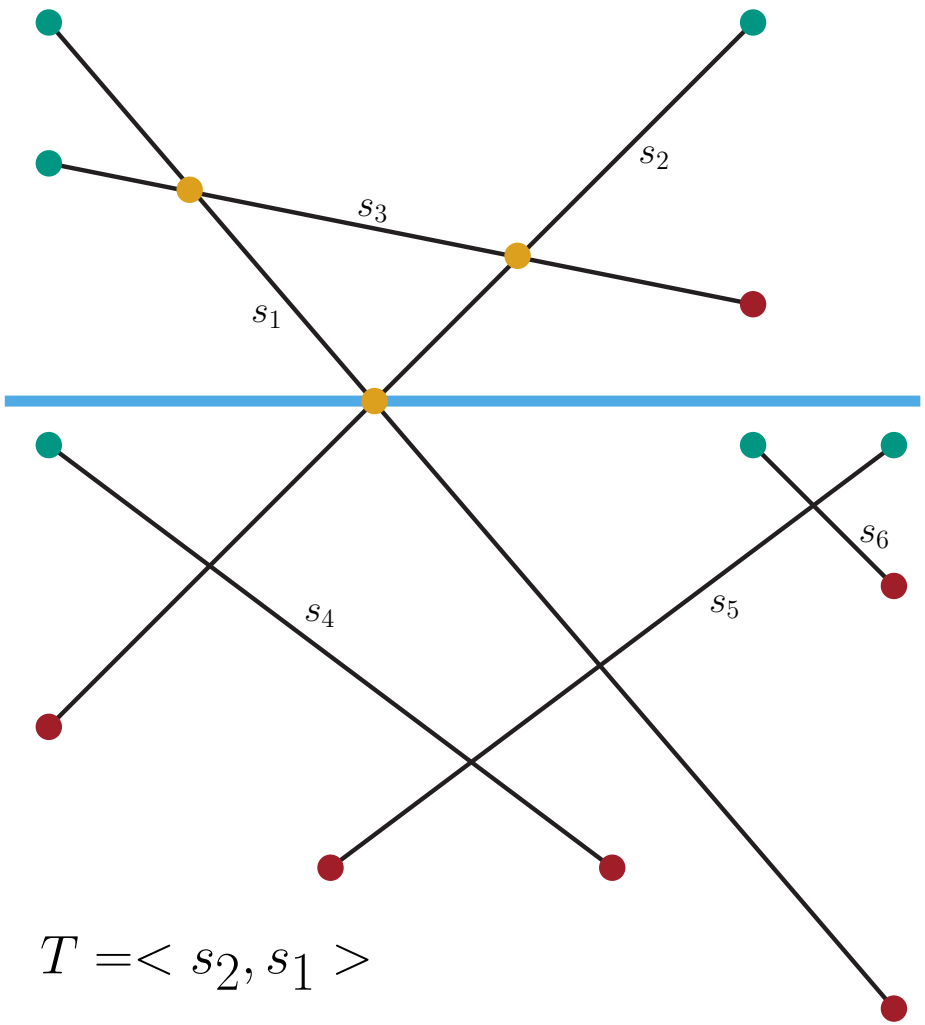
Linienschnitt

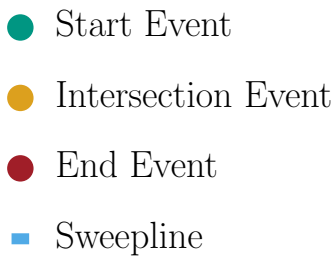
Illustration Sweepline-Algorithmus



Linienschnitt

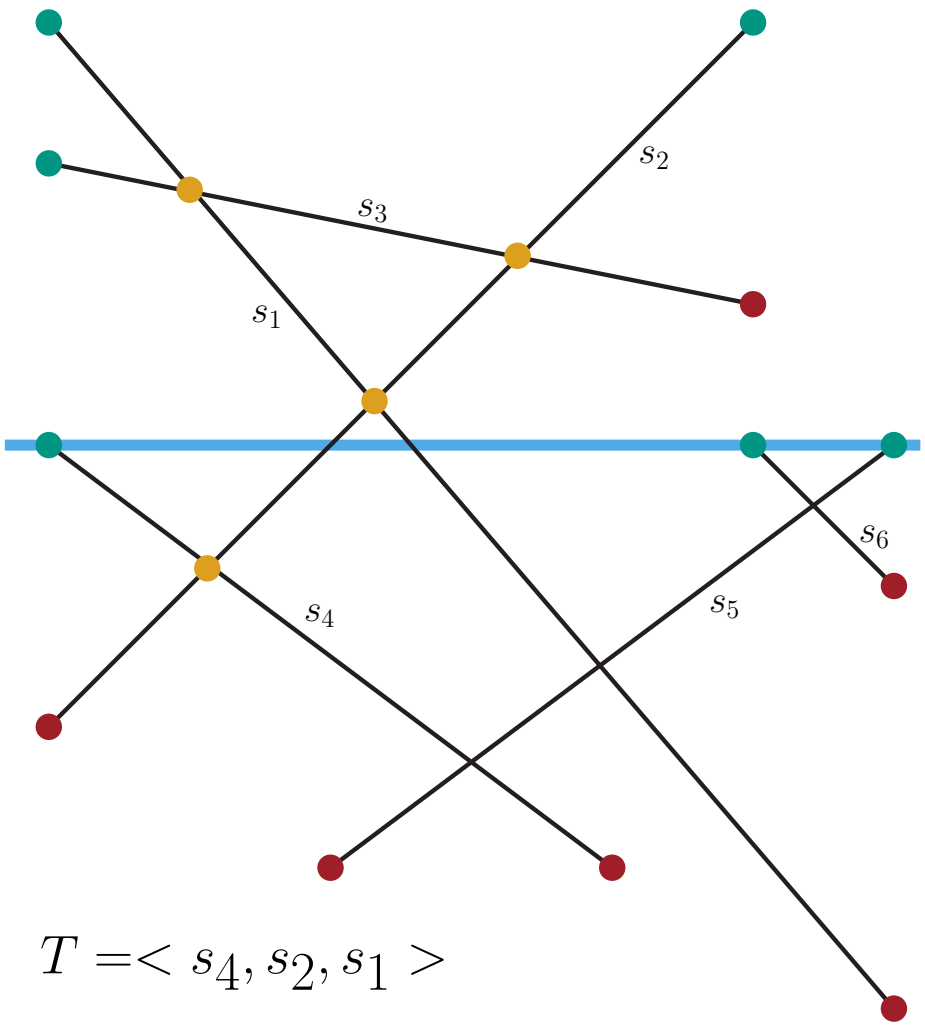
Illustration Sweepline-Algorithmus





Linienschnitt

Illustration Sweepline-Algorithmus



Start Event

Intersection Event

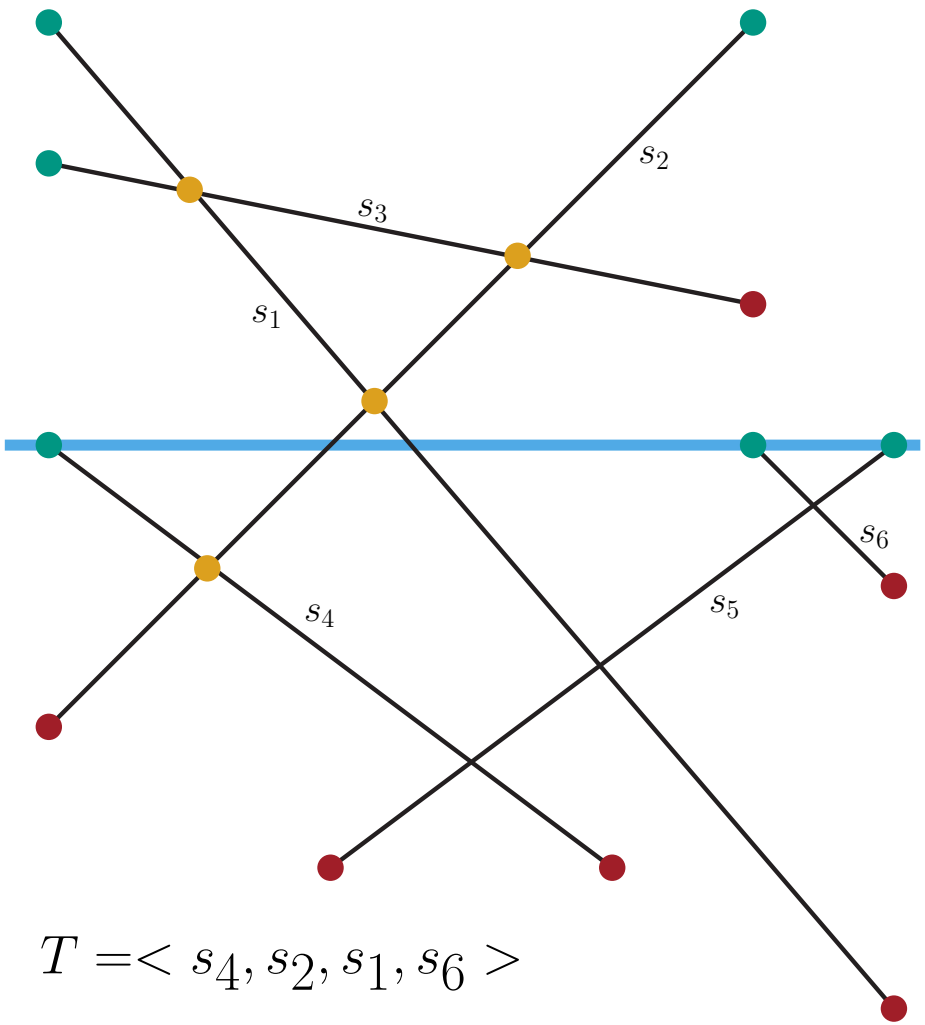
End Event

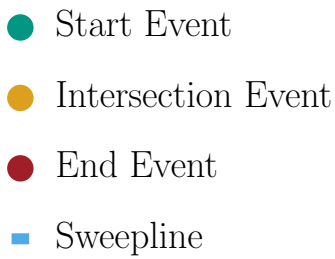
Sweepline

$T =< s_4, s_2, s_1 >$

Linienschnitt

Illustration Sweepline-Algorithmus

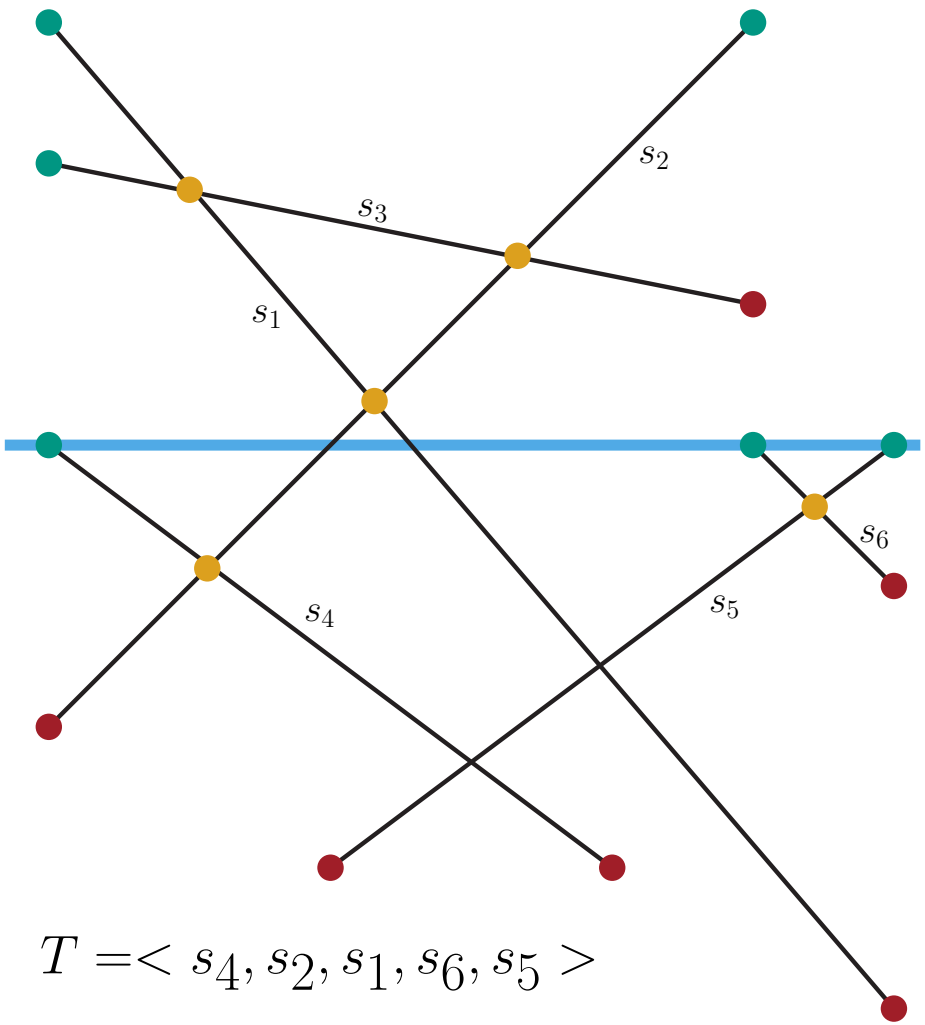




$T =< s_4, s_2, s_1, s_6 >$

Linienschnitt

Illustration Sweepline-Algorithmus

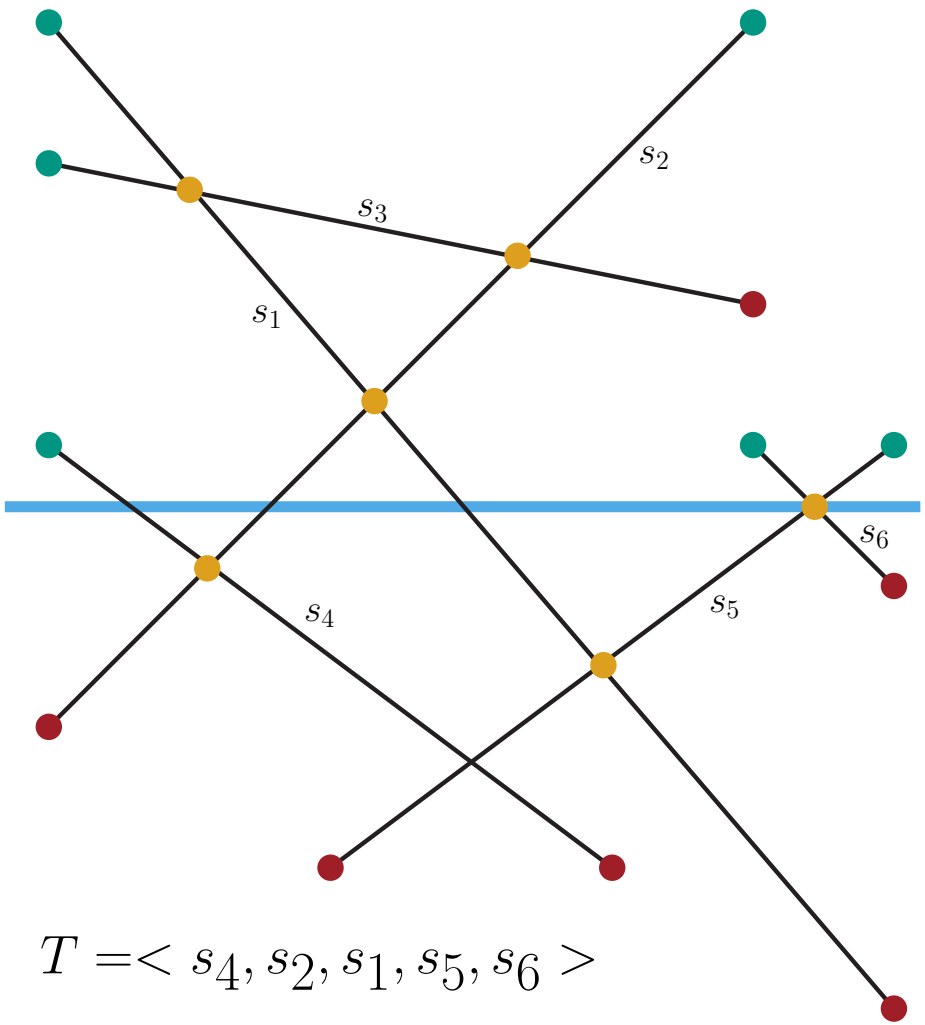


- Start Event
- Intersection Event
- End Event
- Sweepline

$$T =< s_4, s_2, s_1, s_6, s_5 >$$

Linienschnitt

Illustration Sweepline-Algorithmus

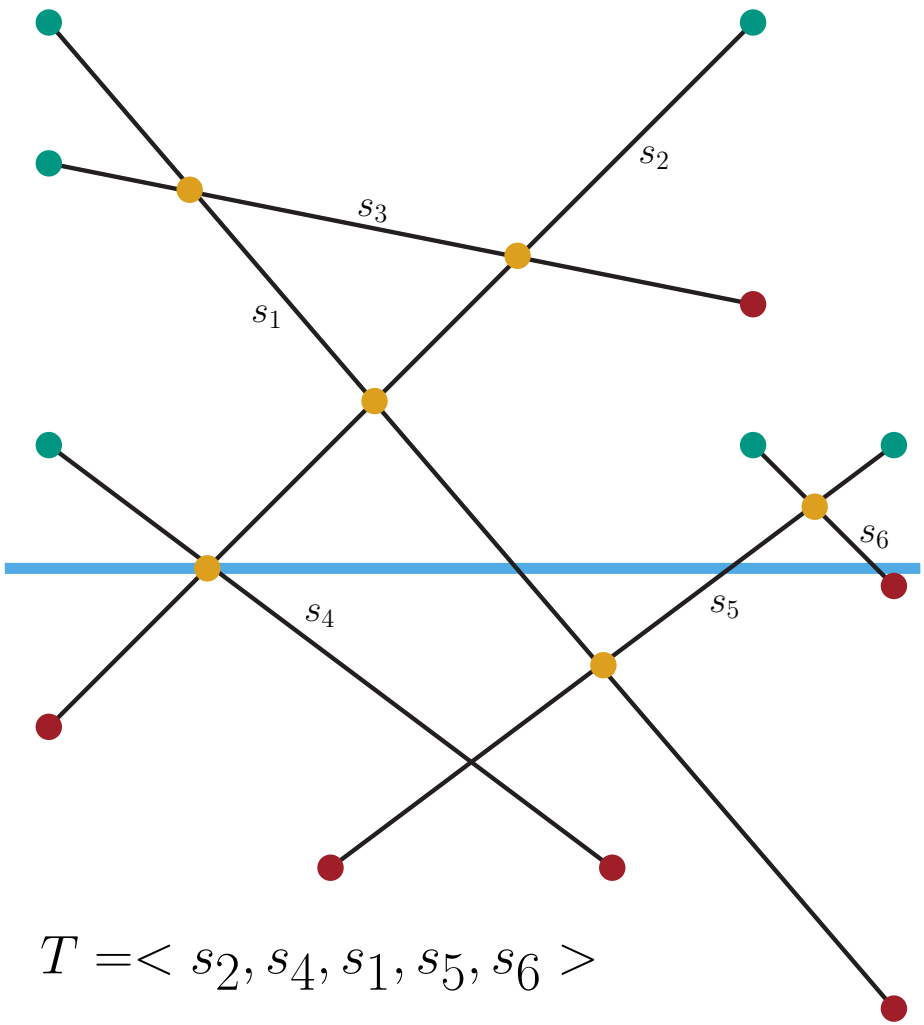


- Start Event
- Intersection Event
- End Event
- Sweepline

$$T =< s_4, s_2, s_1, s_5, s_6 >$$

Linienschnitt

Illustration Sweepline-Algorithmus

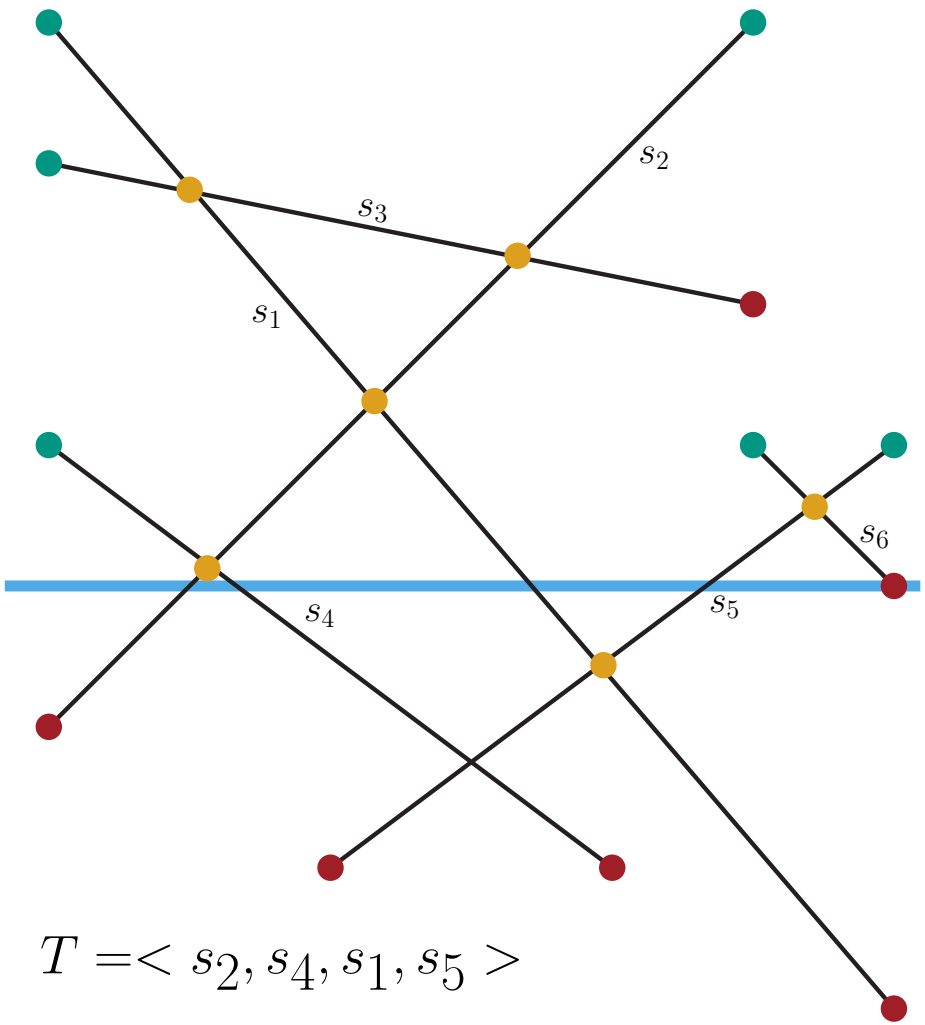


- Start Event
- Intersection Event
- End Event
- Sweepline

$$T =< s_2, s_4, s_1, s_5, s_6 >$$

Linienschnitt

Illustration Sweepline-Algorithmus

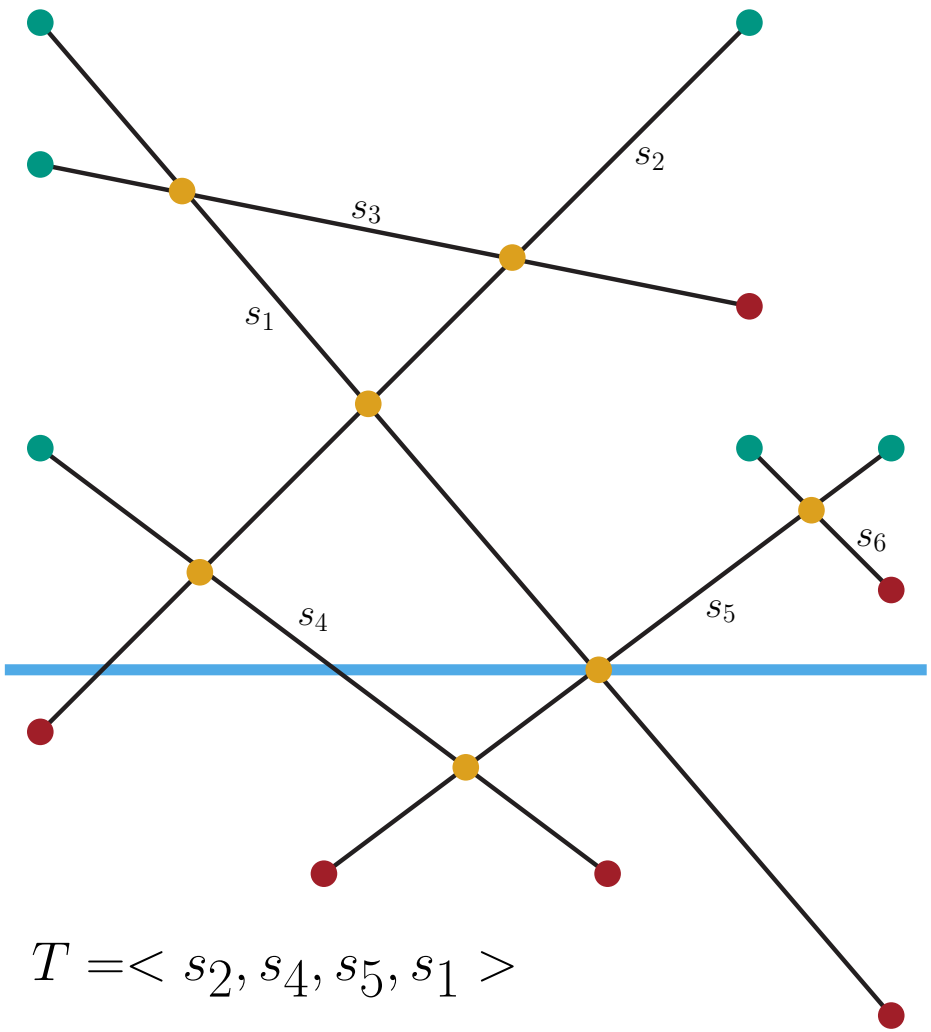


- Start Event
- Intersection Event
- End Event
- Sweepline

$T =< s_2, s_4, s_1, s_5 >$

Linienschnitt

Illustration Sweepline-Algorithmus

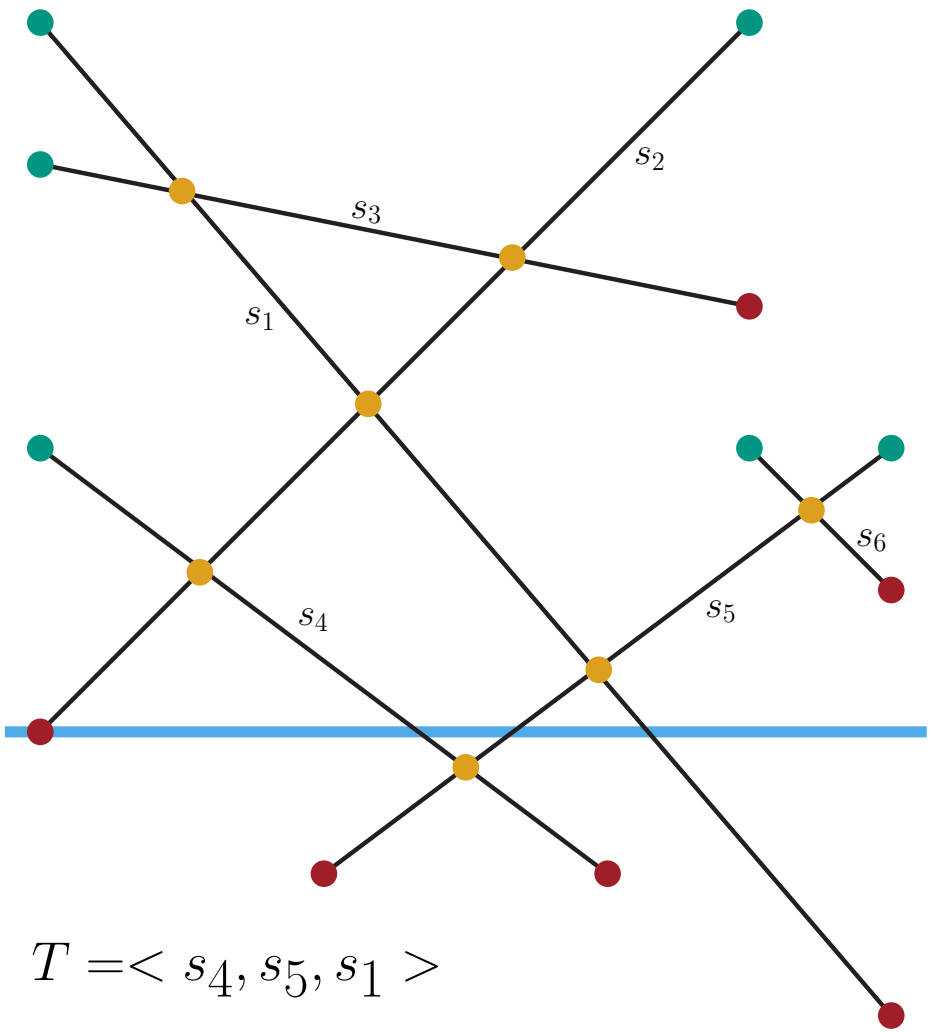


- Start Event
- Intersection Event
- End Event
- Sweepline

$$T =< s_2, s_4, s_5, s_1 >$$

Linienschnitt

Illustration Sweepline-Algorithmus

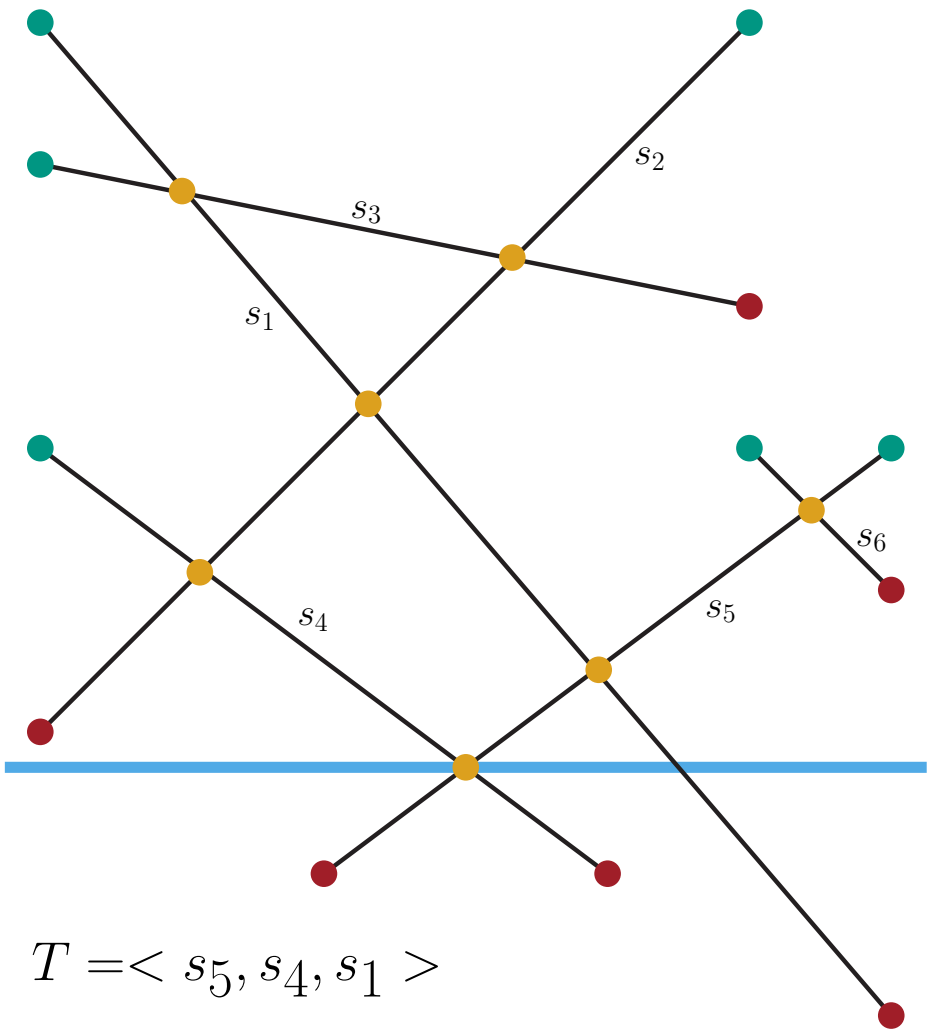


- Start Event
- Intersection Event
- End Event
- Sweepline

$T =< s_4, s_5, s_1 >$

Linienschnitt

Illustration Sweepline-Algorithmus

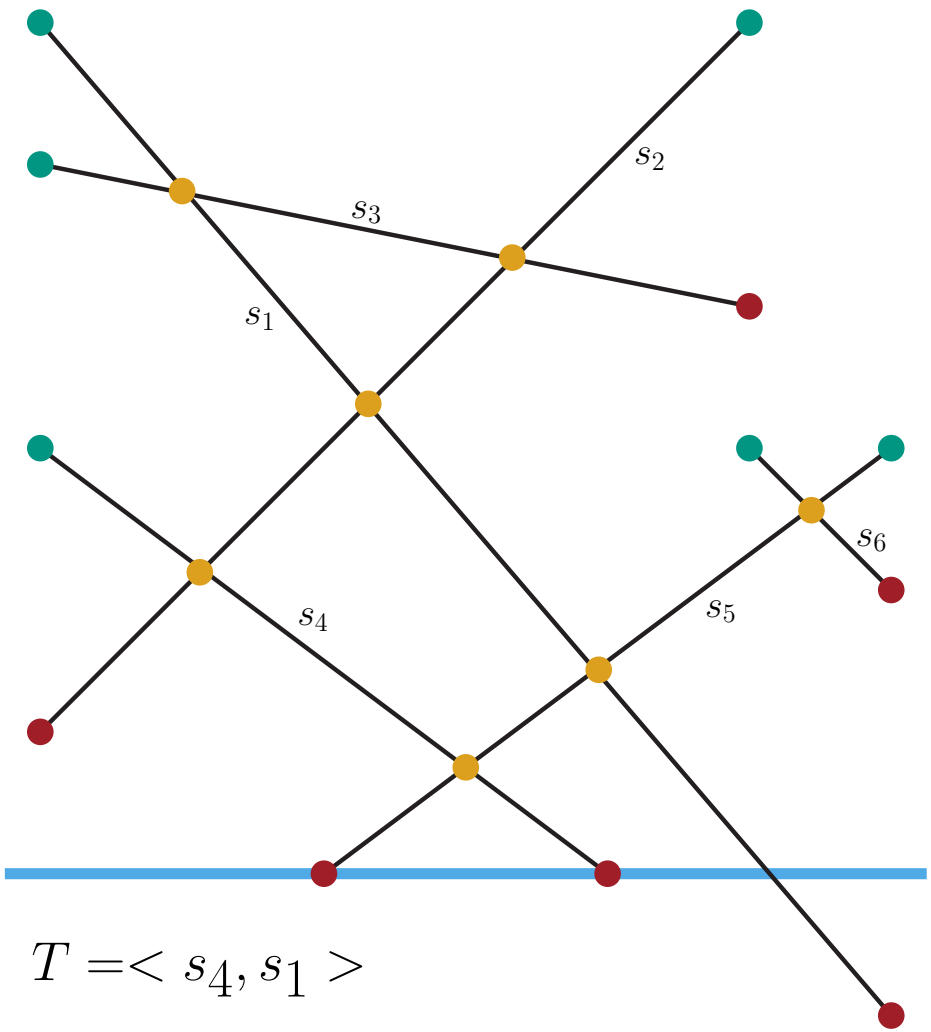


- Start Event
- Intersection Event
- End Event
- Sweepline

$$T =< s_5, s_4, s_1 >$$

Linienschnitt

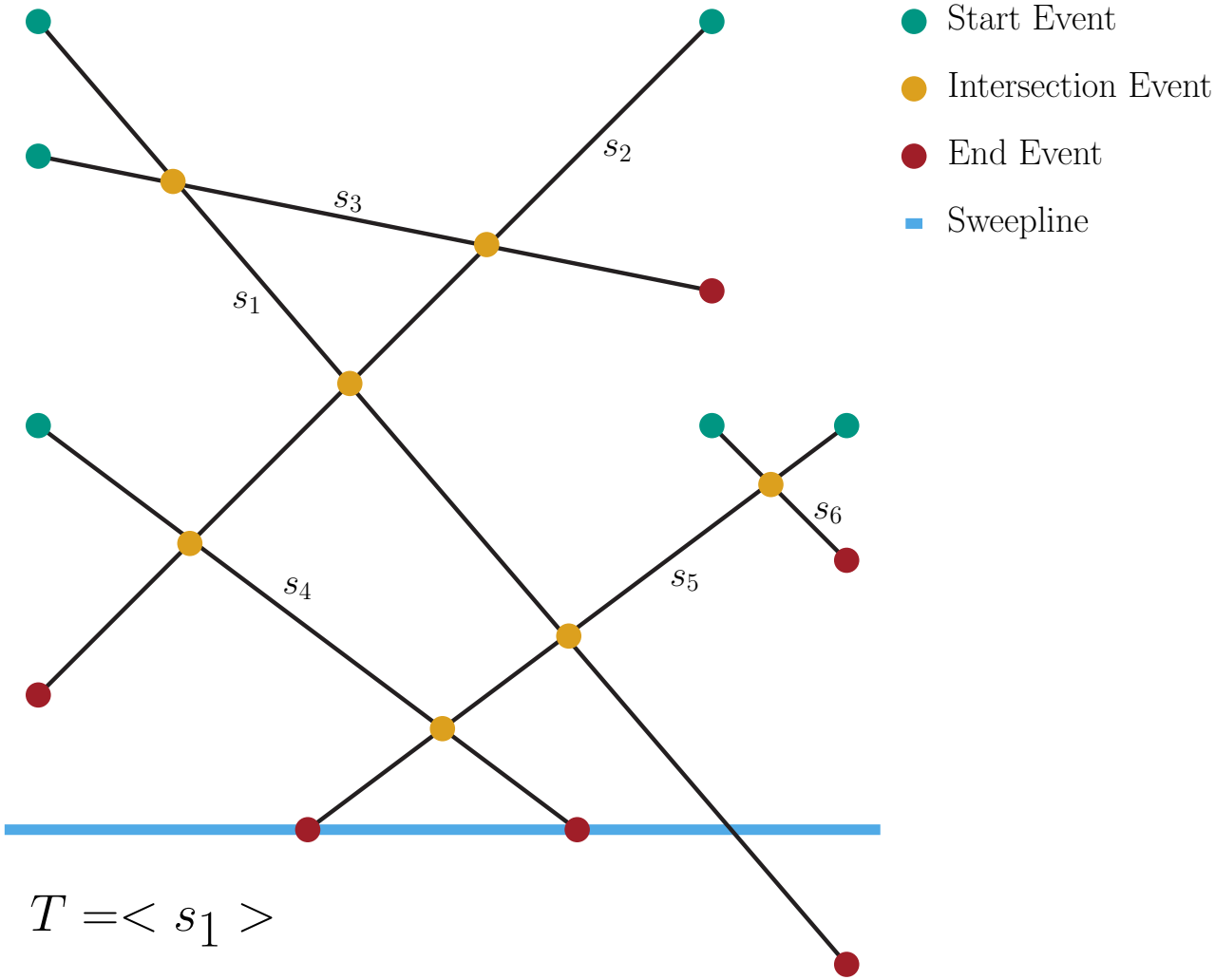
Illustration Sweepline-Algorithmus



- Start Event
- Intersection Event
- End Event
- Sweepline

Linienschnitt

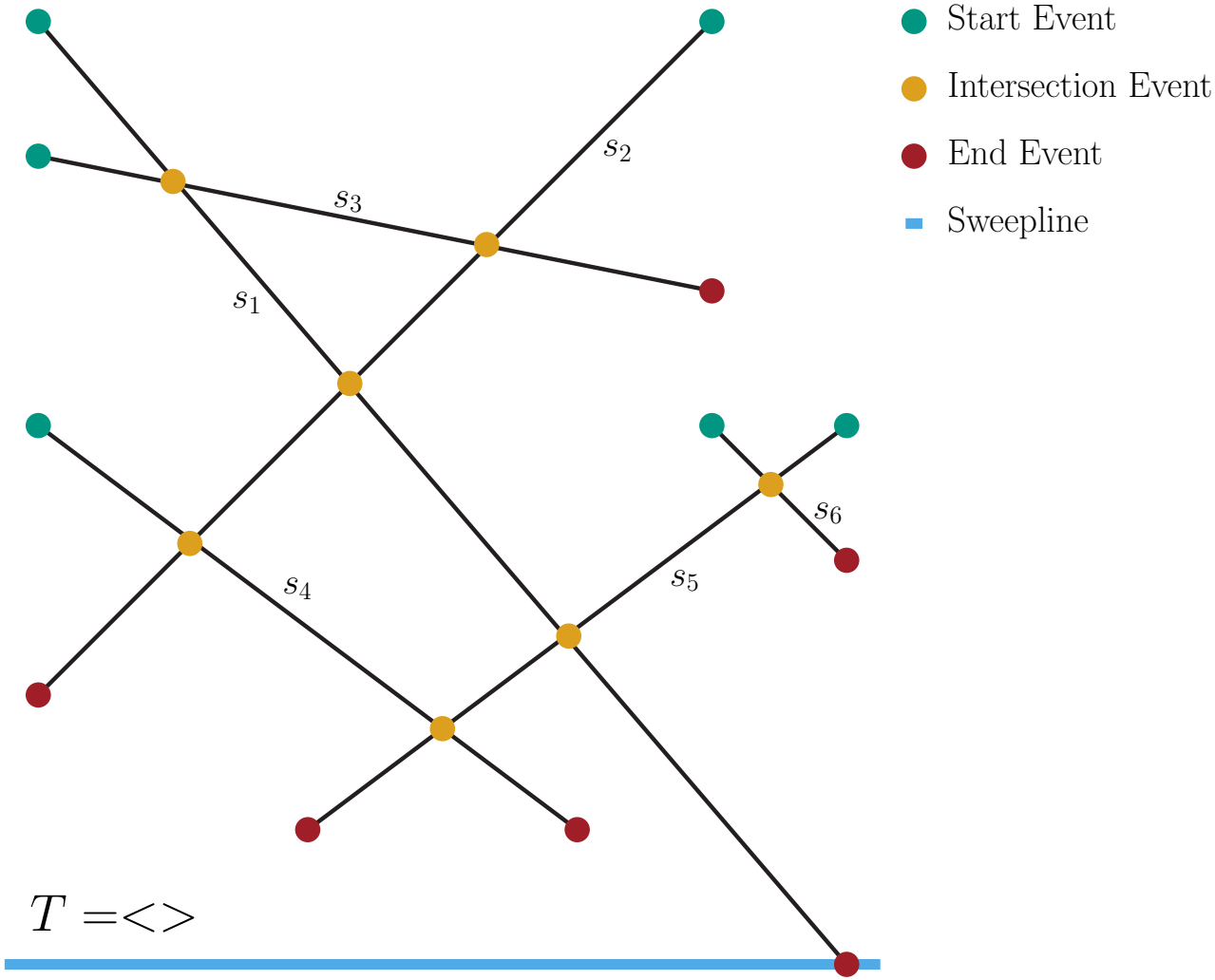
Illustration Sweepline-Algorithmus



- Start Event
- Intersection Event
- End Event
- Sweepline

Linienschnitt

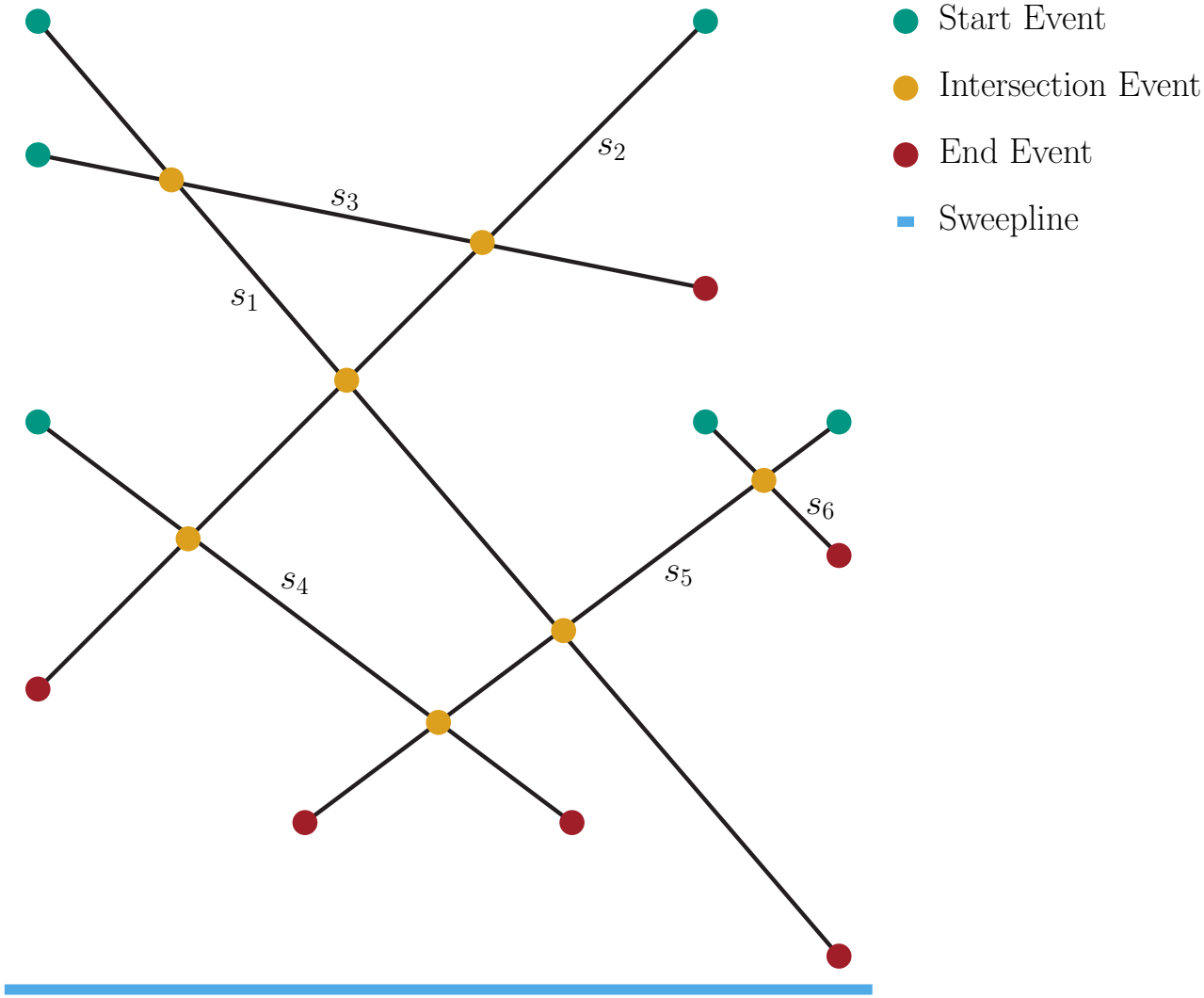
Illustration Sweepline-Algorithmus



- Start Event
- Intersection Event
- End Event
- Sweepline

Linienschnitt

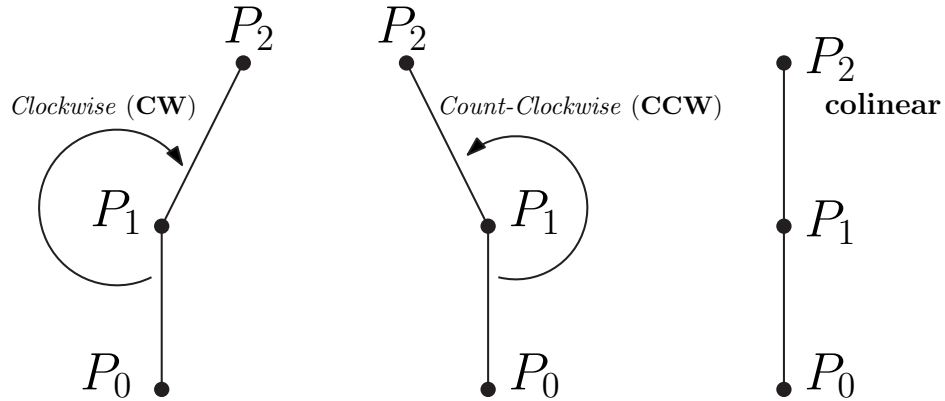
Illustration Sweepline-Algorithmus



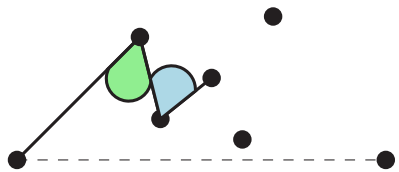
- Start Event
- Intersection Event
- End Event
- Sweepline

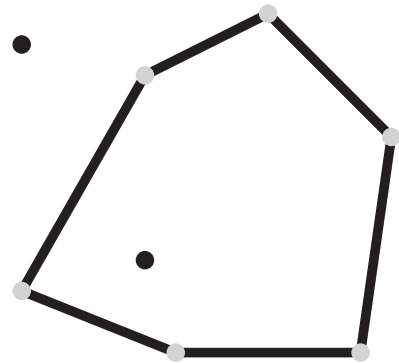
Punktorientierung

- Gegeben drei Punkte P_0, P_1, $P_2 \Rightarrow$ bestimme Orientierung
- Sei $\overrightarrow{a} = \overrightarrow{P_0P_1}$ und $\overrightarrow{b} = \overrightarrow{P_1P_2}$
- $CCW(P_0, P_1, P_2) = a_x \cdot b_y - a_y \cdot b_x$
- $CCW(P_0, P_1, P_2) > 0 \Rightarrow$ **CCW**
- $CCW(P_0, P_1, P_2) = 0 \Rightarrow$ **Colinear**
- $CCW(P_0, P_1, P_2) < 0 \Rightarrow$ **CW**



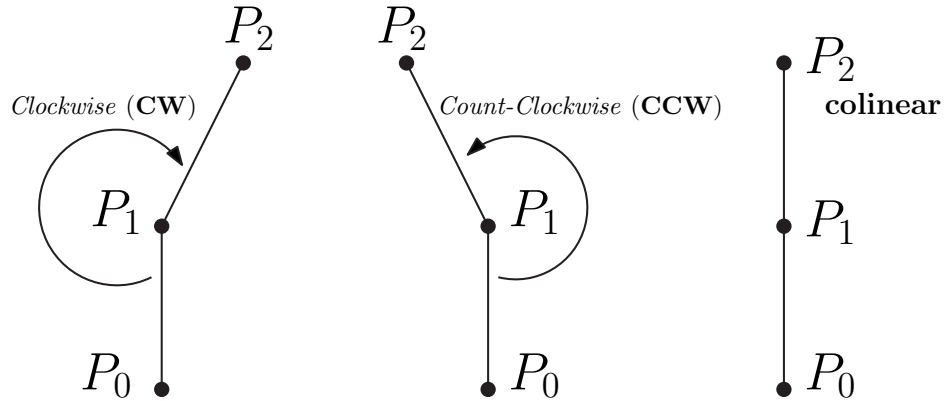
- Unterproblem von Algorithmen
  - Graham Scan
  - Test auf Enthaltensein
    Punkt in konvexem Polygon



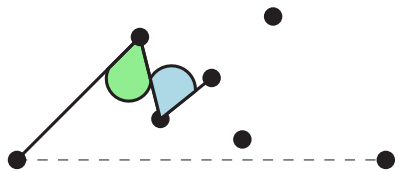


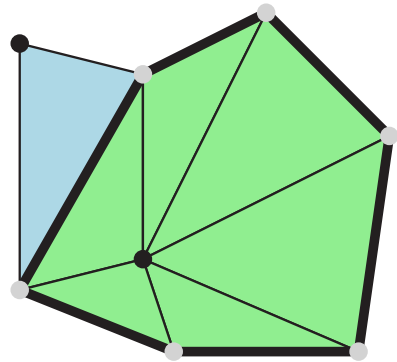
Punktorientierung

- Gegeben drei Punkte P_0, P_1, $P_2 \Rightarrow$ bestimme Orientierung
- Sei $\overrightarrow{a} = \overrightarrow{P_0P_1}$ und $\overrightarrow{b} = \overrightarrow{P_1P_2}$
- $CCW(P_0, P_1, P_2) = a_x \cdot b_y - a_y \cdot b_x$
- $CCW(P_0, P_1, P_2) > 0 \Rightarrow$ **CCW**
- $CCW(P_0, P_1, P_2) = 0 \Rightarrow$ **Colinear**
- $CCW(P_0, P_1, P_2) < 0 \Rightarrow$ **CW**



- Unterproblem von Algorithmen
  - Graham Scan
  - Test auf Enthaltensein
    Punkt in konvexem Polygon

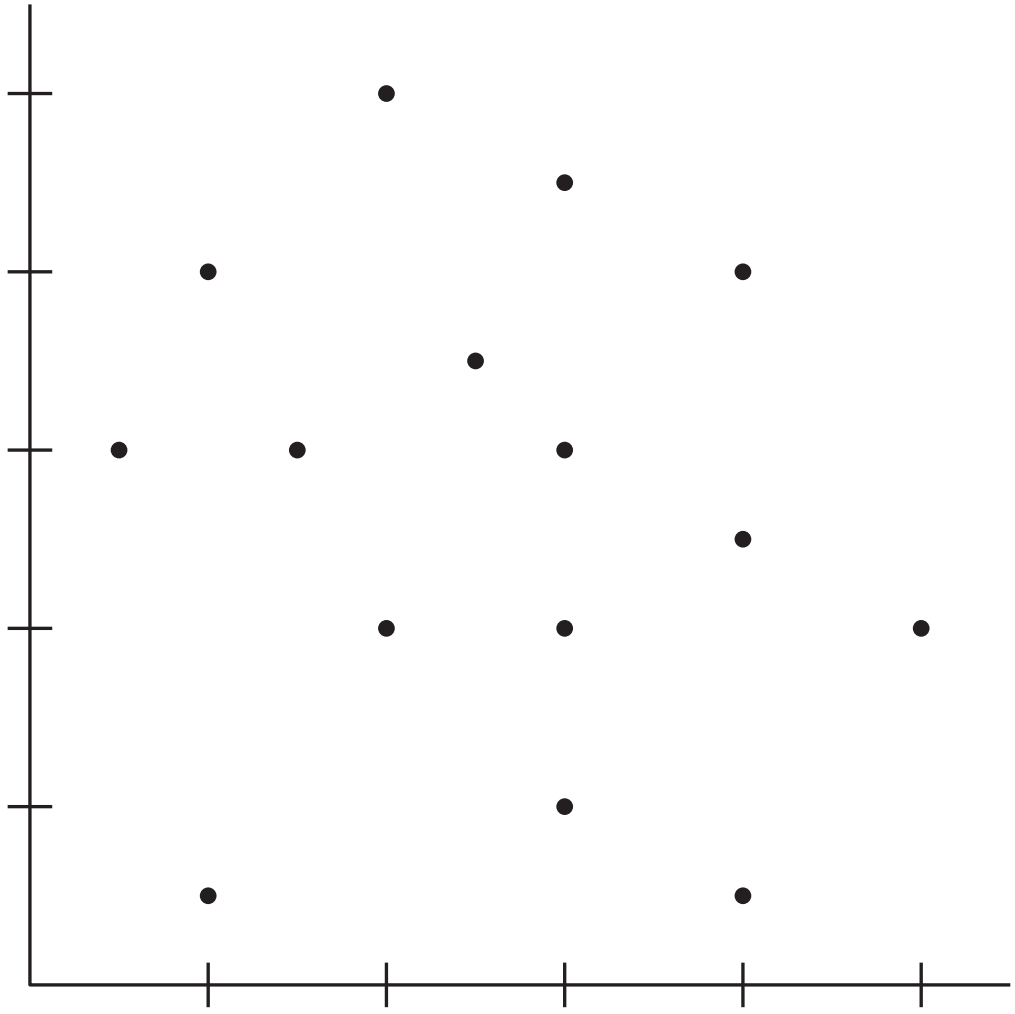




Konvexe Hülle

Graham-Scan

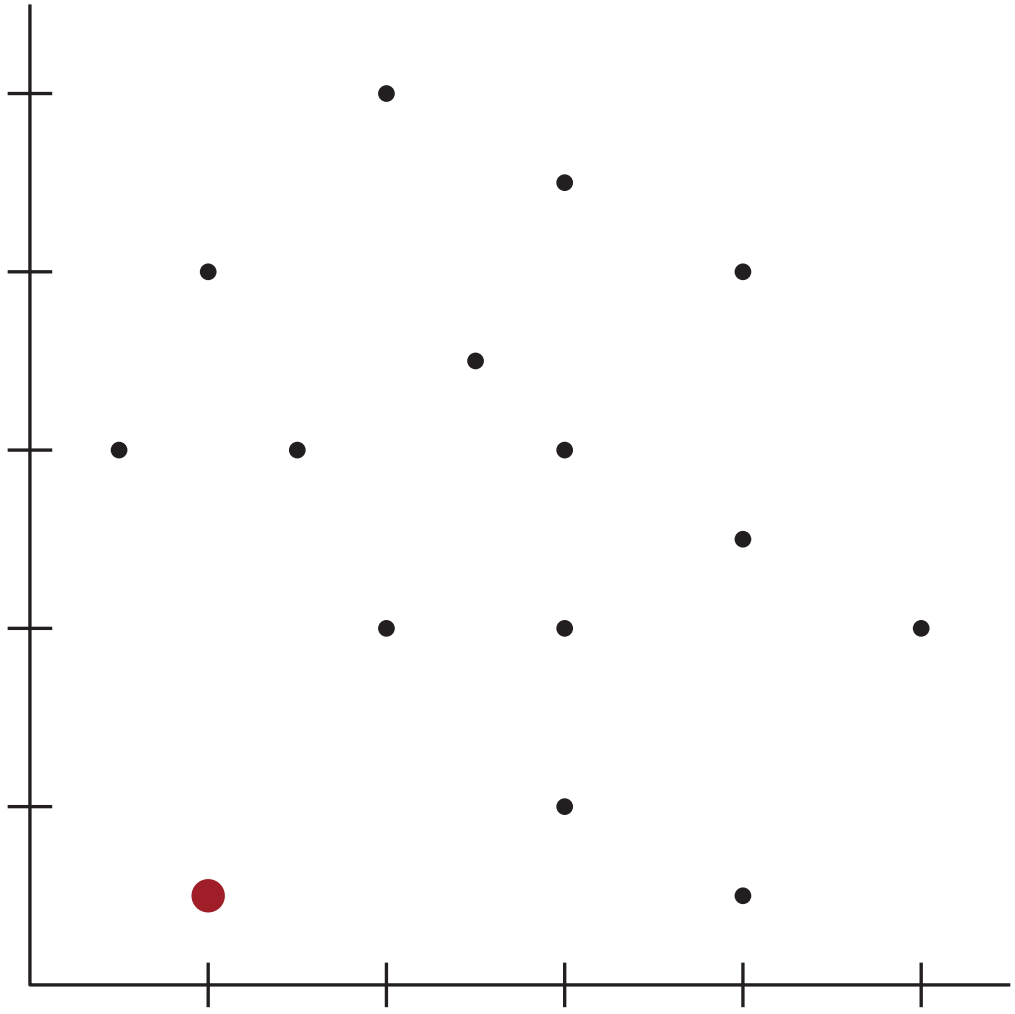
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

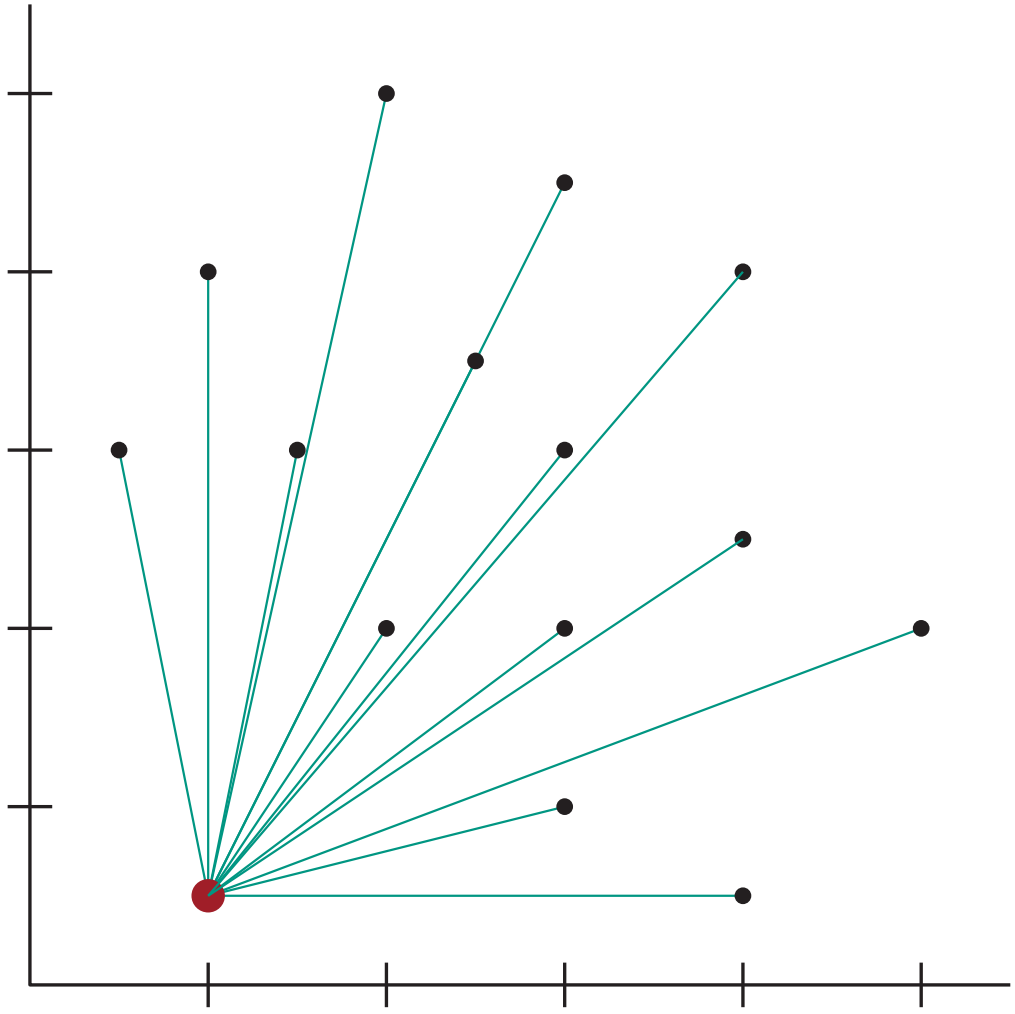
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

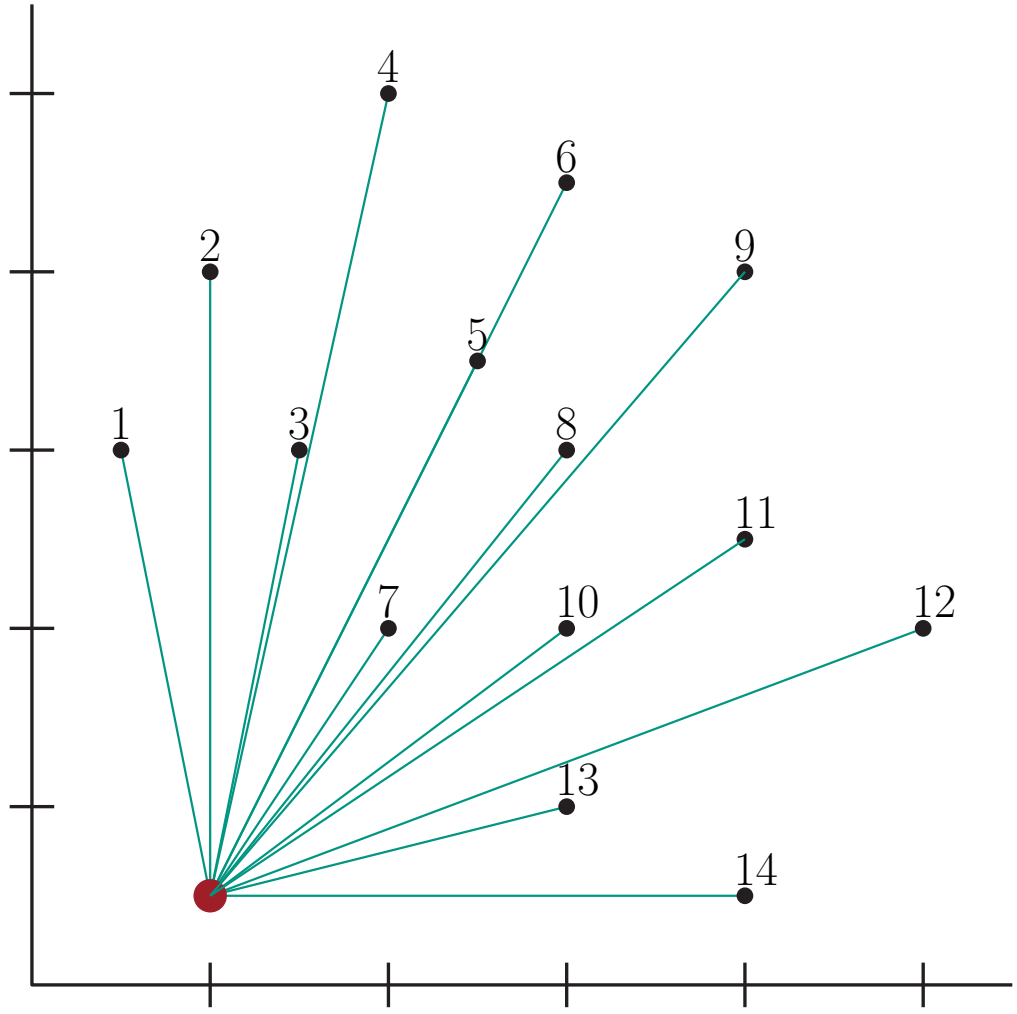
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

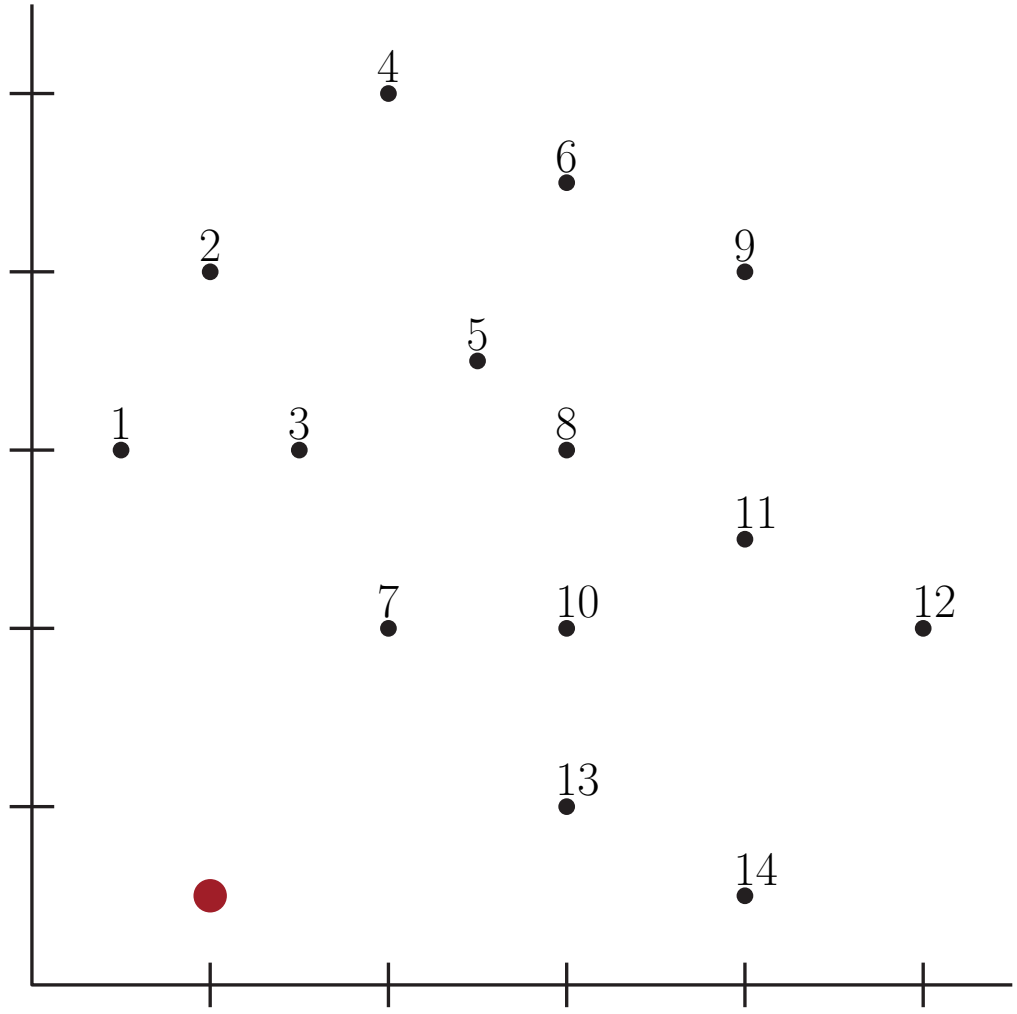
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

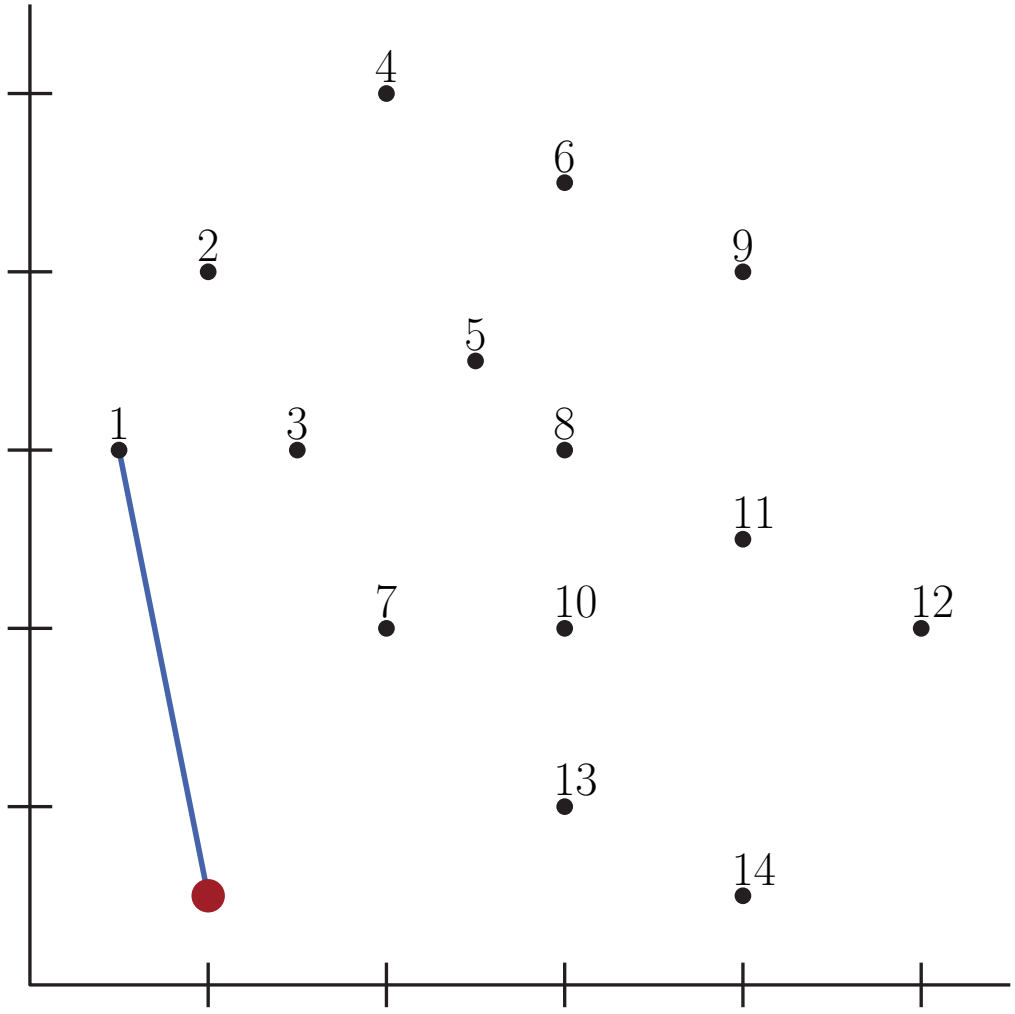
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

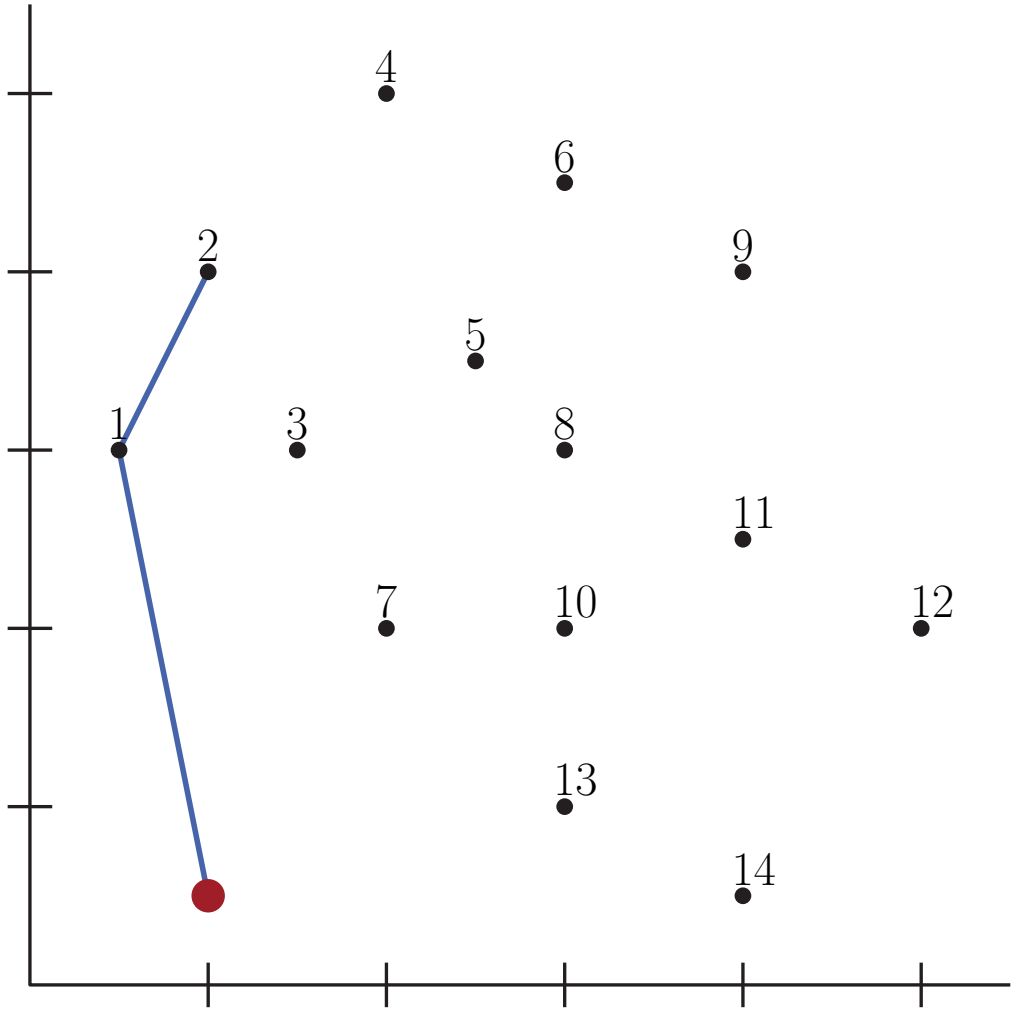
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

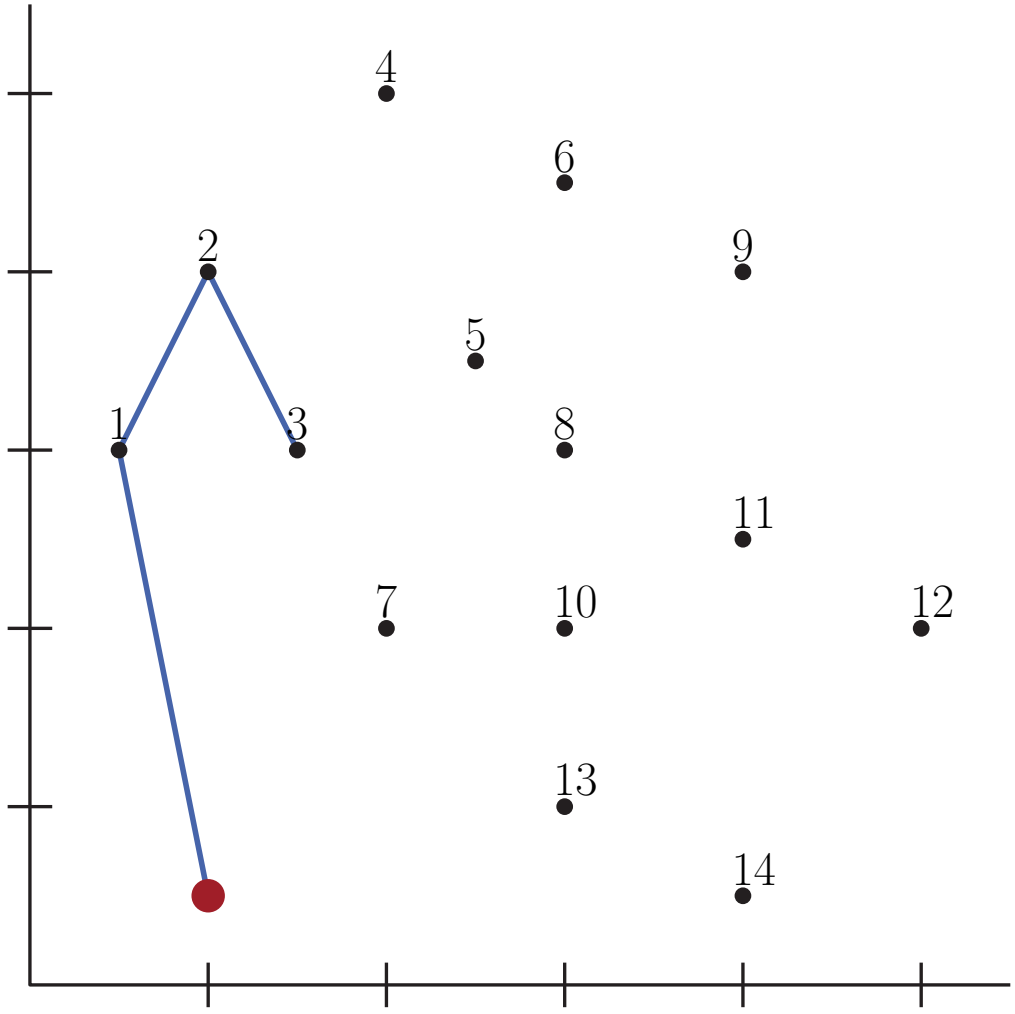
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

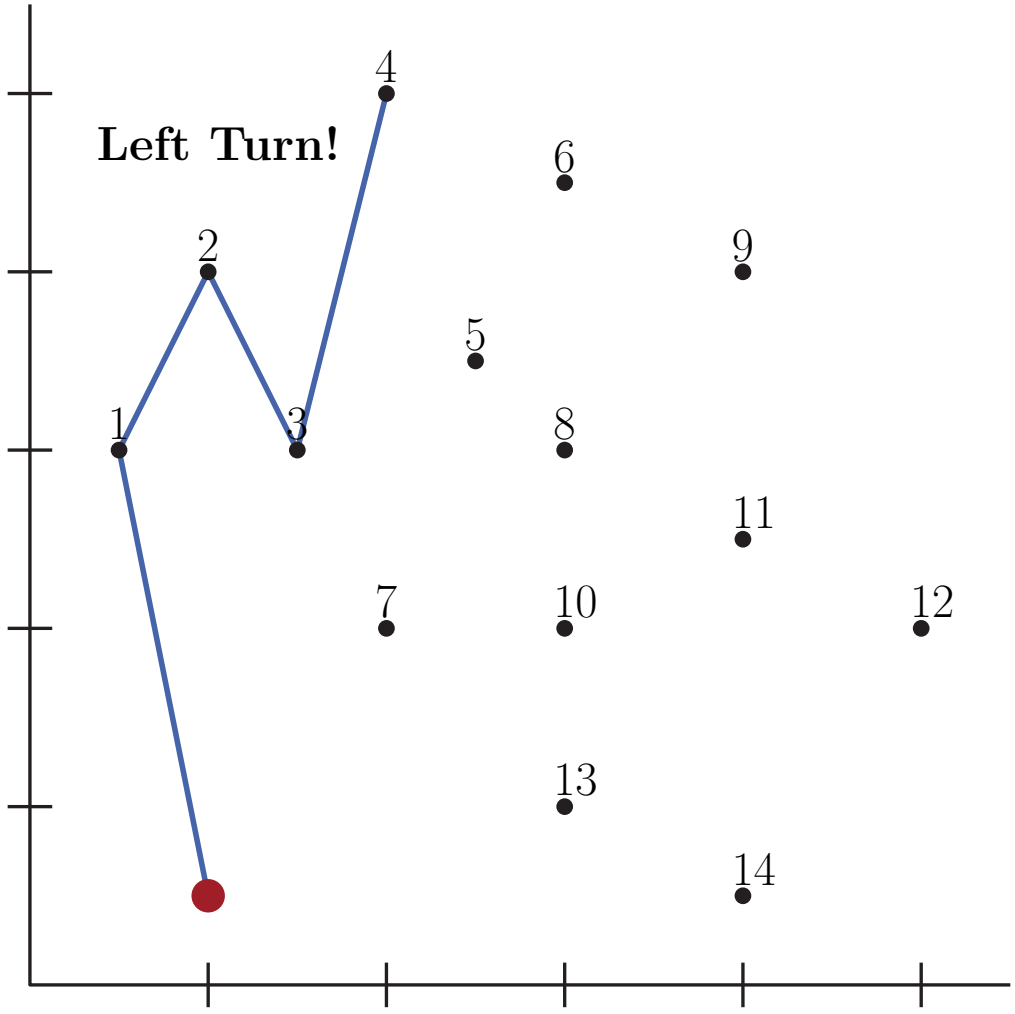
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

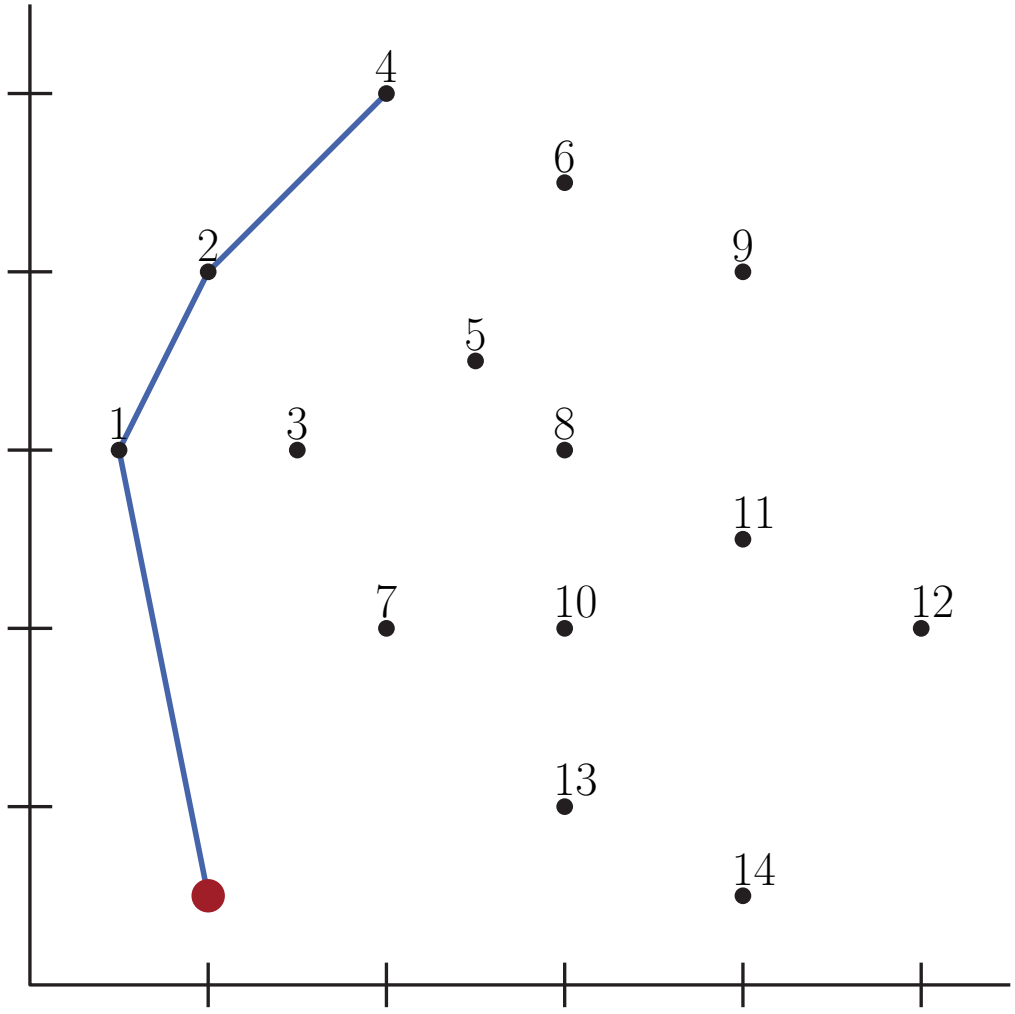
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

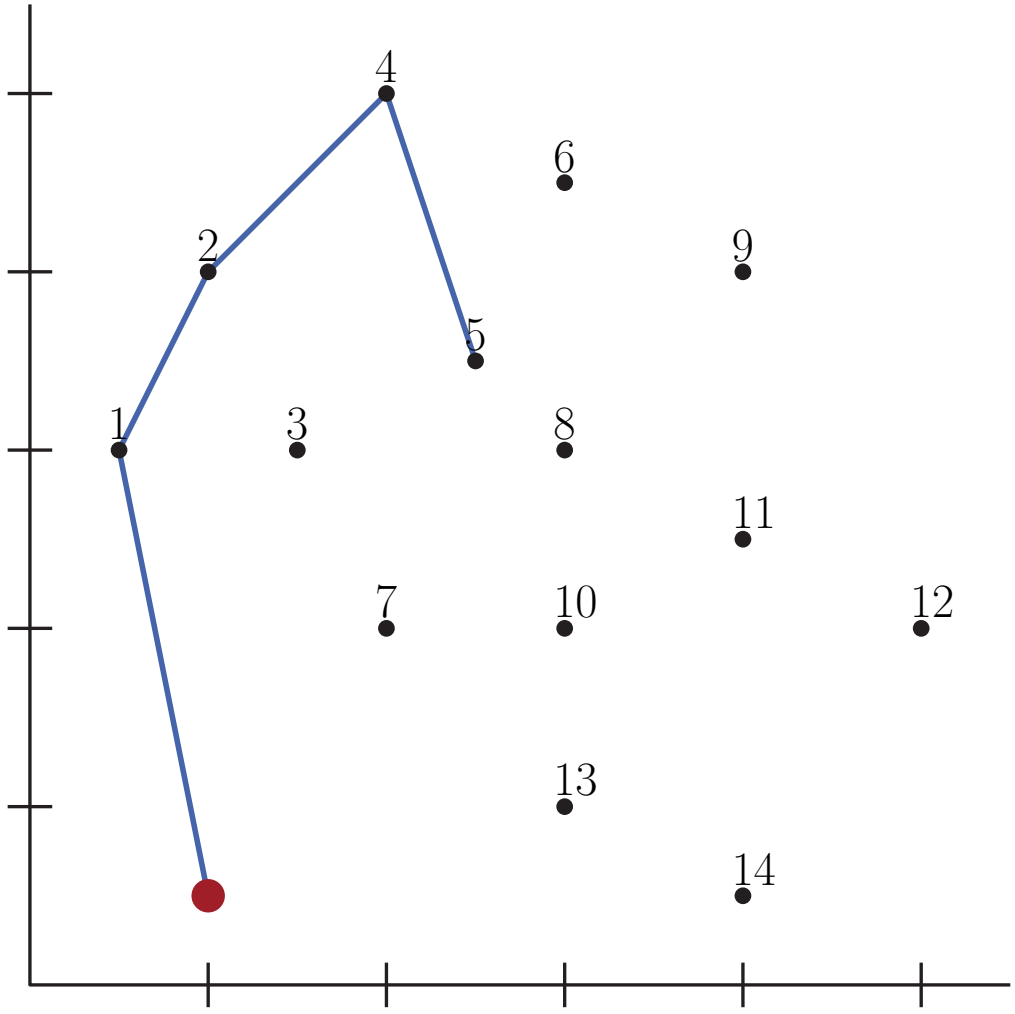
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

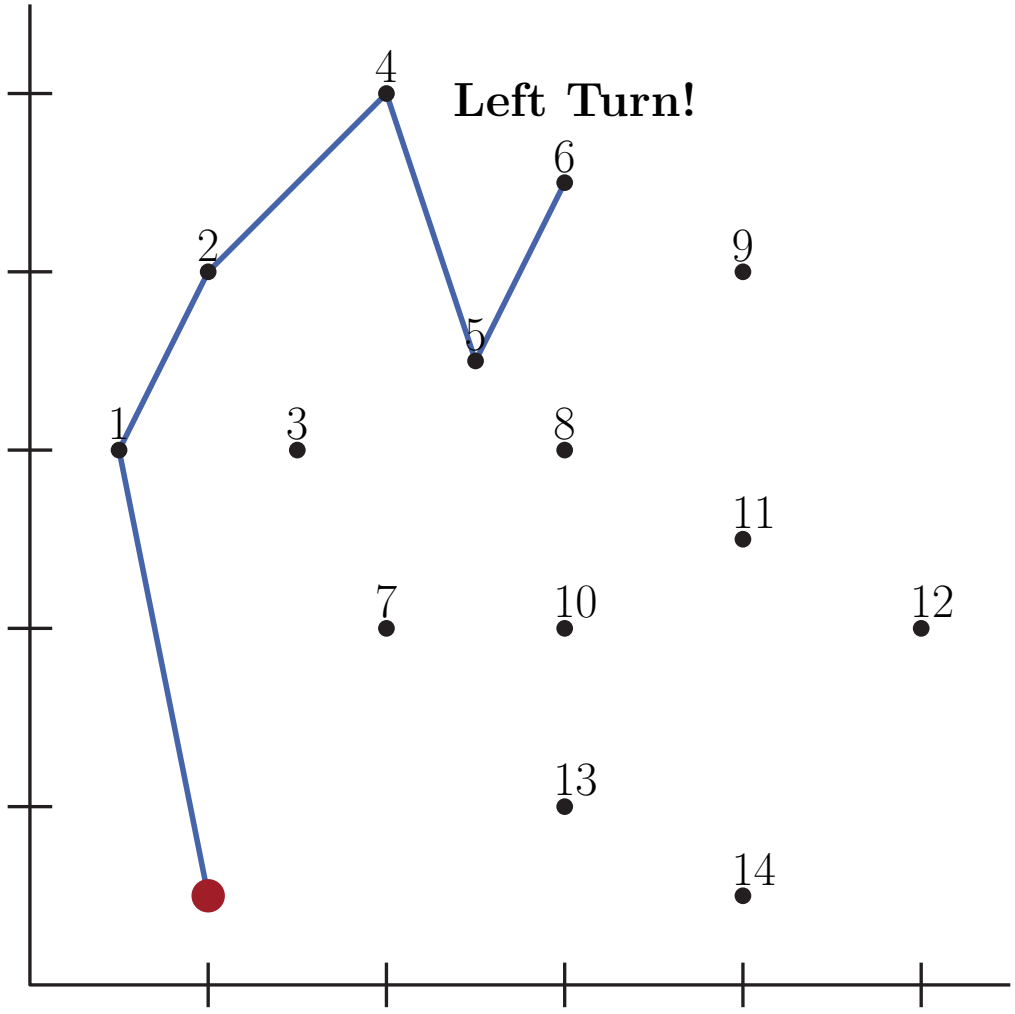
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

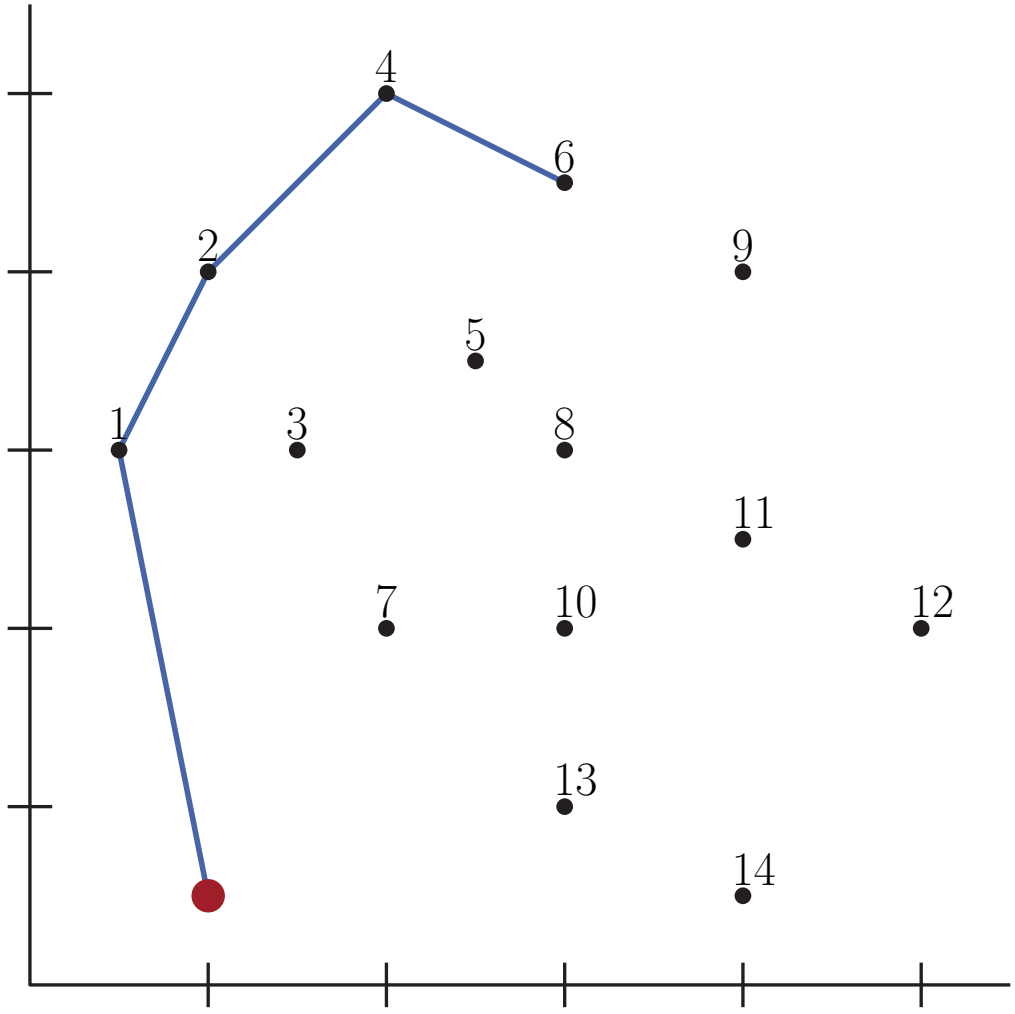
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

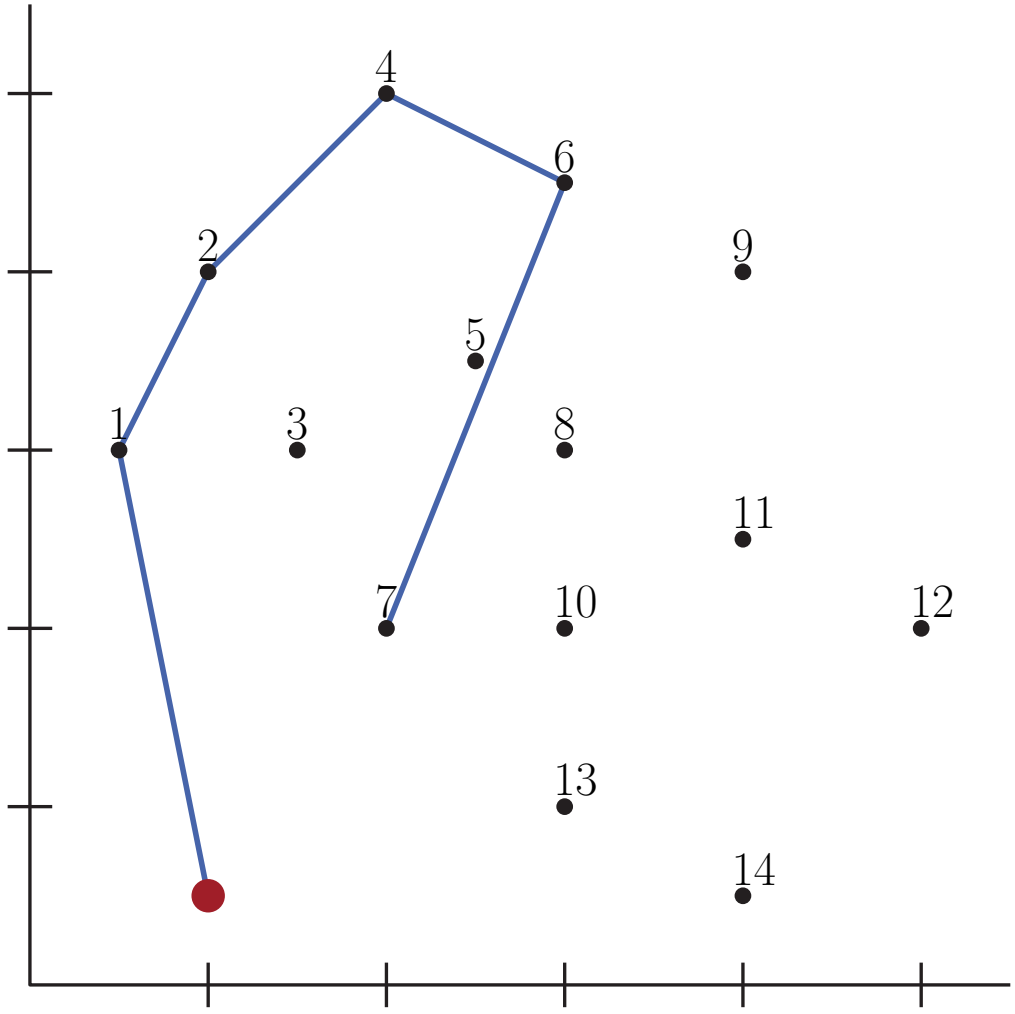
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

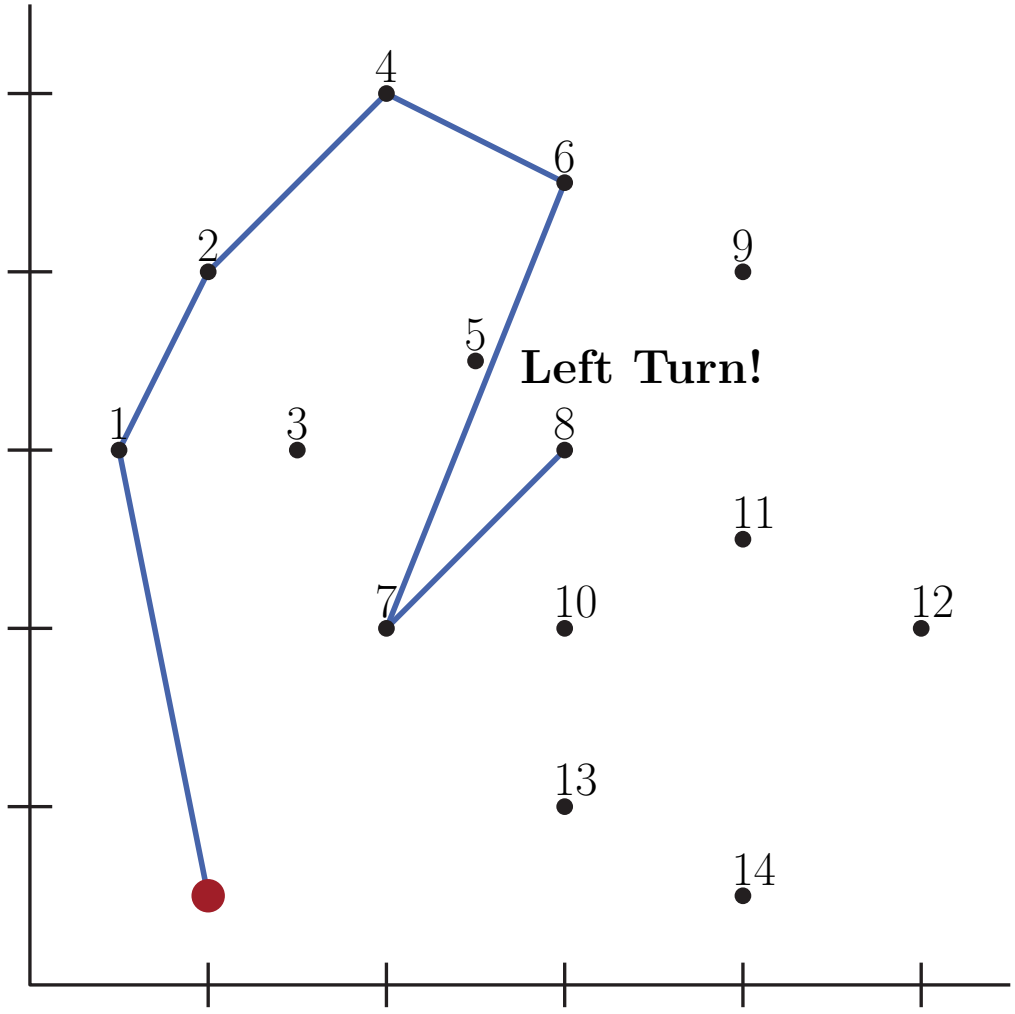
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

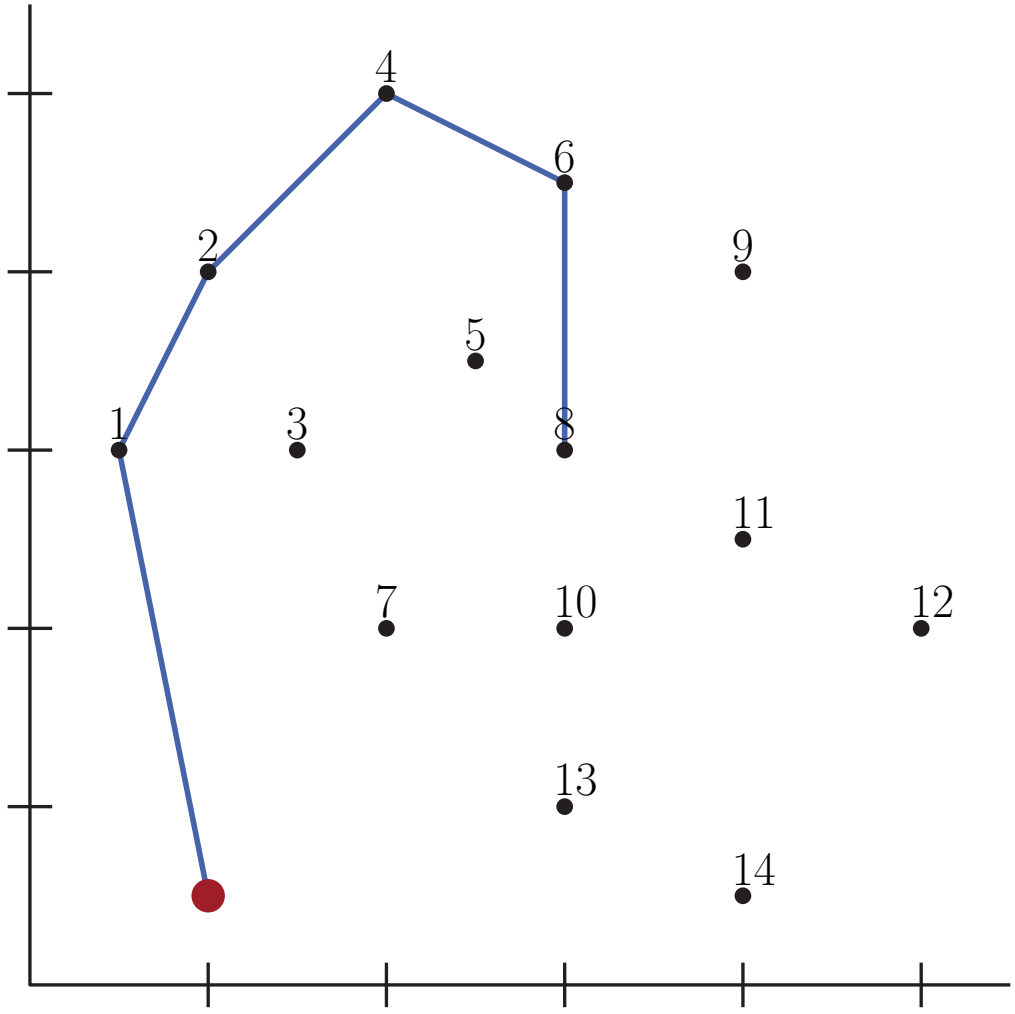
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

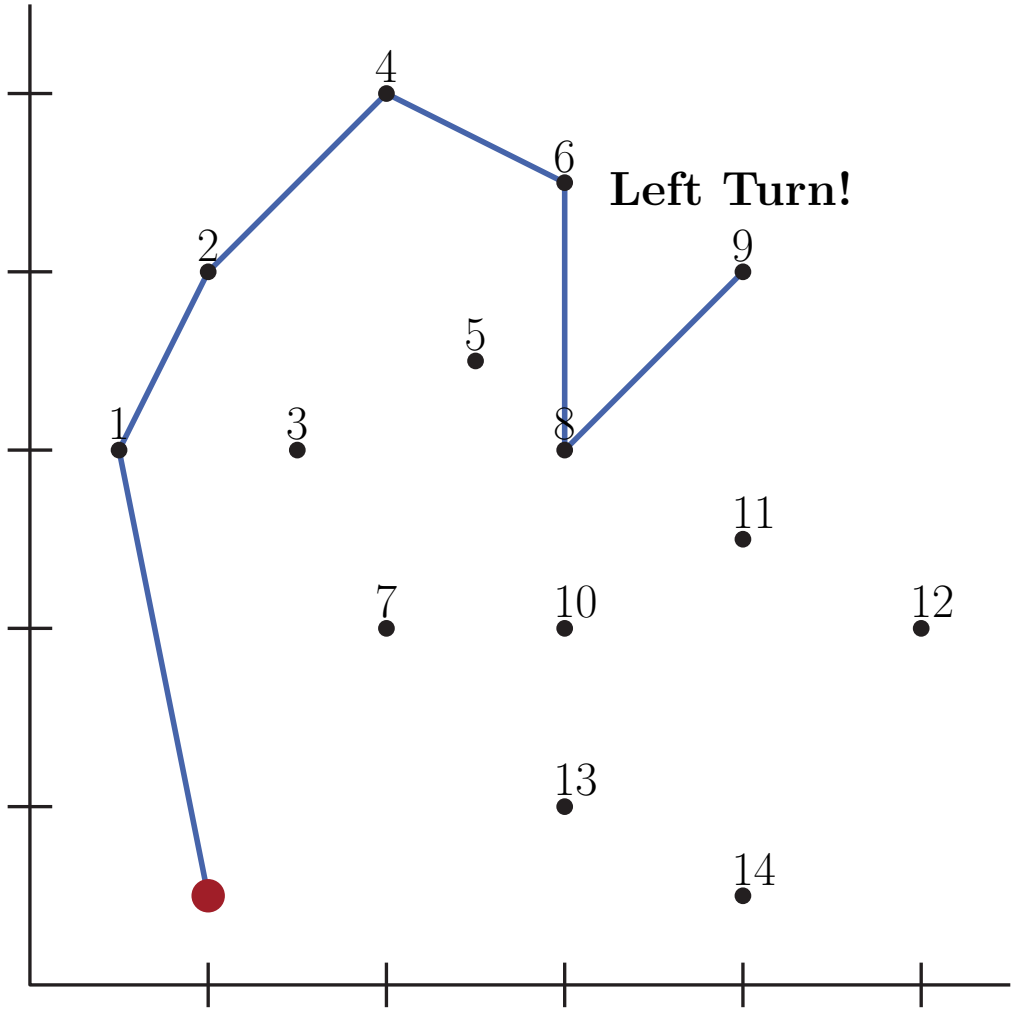
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

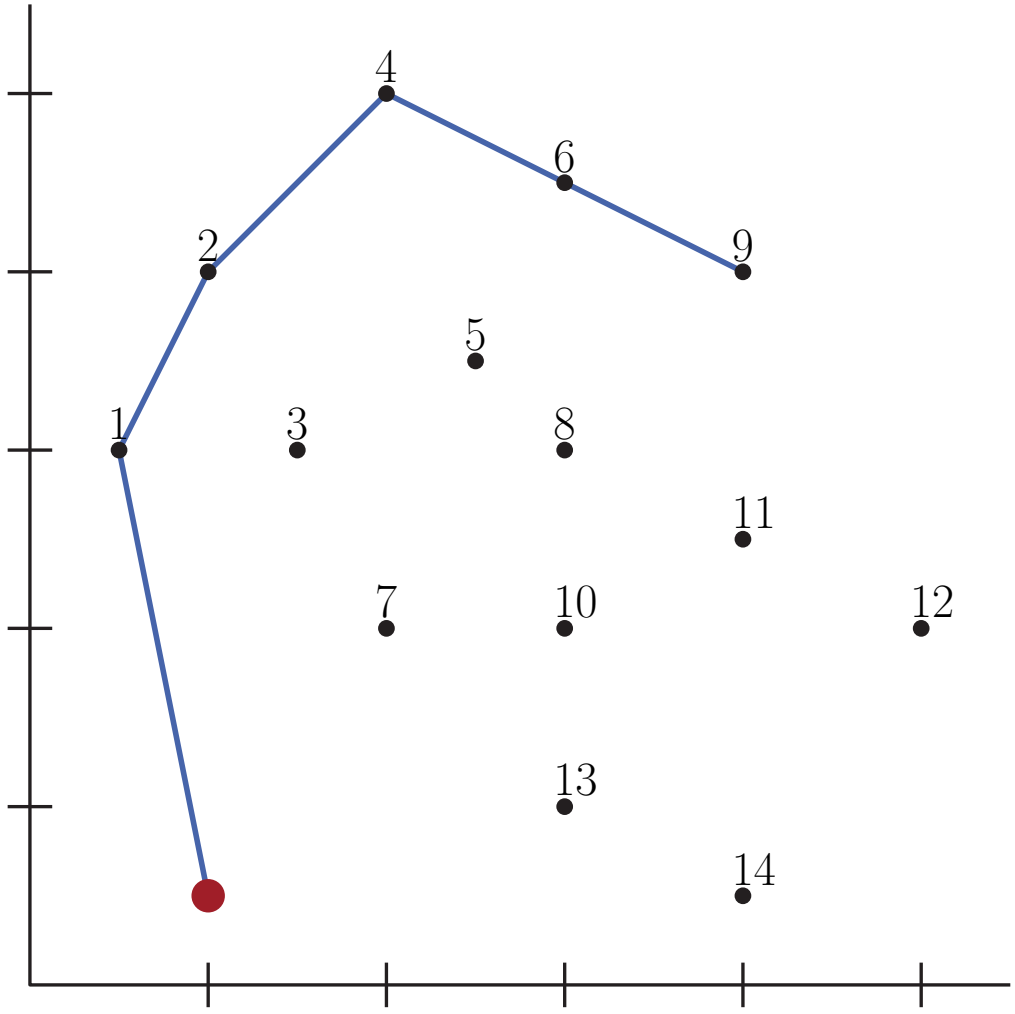
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

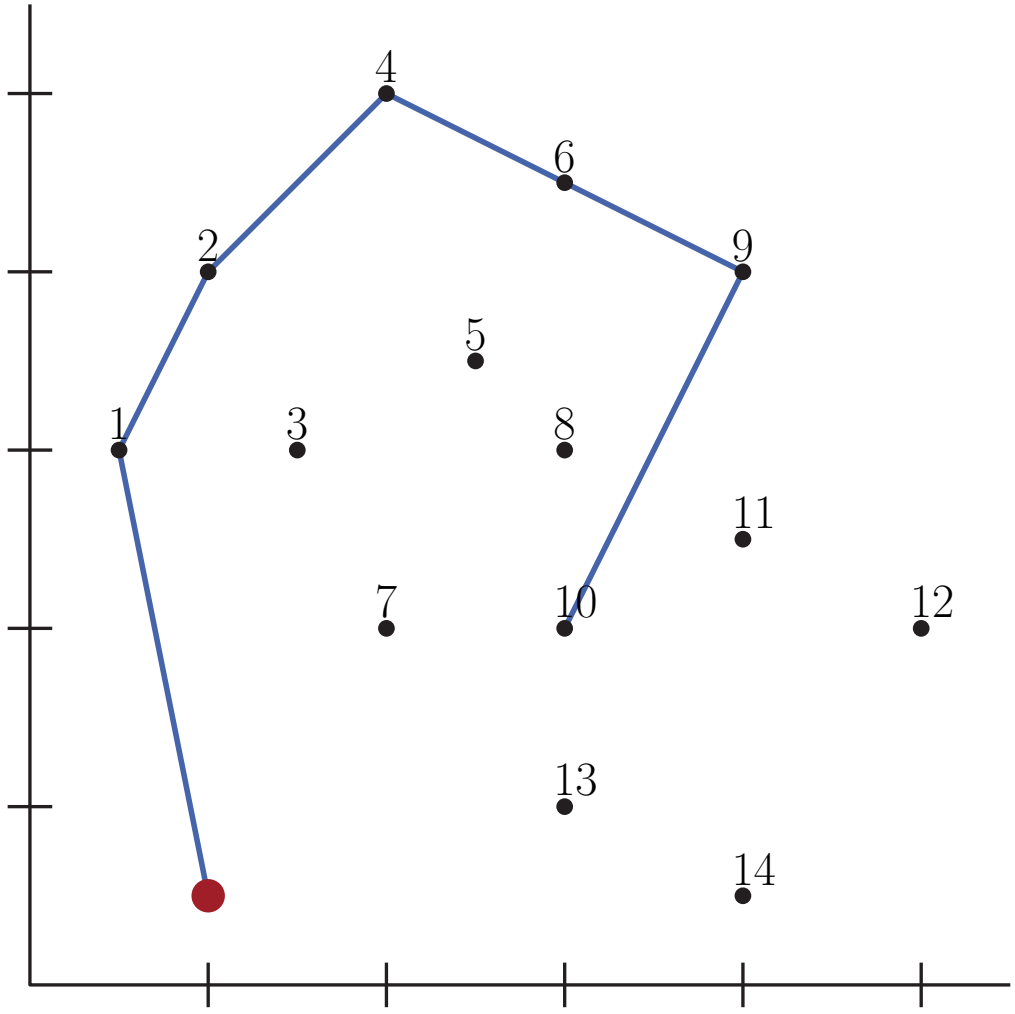
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

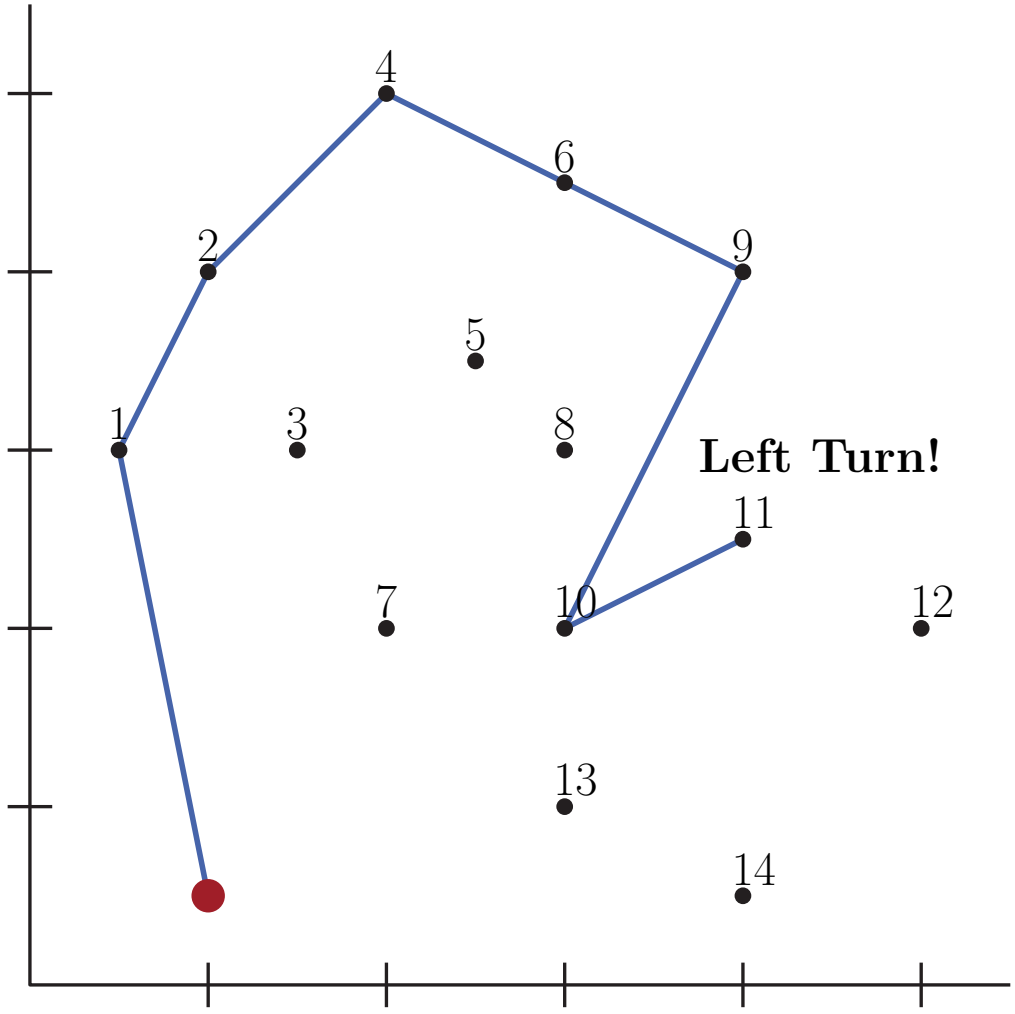
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

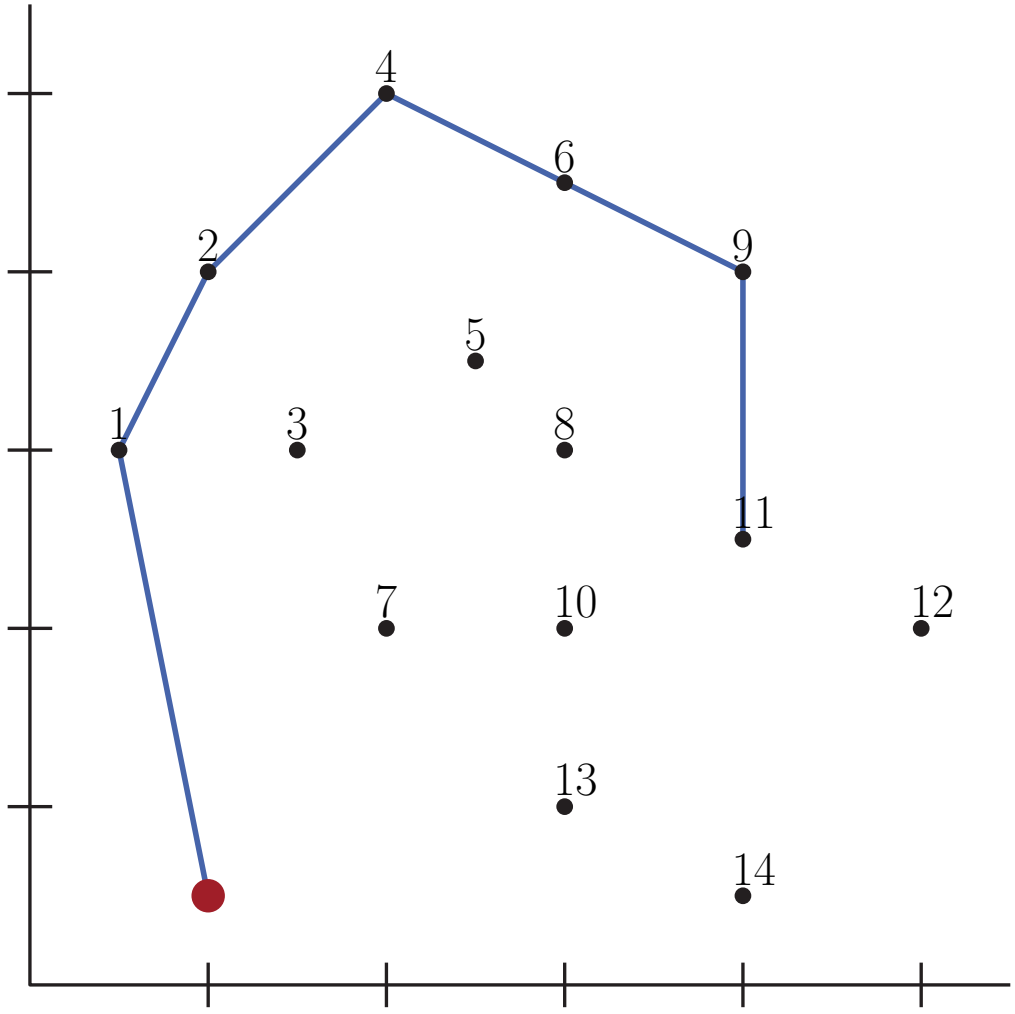
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

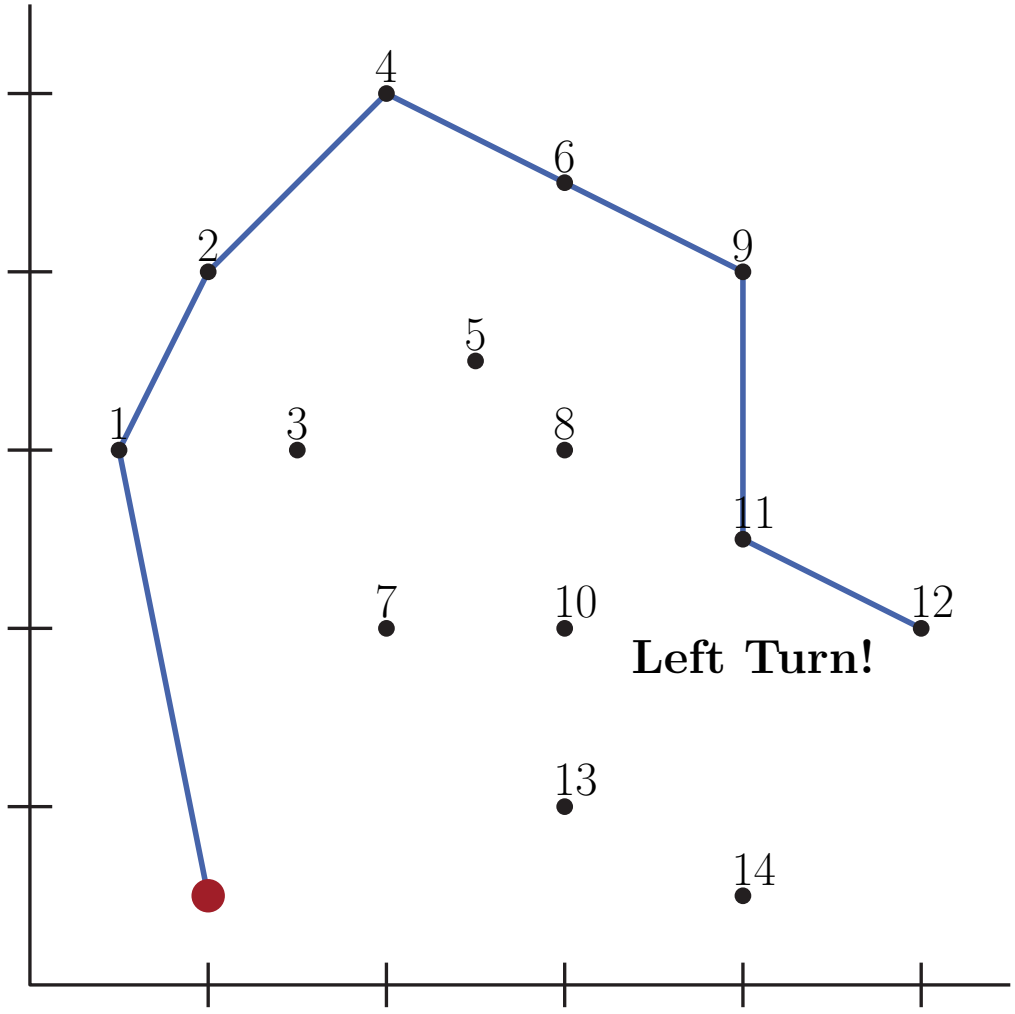
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

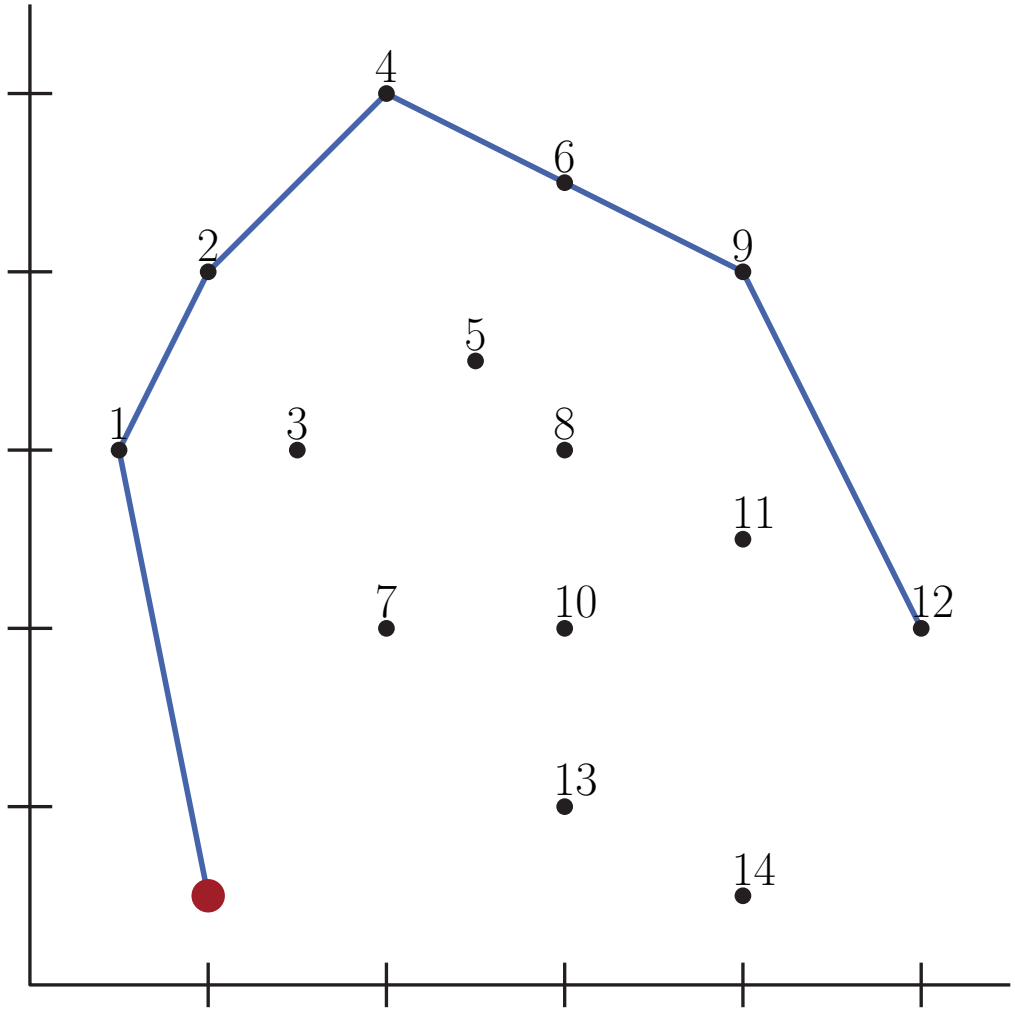
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

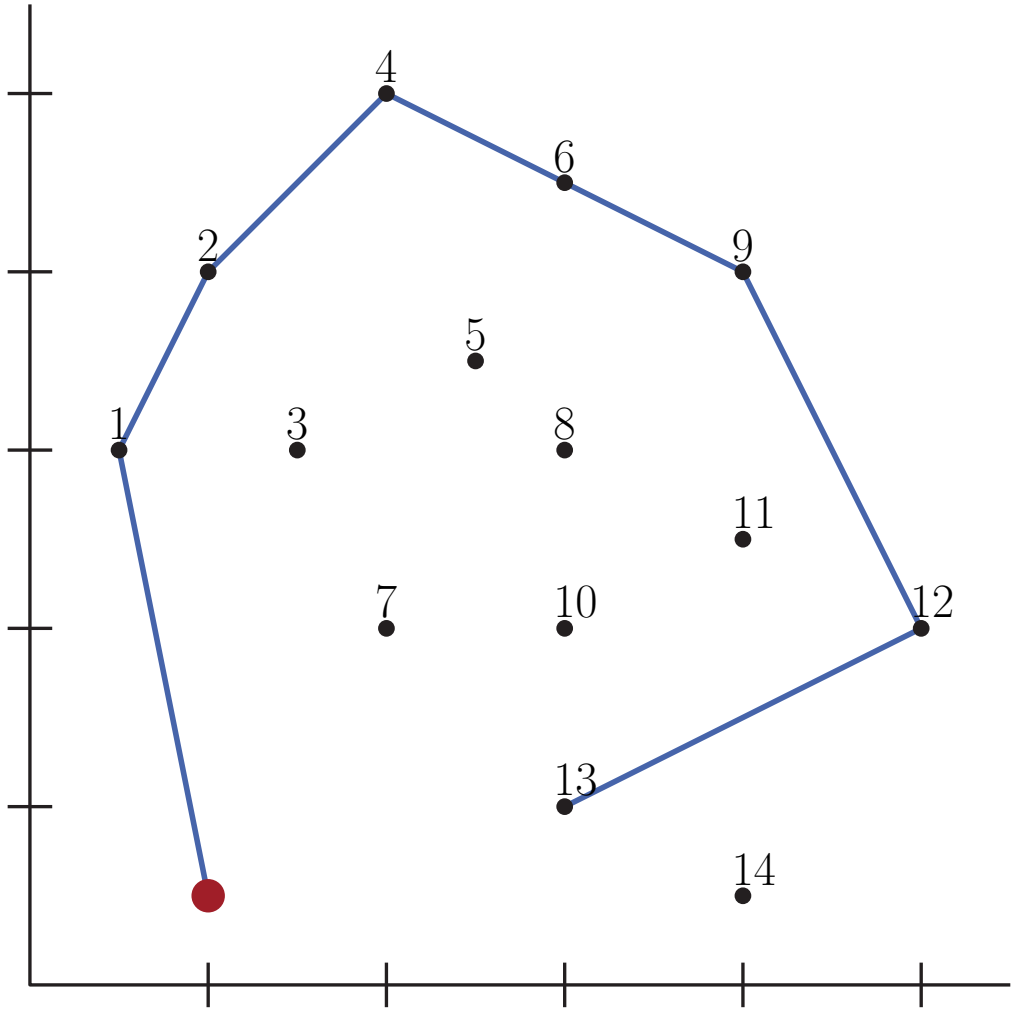
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

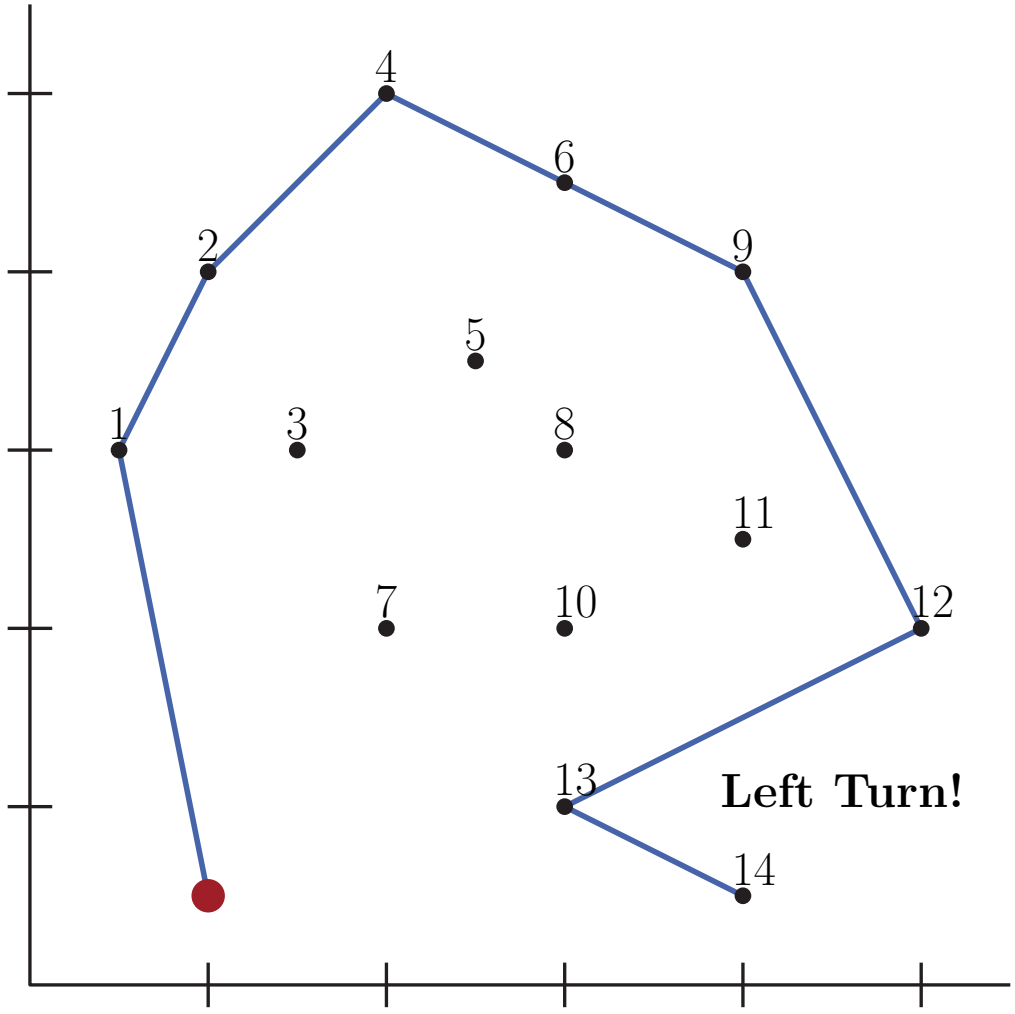
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

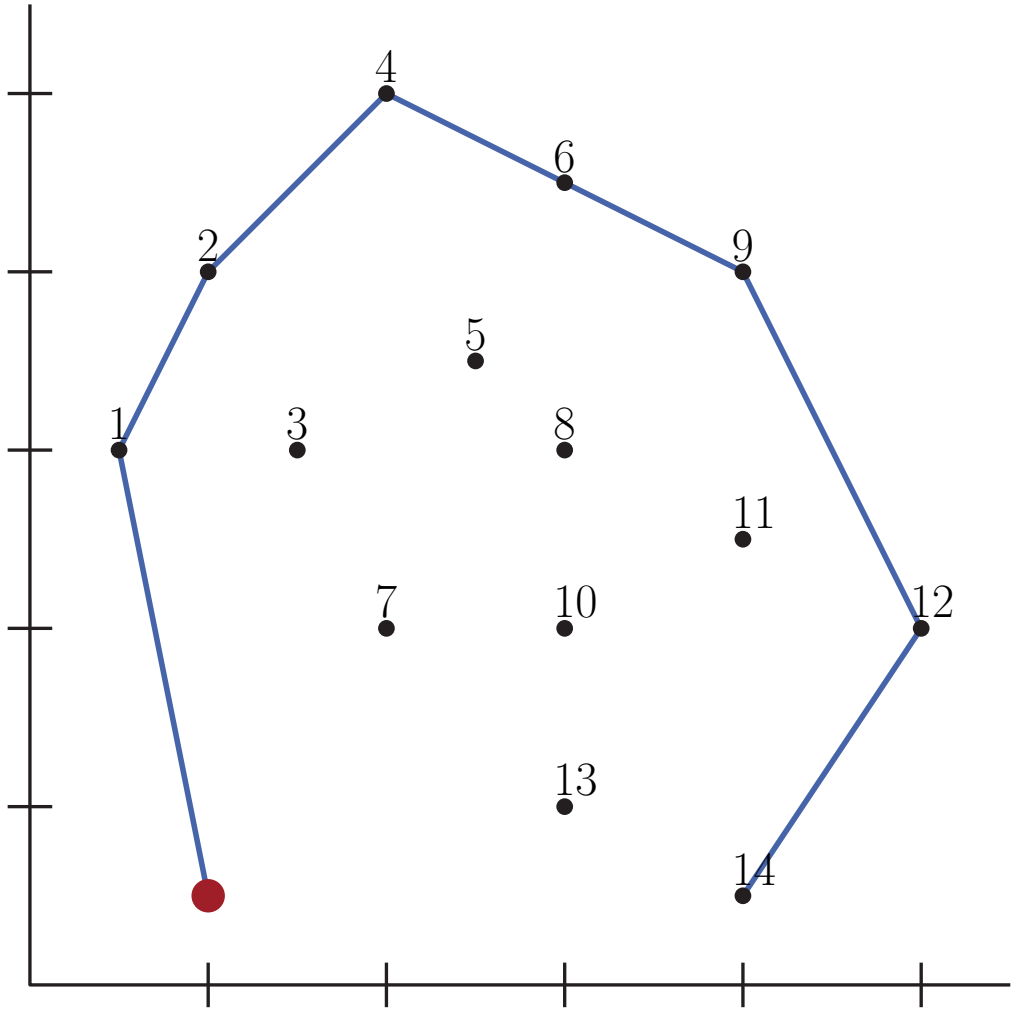
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

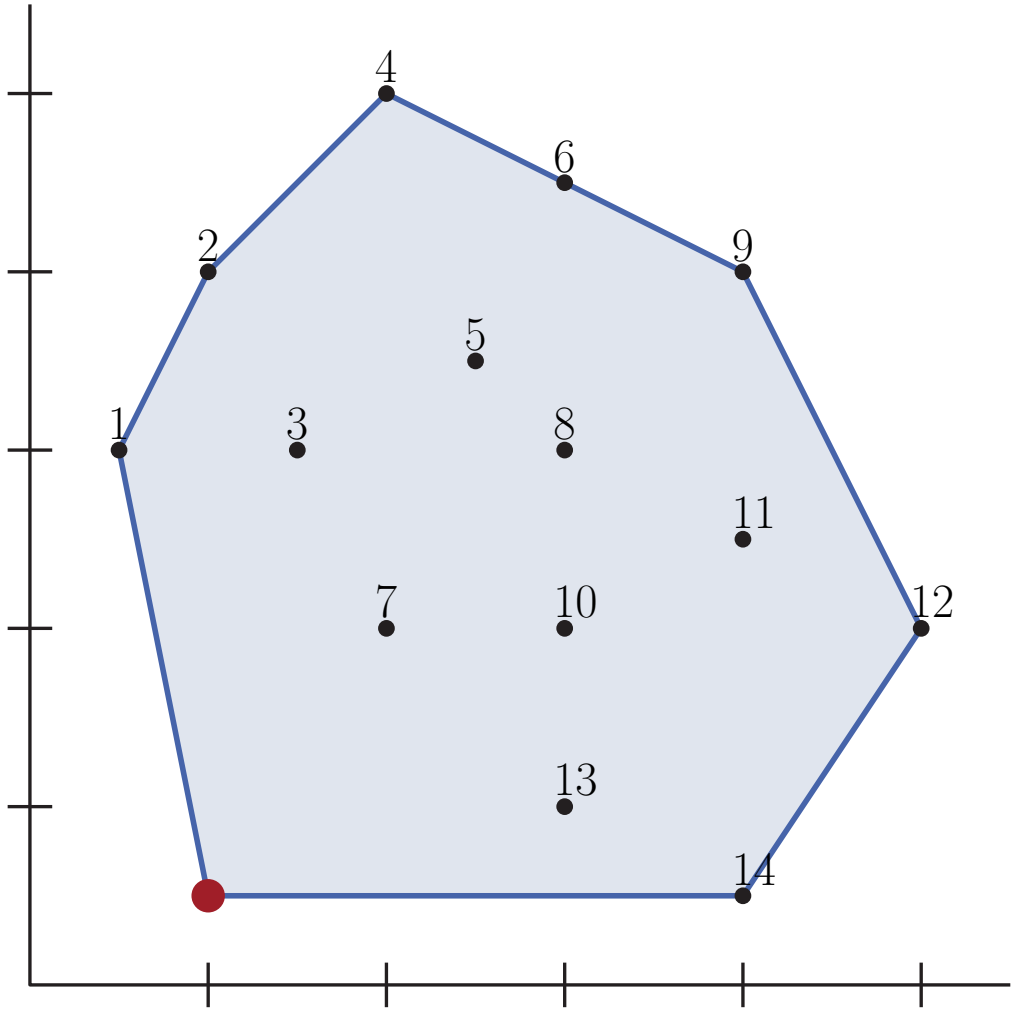
1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1} H_k P_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Konvexe Hülle

Graham-Scan

1. Finde Punkt P_0 mit kleinster y-Koordinate
   - Gibt es mehrere Punkte mit gleicher y-Koordinate, dann nehme Punkt mit kleinster x-Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Itereriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i-ter Punkt in der aktuellen konvexen Hülle)
   - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
   - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Ende!

Feierabend!