Algorithmen / Algorithms II

Peter Sanders

Exercise:

Daniel Seemaier, Tobias Heuer

Institute of Theoretical Informatics

Web:

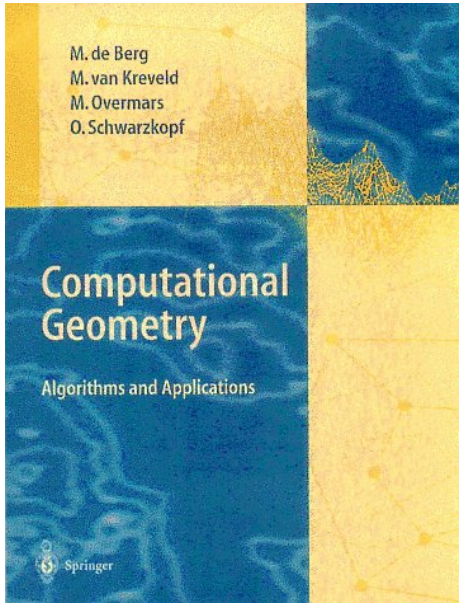
`http://algo2.iti.kit.edu/AlgorithmenII_WS20.php`

12 Geometric Algorithms

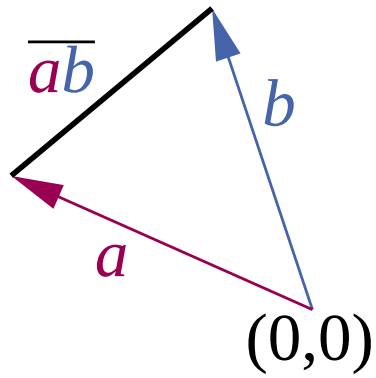
- What are geometric algorithms concerned with?
- Intersection of line segments: Bentley-Ottmann-Algorithm
- Convex hulls
- Smallest enclosing ball
- Range search

Source:

[Computational Geometry – Algorithms and Applications
de Berg, van Kreveld, Overmars, Schwartzkopf
Springer, 1997]



Elementary Geometric Objects



Points: $x \in \mathbb{R}^d$

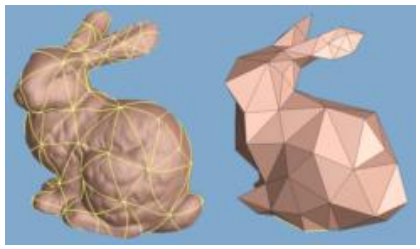
Line segments: $\overline{ab} := \{\alpha a + (1-\alpha)b : \alpha \in [0,1]\}$

Many more: half spaces, planes, curves,...

Dimensionality d:

1: usually trivial, often not considered geometric problem

2: Geographic information systems, image processing,...



3: Computer graphics, simulations,...

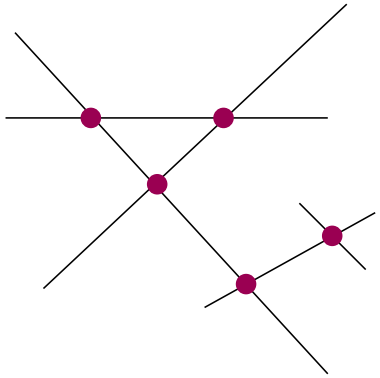
≥ 4: Optimization, databases, machine learning,...

curse of dimensionality!

n: number of input points

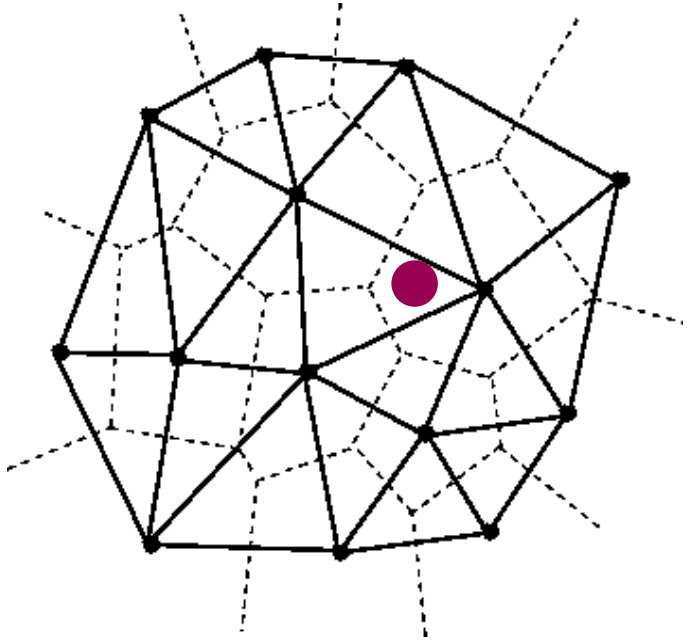
Typical Problems

- Intersection between n line segments



Typical Problems

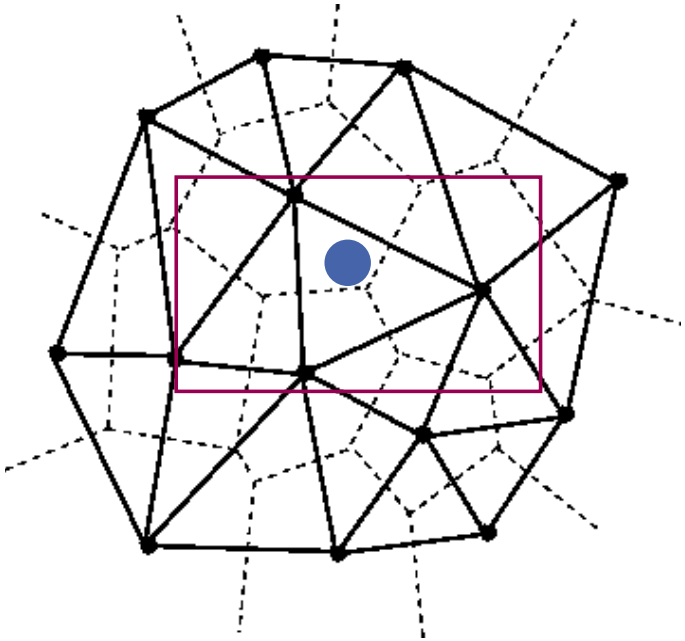
- Intersection between n line segments
- Convex hull
- Triangulating point clouds
  (2D and higher)
  e.g. Delaunay triangulation:
  empty circumcircle criterion



- Voronoi diagram: let $x \in M \subseteq \mathbb{R}^d$.
  $\forall y \in \mathbb{R}^d$ determine closest element in M.
  (Tesselation of M in n Voronoi cells)
- Point location: Given tesselation of R^d, $x \in R^d$:
  in which cell lies x ?

Data Structures for Point Clouds

- Find closest neighbor
- Range queries
- ...



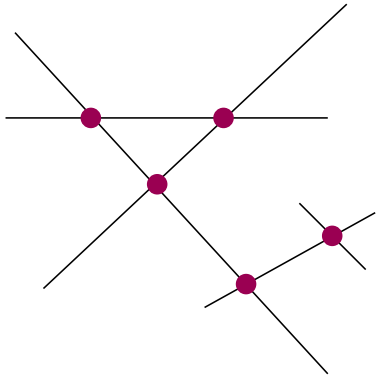
More Problems

- Line of sight
- Linear programming
- Geometric versions of optimization problems
  - Shortest paths, e.g.

    energy efficient communication in radio networks
  - Minimum spanning trees

    reducible to Delaunay triangulation + graph algorithms
  - Matchings
  - Traveling salesmen problem
  - ...
- ...

12.1 Line Segment Intersection (LSI))

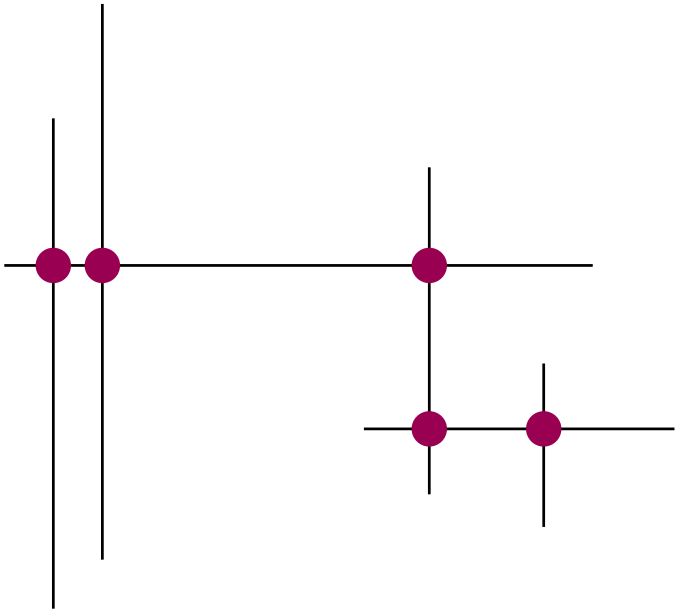


Given: $S = \{s_1, \ldots, s_n\}$, n line segments

Find: intersections $\bigcup_{s,t \in S} s \cap t$

Warm up: Orthogonal line segments

– segments are parallel to x or y axis



LSI: Applications

- Circuit design: where do conductors intersect?
- GIS: road intersections, bridges,. . .
- Extensions:

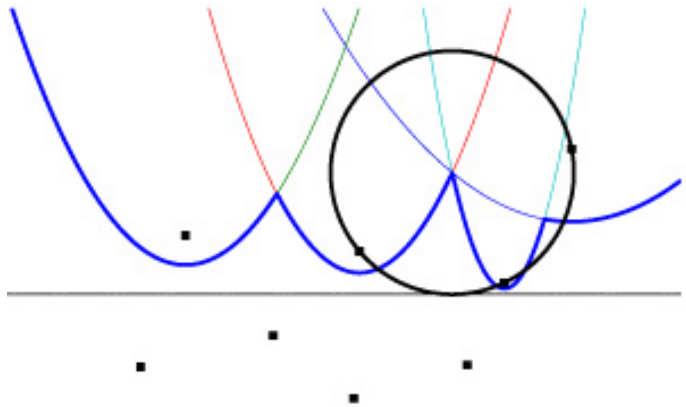
  e.g. construct graphs of neighboring segments/areas
- More general:

  plane-sweep-algorithms

  e.g. construction of

  convex hulls or

  Voronoi diagrams)



LSI: Naive Algorithm

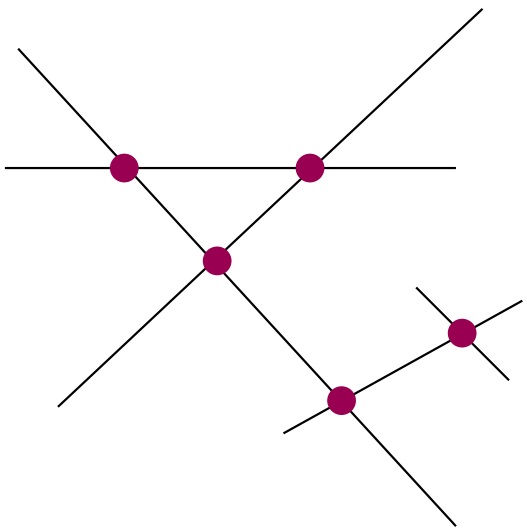
foreach $\{s,t\} \subseteq S$ **do**

  if $s \cap t \neq \emptyset$ **then**

    output $\{s,t\}$

Problem: time $\Theta\left(n^2\right)$.

infeasible for large inputs

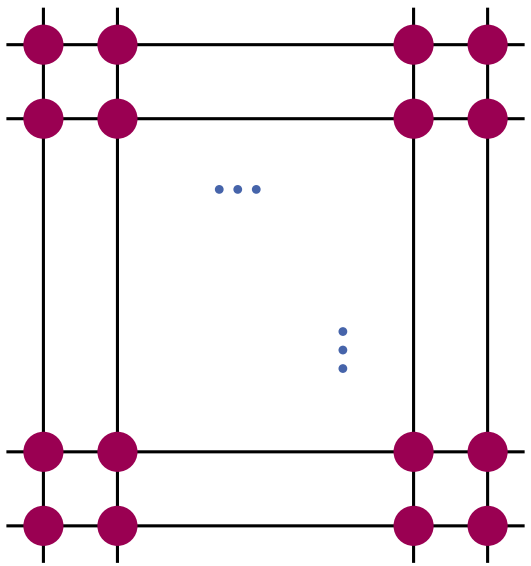


LSI: Lower Bound

$\Omega(n+k)$ with k:= number of found intersections

Comparison-based: $\Omega(n \log n + k)$ (Proof: not here)

Note: $k = \Theta(n^2)$ possible, but in real inputs often $k = \mathrm{O}(n)$.

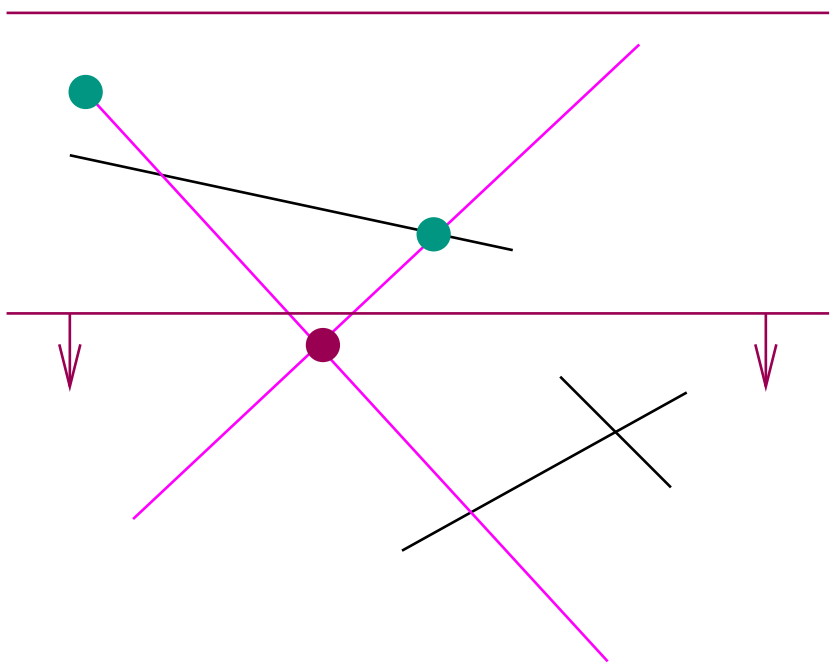


Idea: Plane-Sweep-Algorithm

(Horizontal) line ℓ sweeps from top to bottom.

Invariant: intersections above ℓ correctly found

Idea: save segments intersecting ℓ and find their intersections.



Plane-Sweep for Orth. Segments

For now: Intersections of vertical and horizontal segments.

$T = \langle\rangle$: SortedSequence **of** Segment // e.g., (a,b)-tree

invariant T stores the vertical segments intersecting ℓ

Q:= sort($\langle (y,s) : \exists$hor. seg. s at y or $\exists$vert. seg. s starting/ending at $y \rangle$)

// tie breaking: vert. starting events first, vert. finishing events last

foreach $(y,s) \in Q$ in descending order **do**

  if s is a vertical segment and starts at y **then** T.insert(s)

  else if s is a vertical segment and ends at y **then** T.remove(s)

  else // we have a horizontal segment $s = \overline{(x_1,y)(x_2,y)}$

    foreach $t = \overline{(x,y_1)(x,y_2)} \in T$ with $x \in [x_1,x_2]$ **do**

      output $\{s,t\}$

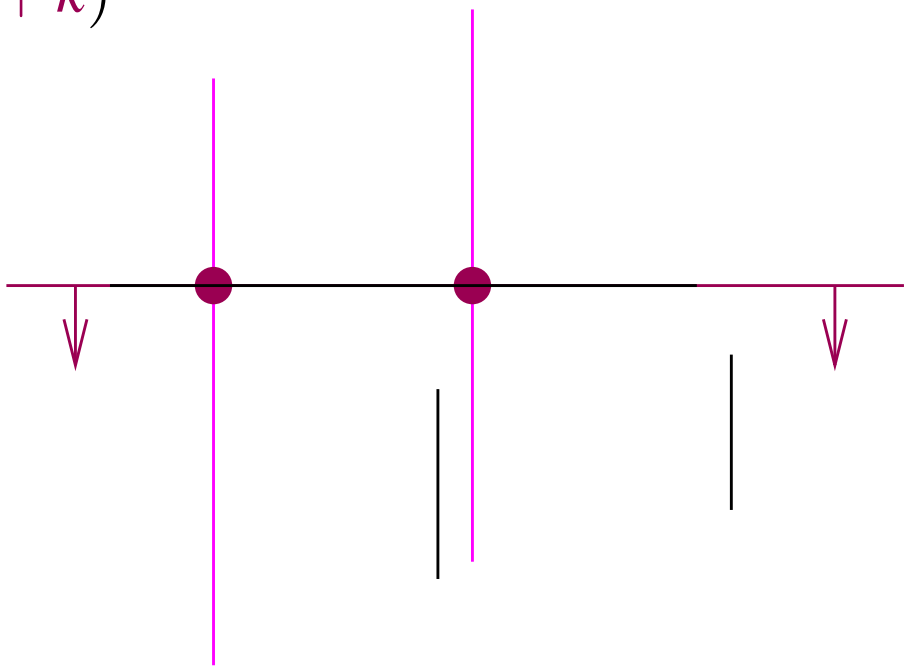
Analysis Orth. Segments

insert: $\mathrm{O}(\log n)$ $(\leq n\times)$

remove: $\mathrm{O}(\log n)$ $(\leq n\times)$

rangeQuery: $\mathrm{O}(\log n + k_s)$, k_s intersections with hor. segments s

Total: $\mathrm{O}(n\log n + \sum_s k_s) = \mathrm{O}(n\log n + k)$

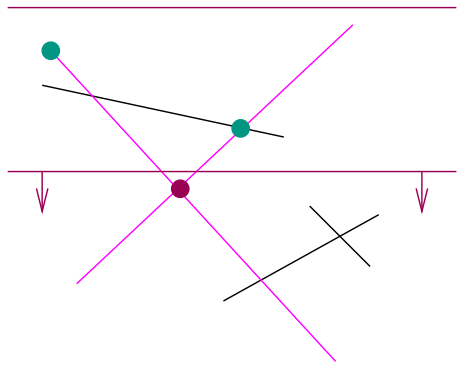


"Almost" General Case'

Assume: general position, i.e.

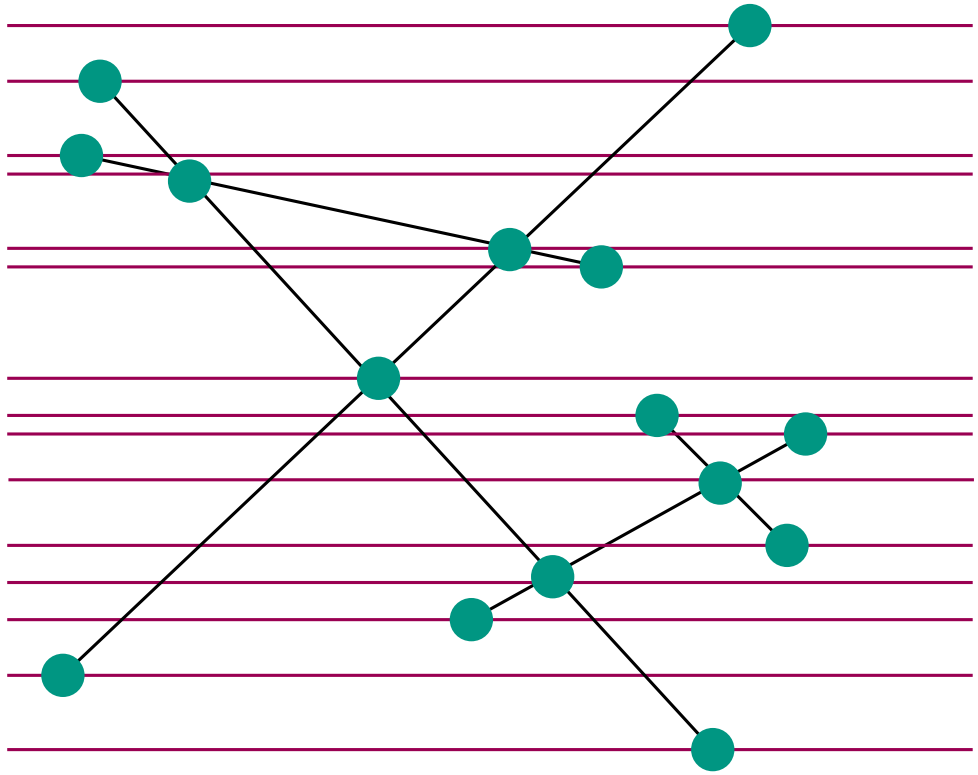
- ☐ No horizontal segments
- ☐ No overlapping segments
- ☐ Exactly two segments intersect in one point
- ☐ Intersections in interior of segments

Note: small random perturbations can produce general position



General Case – Basic Idea

- Plane sweep with sweep line ℓ
- State T := segments intersecting ℓ – ordered by x
- Event:= state change
  - Starting point
  - End point
  - Intersection
- Intersection test
  only for segments,
  neighboring an event point in T.



General Case – Correctness

Lemma:

$s \cap t = \{(x, y)\} \longrightarrow \exists$ event : s, t become neighbors on ℓ

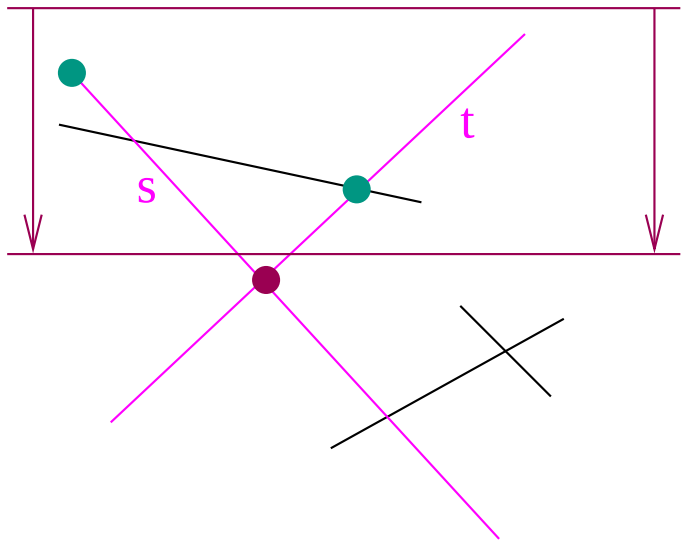
Proof:

Initially : $T = \langle\rangle \longrightarrow s, t$ not neighboring in ℓ.

@$y + \varepsilon$: s, t are neighboring in ℓ.

$\longrightarrow$

$\exists$ event: s and t become neighbors.



General Case – Implementation

$T = \langle\rangle$: SortedSequence **of** Segment

invariant T stores the relative order of the segments intersecting ℓ

Q : MaxPriorityQueue

$Q := Q \cup \left\{ (\max\{y, y'\}, \text{start}, s) : s = \overline{(x,y)(x',y')} \in S \right\}$ // $O(n \log n)$

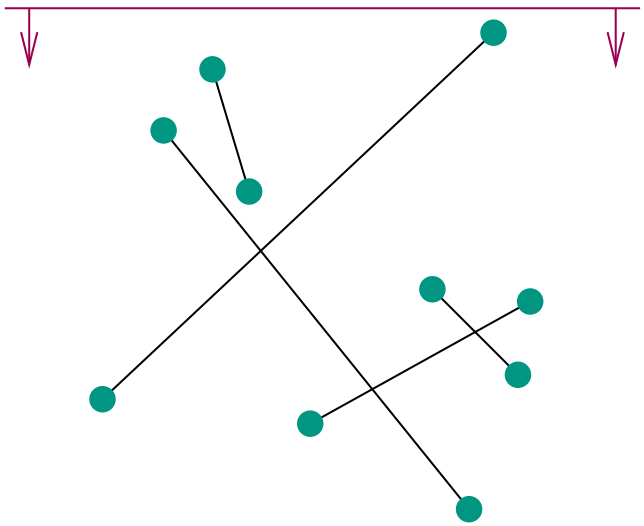
$Q := Q \cup \left\{ (\min\{y, y'\}, \text{finish}, s) : s = \overline{(x,y)(x',y')} \in S \right\}$ // $O(n \log n)$

while $Q \neq \emptyset$ **do**

  $(y, \text{type}, s) := Q.\text{deleteMax}$

  // $O((n+k)\log n)$

  handleEvent$(y, \text{type}, s, T, Q)$



handleEvent$(y, \text{start}, s, T, Q)$ // $n\times$

h:= T.insert(s) // $\mathrm{O}(\log n)$

prev:= pred(h) // $\mathrm{O}(1)$

next:= succ(h) // $\mathrm{O}(1)$

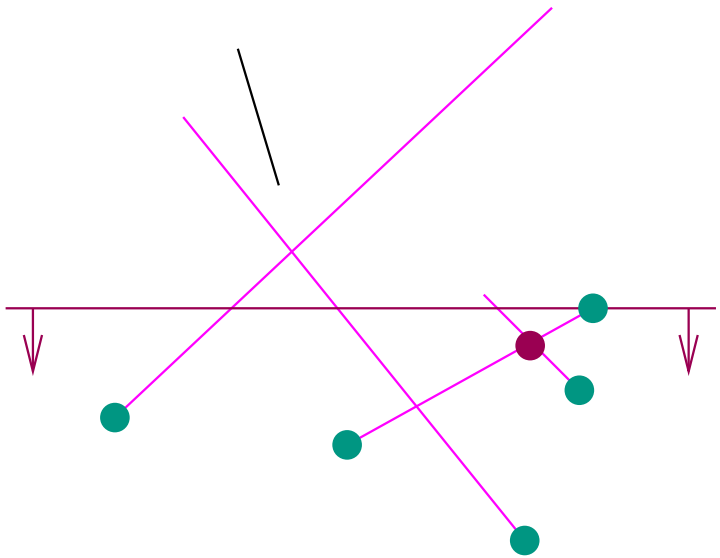
findNewEvent(prev, h)

findNewEvent(h, next)

Procedure findNewEvent(s,t) // $\mathrm{O}(1+\log n)$

if $*s$ and $*t$ intersect at $y' < y$ **then**

Q.insert$((y', \text{intersection}, (s,t)))$



handleEvent$(y, \text{finish}, s, T, Q)$ // $n\times$

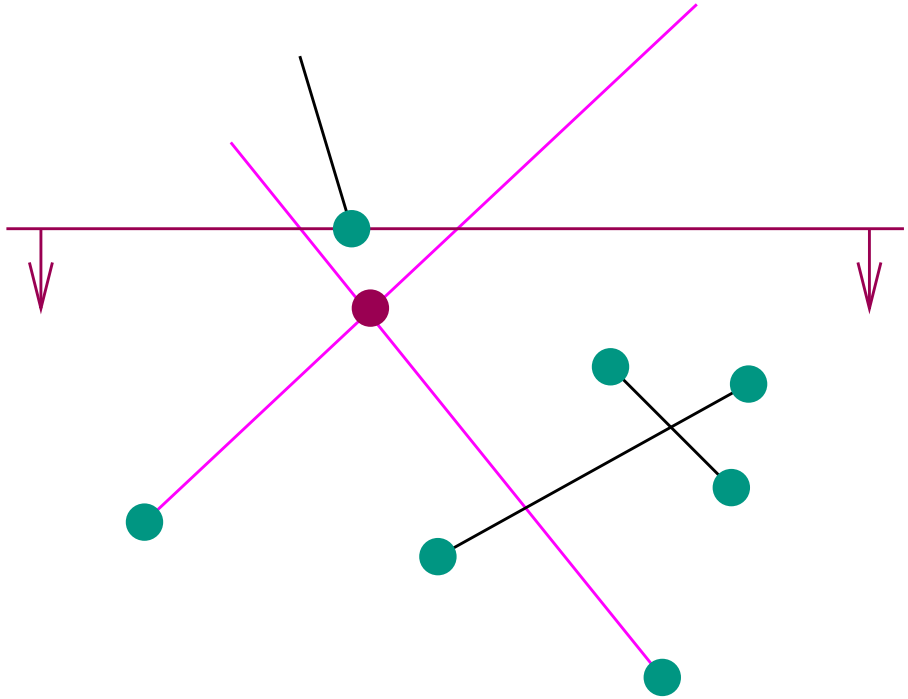
h:= T.locate(s) // $\mathrm{O}(\log n)$

prev:= pred(h) // $\mathrm{O}(1)$

next:= succ(h) // $\mathrm{O}(1)$

T.remove(s) // $\mathrm{O}(\log n)$

findNewEvent(prev, next)



handleEvent$(y,$ intersection$,(a,b),T,Q)$ // $k\times$

output $(*s \cap *t)$ // $\mathrm{O}(1)$

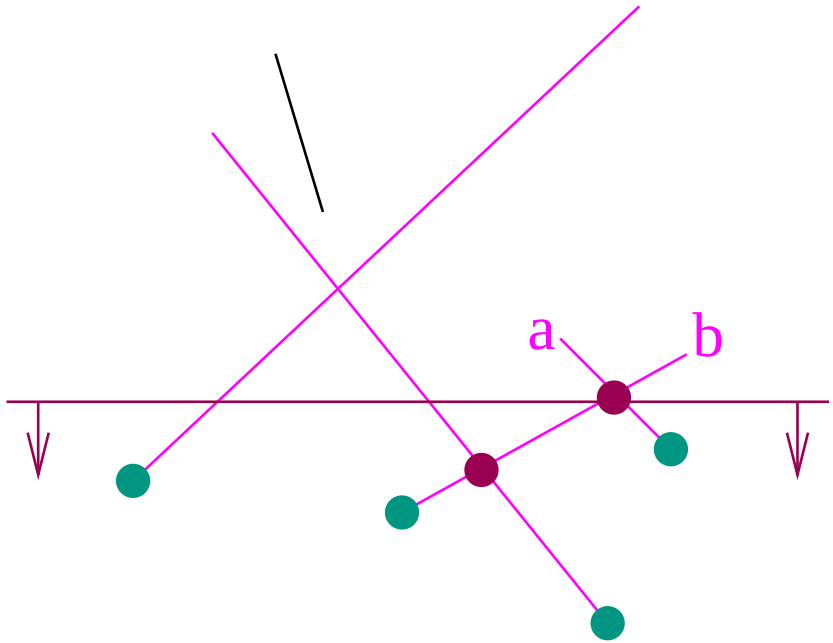
T.swap(a,b) // $\mathrm{O}(\log n)$

prev:= pred(b) // $\mathrm{O}(1)$

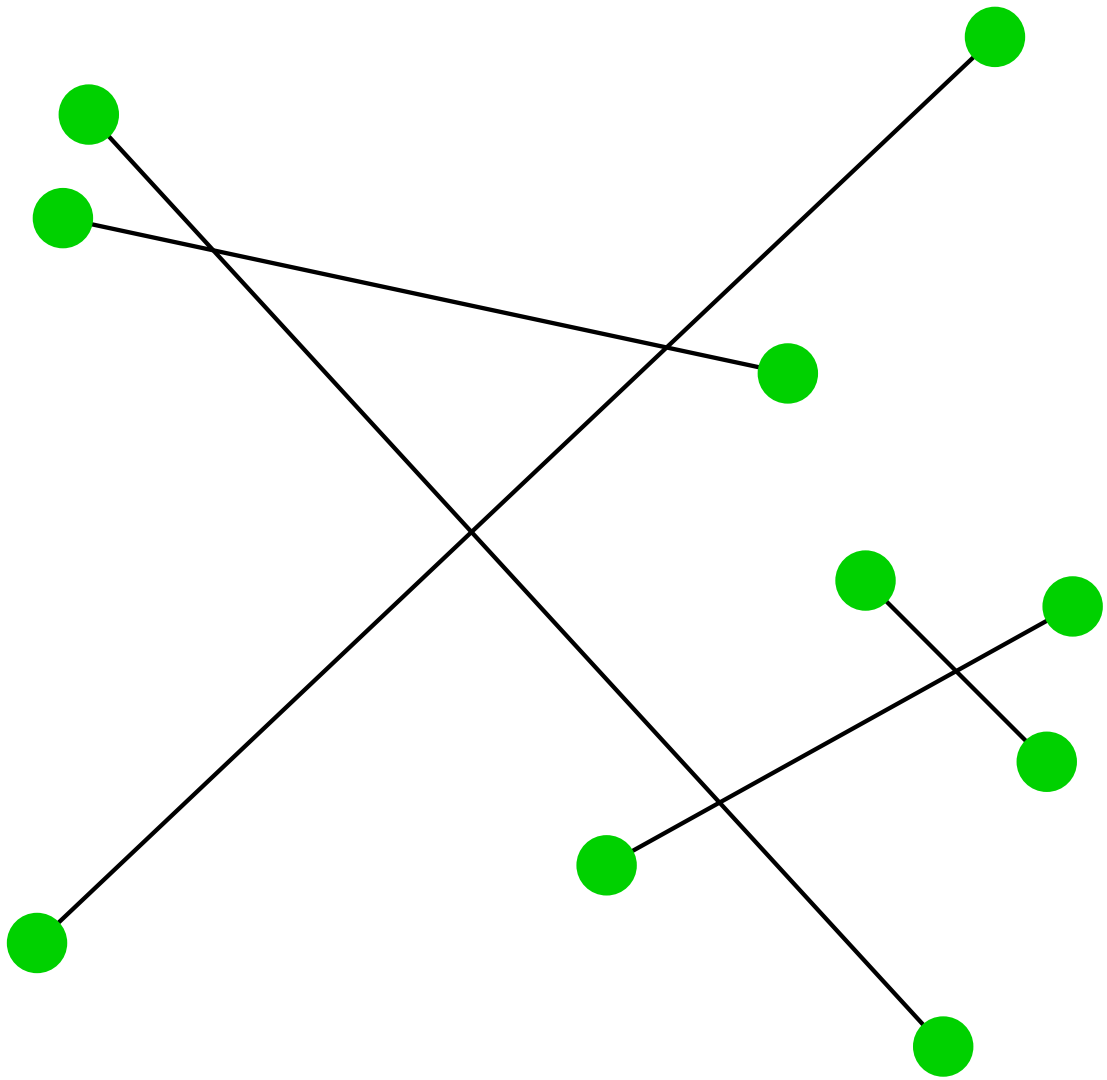
next:= succ(a) // $\mathrm{O}(1)$

findNewEvent(prev, b)

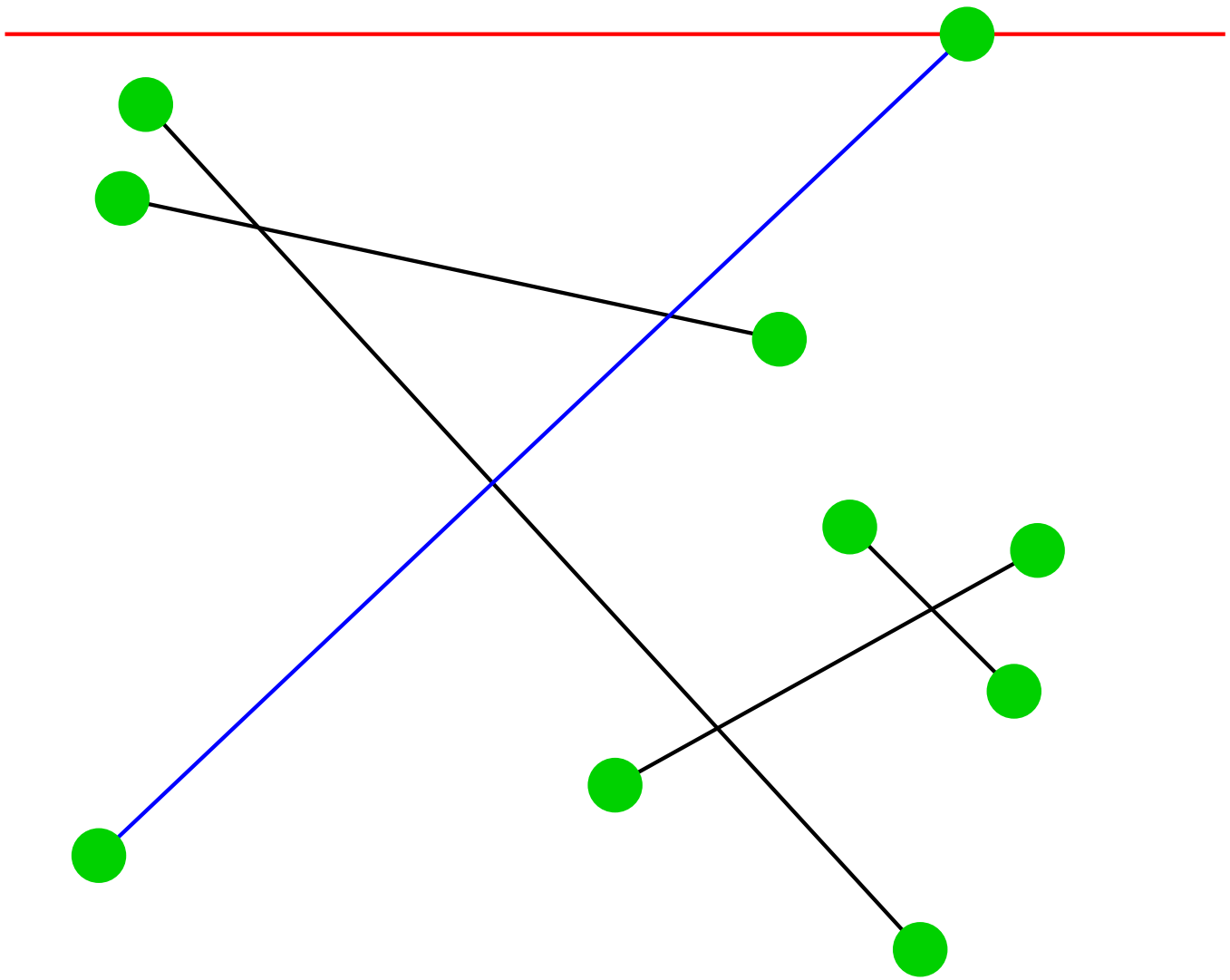
findNewEvent(a, next)



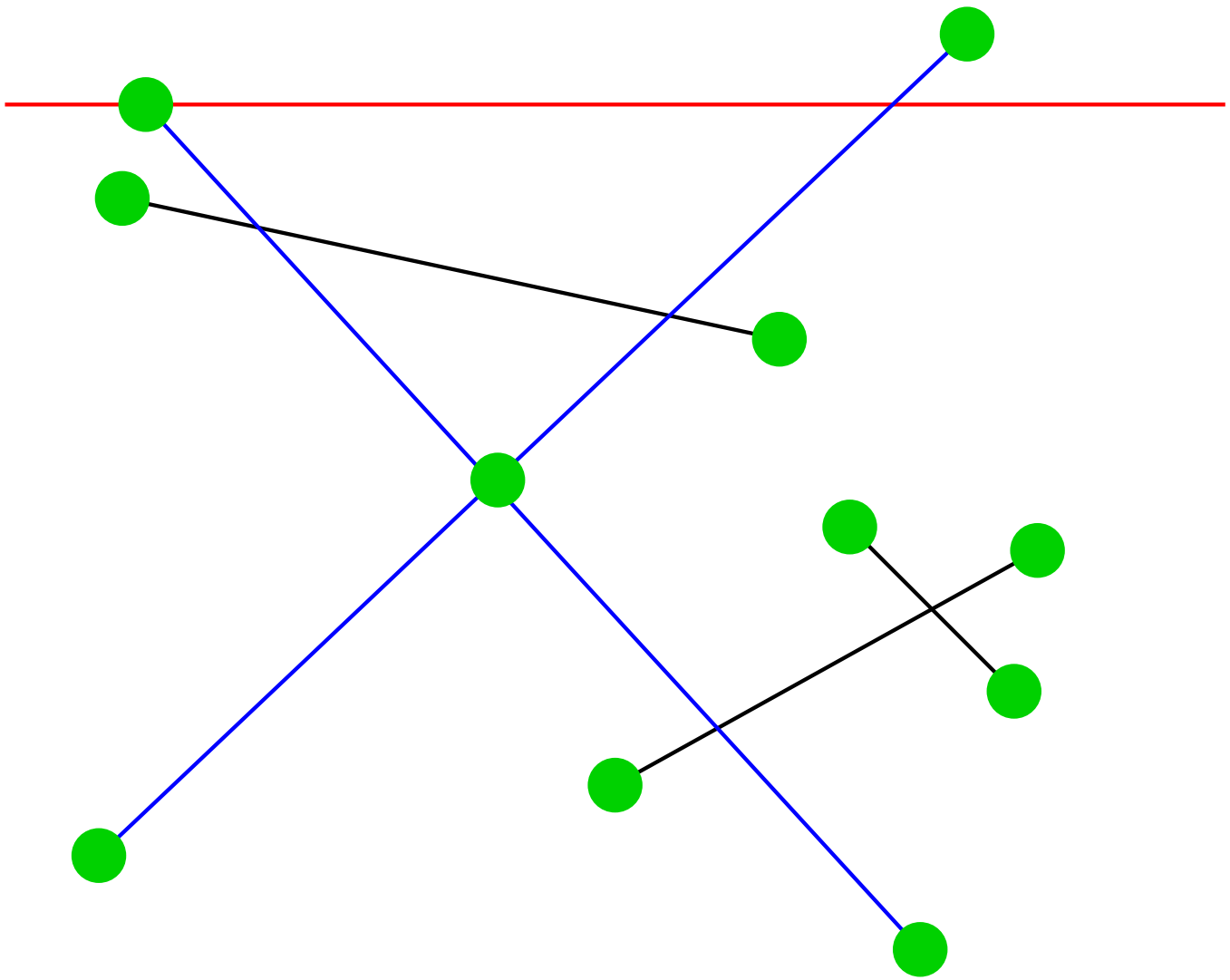
General Case – Example



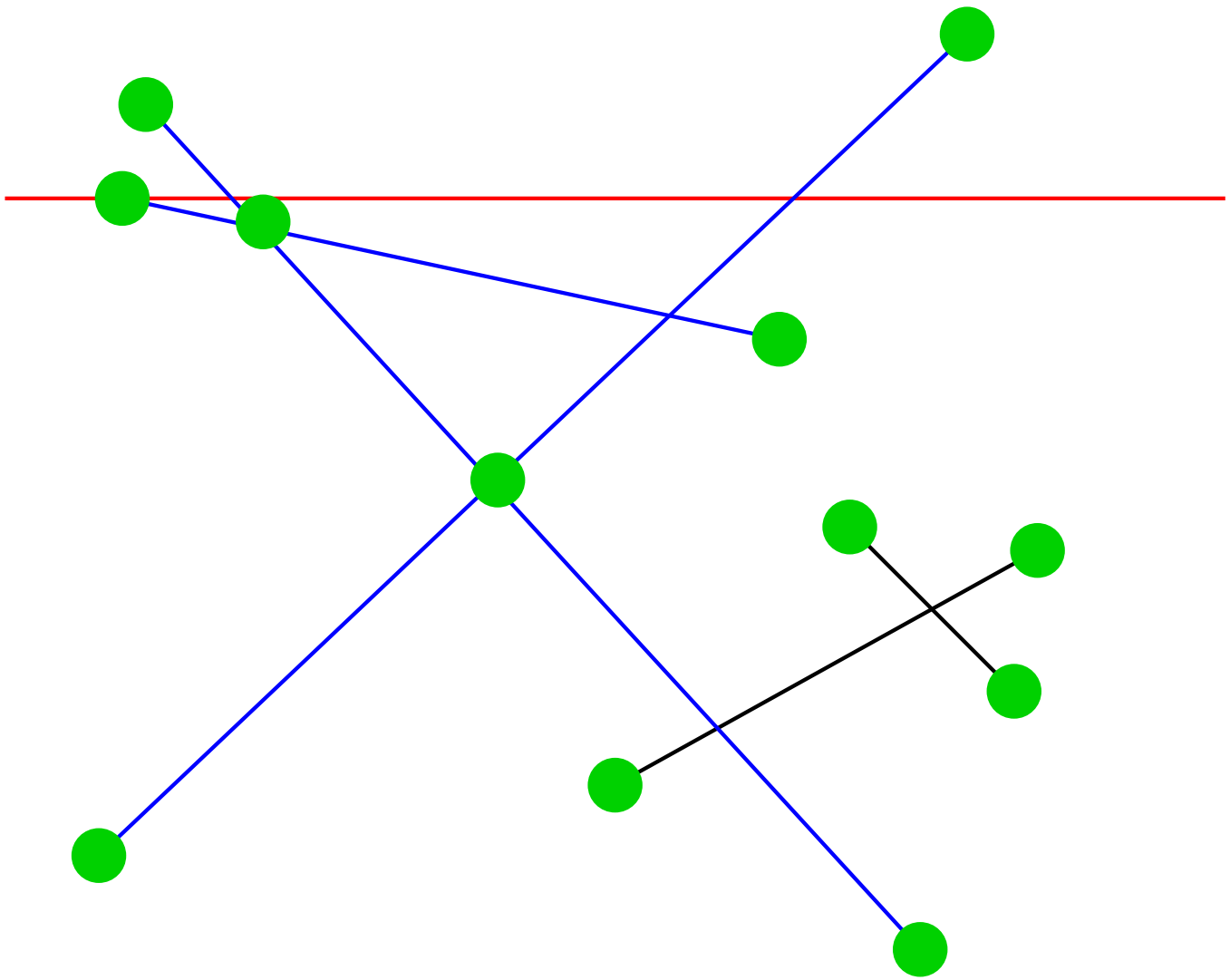
General Case – Example



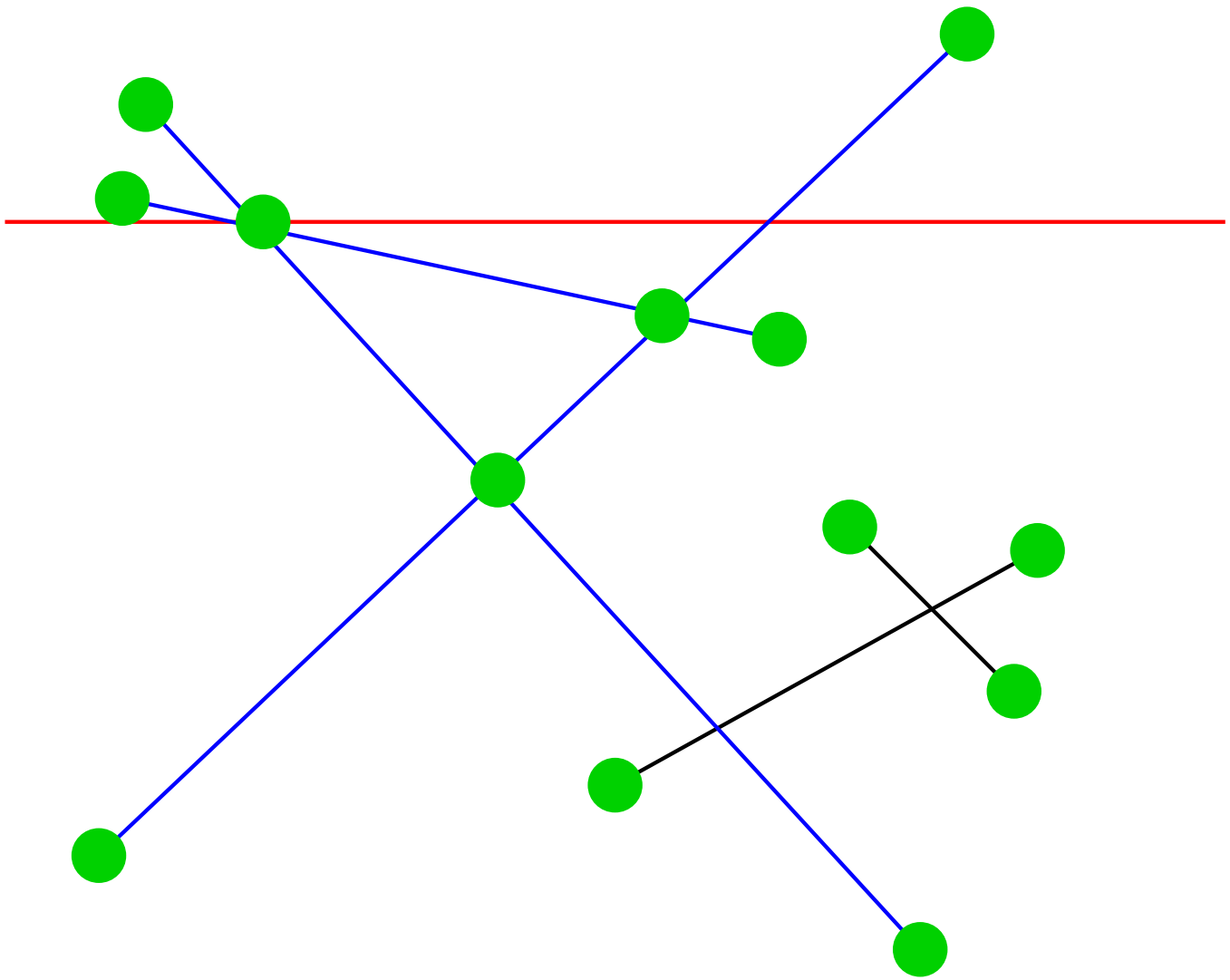
General Case – Example



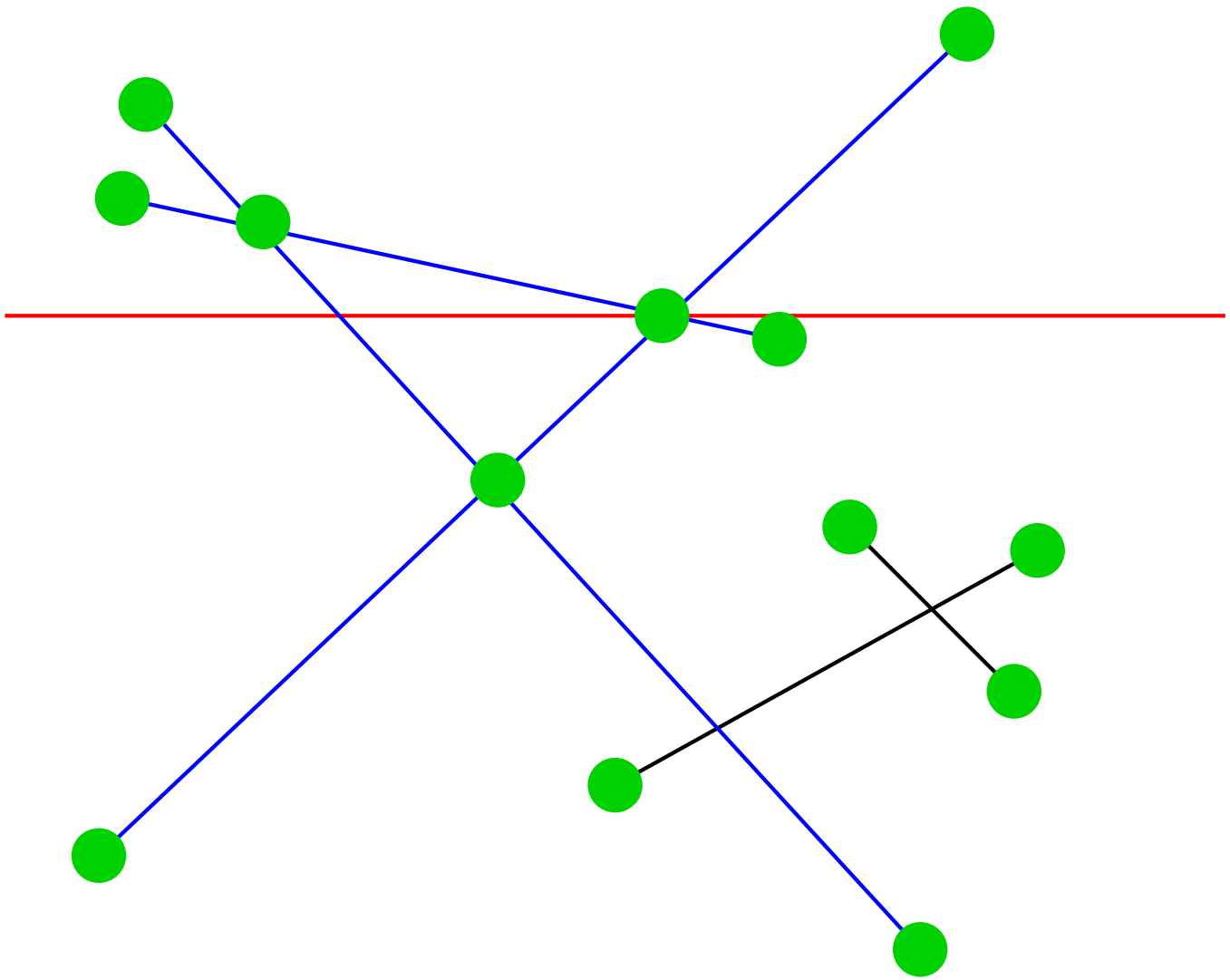
General Case – Example



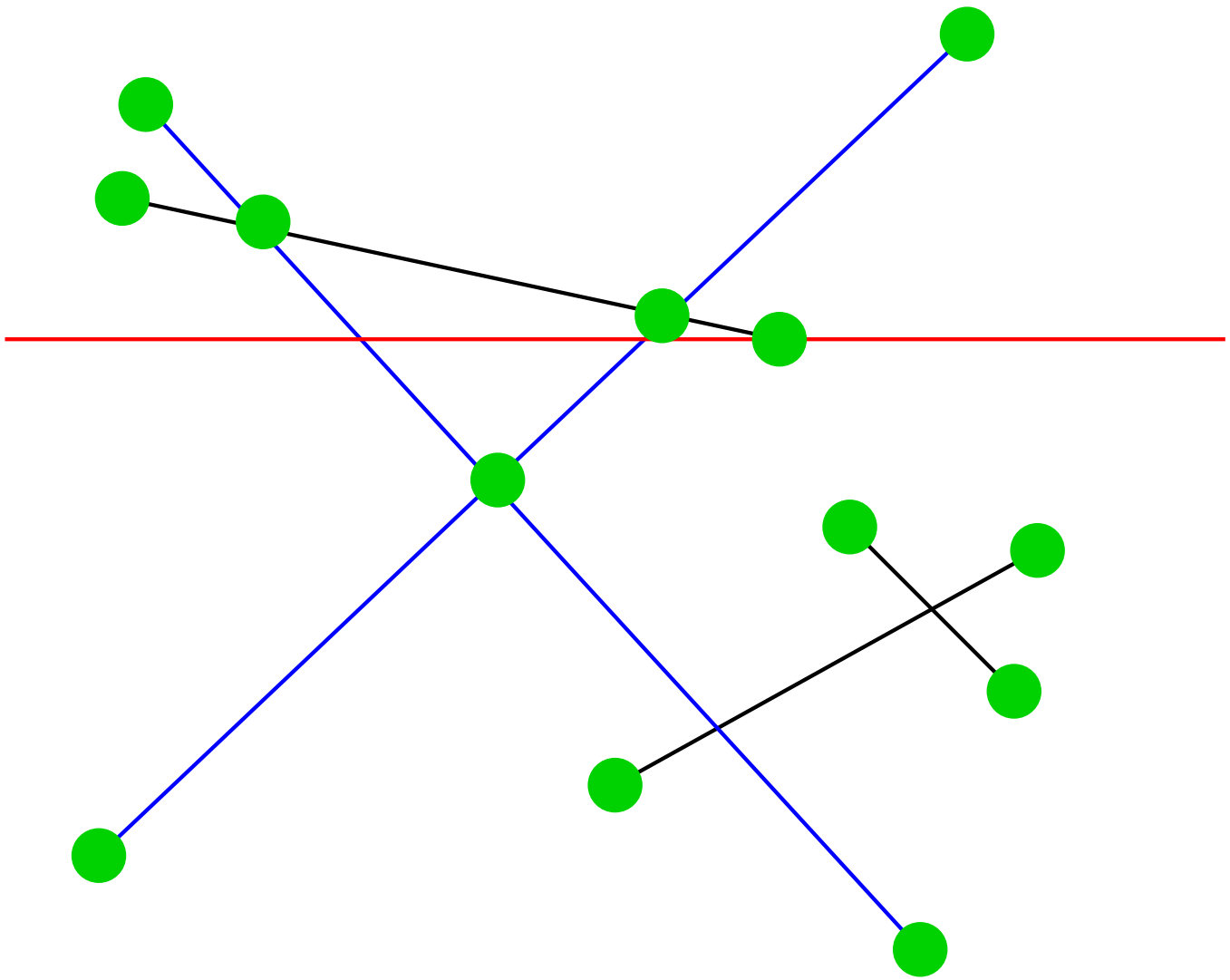
General Case – Example



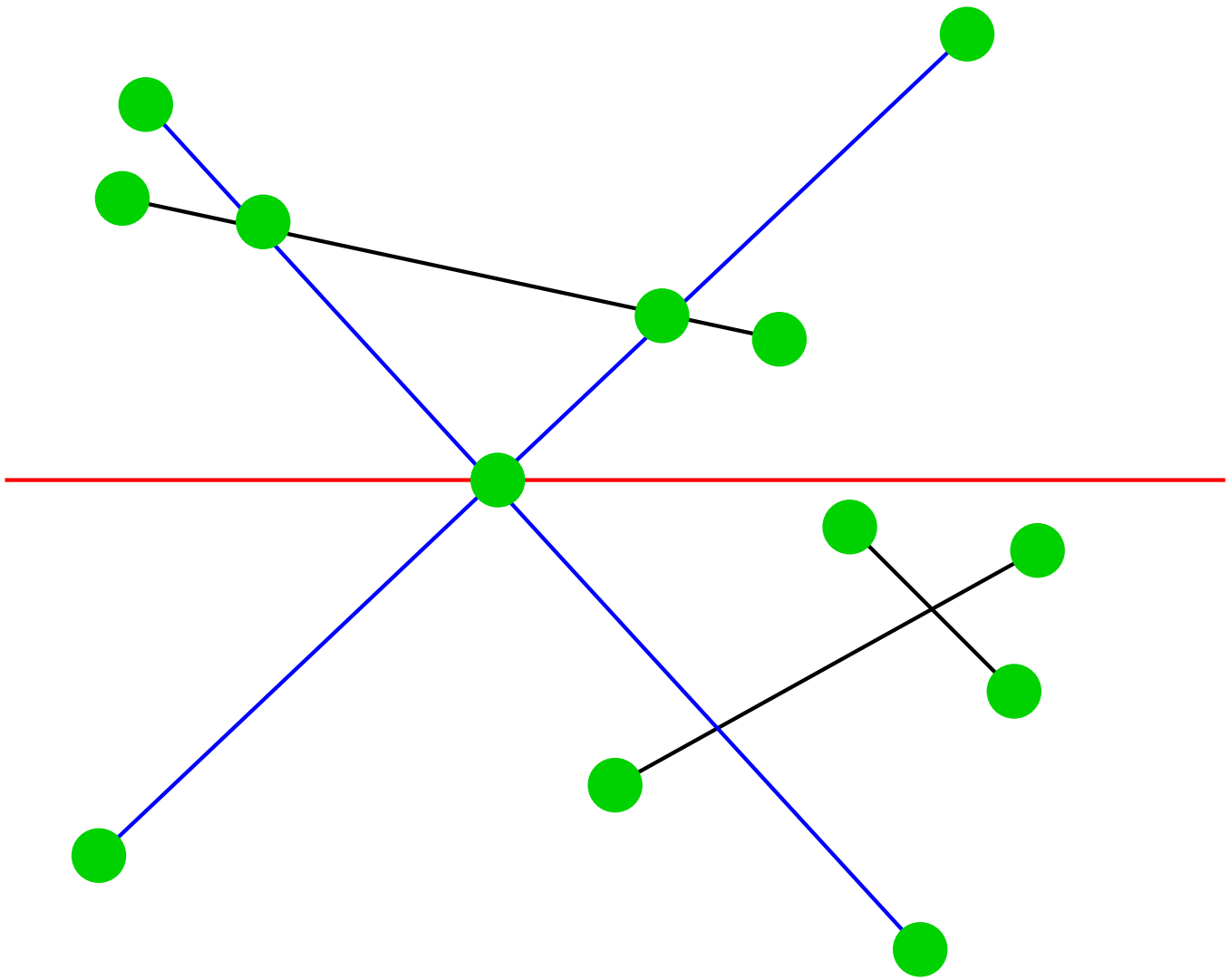
General Case – Example



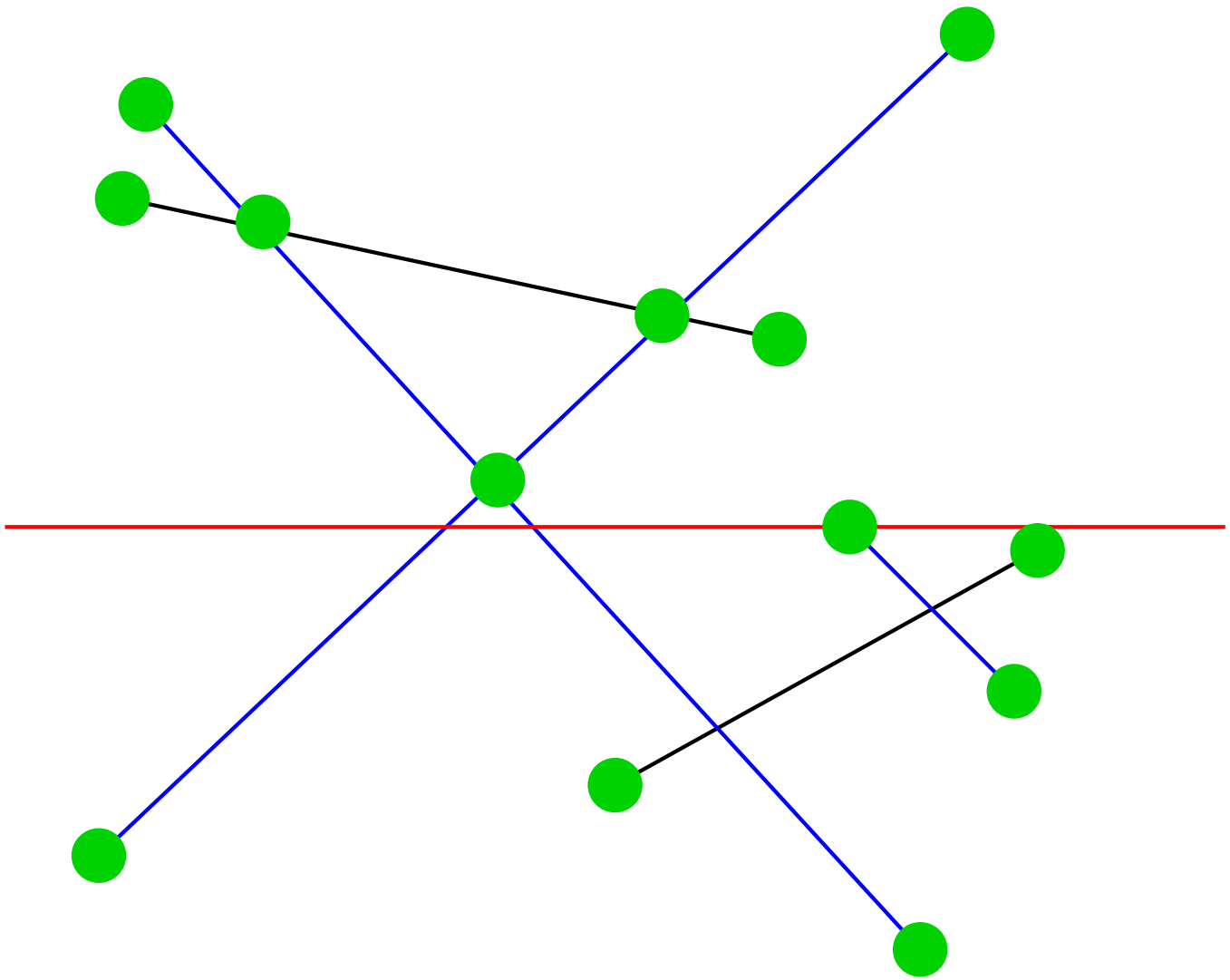
General Case – Example



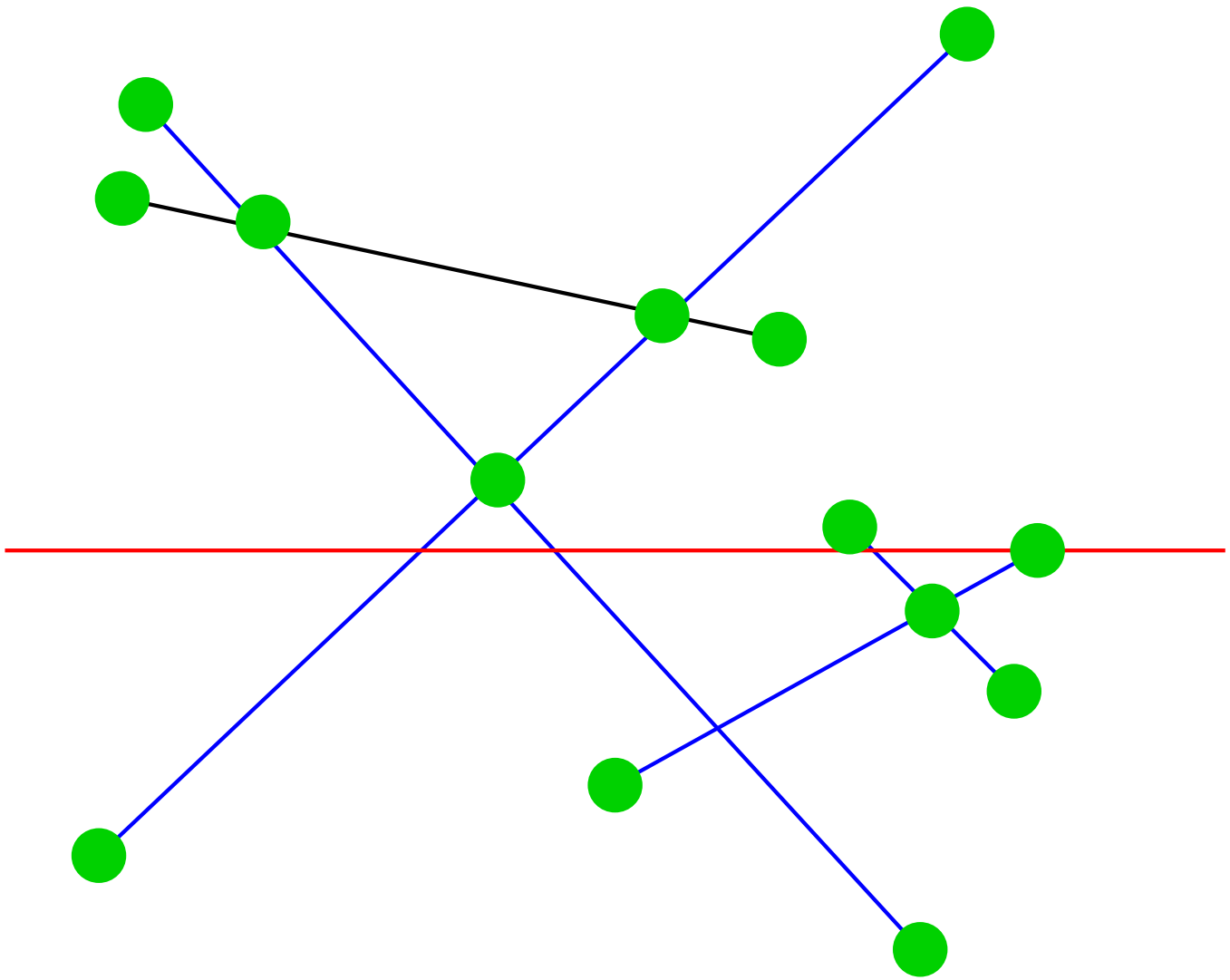
General Case – Example



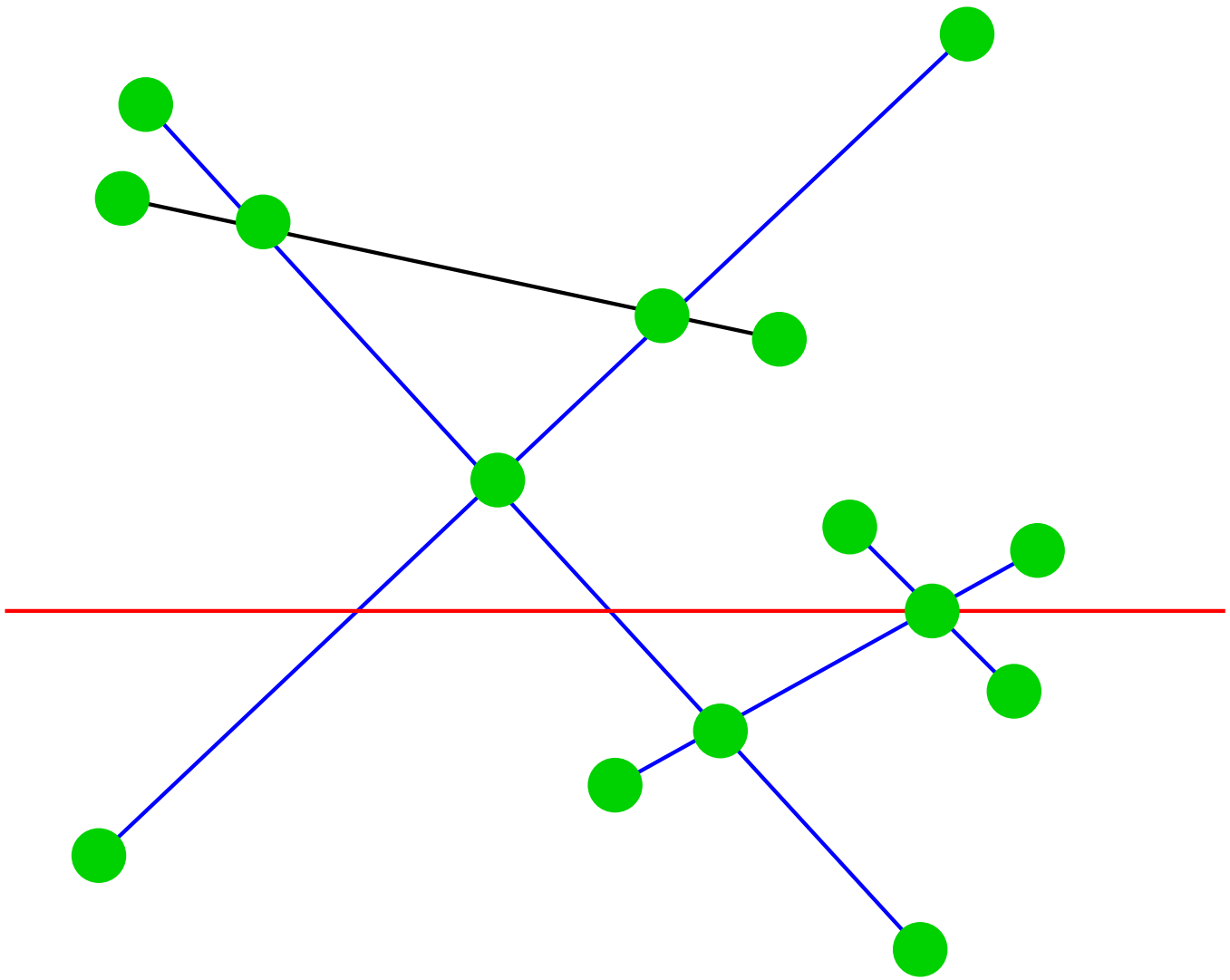
General Case – Example



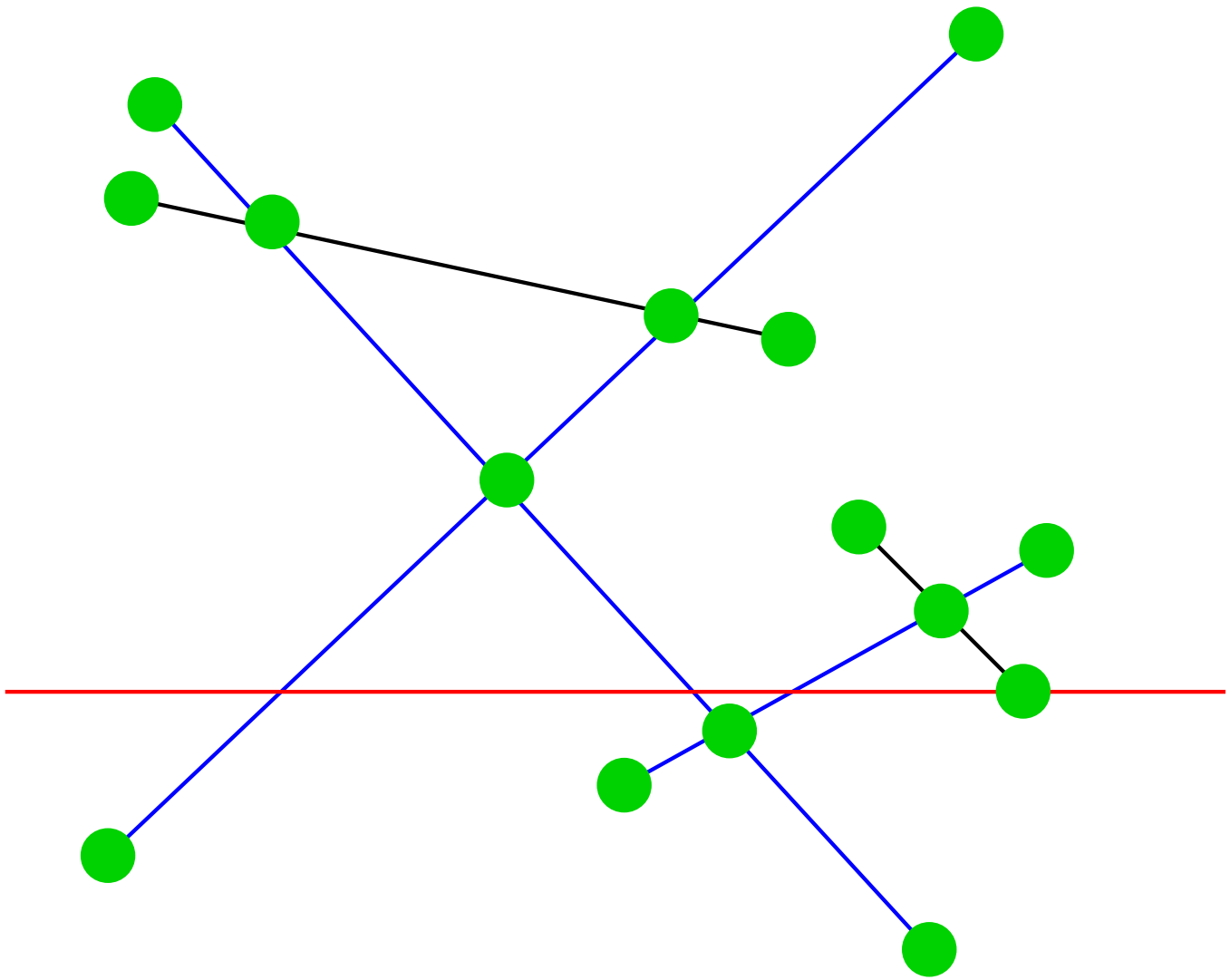
General Case – Example



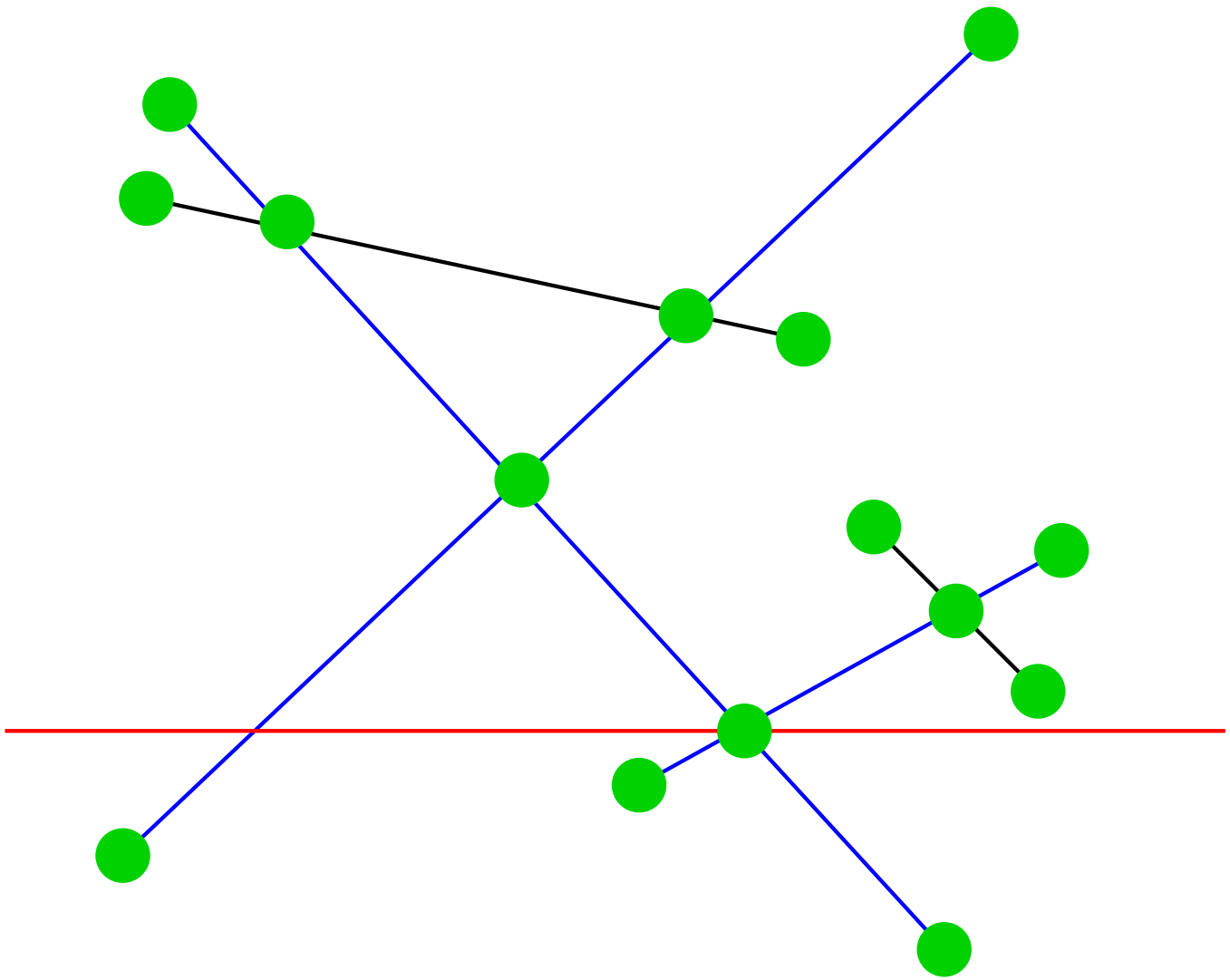
General Case – Example



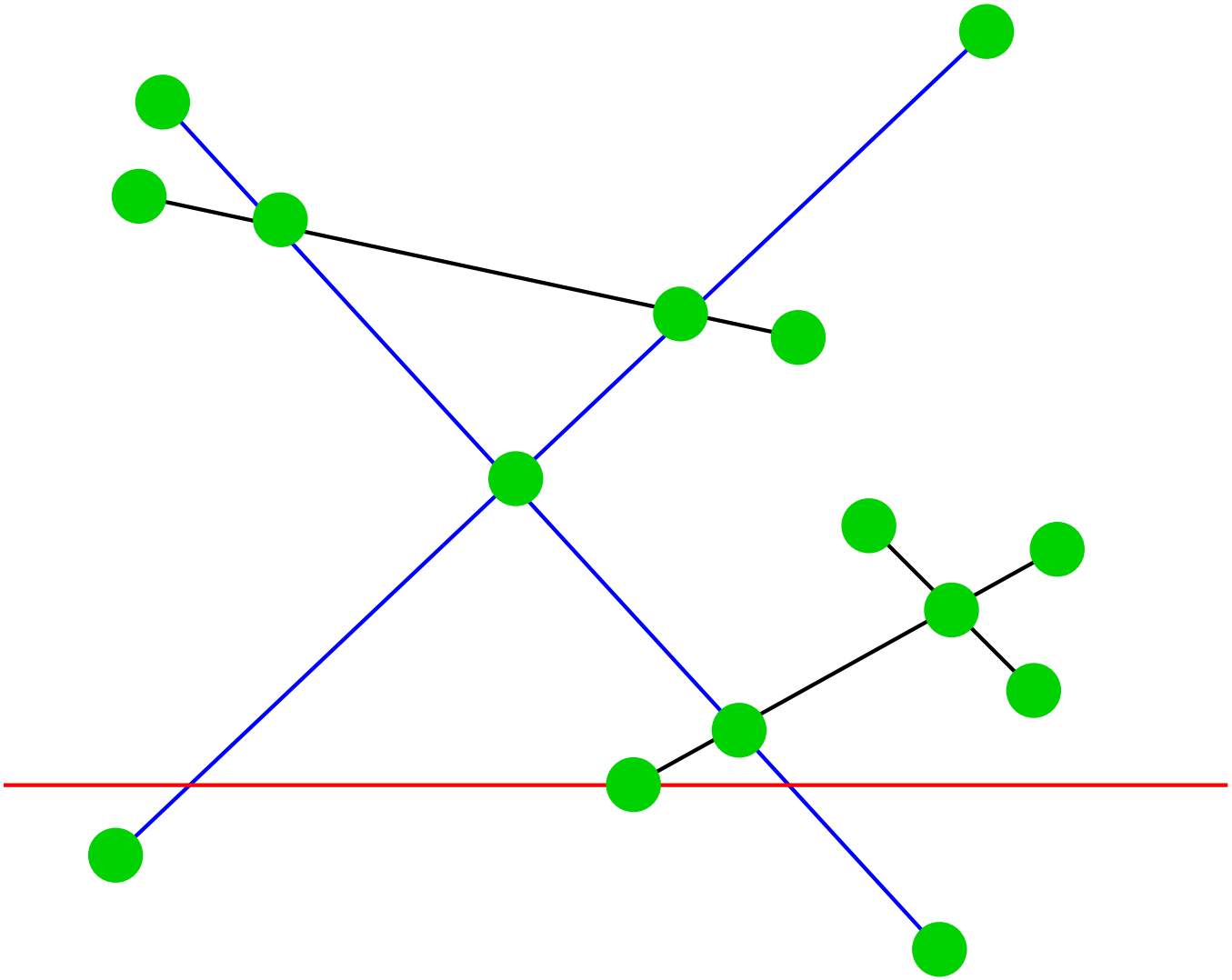
General Case – Example



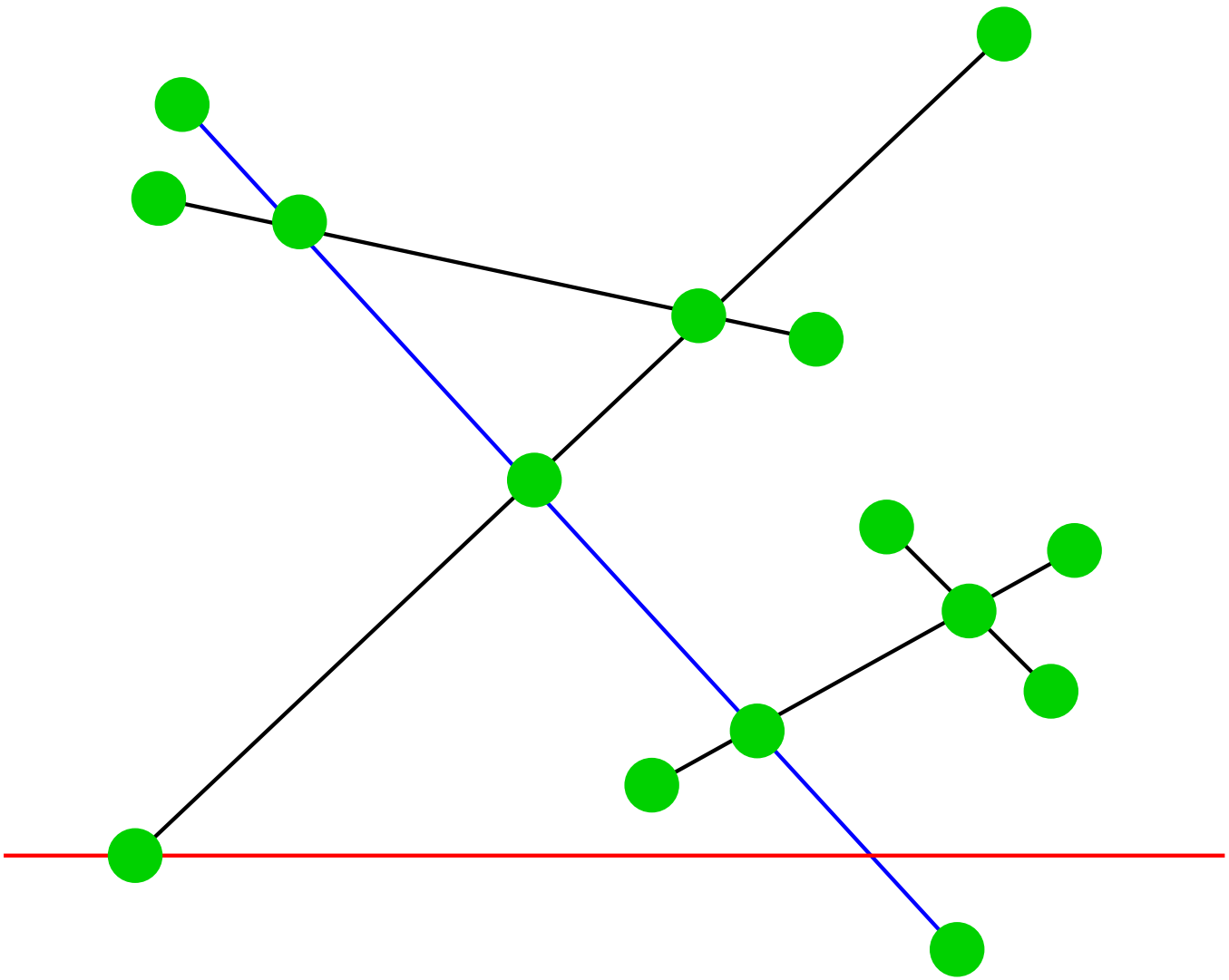
General Case – Example



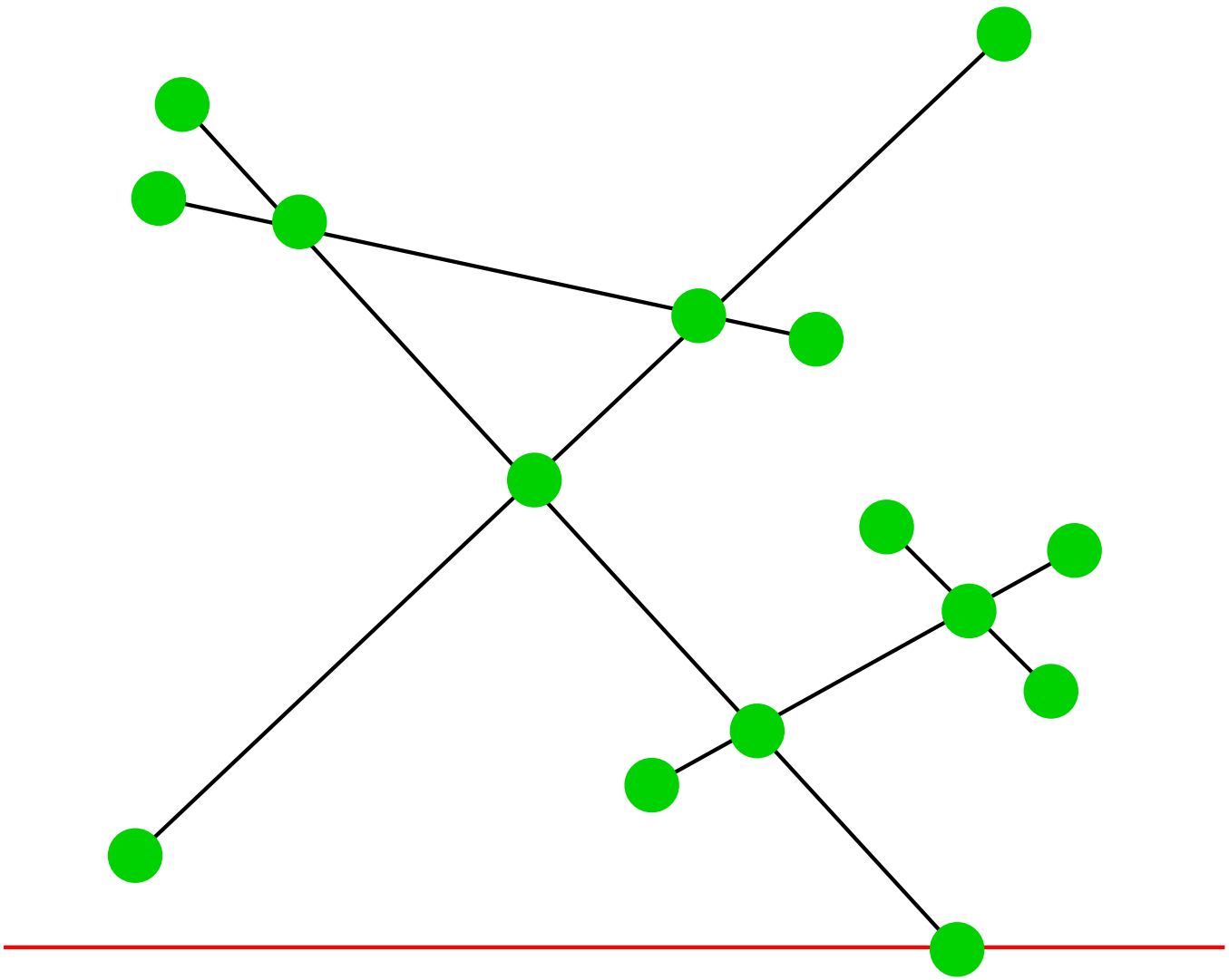
General Case – Example



General Case – Example



General Case – Example



"Almost" General Case – Analysis

In total: $\mathrm{O}((n+k)\log n)$

General Case – now really "almost"

Remaining assumption: no overlaps

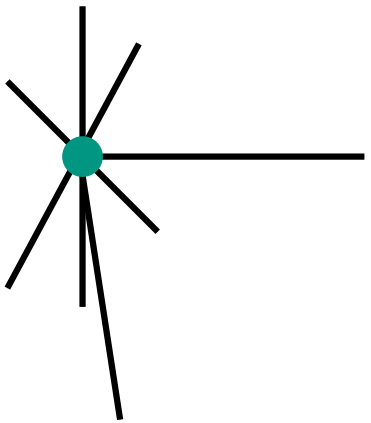
Order for Q : $(x,y) \prec (x',y') \Leftrightarrow y > y' \vee y = y' \wedge x < x'$

(skewed lexicographic order)

Interpretation: infinitesimal ascending sweep line

Q stores multiple segments intersecting at (x,y)

as a single event



handleEvent$(p = (x, y))$

$U :=$ segments starting at p // from Q

$C :=$ segments with p in their interior // from T

$L :=$ segments finishing at p // from Q

if $|U| + |C| + |L| \geq 2$ **then** report intersection @ p

T.remove$(L \cup C)$

T.insert$(C \cup U)$ such that order just below p is correct

if $U \cup C = \emptyset$ **then**

  findNewEvent$(T$.findPred$(p), T$.findSucc$(p), p)$

else

  findNewEvent$(T$.findPred$(p), T$.findLeftmost$(p), p)$

  findNewEvent$(T$.findRightmost$(p), T$.findSucc$(p), p)$

findNewEvent(s,t,p)

if s and t intersect at a point $p' \succ p$ **then**

if $p' \notin Q$ **then** Q.insert(p')

Detect Overlaps

For every line segment s calculate corresponding line $g(s)$

Sort S according to $g(s)$

1D overlap problem for every occurring line.

Space Requirement

Currently: $\Theta(n+k)$

Reduction to $\mathrm{O}(n)$:

Delete intersections of non-neighboring segments from T.

They will be re-inserted when they become neighbors.

More on ILS

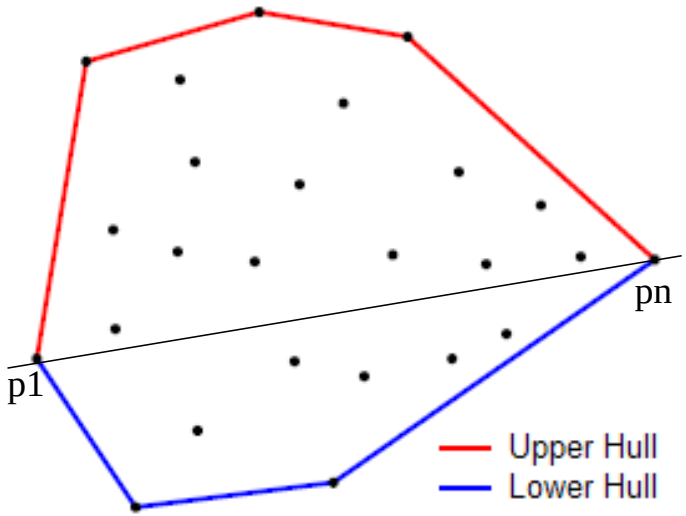
- [Bentley Ottmann 1979] Time $\mathrm{O}((n+k)\log n)$
- [Chazelle Edelsbrunner 1988] Time $\mathrm{O}(n\log n+k)$
- [Pach Sharir 1991] Time $\mathrm{O}((n+k)\log n)$, space $\mathrm{O}(n)$
- [Mulmuley 1988] Expected time $\mathrm{O}(n\log n+k)$, space $\mathrm{O}(n)$
- [Balaban 1995] Time $\mathrm{O}(n\log n+k)$, space $\mathrm{O}(n)$

12.2 2D Convex Hull

Given: Set $P = \{p_1, \ldots, p_n\}$ of points in $\mathbb{R}^2$

Find: Convex polygon K with corners in P and $P \subseteq K$.

We present a simple algorithm requiring time $\mathrm{O}(\mathrm{sort}(n))$.



Convex Hull

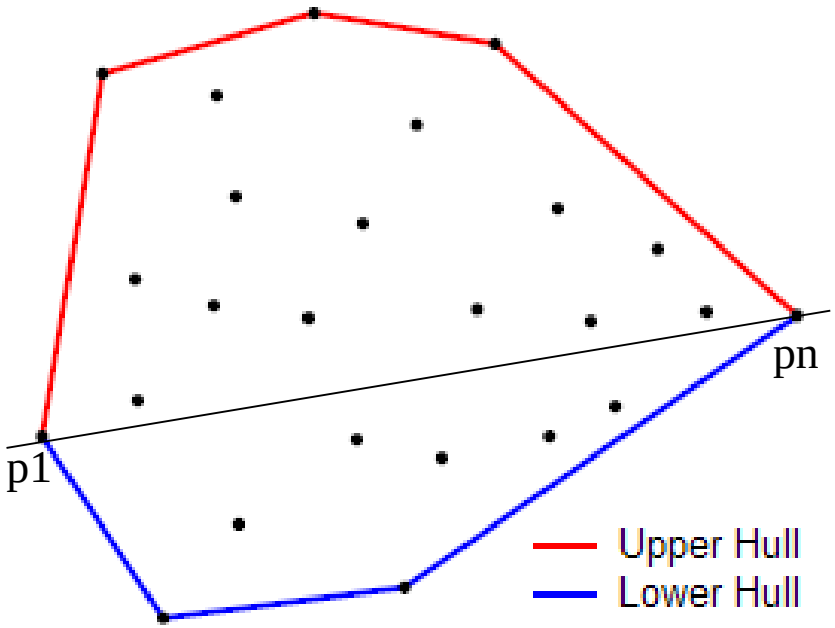
sort P lexicographically by (x, y), i.e., from now on

$p_1 < p_2 < \cdots < p_n$

Wlog:

We only compute upper hull of points above $\overline{p_1 p_n}$

(computing the lower cull works analogously)



Graham's Scan [Graham 1972, Andrew 1979]

Function upperHull$(p_1, \ldots, p_n)$

$L = \langle p_n, p_1, p_2 \rangle$: Stack **of** Point

invariant L is the upper hull of $\langle p_n, p_1, \ldots, p_i \rangle$

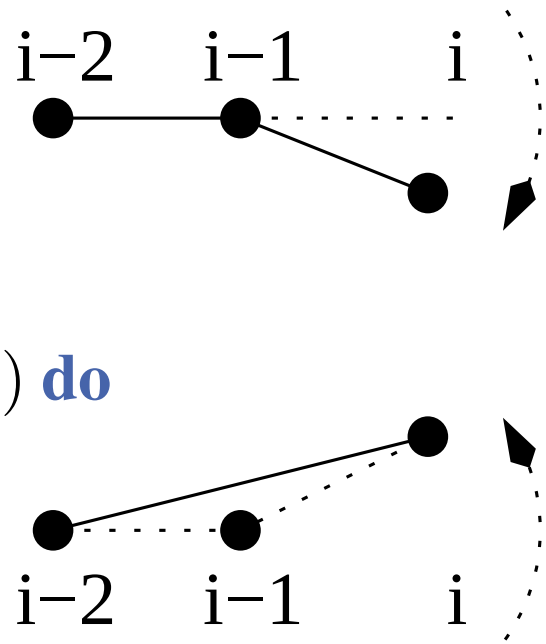
for $i := 3$ **to** n **do**

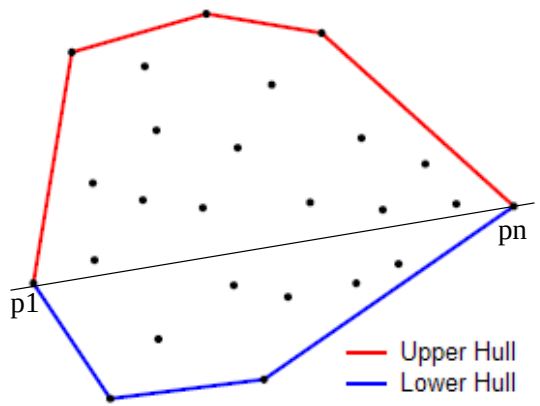
while $\neg$rightTurn$(L$.secondButlast$, L$.last$, p_i)$ **do**

L.pop

$L := L \circ \langle p_i \rangle$

return L





Graham's Scan – Example

Function upperHull$(p_1, \ldots, p_n)$

$L = \langle p_n, p_1, p_2 \rangle$: Stack **of** Point

invariant L is the upper hull of $\langle p_n, p_1, \ldots, p_i \rangle$

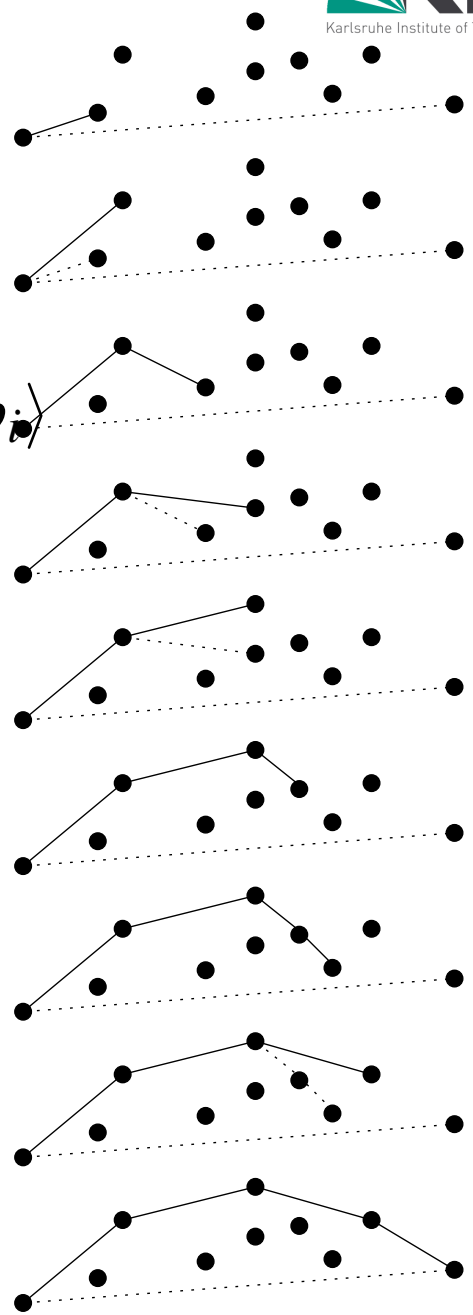
for $i := 3$ **to** n **do**

while $\neg$rightTurn$(L$.secondButlast, L.last, $p_i)$ **do**

L.pop

$L := L \circ \langle p_i \rangle$

return L



Graham's Scan – Analysis

Function upperHull$(p_1, \ldots, p_n)$

$L = \langle p_n, p_1, p_2 \rangle$: Stack **of** Point

invariant L is the upper hull of $\langle p_n, p_1, \ldots, p_i \rangle$

for $i := 3$ **to** n **do**

while $\neg$rightTurn$(L$.secondButlast$, L$.last$, p_i)$ **do**

L.pop

$L := L \circ \langle p_i \rangle$

return L

Sorting $+\mathrm{O}(n)$

How many iterations of while loop in total?

3D Convex Hull

[Preparata Hong 1977] Time $\mathrm{O}(n \log n)$ algorithm

Convex Hull, $d \geq 4$

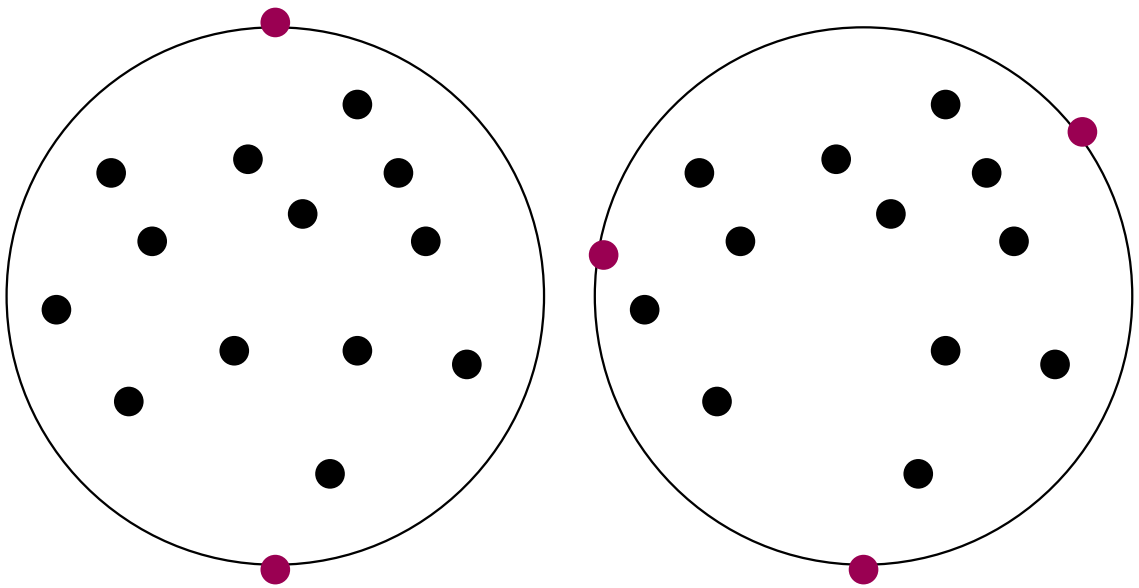
Output complexity $\mathrm{O}\left(n^{\lfloor d/2 \rfloor}\right)$

12.3 Smallest Enclosing Ball

Given: Set $P = \{p_1, \ldots, p_n\}$ of points in $\mathbb{R}^d$

Find: Ball B with minimal radius, such that $P \subseteq B$.

We present a simple algorithm with expected time $\mathrm{O}(n)$. [Welzl 1991].



Function smallestEnclosingBallWithPoints(P, Q)

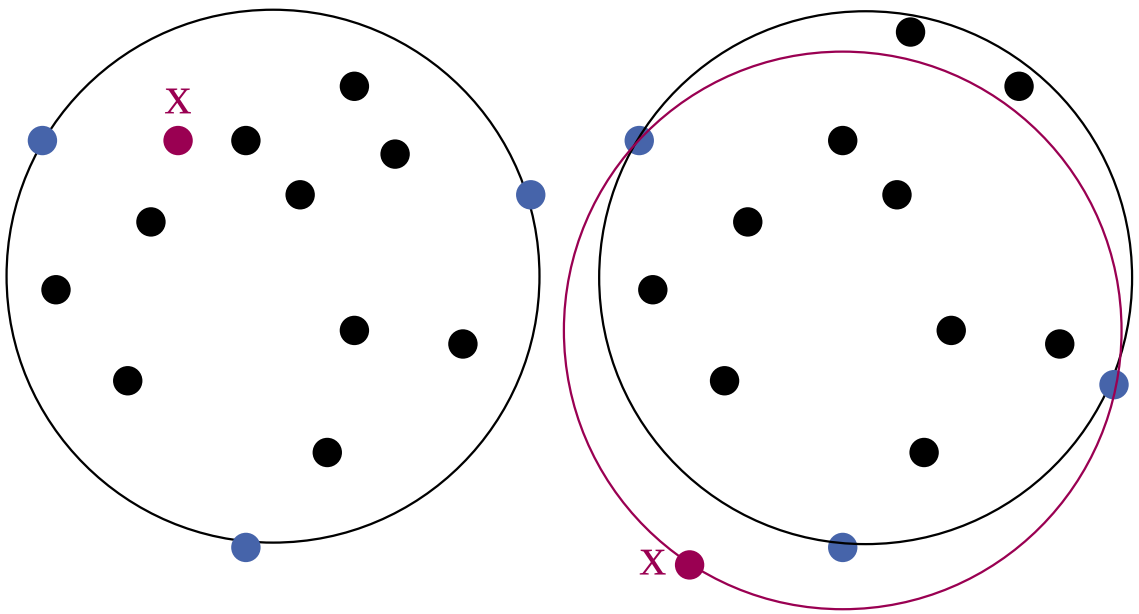
if $|P| = 0 \vee |Q| = d + 1$ **then return** ball(Q)

pick random $x \in P$

B:= smallestEnclosingBallWithPoints$(P \setminus \{x\}, Q)$

if $x \in B$ **then return** B

return smallestEnclosingBallWithPoints$(P \setminus \{x\}, Q \cup \{x\})$



Smallest Enclosing Ball – Correctness

Function smallestEnclosingBallWithPoints(P, Q)

  if $|P| = 1 \vee |Q| = d + 1$ **then return** ball(Q)

  pick random $x \in P$

  B:= smallestEnclosingBallWithPoints$(P \setminus \{x\}, Q)$

  if $x \in B$ **then return** B

  return smallestEnclosingBallWithPoints$(P \setminus \{x\}, Q \cup \{x\})$

Show: $x \notin B \rightarrow x$ is on boundary of sEB(P)

Proof by contradiction:

x not on boundary of sEB(P)

$\longrightarrow$ sEB$(P) =$ sEB$(P \setminus \{x\}) = B$ Show.: sEBs is not unique!

Therefore $x \in B$

Lemma: sEB(P) is unique.

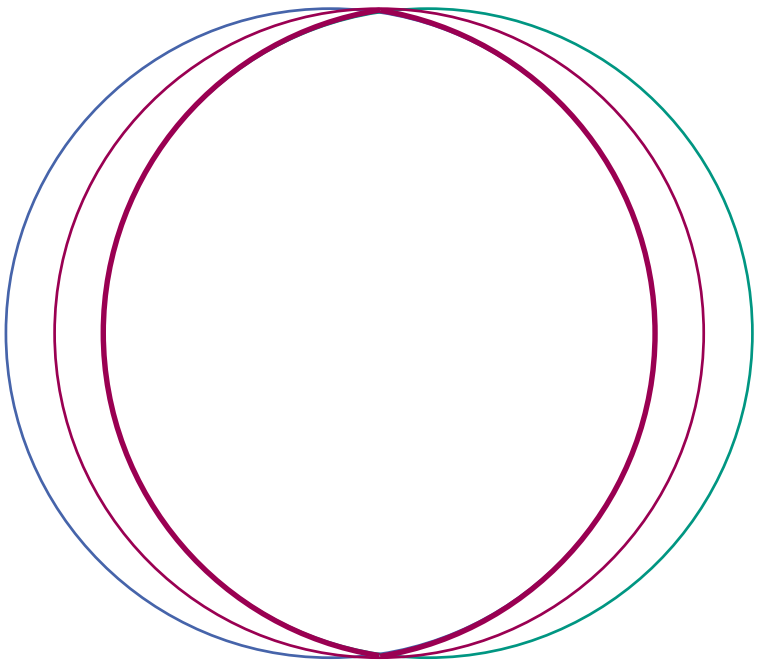
Proof: Assume $\exists$sEBs $B_1 \neq B_2$

$\longrightarrow P \subseteq B_1 \wedge P \subseteq B_2$

$\longrightarrow P \subseteq B_1 \cap B_2 \subseteq$ sEB$(B_1 \cap B_2) =: B$

But then radius$(B) <$ radius(B_1)

Contradiction to assumption that B_1 is sEB. $\square$



Smallest Enclosing Ball – Analysis

We count the expected number of tests $x \in B$, $T(p,q)$.

$$T(p,d+1) = T(1,p) = 0 \qquad \text{base of recurrence}$$

$$\begin{aligned} T(p,q) &\leq 1 + T(p-1,q) + \mathbb{P}[x \notin B]\, T(p,q+1) \\ &\leq 1 + T(p-1,q) + \frac{d+1-q}{p} T(p,q+1) \end{aligned}$$

Smallest Enclosing Ball – Analysis, $d = 2$

$$T(p,d+1) = T(1,p) = 0$$

$$T(p,q) \leq 1 + T(p-1,q) + \frac{d+1-q}{p} T(p,q+1)$$

$$T(p,2) \leq 1 + T(p-1,2) + \frac{1}{p} T(p,3) \leq 1 + T(p-1,2) \leq p$$

$$T(p,1) \leq 1 + T(p-1,1) + \frac{2}{p} T(p,2)$$

$$\leq 1 + T(p-1,1) + \frac{2}{p} p = 3 + T(p-1,1) \leq 3p$$

$$T(p,0) \leq 1 + T(p-1,0) + \frac{3}{p} T(p,1)$$

$$\leq 1 + T(p-1,0) + \frac{3}{p} 3p = 10 + T(p-1,0) \leq 10p$$

Smallest Enclosing Ball – Analysis

d	$T(p,0)$
1	$3n$
2	$10n$
3	$41n$
4	$206n$

In general $T(p,0) \geq d!n$

Similar Randomized Linear Time Algorithms

- More efficient variants for large constant d
- Linear programming with constant d [Seidel 1991]
- Smallest enclosing ellipsoid, annulus,...
- Support vector machines (machine learning)
- All LP-type problems [Sharir Welzl 1992])
  - $\mathrm{O}(1)$ objects define optimum
  - Insertion of object x

    $\longrightarrow$ solution remains the same or x part of new solution

12.4 2D Range Queries

Given: $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$

Queries: axis-aligned rectangles $Q = [x, x'] \times [y, y']$

find $P \cap Q$ (range reporting)

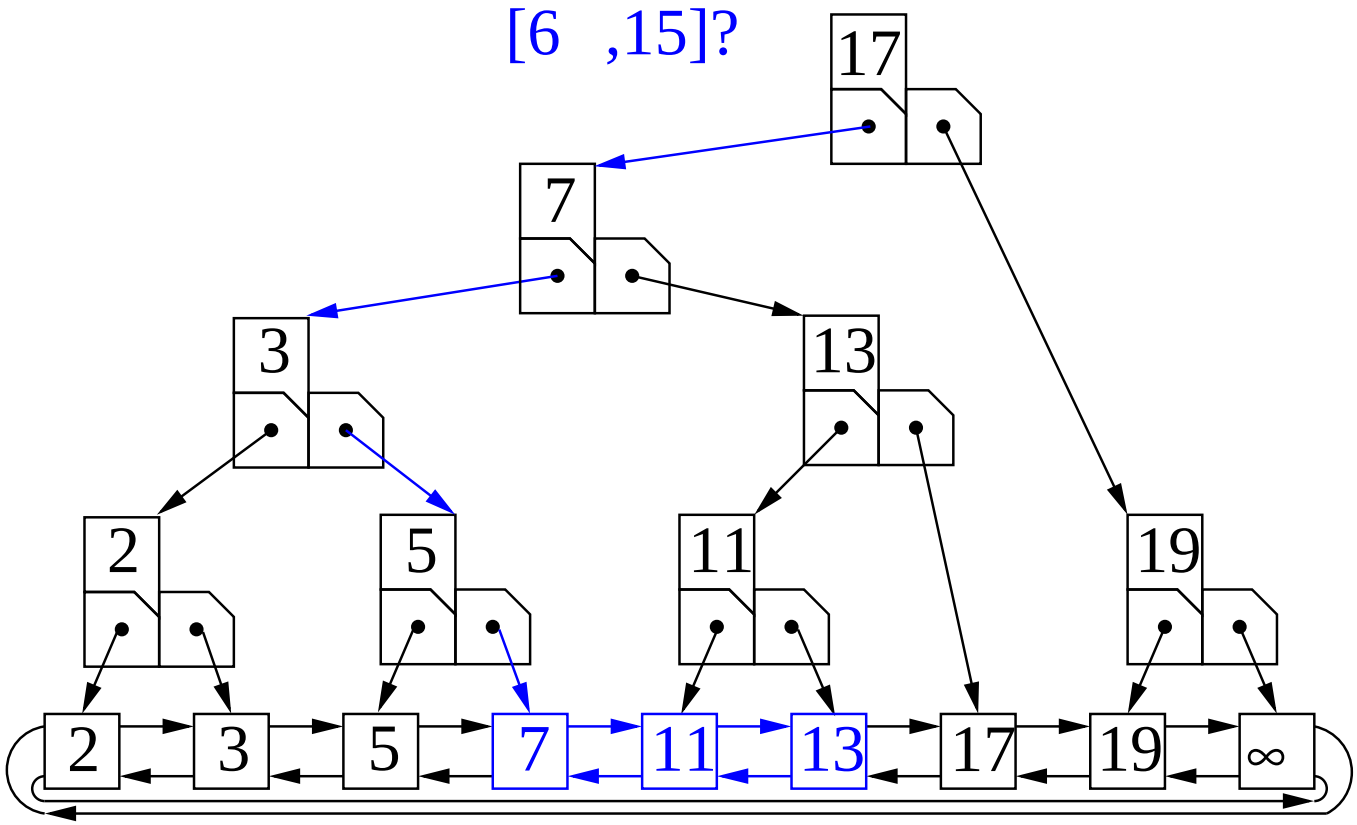
or $k = |P \cap Q|$ (range counting)

Preprocessing is permitted

- ☐ Preprocessing time? $\mathrm{O}(n \log n)$
- ☐ Space? $\mathrm{O}(n)$?
- ☐ Query time?
  - Counting: $\mathrm{O}(\log n)$
  - Reporting: $\mathrm{O}(k + \log n)$ or at least $\mathrm{O}(k \cdot \log n)$

1D Range Query

Search Tree



Count queries: save size of subtrees

Even dynamic!

Reduction to $1..n \times 1..n$

Simplifying assumption: coordinates are pairwise distinct.

Substitute coordinates $P = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ by their rank:

$x_i \rightarrow$ rank of x_i in $\{x_1, \ldots, x_n\}$

$y_i \rightarrow$ rank of y_i in $\{y_1, \ldots, y_n\}$

Reduction to $1..n \times 1..n$

Substitute coordinates $P = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ by their rank:

P_x:= sort$(\langle x_1, \ldots, x_n \rangle)$; P_y:= sort$(\langle y_1, \ldots, y_n \rangle)$

P:= $\{(\text{binarySearch}(x, P_x), \text{binarySearch}(y, P_y)) : (x, y) \in P\}$

Function rangeQuery$([x, x'] \times [y, y'])$

  x:= binarySearchSucc(x, P_x); x':= binarySearchPred(x', P_x)

  y:= binarySearchSucc(y, P_y); y':= binarySearchPred(y', P_y)

  R:= intRangeQuery$([x, x'] \times [y, y'])$

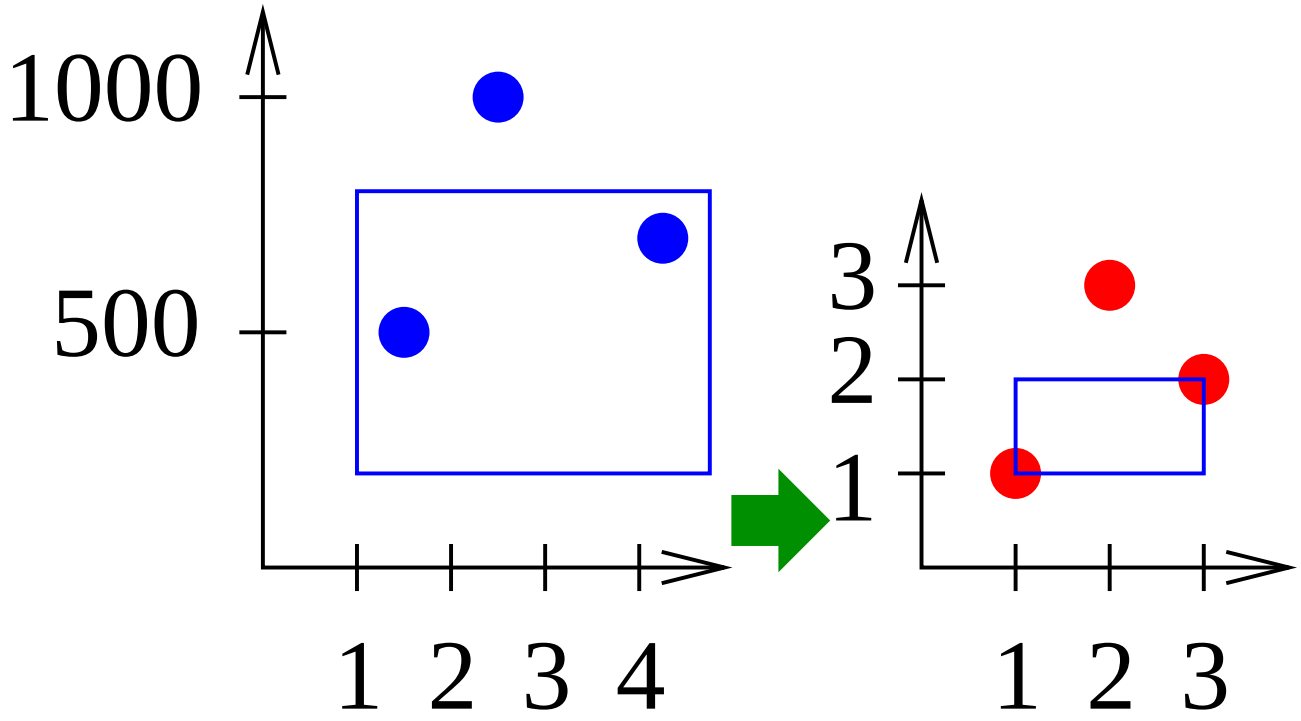
  return $\{(P_x[x], P_y[y]) : (x, y) \in R\}$

Construction time $\mathrm{O}(n \log n)$

Query time $\mathrm{O}(\log n) + T_{\text{intRangeQuery}}(n)$

Example

$\{(2.5,1000),(1.4,500),(4.2,700)\} \rightarrow \{(2,3),(1,1),(3,2)\}$



Wavelet Tree

[Chazelle 1988, Grossi/Gupta/Vitter 2003, Mäkinen/Navarro 2007]

Class WaveletTree$(X = \langle x_1, \ldots, x_n \rangle)$// represents $(x_1, 1), \ldots, (x_n, n)$

//Constructor:

if $n < n_0$ **then** store X directly

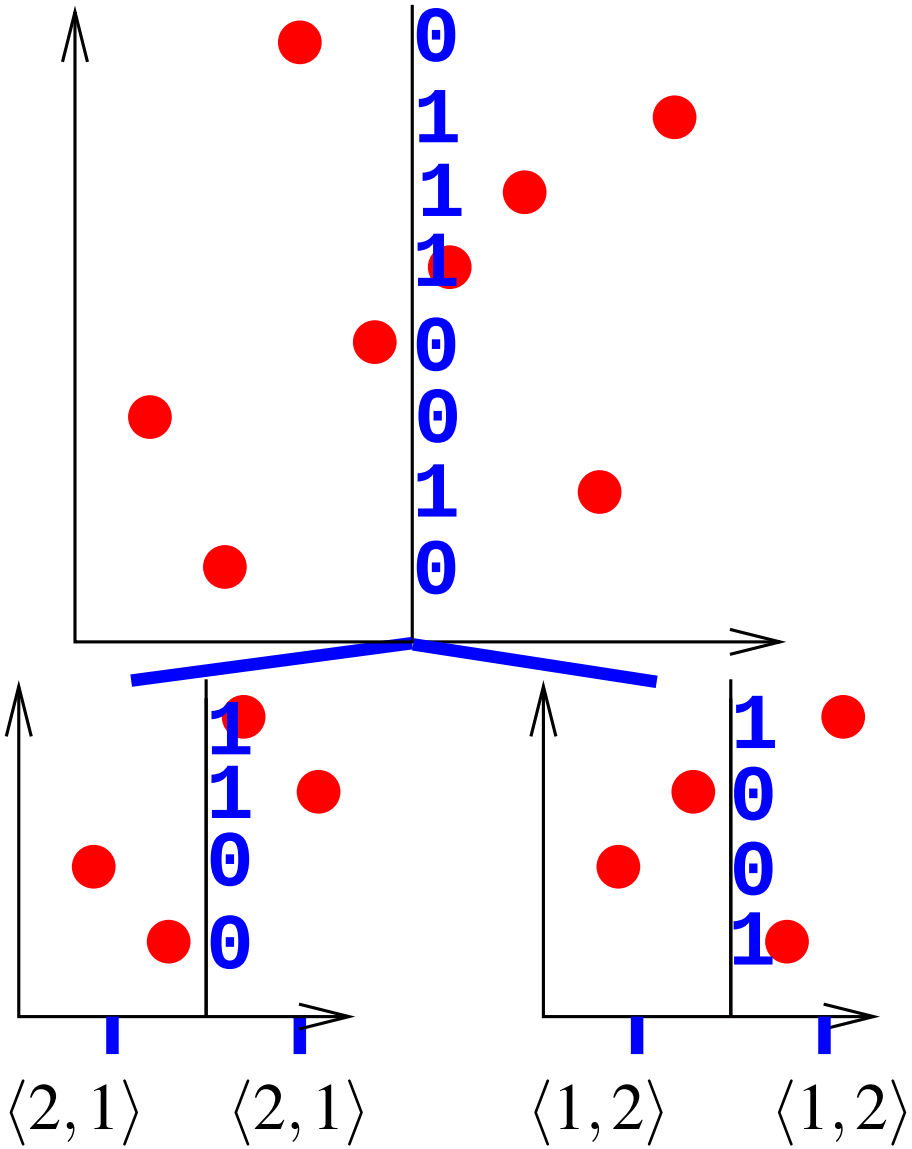
else

store bitvector b with $b[i] = 1$ iff $x_i > \lfloor n/2 \rfloor$

ℓ:= WaveletTree$(\langle x_i : x_i \leq \lfloor n/2 \rfloor \rangle)$

r:= WaveletTree$(\langle x_i - \lfloor n/2 \rfloor : x_i > \lfloor n/2 \rfloor \rangle)$

Example



Wavelet Tree Counting Query

Function intRangeCount$([x,x'] \times [y,y'])$

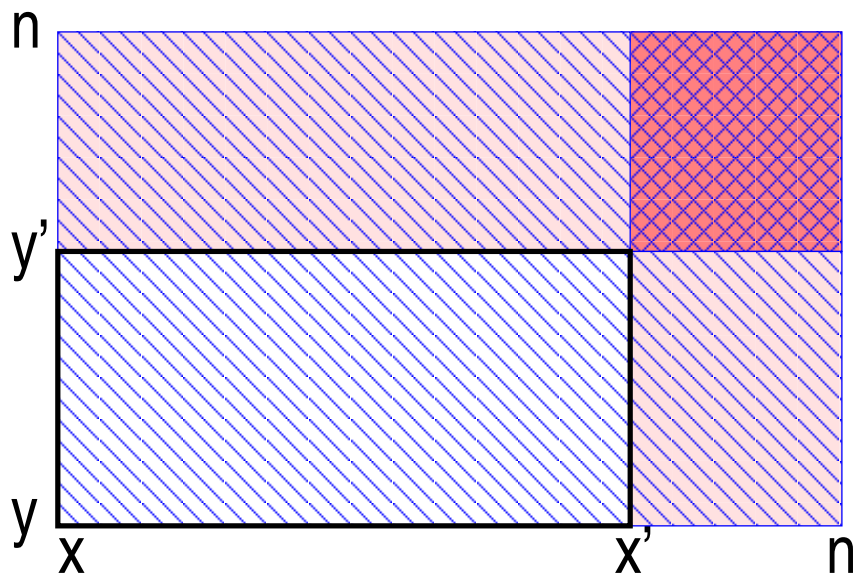
return

intDominanceCount$(x,y)-$

intDominanceCount$(x',y)-$

intDominanceCount$(x,y')+$

intDominanceCount(x',y')



Wavelet Tree Dominance Counting Query

Function intDominanceCount(x,y) // $|[x,n]\times[y,n]\cap P|$

  if $n \le n_0$ **then return** $|[x,n]\times[y,n]\cap P|$ // brute force

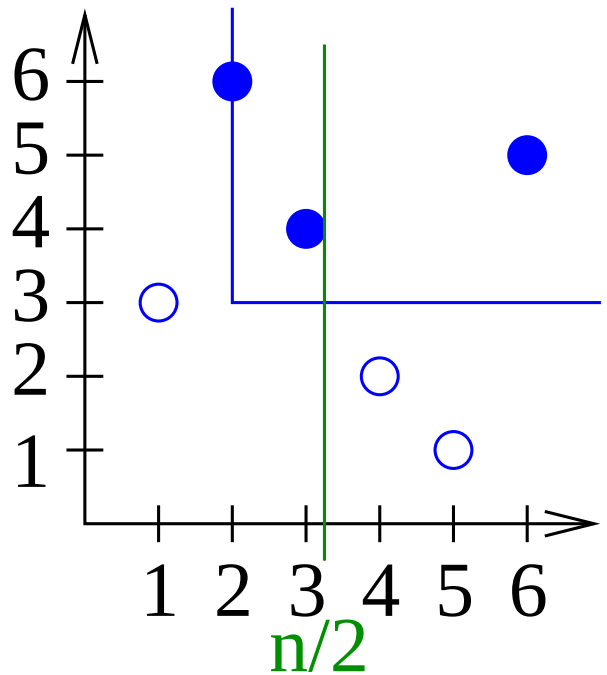
  y_r:= b.rank(y) // Number of els $\le y$ in right half

  if $x \le \lfloor n/2 \rfloor$ **then**

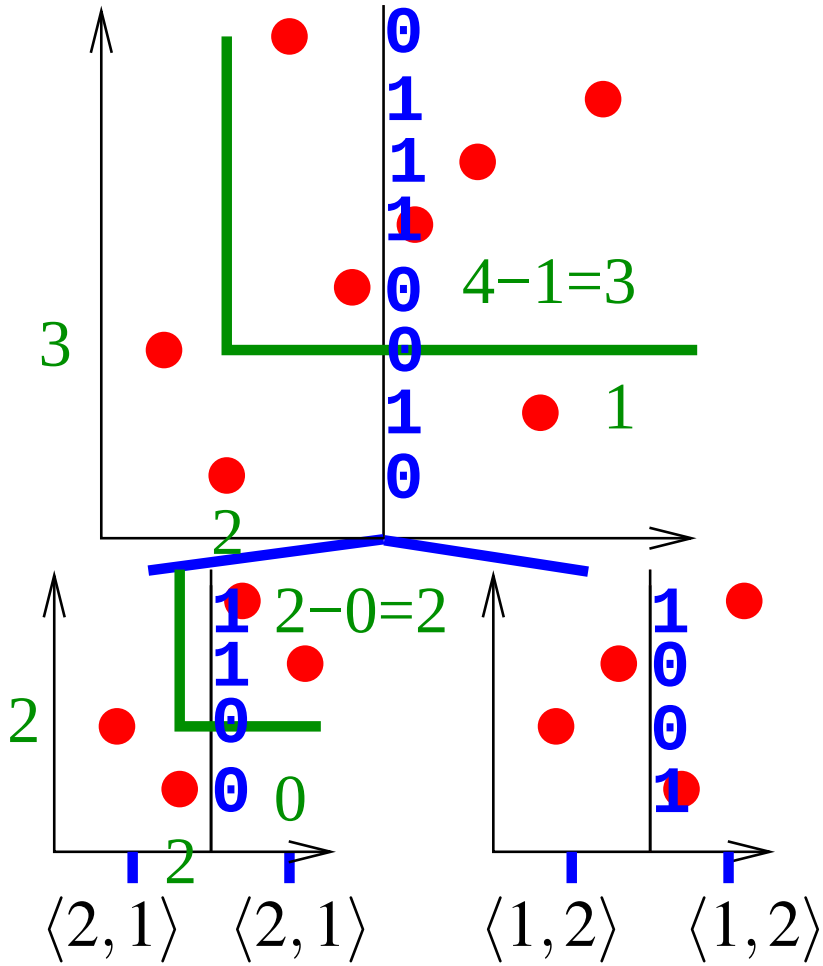
    return ℓ.intDominanceCount$(x, y-y_r) + \lceil n/2 \rceil - y_r$

  else

    return r.intDominanceCount$(x - \lfloor n/2 \rfloor, y_r)$



Example



Analysis

Only one recursive call

Recursion depth $\mathrm{O}(\log n)$.

rank in constant time (see below)

Time $\mathrm{O}(\log n)$

Wavelet Tree Dominance Reporting Query

Function intDominanceReporting(x,y) // $[x,n]\times[y,n]\cap P$

  if $n \le n_0$ **then return** $[x,n]\times[y,n]\cap P$ // brute force

  $R := \emptyset$ // Result

  $y_r := b.\text{rank}(y)$ // Number of els $\le y$ in right half

  if $x \le \lfloor n/2 \rfloor$ **then** // Both halves interesting

    if $y - y_r < \frac{n}{2}$ **then** $R := R \cup \ell.\text{intDominanceReporting}(x, y - y_r)$

    if $y_r < \frac{n}{2}$ **then** $R := R \cup r.\text{oneSidedReporting}(y_r)$

  else if $y_r < \frac{n}{2}$ **then** $R := R \cup r.\text{intDominanceReporting}(x - \lfloor n/2 \rfloor, y_r)$

  return R

Function oneSidedReporting(y) // $[1,n] \times [y,n] \cap P$

  if $n \leq n_0$ **then return** $[1,n] \times [y,n] \cap P$ // brute force

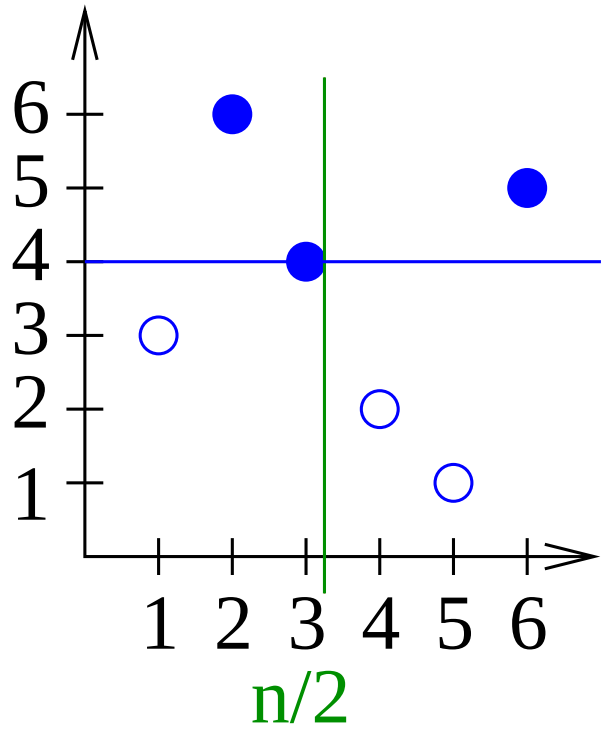
  y_r:= b.rank(y) // Number of els $\leq y$ in right half

  R:= $\emptyset$

  if $y_r < \frac{n}{2}$ **then** R:= $R \cup r$.oneSidedReporting(y_r)

  if $y - y_r < \frac{n}{2}$ **then** R:= $R \cup \ell$.oneSidedReporting$(y - y_r)$

  return R



Analysis

Recurrence

$T(n_0,0) = \mathrm{O}(1)$

$T(n,0) = T(n/2,0) + O(1) \Longrightarrow T(n,0) = \mathrm{O}(\log n)$

$T(n,k) = T(n/2,k') + T(n/2,k-k') + O(1)$ thus

$T(n,k) \leq ck' \log n + c(k-k')\log n + c = c(1+k\log n) =$
$\mathrm{O}(k\log n)$

Time $\mathrm{O}(k+\log n)$ needs additional factor $\log n$ space.

E.g. save complete list on all levels

Exercise?

General Reporting Queries

4-sided

$\rightsquigarrow$

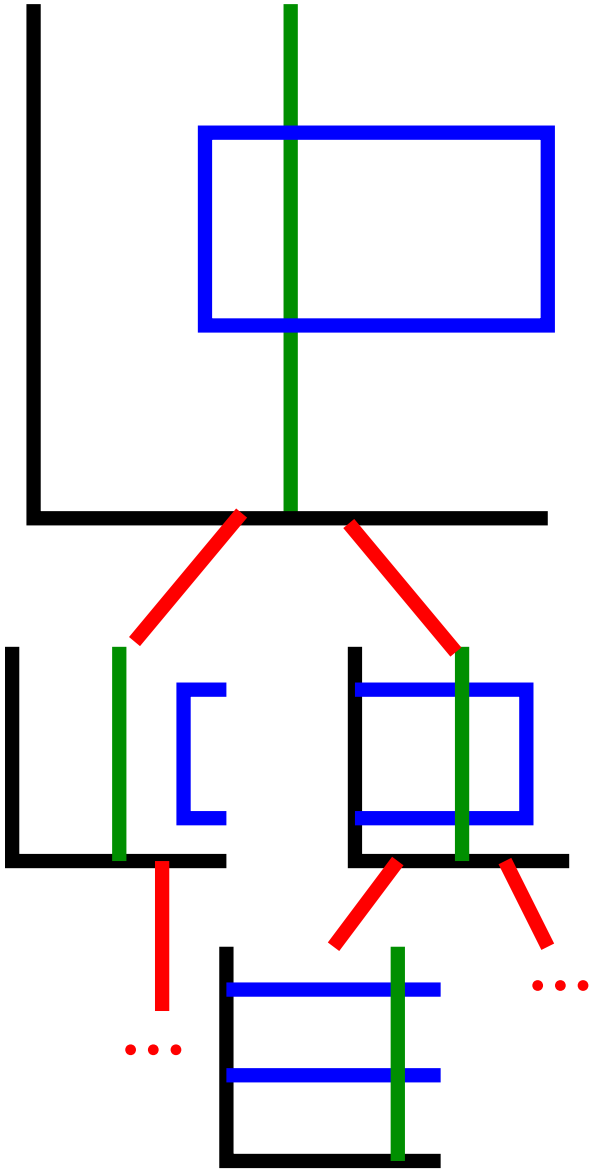
3-sided (2 variants)

$\rightsquigarrow$

y-range

Analogously to oneSidedReporting

(two ranks instead of one)



Bit Vectors v with rank in $\mathrm{O}(1)$

Choose $B = \Theta(\log n)$.

Preprocessing bRank$[i]$:= v.rank(iB)

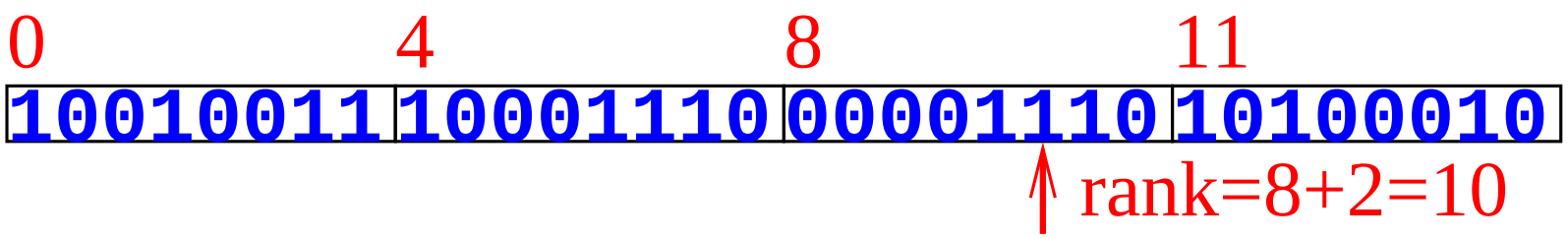
Time $\mathrm{O}(n)$, Space $\mathrm{O}(n)$ bits

Reduction to logarithmic input size:

Function rank(j) **return** bRank$[j$ **div** $B] +$ rank$(v[B(j$ **div** $B)..j])$

Logarithmic size:

Machine instruction (population count) or table lookup (e.g. size $\sqrt{n}$ table)



More on Bit Vectors

- Additional important operation $b.\text{select}(i) :=$ position of ith 1-bit. Also in time $\mathrm{O}(1)$
- Information theoretic asympt. optimal space $n + o(n)$ bits possible.
- Basis for further succinct data structures
- Examples: tree with space $2n + o(n)$ bits and navigation in constant time.