Winter Semester 2023/24

Advanced Algorithms

http://www14.in.tum.de/lehre/2023WS/ada/index.html.en

Susanne Albers

Department of Computer Science

TU München

Organization

Lectures: 3 SWS
Tue, Thu 12–14
Lecture hall: Galileo 8120.EG.001

Exercises: 2 SWS
Teaching assistant: Sebastian Schubert,
Malte Kriegelsteiner

Tue 14–16: Room MI 02.04.011
Thu 14–16: Room MI 01.07.023

Organization

Problem sets: Made available on Tuesday by 10:00 am via Moodle.

Exam: Written exam, date will be announced.

Valuation: 6 ECTS (3 + 2 SWS)

Prerequisites: Grundlagen: Algorithmen und Datenstrukturen (GAD)
Diskrete Strukturen (DS)
Diskrete Wahrscheinlichkeitstheorie (DWT)

Literature

- Th. Cormen, C. Leiserson, R. Rivest, and C. Stein. Introduction to Algorithms, Third Edition, MIT Press, 2009.
- J. Kleinberg and E. Tardos. Algorithm Design. Pearson, Addison Wesley, 2006.
- M. Mitzenmacher and E. Upfal. Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis. Second Edition, Cambridge University Press, 2017.
- Th. Ottmann und P. Widmayer: Algorithmen und Datenstrukturen. 6. Auflage, Springer Verlag, 2017.
- Research papers

Content

Design and analysis techniques for algorithms

- Divide and conquer
- Greedy approaches
- Dynamic programming
- Randomization
- Amortized analysis

Content

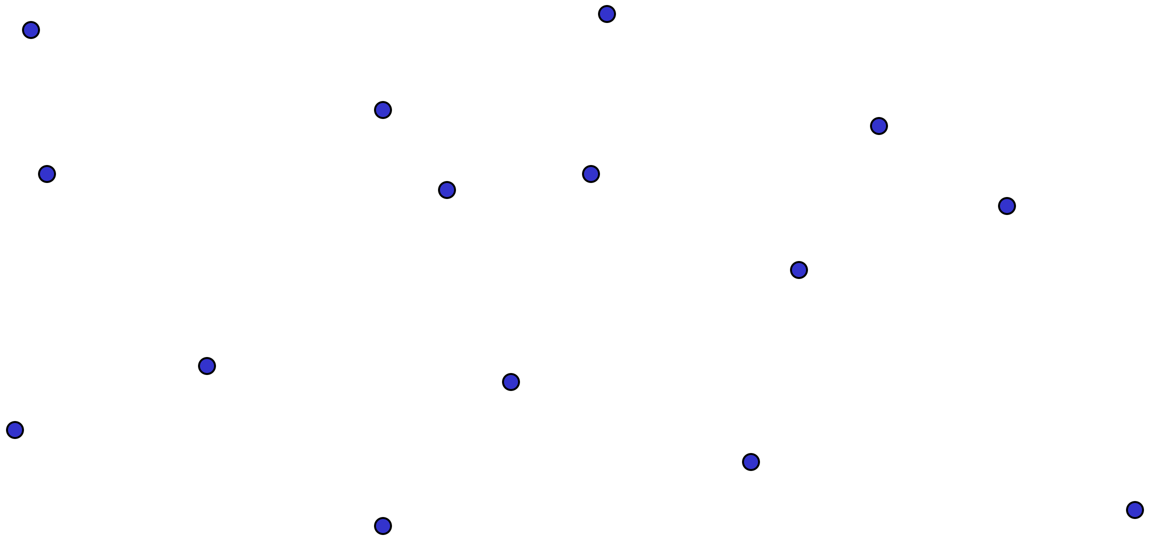
Problems and application areas:

- Geometric algorithms
- Algebraic algorithms
- Graph algorithms
- Data structures
- Algorithms on strings
- Optimization problems
- Complexity

Geometric divide-and-conquer

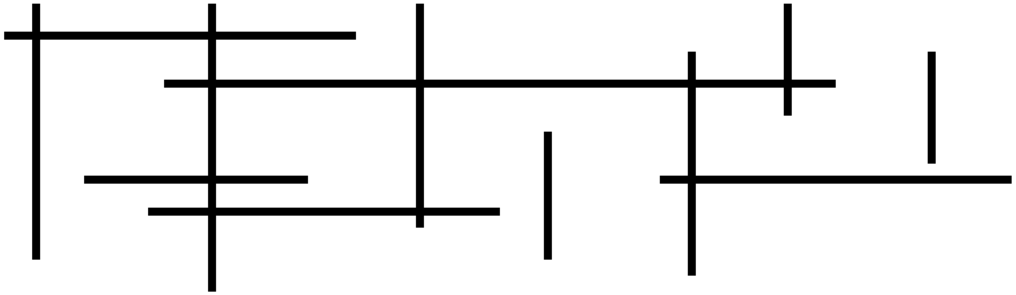
Closest Pair Problem:

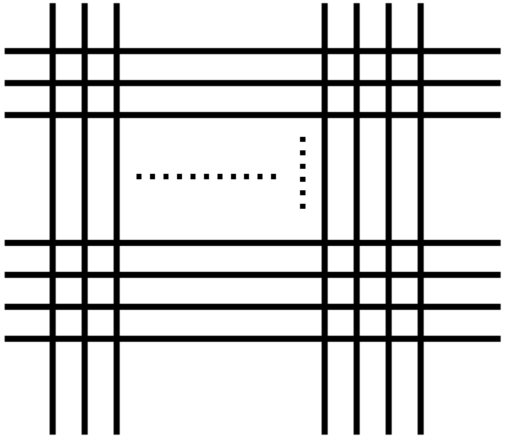
Given a set *S* of *n* points in the plane, find a pair of points with the smallest distance.



Line segment intersection

Find all pairs of intersecting line segments.





Fast Fourier Transform

$p, q \in R[x]$

$$p(x) = a_n x^n + \ldots + a_1 x^1 + a_0$$
$$q(x) = b_n x^n + \ldots + b_1 x^1 + b_0$$

Multiplication

$$p(x)q(x) = (a_n x^n + \ldots + a_0)(b_n x^n + \ldots + b_0)$$
$$= c_{2n} x^{2n} + \ldots + c_1 x^1 + c_0$$

Fast Fourier Transform

- FFT algorithms compute the discrete Fourier transform (DFT).

- Many applications in engineering, science and mathematics.
  - Digital signal processing (e.g. UMTS and LTE)
  - Image processing
  - Data compression
  - Partial differential equations

- Popular algorithm by J. Cooley and J.W. Tukey, 1965, based on earlier ideas of C.F. Gauß in 1805.

- Included in Top 10 Algorithms of the 20th Century by IEEE journal *Computing in Science & Engineering* (2000).

Randomization

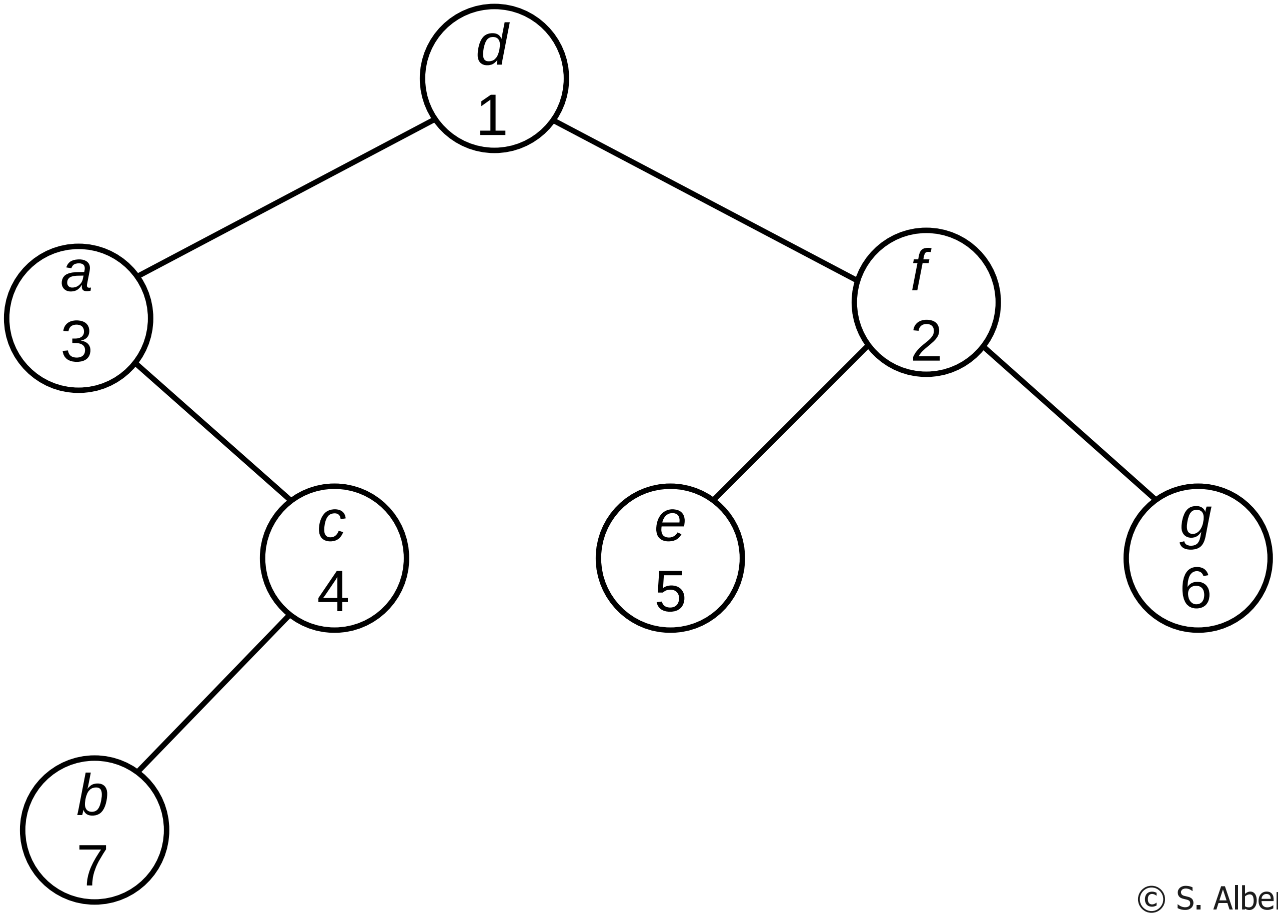
Algorithm may make random choices.

Advantages: Speed and simplicity

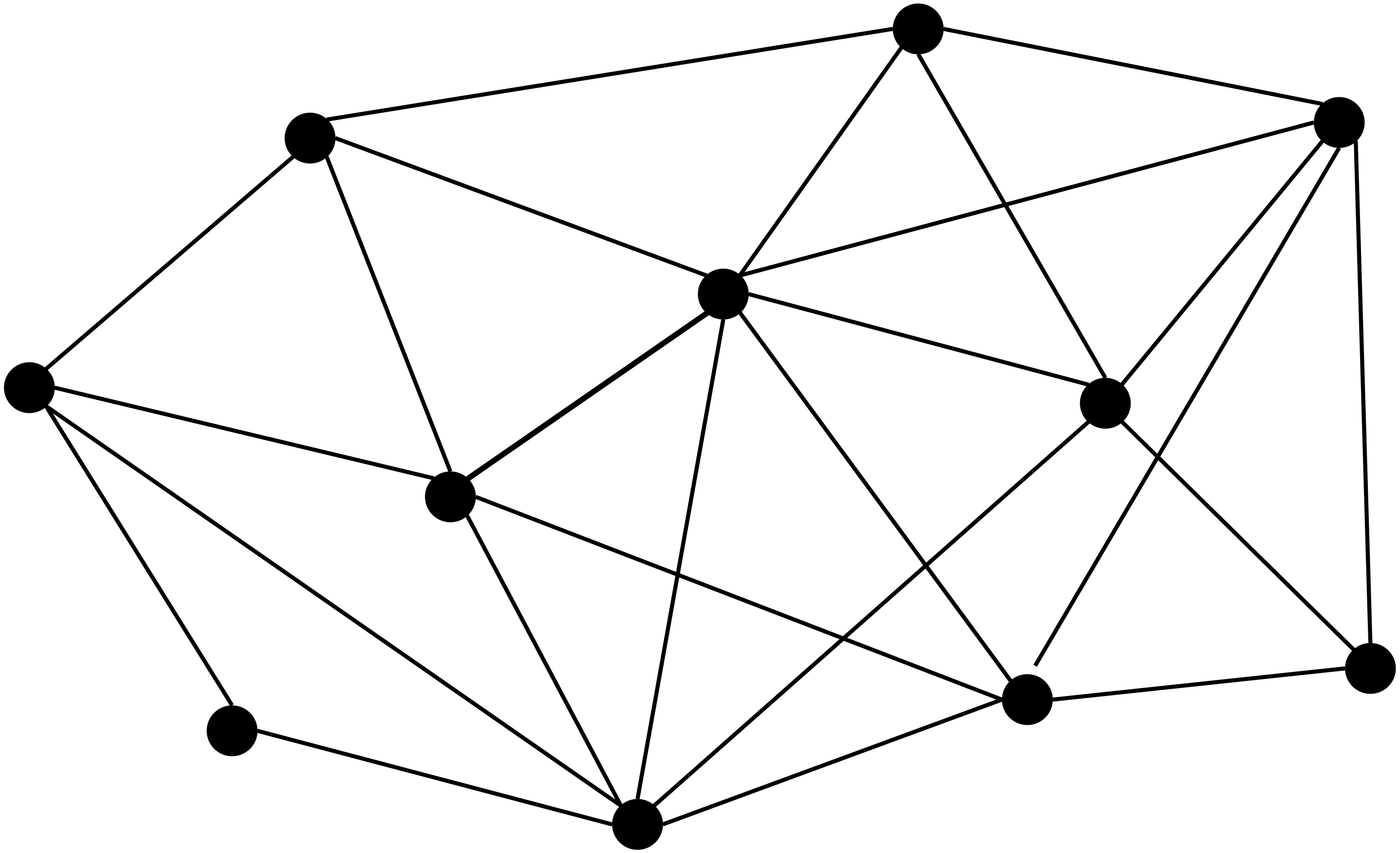
- Types of randomized algorithms
- Randomized primality test
- Cryptography: RSA algorithm

Randomized Search Trees

key	*a*	*b*	*c*	*d*	*e*	*f*	*g*
priority	3	7	4	1	5	2	6



Minimum cuts



Suffix Trees

Static texts

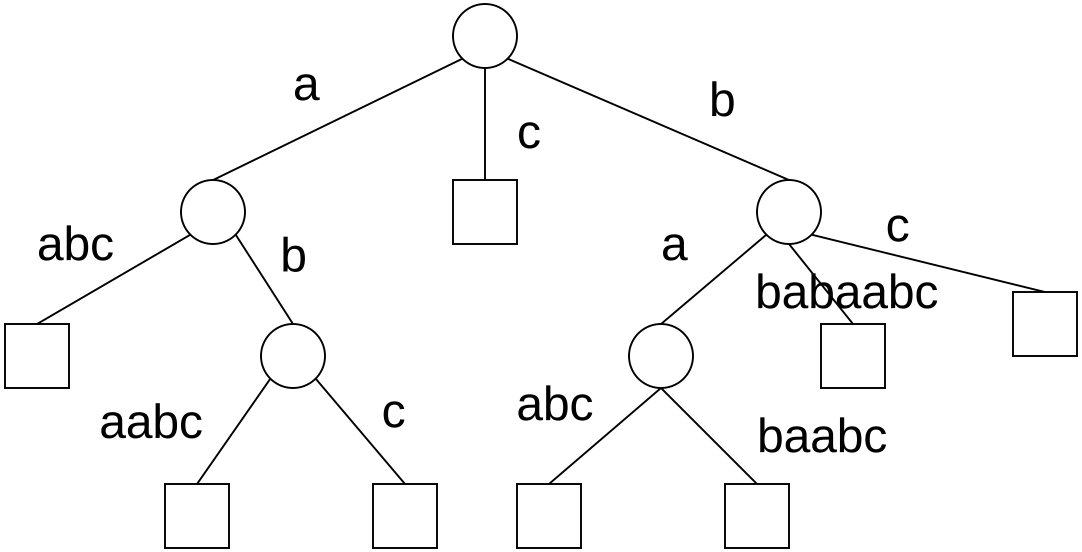
- Literature databases
- Library systems
- Gene databases
- World Wide Web

Search index

for a text σ in order to search for several patterns α.

Substring search in time $O(|\alpha|)$.

Suffix tree



σ = bbabaabc

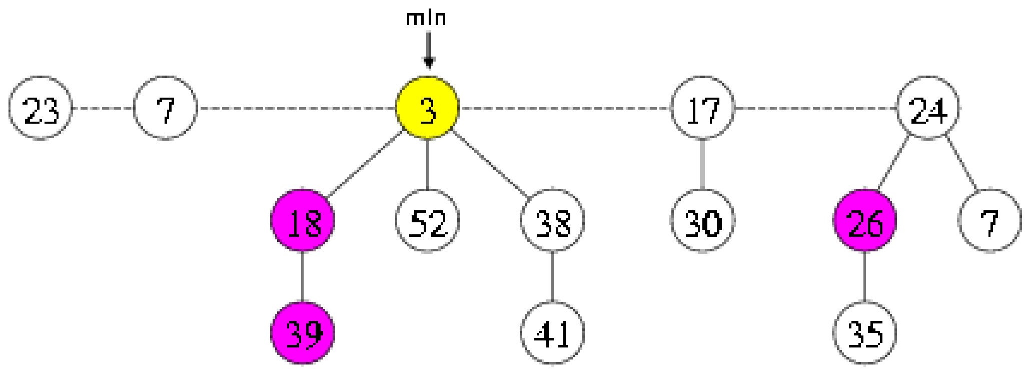
Amortized analysis

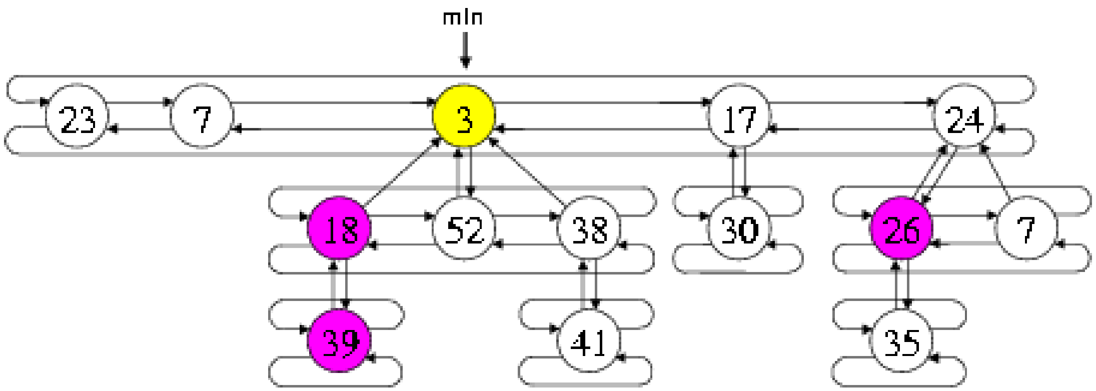
- Best case
- Worst case
- Average case
- Amortized worst case

What is the average cost of an operation in a worst case sequence of operations?

Average execution time of an operation is small, even though a single operation can have a high execution time.

Fibonacci heaps





Greedy algorithms

In each step make the choice that looks best at the moment

Basic examples

- The coin-changing problem
- The Traveling Salesman Problem

Scheduling problems

- Interval scheduling
- Scheduling to minimize lateness

Discussion: Shortest paths and minimum spanning trees

Dynamic programming

Recursive approach: Solve a problem by solving several smaller analogous subproblems of the same type. Then combine these solutions to generate a solution to the original problem.

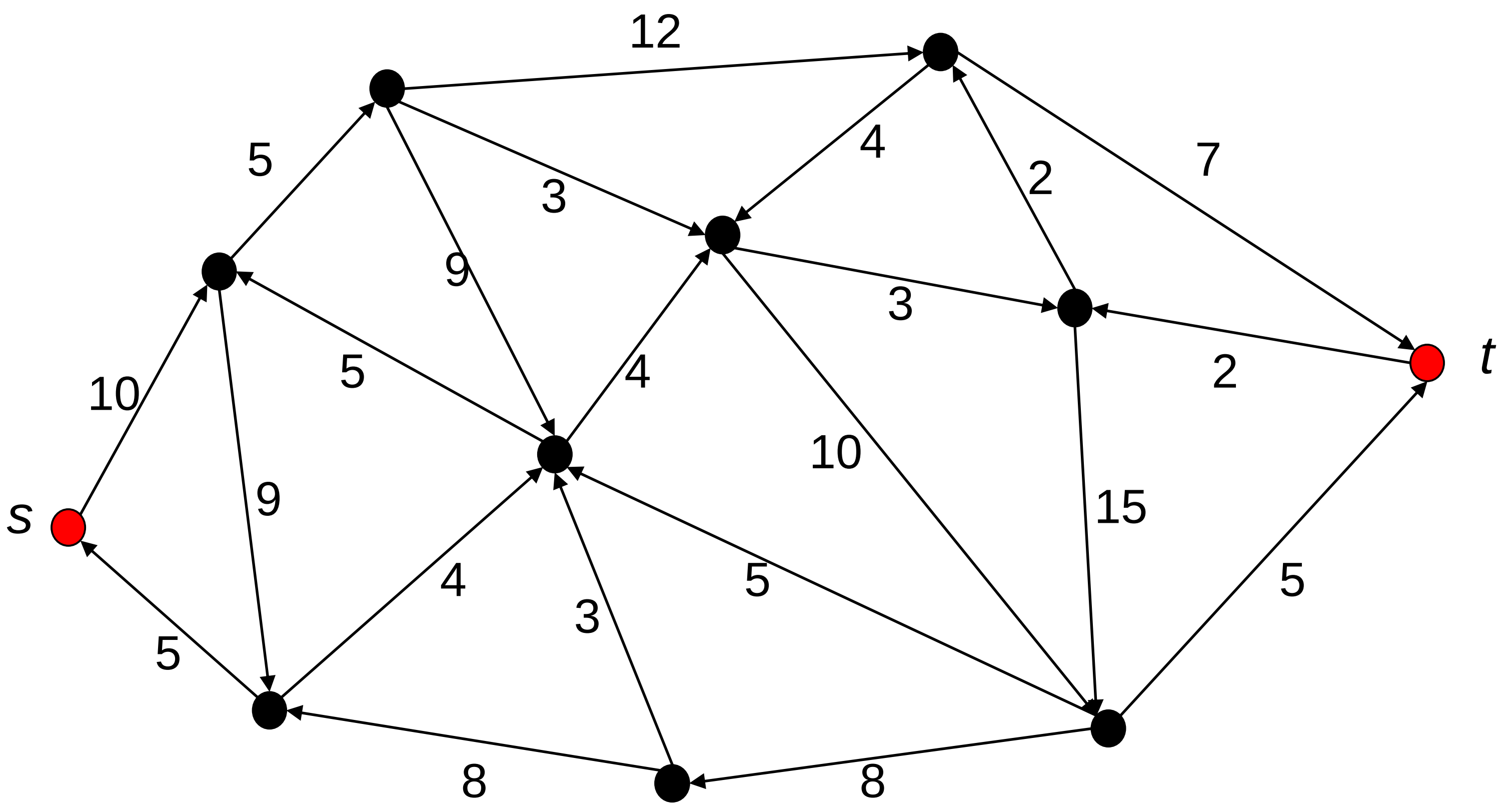
Drawback: Repeated computation of solutions

Dynamic-programming method: Once a subproblem has been solved, store its solution in a table so that it can be retrieved later by simple table lookup.

Dynamic programming

- Matrix chain multiplication
- Segmented least squares
- Optimal binary search trees
- Subset sums & knapsacks

Maximum flow problem

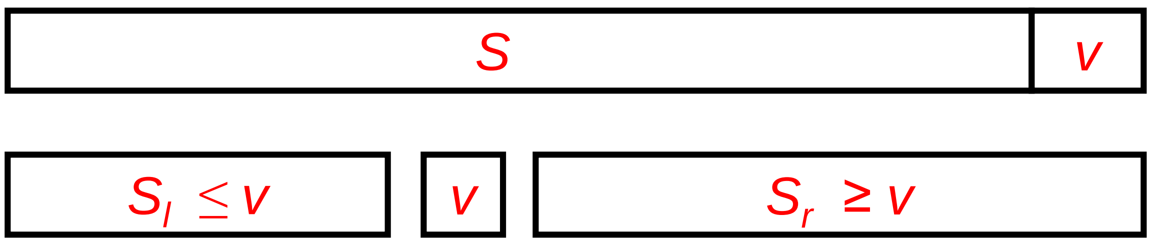


01 - Divide and Conquer

The divide-and-conquer paradigm

- Quicksort
- Formulation and analysis of the paradigm
- Geometric divide-and-conquer
  - Closest pair problem
  - Line segment intersection
  - Voronoi diagrams

Quicksort: Sorting by partitioning



function Quick (S: sequence): sequence;

{returns the sorted sequence S}

begin

if $\#S \leq 1$ then Quick:=S;

else { choose pivot/splitter element v in S;

partition S into S_l with elements $\leq v$,

and S_r with elements $\geq v$;

Quick:= $\boxed{\text{Quick}(S_l) \,|\, v \,|\, \text{Quick}(S_r)}$ }

end;

Formulation of the D&C paradigm

Divide-and-conquer method for solving a problem instance of size n:

1. Divide

n > c: Divide the problem into k subproblems of sizes $n_1,...,n_k$ ($k \geq 2$).

$n \leq c$: Solve the problem directly.

2. Conquer

Solve the k subproblems in the same way (recursively).

3. Merge

Combine the partial solutions to generate a solution for the original instance.

Analysis

$T(n)$: maximum number of steps necessary for solving an instance of size n

$$T(n) = \begin{cases} a & n \le c \\ T(n_1) + \ldots + T(n_k) & n > c \\ + \text{ cost for divide and merge} & \end{cases}$$

Special case: $k = 2$, $n_1 = n_2 = n/2$

cost for divide and merge: DM(n)

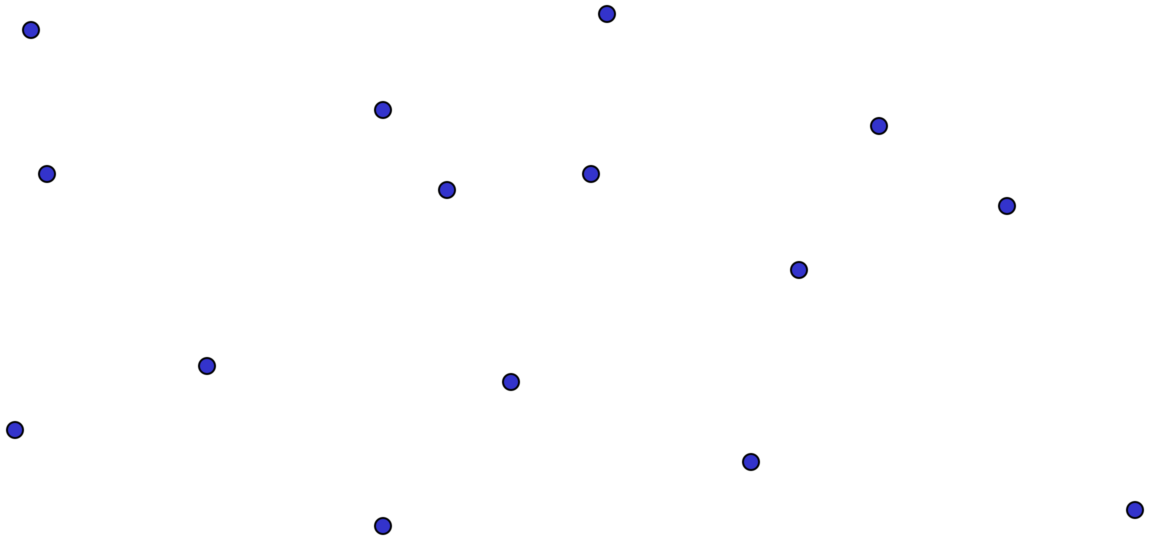
$T(1) = a$

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

Geometric divide-and-conquer

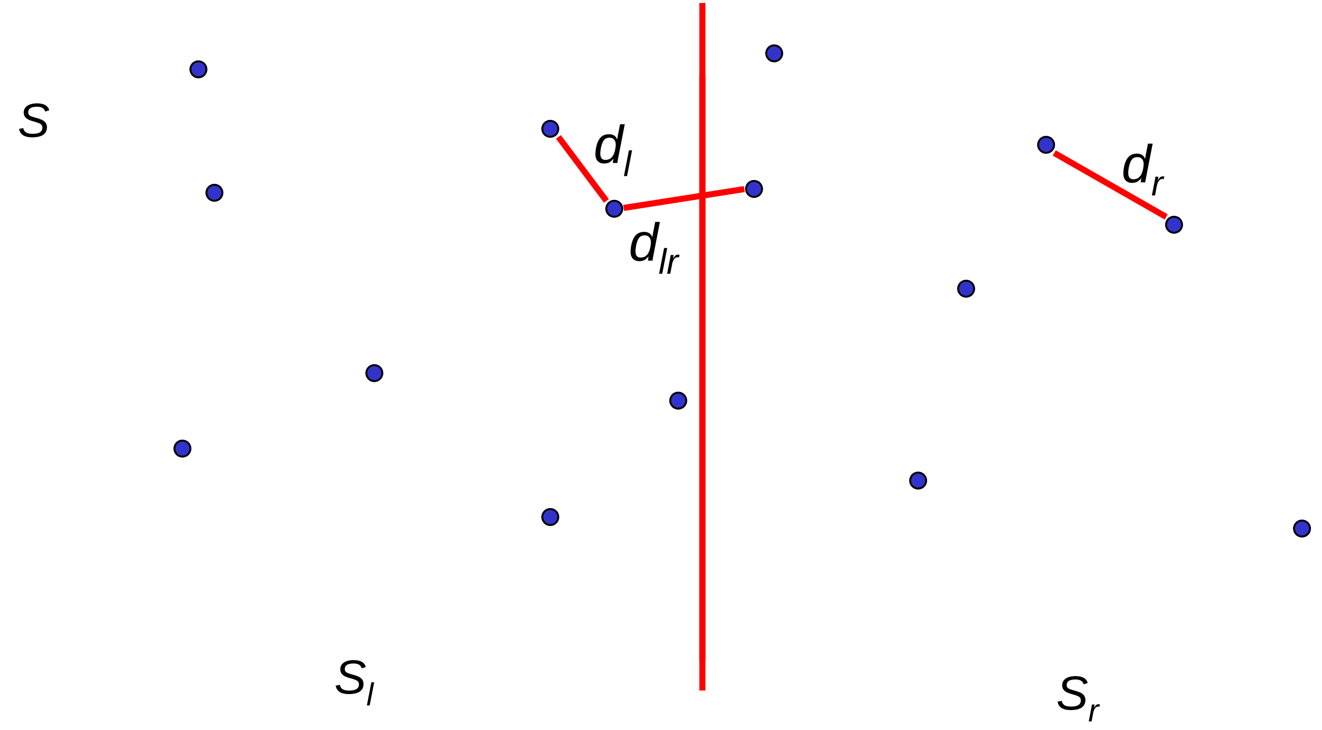
Closest Pair Problem:

Given a set *S* of *n* points in the plane, find a pair of points with the smallest distance.



Divide-and-conquer method

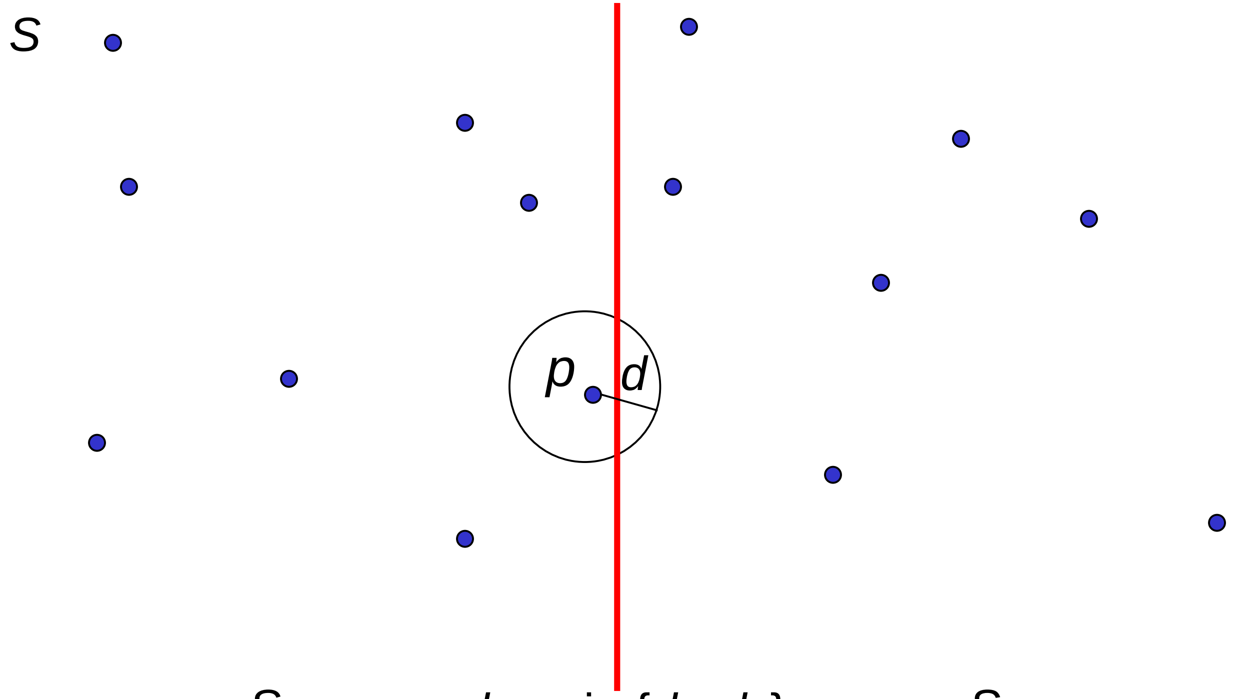
1. **Divide:** Divide S into two equal sized sets S_l und S_r .
2. **Conquer:** d_l = mindist(S_l) d_r = mindist(S_r)
3. **Merge:** $d_{lr} = \min\{ d(p_l, p_r) \mid p_l \in S_l, p_r \in S_r \}$
   return $\min\{d_l, d_r, d_{lr}\}$



Divide-and-conquer method

1. **Divide:** Divide S into two equal sets S_l und S_r .
2. **Conquer:** $d_l = \text{mindist}(S_l)$ $d_r = \text{mindist}(S_r)$
3. **Merge:** $d_{lr} = \min\{ \text{d}(p_l, p_r) \mid p_l \in S_l, \text{p}_\text{r} \in S_r \}$
   return $\min\{d_l, d_r, d_{lr}\}$

Computation of d_{lr} :



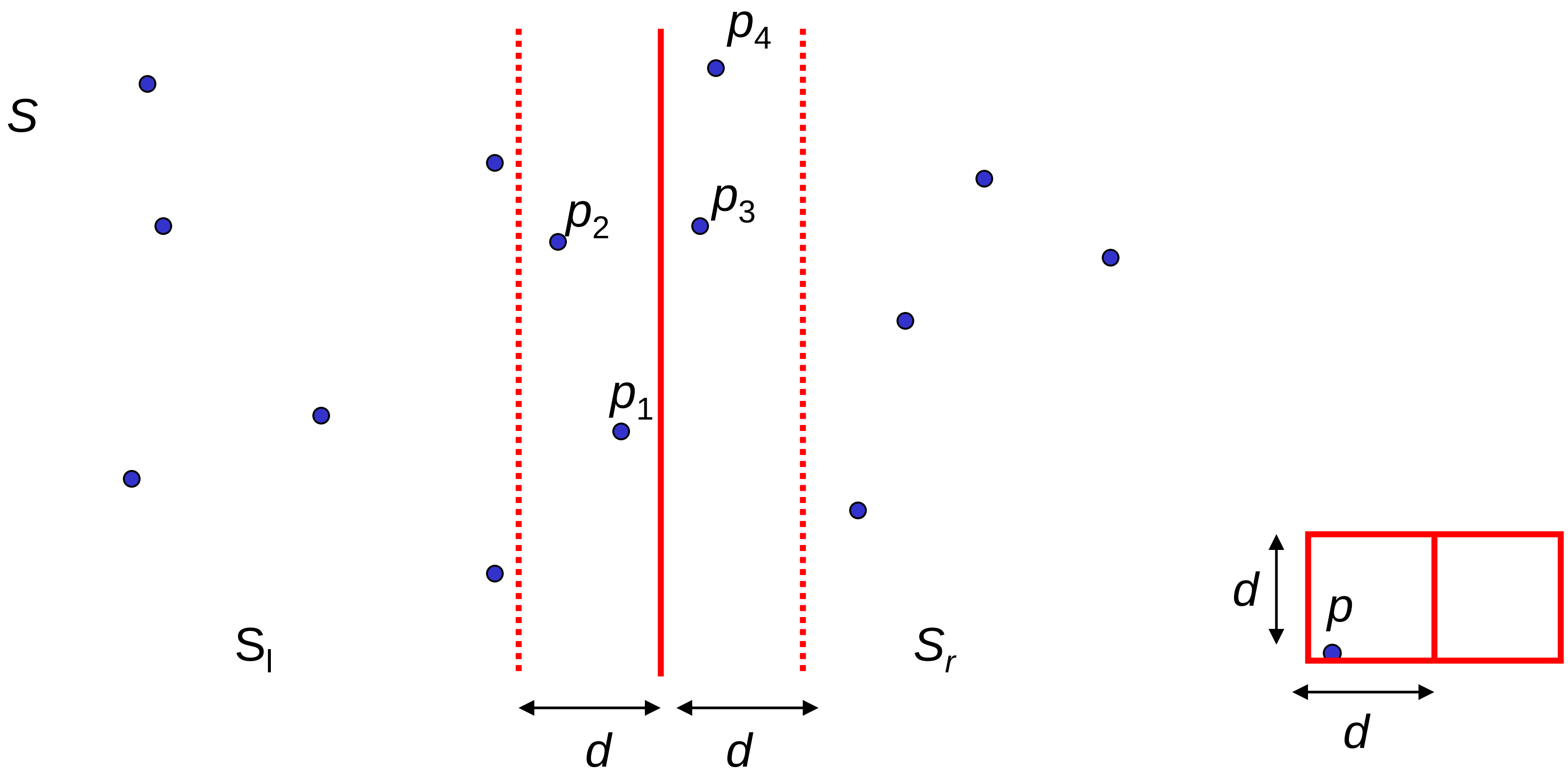
S_l $d = \min \{d_l, d_r\}$ S_r

Merge step

1. Consider only points within distance *d* of the bisection line, in the order of increasing y-coordinates.

2. For each point *p* consider all points *q* within *y*-distance at most *d*; there are at most 7 such points.

Merge step



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

Implementation

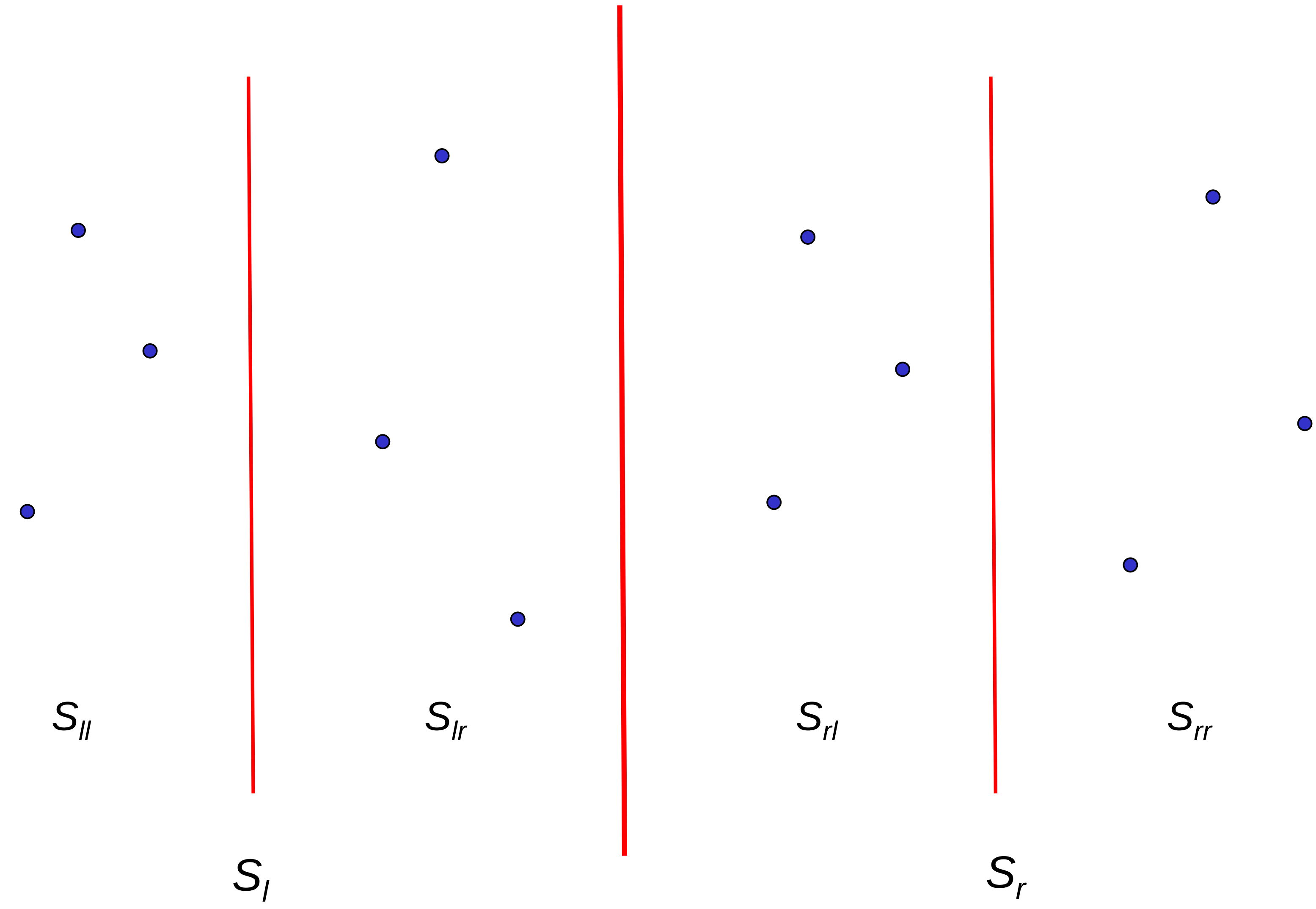
- Initially sort the points in S in order of increasing x-coordinates $O(n \log n)$.

  Each bisection line can be determined in $O(1)$ time.

- Once the subproblems S_l , S_r are solved, generate a list of the points in S in order of increasing y-coordinates.

  This can be done by merging the sorted lists of points of S_l , S_r (merge sort).

Sorted lists



Running time (divide-and-conquer)

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

- Guess the solution by repeated substitution.
- Verify by induction.

Solution: O(*n* log *n*)

Guess by repeated substitution

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

$$\begin{aligned} T(n) &= 2T(n/2) + an = 2(2T(n/4) + an/2) + an \\ &= 4T(n/4) + 2an = 4(2T(n/8) + an/4) + 2an \\ &= 8T(n/8) + 3an = 8(2T(n/16) + an/8) + 3an \\ &= 16T(n/16) + 4an \end{aligned}$$

Verify by induction

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

$n = 2^i$

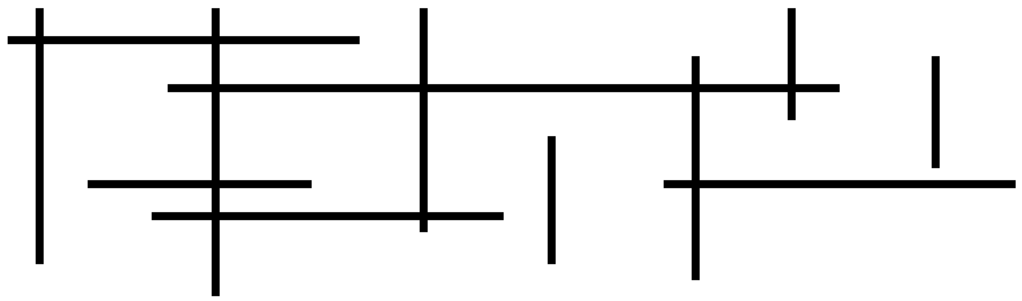
$i = 1$: ok

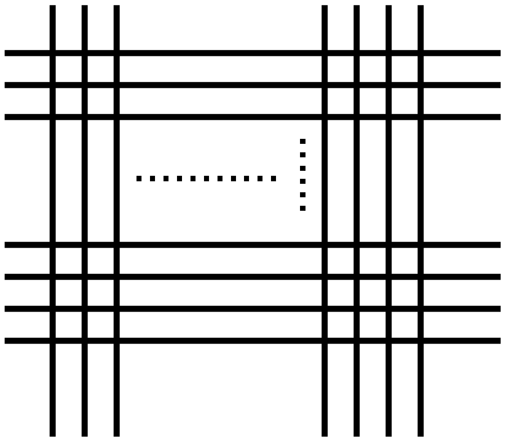
$i > 1$

$$\begin{aligned} T(2^i) &= 2T(2^{i-1}) + a2^i \\ &\leq 2a2^{i-1}(i-1) + a2^i \\ &= a2^i(i-1) + a2^i \\ &= a2^i i \\ &= an \log n \end{aligned}$$

Line segment intersection

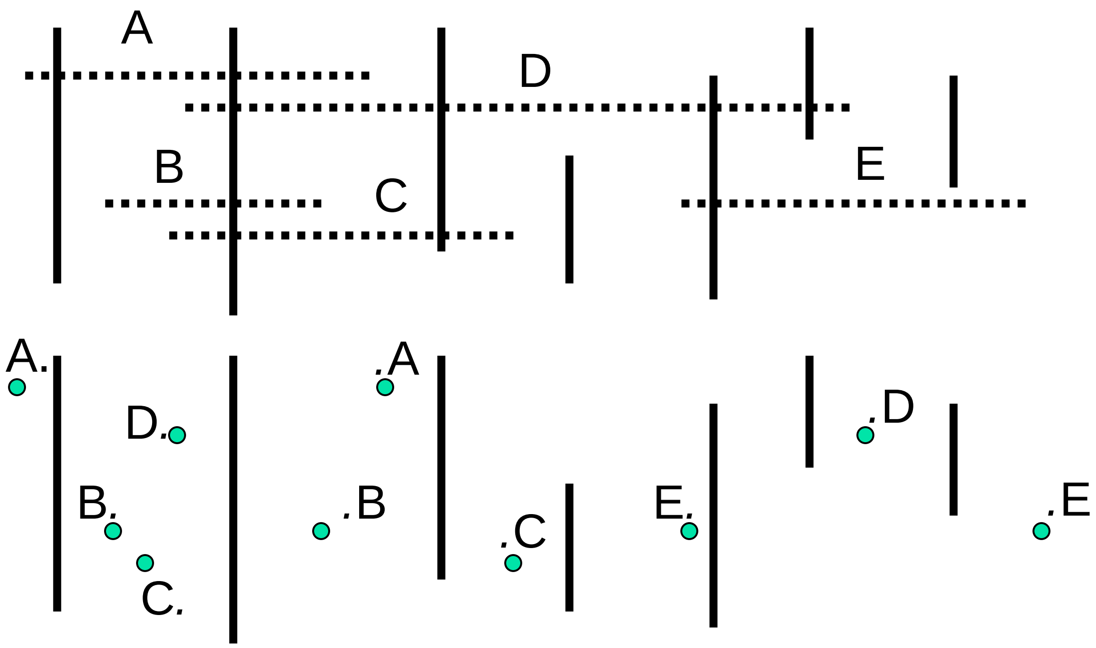
Find all pairs of intersecting line segments.





Line segment intersection

Find all pairs of intersecting line segments.



The representation of the horizontal line segments by their endpoints allows for a vertical partitioning of all objects.

ReportCuts

Input: Set S of vertical line segments and endpoints of horizontal line segments.

Output: All intersections of vertical line segments with horizontal line segments, for which at least one endpoint is in S.

1. Divide

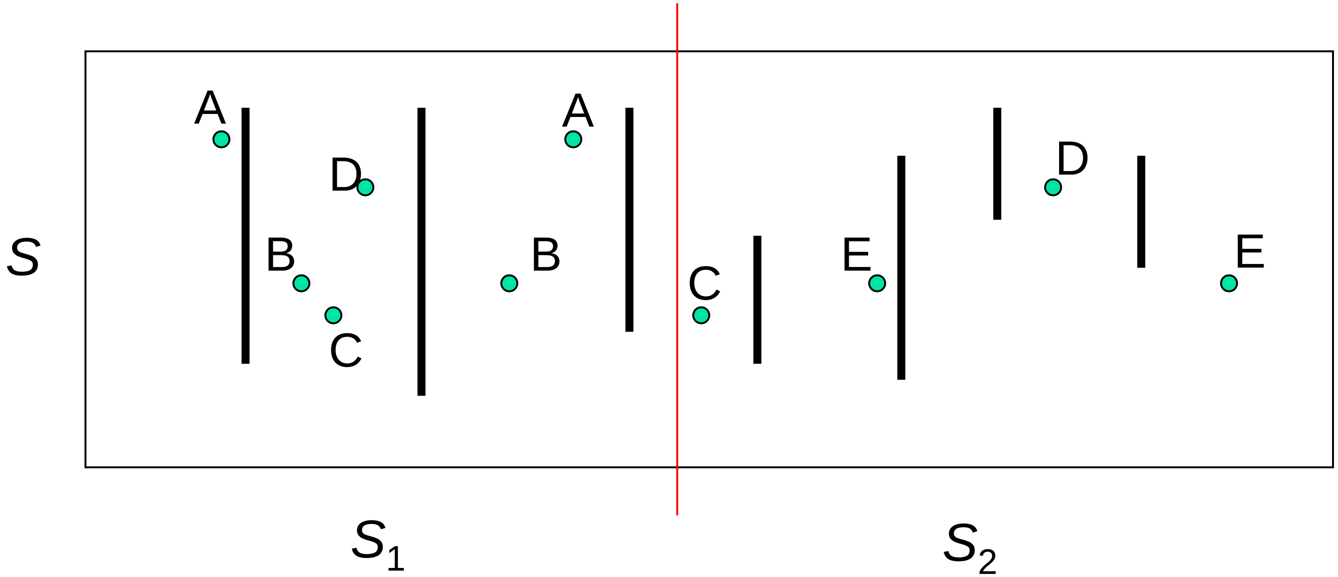
if $|S| > 1$

then using vertical bisection line L, divide S into equal size sets S_1 (to the left of L) and S_2 (to the right of L)

else S contains no intersections

ReportCuts

1. Divide:



2. Conquer:

ReportCuts(S_1); ReportCuts(S_2)

ReportCuts

3. Merge: ???

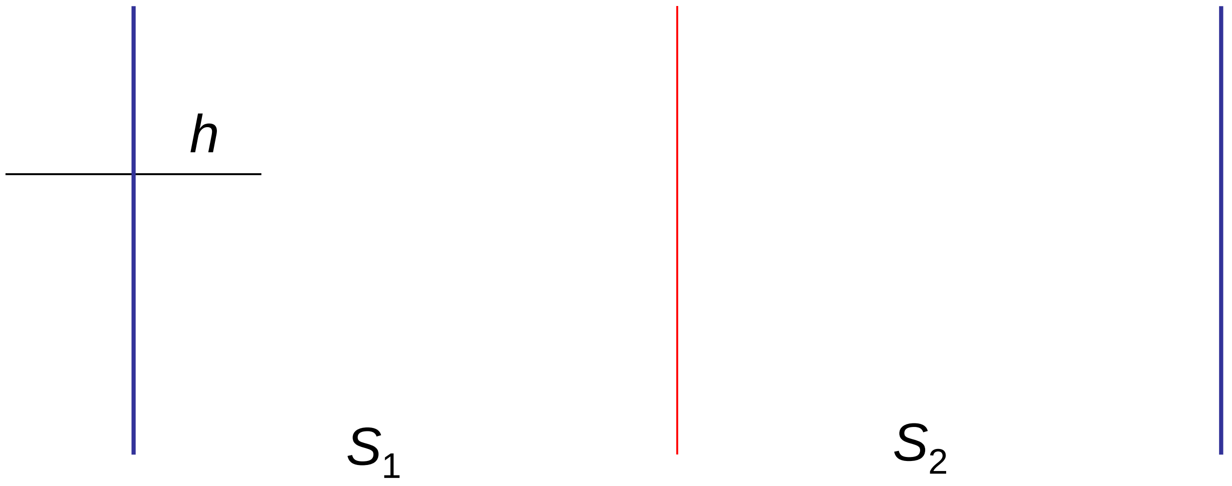
Possible intersections of a horizontal line segment h in S_1

Case 1: both endpoints in S_1

ReportCuts

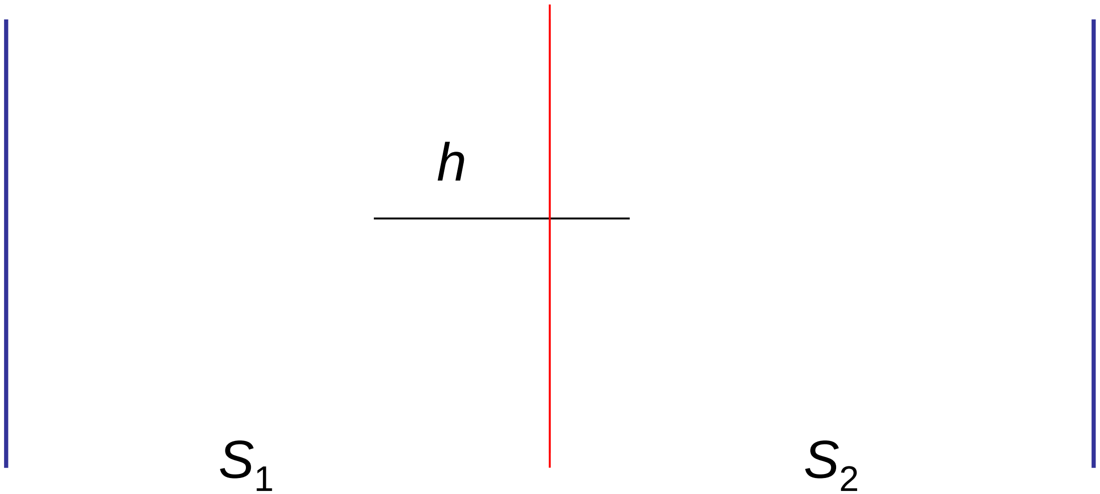
Case 2: only one endpoint of h in S_1

2 a) right endpoint in S_1

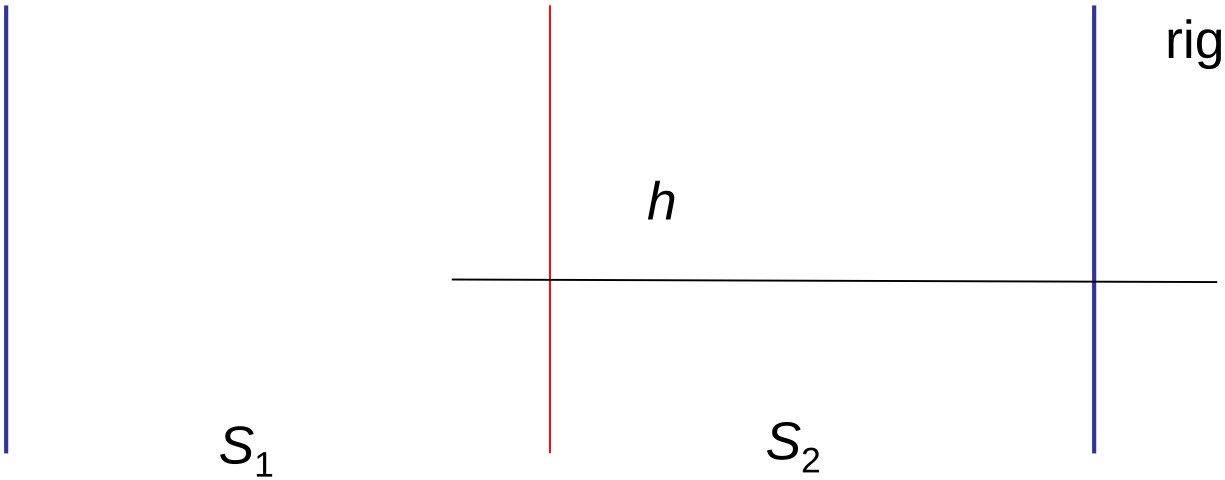


ReportCuts

2 b) left endpoint of h in S_1



right endpoint in S_2

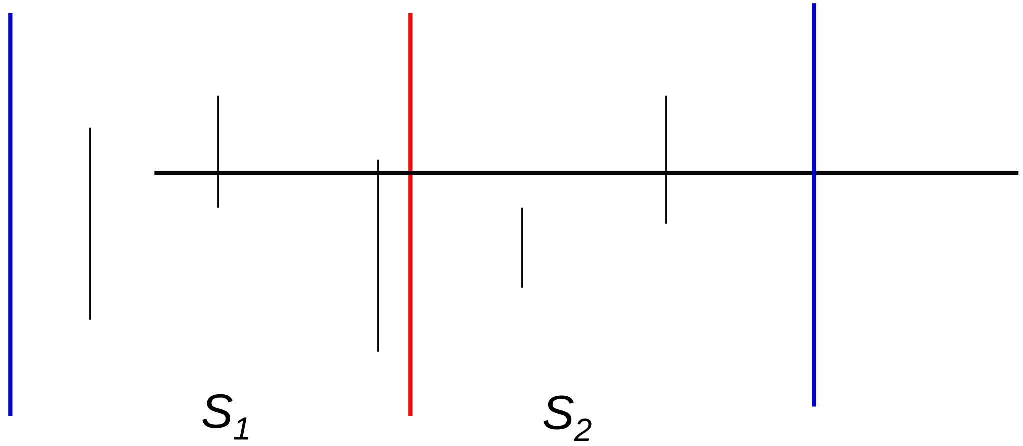


right endpoint not in S_2

Procedure: ReportCuts(S)

3. Merge:

Return the intersections of vertical line segments in S_2 with horizontal line segments in S_1, for which the left endpoint is in S_1 and the right endpoint is neither in S_1 nor in S_2 .
Proceed analogously for S_1 .



Implementation

Set S

$L(S)$: y-coordinates of all segments whose left endpoint in S, but right endpoint is not in S.

$R(S)$: y-coordinates of all segments whose right endpoint is in S, but left endpoint is not in S.

$V(S)$: y-intervals of all vertical line segments in S.

Base cases

S contains only one element e.

Case 1: $e = (x,y)$ is a left endpoint of horizontal line segment s

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

Case 2: $e = (x,y)$ is a right endpoint of horizontal line segment s

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

Case 3: $e = (x, y_1, y_2)$ is a vertical line segment s

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

Merge step

Assume that $L(S_i)$, $R(S_i)$, $V(S_i)$ are known for i = 1,2.
$S = S_1 \cup S_2$

$$L(S) = L(S_1)\backslash R(S_2) \cup L(S_2)$$

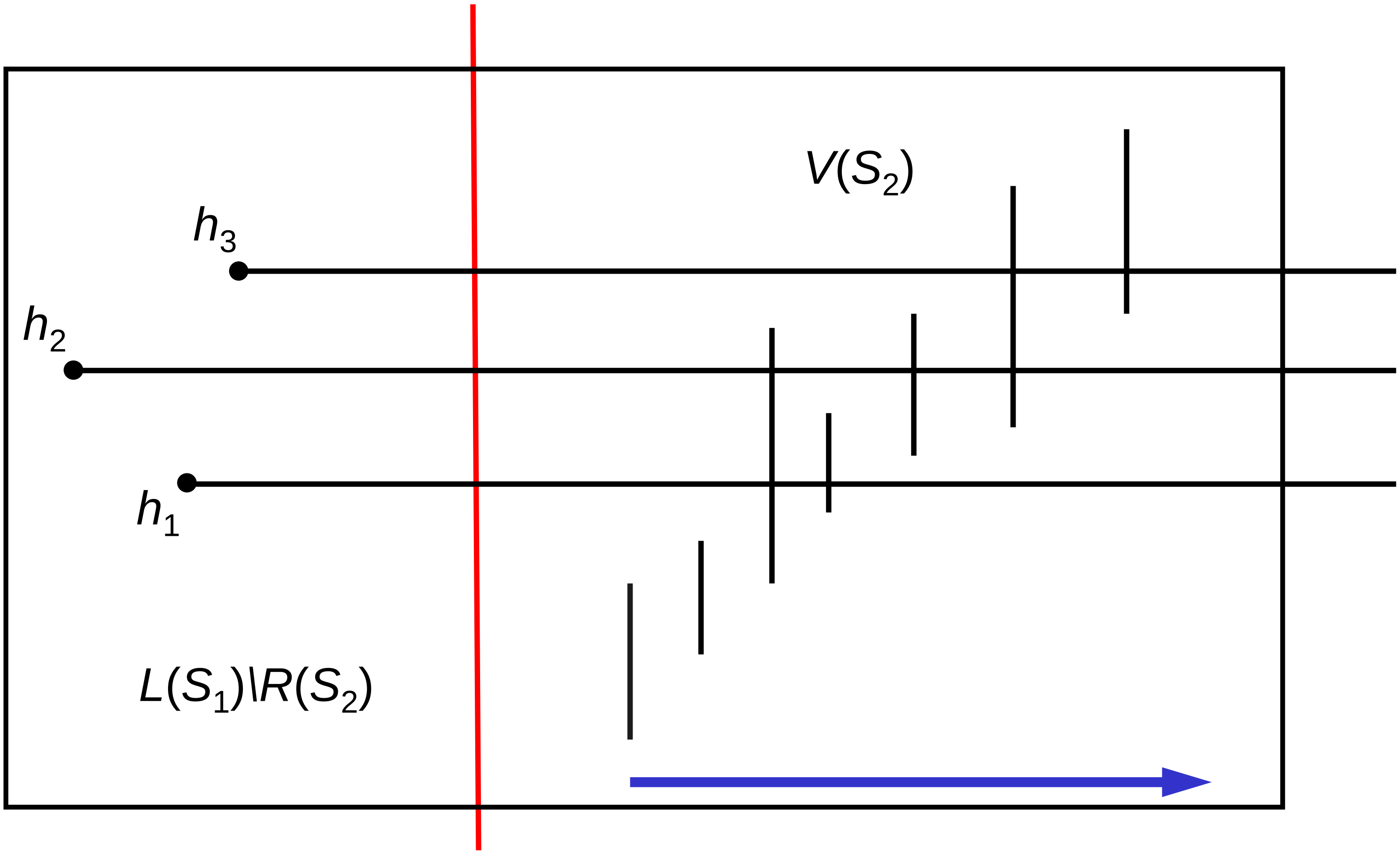
$$R(S) = R(S_2)\backslash L(S_1) \cup R(S_1)$$

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

L, R: ordered by increasing y-coordinates (and segment number)
linked lists

V: ordered by increasing lower endpoints
linked list

Output of the intersections



Running time

Initially, the input (vertical line segments, left/right endpoints of horizontal line segments) has to be sorted and stored in an array.

Divide-and-conquer:

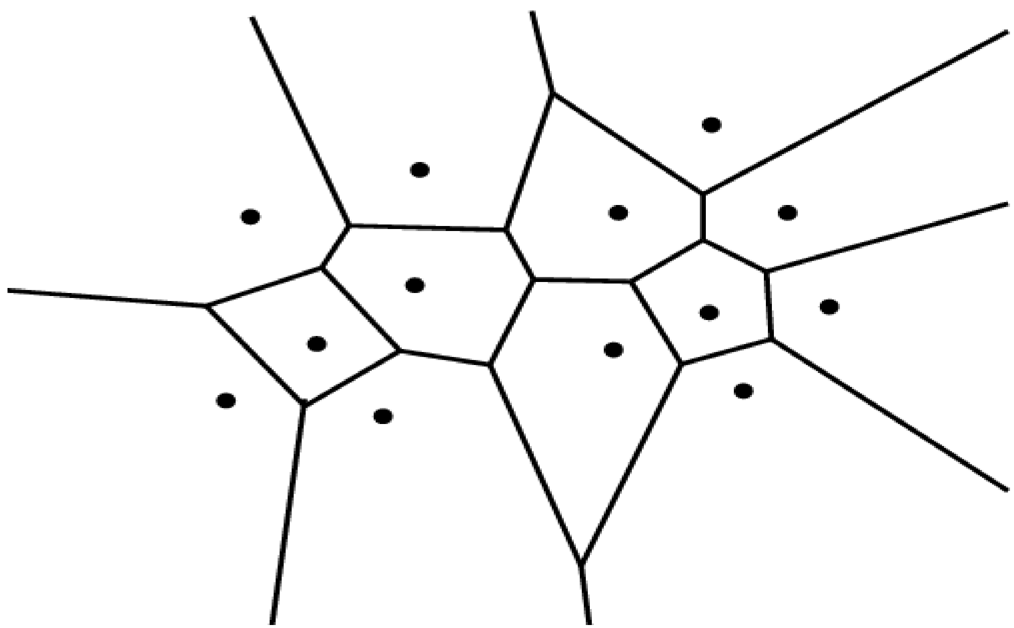
$$T(n) = 2T(n/2) + a \cdot n + \text{size of output}$$
$$T(1) = O(1)$$

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

Computation of a Voronoi diagram

Input: Set of sites

Output: Partition of the plane into regions, each consisting of the points closer to one particular site than to any other site.



Definition of Voronoi diagrams

P : Set of sites

$H(p \mid p') = \{x \mid x$ is closer to p than to $p'\}$

Voronoi region of *p:*

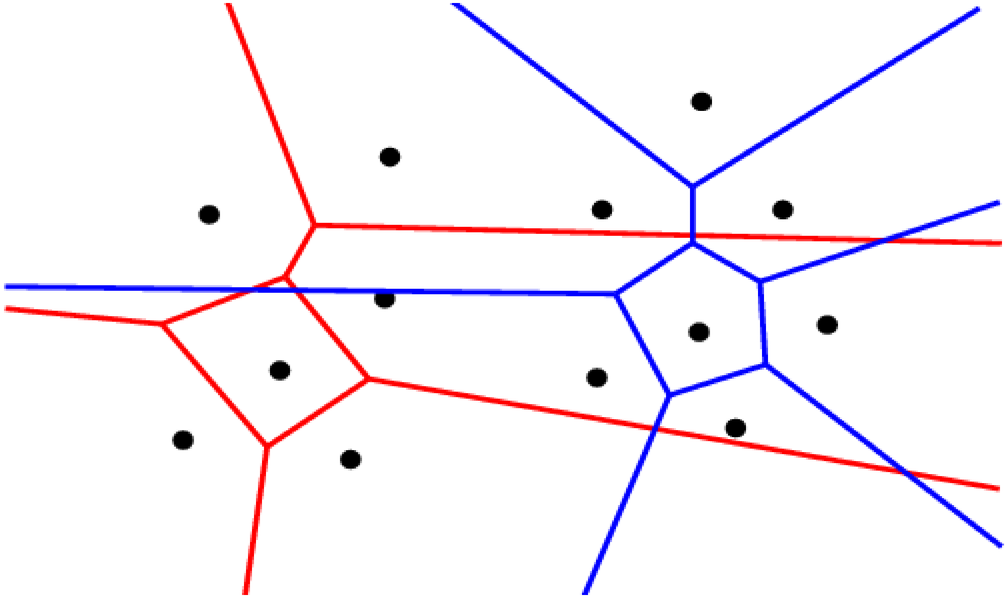
$$VR(p) = \bigcap_{p' \in P \setminus \{p\}} H(p \mid p')$$

Computation of a Voronoi Diagram

Divide: Partition the set of sites into two equal sized sets.

Conquer: Recursive computation of the two smaller Voronoi diagrams.

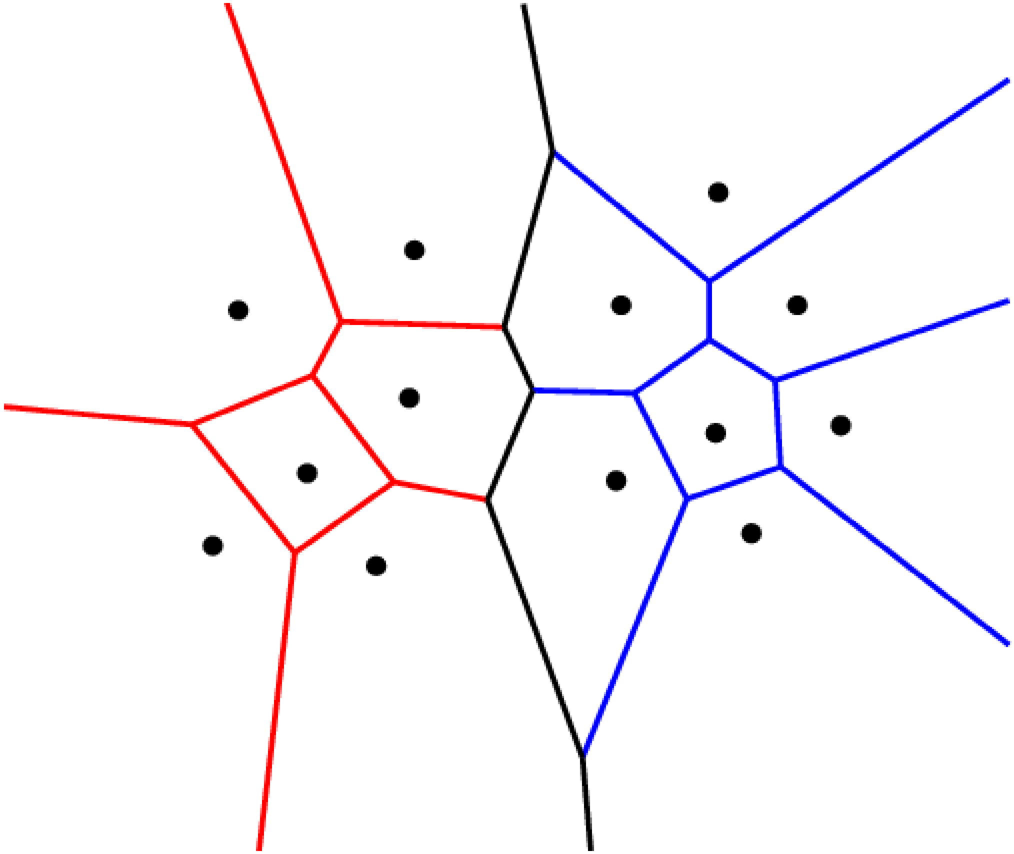
Stopping condition: The Voronoi diagram of a single site is the whole plane.



Merge: Connect the diagrams by adding new edges.

Computation of a Voronoi diagram

Output: The complete Voronoi diagram.



Running time: O(n log n), where n is the number of sites.