Algorithm Theory

01 - Introduction

Dr. Alexander Souza

Winter term 11/12

Organization

Lectures:	Tue	14-16	101-00-026
	Wed	16-17	101-00-026
Exercises:	Thu	8-10	101-01-018
	Thu	8-10	051-00-034
	Fri	12-14	101-00-018
	Fri	12-14	078-00-014

3 out of these 4 groups will take place biweekly
Registration during this class, teamwork up to 3 students

Sheet 1 will be out on Wed.,26.10.

Hand-in biweekly during Wed.-class

First hand-in Wed.,2.11., first tutorials Thu.,10.11./Fri.,11.11.

Web page: Contains slides, recording, schedule, sheets, grouping etc.

http://lak.informatik.uni-freiburg.de/
→ Teaching → Winter Term 2011/12 → Algorithm Theory

Organization

Final exam: Date and Time: t.b.a.

Admission

1 exercise presented during the tutorials

50% of total exercise points

More Details: Kursvorlesung, 3+1 SWS

6 ECTS Credits

Lectures in English

Tutorials supervised by Thomas Janson

English: Mahdi

German: Geißer, Jarecki

Camtasia recording available

Literature

Th. Ottmann, P. Widmayer:
Algorithmen und Datenstrukturen
4th Edition, Spektrum Akademischer Verlag,
Heidelberg, 2002

Th. Cormen, C. Leiserson, R. Rivest, C. Stein:
Introduction to Algorithms, Second Edition
MIT Press, 2001

Original literature

Algorithms and data structures

Design and analysis techniques for algorithms

- Divide and conquer
- Greedy approaches
- Dynamic programming — Store subproblem solutions in a table for faster reference
- Randomization
- Amortized analysis

Analysis technique

Algorithms and data structures

Problems and application areas

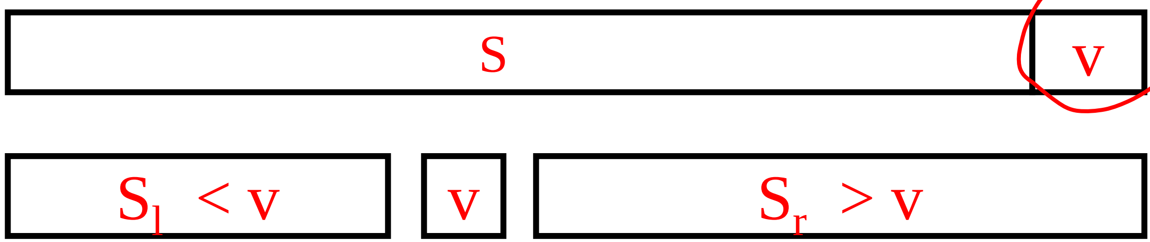
- Geometric algorithms
- Algebraic algorithms
- Graph algorithms
- Data structures
- Internet algorithms
- Optimization methods
- Algorithms on strings

Divide and Conquer

The divide-and-conquer paradigm

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

Quicksort: Sorting by partitioning



Pivot element

function Quick (S: sequence): sequence;

{returns the sorted sequence S}

begin

    if #S <= 1 then Quick:=S

    else { choose pivot element v in S;

        partition S into S_l with elements < v,

        and S_r with elements > v

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

end;

Formulation of the D&C paradigm

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

1. Divide

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

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

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**

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

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

cost for divide and merge: $DM(n)$

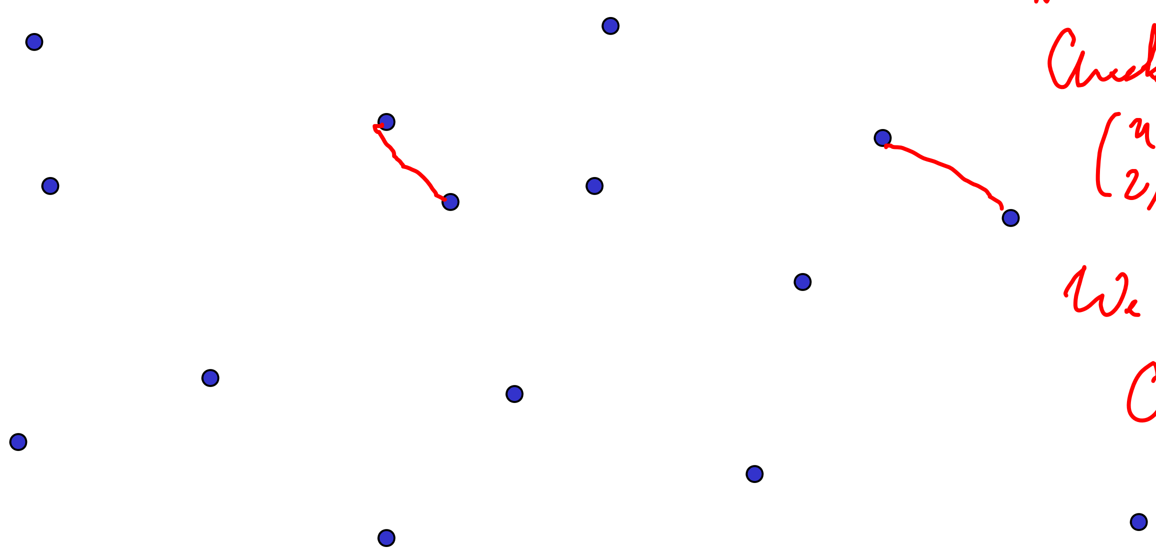
$T(1) = a$

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

Geometric divide-and-conquer

Closest Pair Problem:

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



Naive approach:
Check all pairs
$$\binom{n}{2} = \frac{n \cdot (n-1)}{2} = O(n^2)$$

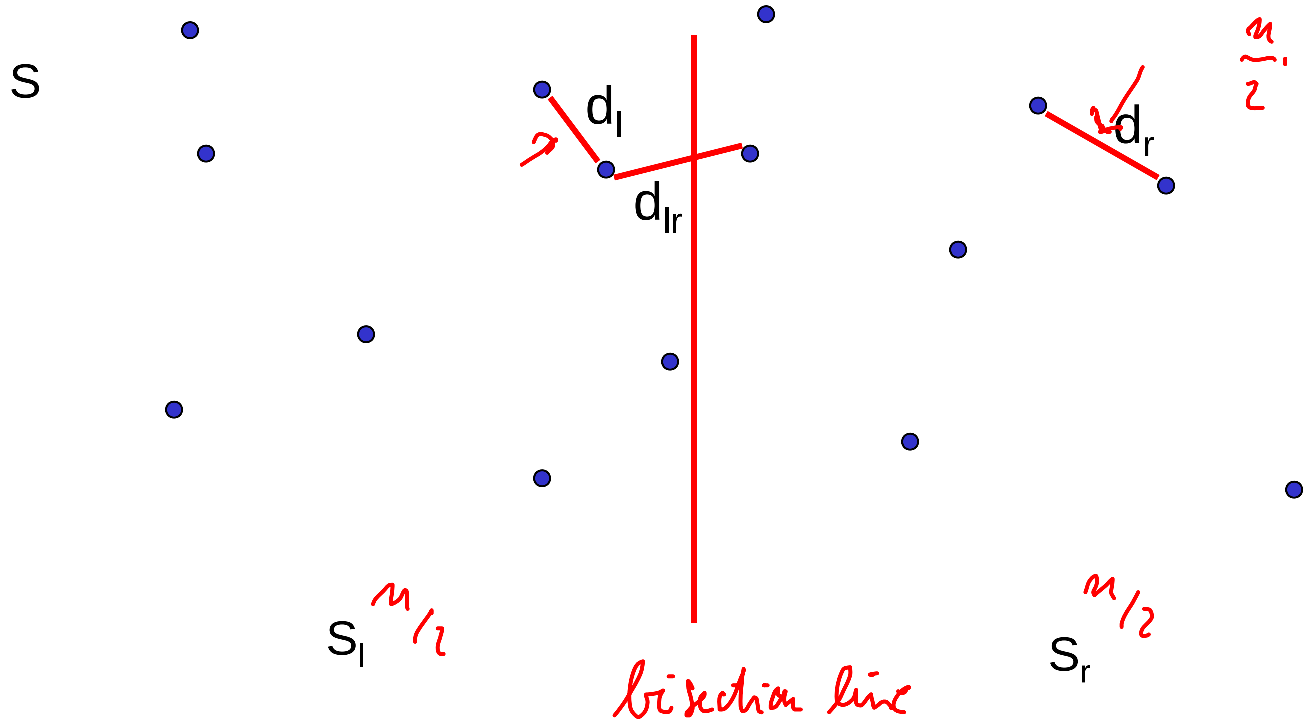
We show:
$$O(n \cdot \log n)$$

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}}

Naive approach of computing d_{lr}

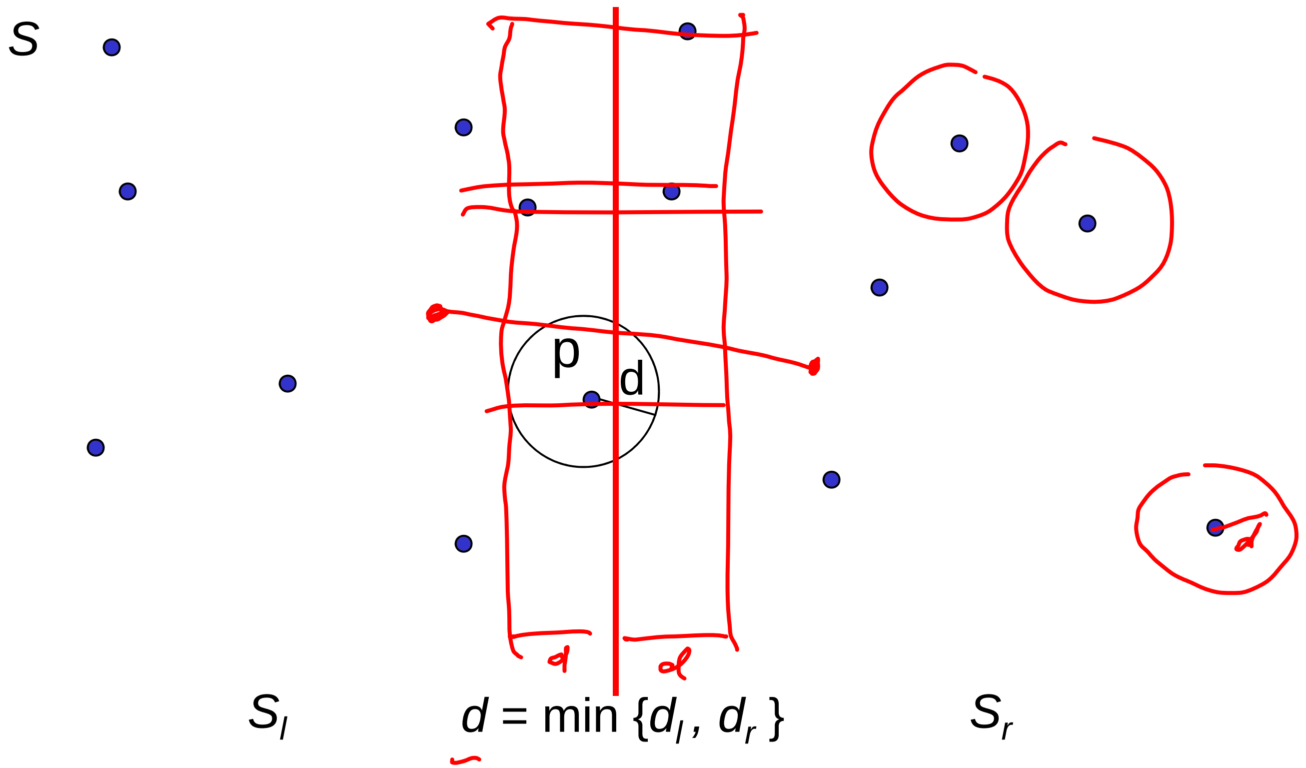
$\frac{n}{2} \cdot \frac{n}{2} = \frac{n^2}{4}$



Divide-and-conquer method

1. **Divide:** Divide S into two equal 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 $\min\{d_l, d_r, d_{lr}\}$

Computation of d_{lr} :

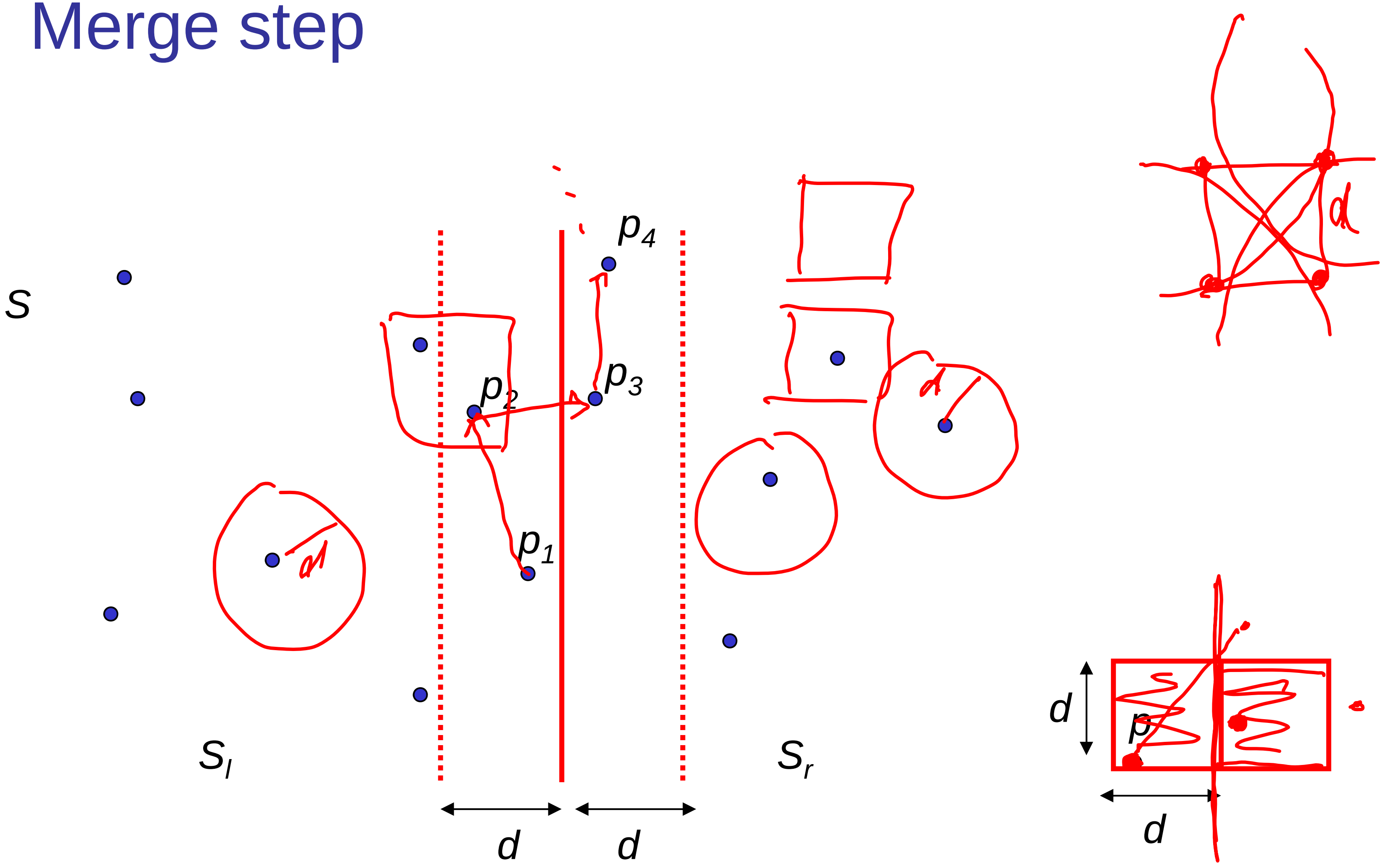


Merge step

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.

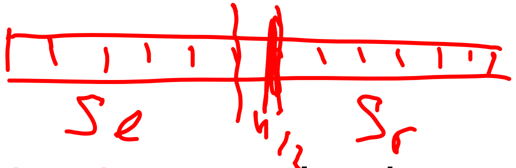
Merge step



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

Implementation

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



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

min_dist = min{d_l, d_r, d_{lr}}

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

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

$$T(n) = 2\,T(n/2) + an = 2\cdot\left(2\cdot T(n/4) + a\cdot\frac{n}{2}\right) + an$$

$$= 4\,T(n/4) + 2an$$

$$= 4\cdot\left(2\cdot T(n/8) + a\cdot n/4\right) + 2an = 8\,T(n/8) + 3an$$

$$= 8\cdot\left(2\cdot T(n/16) + a\cdot\frac{n}{8}\right) + 3an$$

$$= 16\cdot T(n/16) + 4an$$

$T(n) \leq a\cdot n \log_2 n$ Guess

Verify by induction

$T(n) \leq an \log n$

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

$n = 2^i$

$i = 1$: ok $n=2$ $T(2) = a \leq a \cdot 2 \cdot \log 2 = a \cdot 2$ ✓

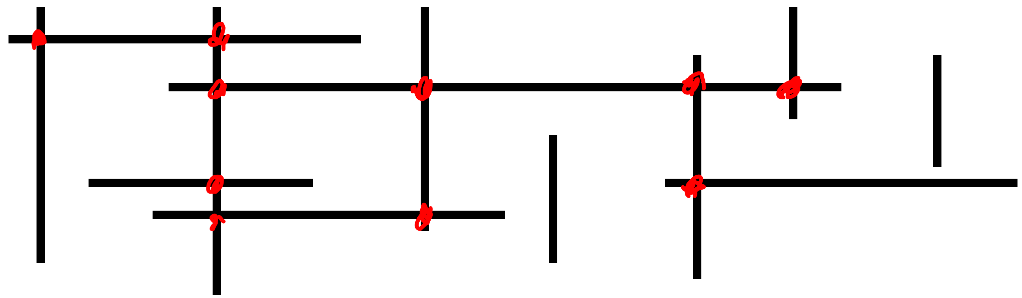
Assume the claim holds for ~~i~~ $i-1$. Show that it also holds i.

$i > 1$

$$\begin{aligned} T(2^i) &= 2 \cdot T(2^{i-1}) + a2^i \\ &\leq 2 \cdot a\, 2^{i-1} \cdot \log 2^{i-1} + a \cdot 2^i \\ &= 2^i \cdot a \cdot (i-1) + a \cdot 2^i \\ &= a \cdot 2^i (i-1+1) = a \cdot 2^i \cdot i \\ &= n \cdot a \cdot \log n \quad ✓ \end{aligned}$$

Line segment intersection

Find all pairs of intersecting line segments.



n segments

Check each line segment with each other

$O(n^2)$

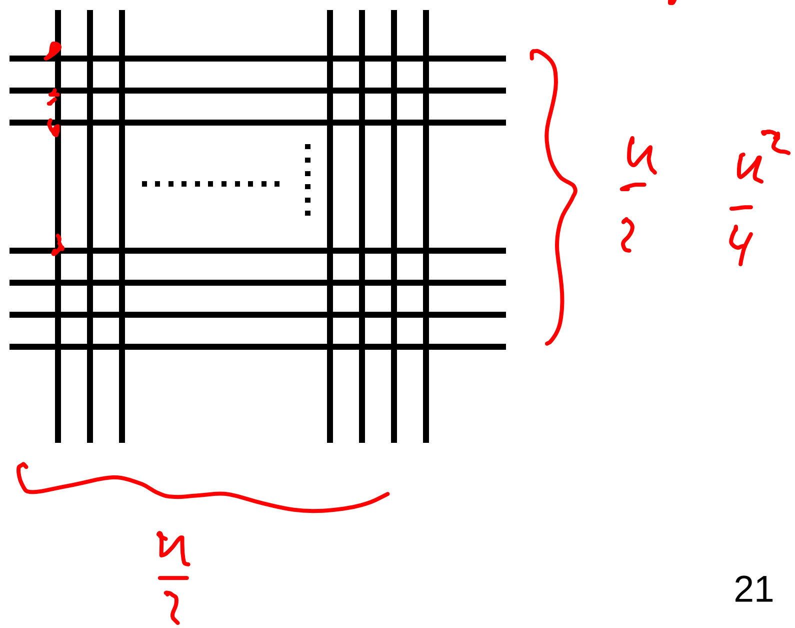
Running time?

$O(n \cdot \log n)$

Output sensitive algorithm

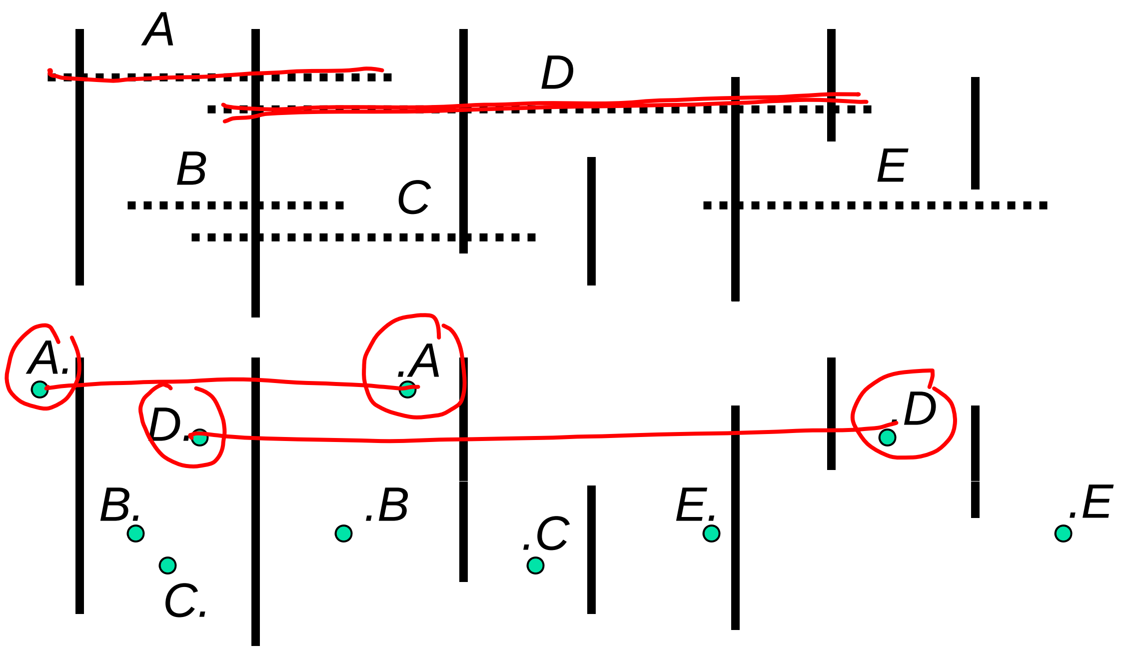
$O(n \log n + k)$

k: number of intersecting points



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

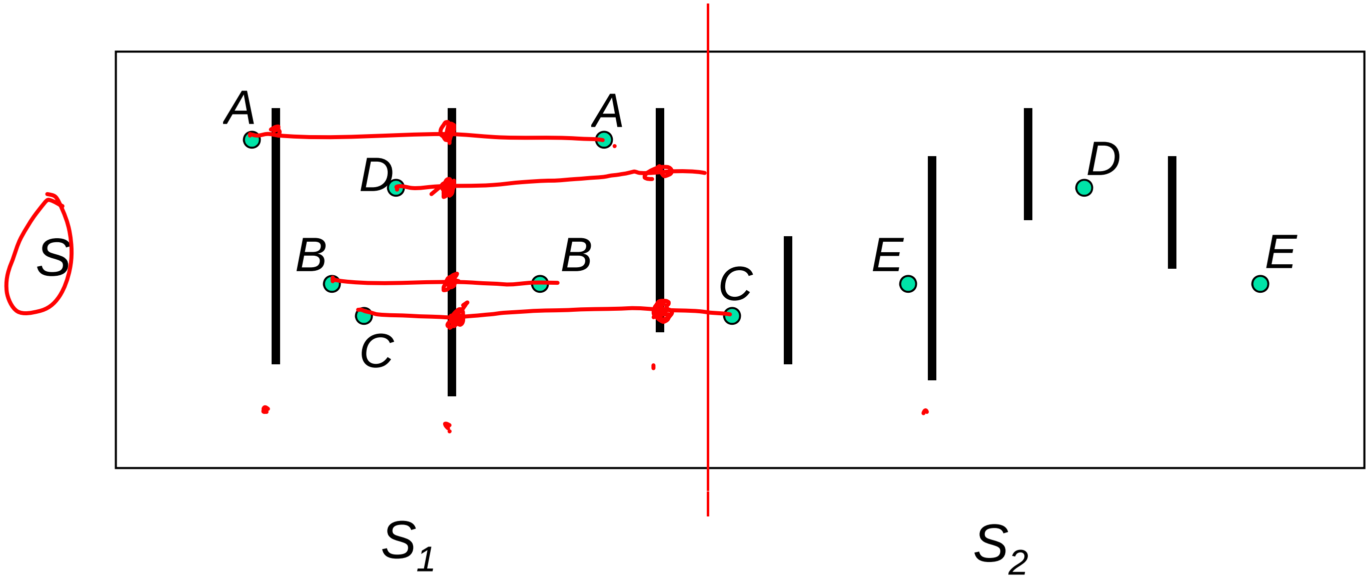
if $|S| > 1$

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:

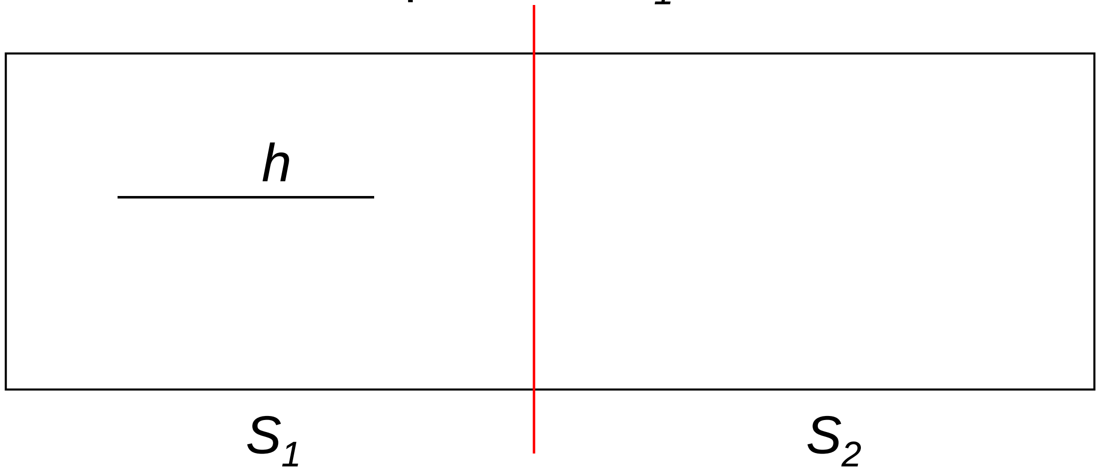
ReportCuts(S_1); ReportCuts(S_2)

ReportCuts

3. Merge: ???

Possible intersections of a horizontal line-segment *h* in S_1

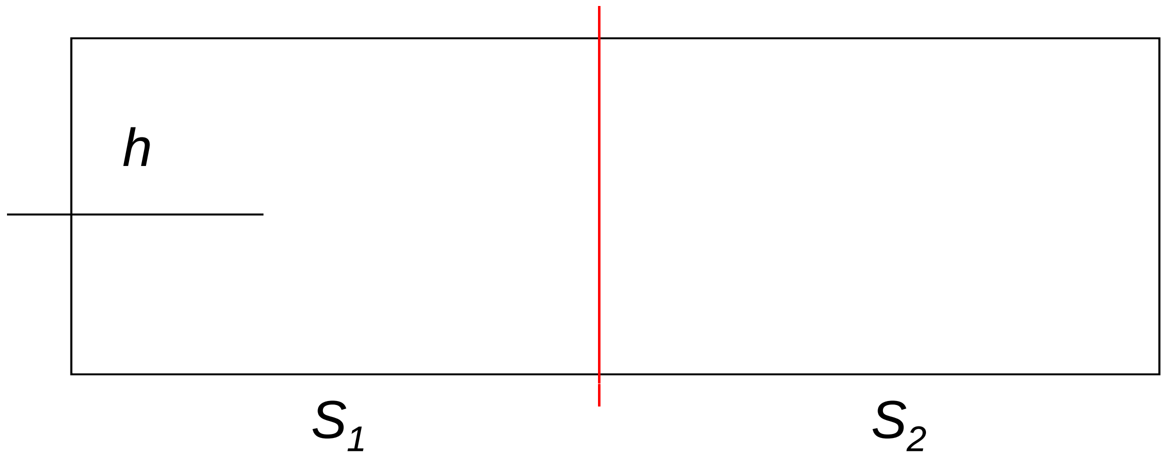
Case 1: both endpoints in S_1



ReportCuts

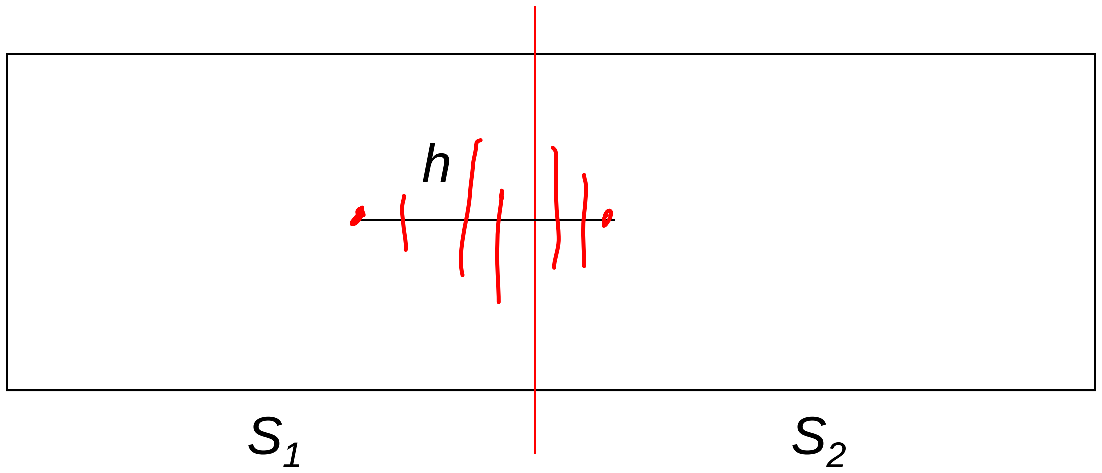
Case 2: only one endpoint of h in S_1

2 a) right endpoint in S_1

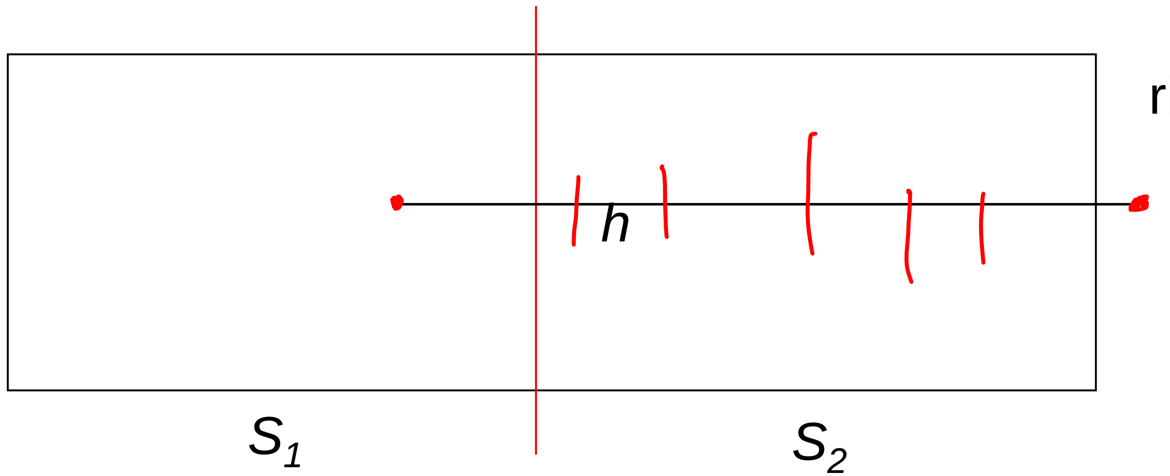


ReportCuts

2 b) left endpoint of h in S_1



right endpoint in S_2

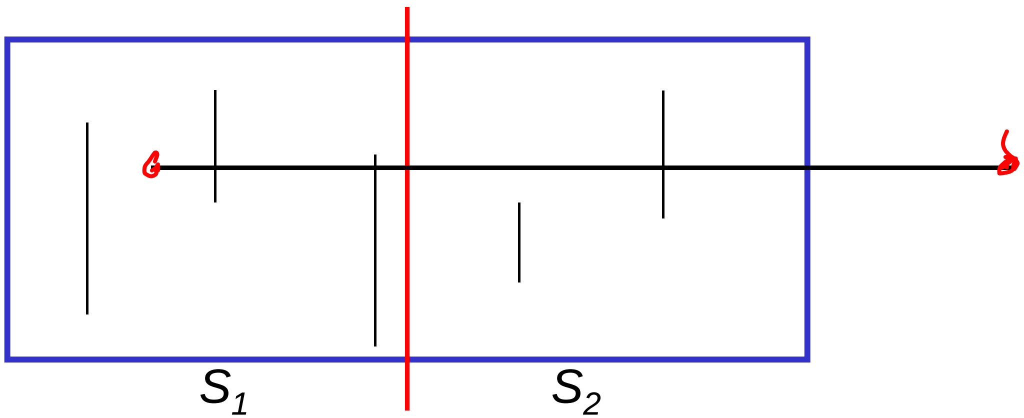


right endpoint not in S_2

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.



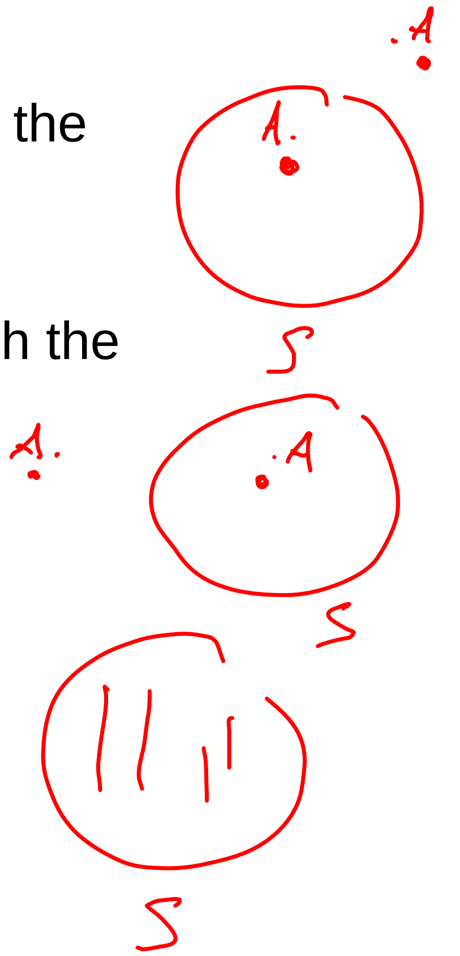
Implementation

Set S

L(S): y-coordinates of all left endpoints in *S*, for which the corresponding right endpoint is not in *S.*

R(S): y-coordinates of all right endpoints in *S*, for which the corresponding left endpoint is not in *S.*

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



Base cases

S contains only one element s.

Case 1: $s = (x,y)$ is a left endpoint

$L(S) = \{y\}$ $R(S) = \varnothing$ $V(S) = \varnothing$

Case 2: $s = (x,y)$ is a right endpoint

$L(S) = \varnothing$ $R(S) = \{y\}$ $V(S) = \varnothing$

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

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

Merge step

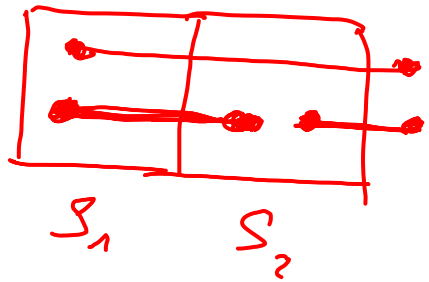
Assume that $L(S_i)$, $R(S_i)$, $V(S_i)$ are known for $i = 1,2$.

$S = S_1 \cup S_2$

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

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

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



L, R: ordered by increasing y-coordinates
linked lists

V: ordered by increasing lower endpoints
linked list