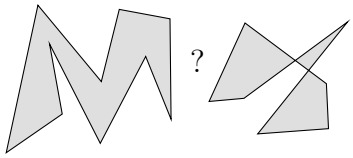
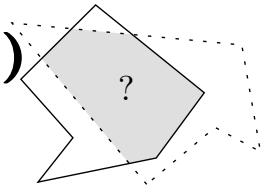
Problem 1

Does $P = (p_1, \ldots, p_n)$ *form a simple Polygon?*



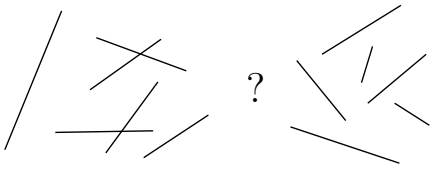
Problem 2 (Polygon Intersection)

Do two simple polygons intersect?



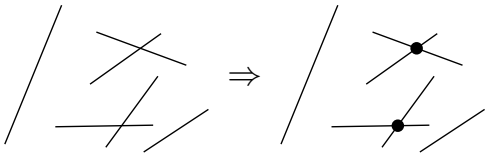
Problem 3 (Segment Intersection Test)

Are n *line segments pairwise disjoint?*



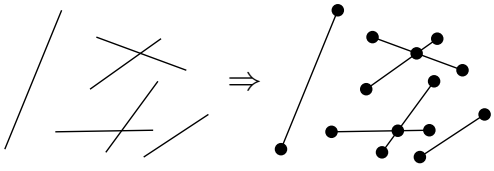
Problem 4 (Segment Intersection)

Given n *line segments, construct all intersections.*



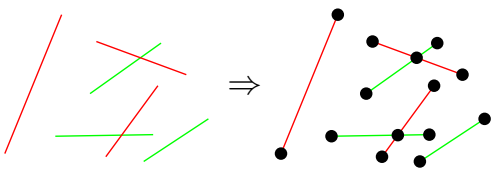
Problem 5 (Segment Arrangement)

Given n *line segments, construct their Arrangement.*



Problem 6 (Map Overlay)

Given sets S *and* T *of pairwise disjoint line segments, construct the Arrangement of* $S \cup T$.



Segment Intersection

Trivial Algorithm. Test all pairs.
$O(n^2)$ time and linear space.
Worst-case optimal. . .

In case of few intersections, we would like to have sub-quadratic time.

Lower bound $\Omega(n \log n)$ for **Element Uniqueness**: Given $x_1, \ldots, x_n \in \mathbb{R}$, are there $i \neq j$ such that $x_i = x_j$?

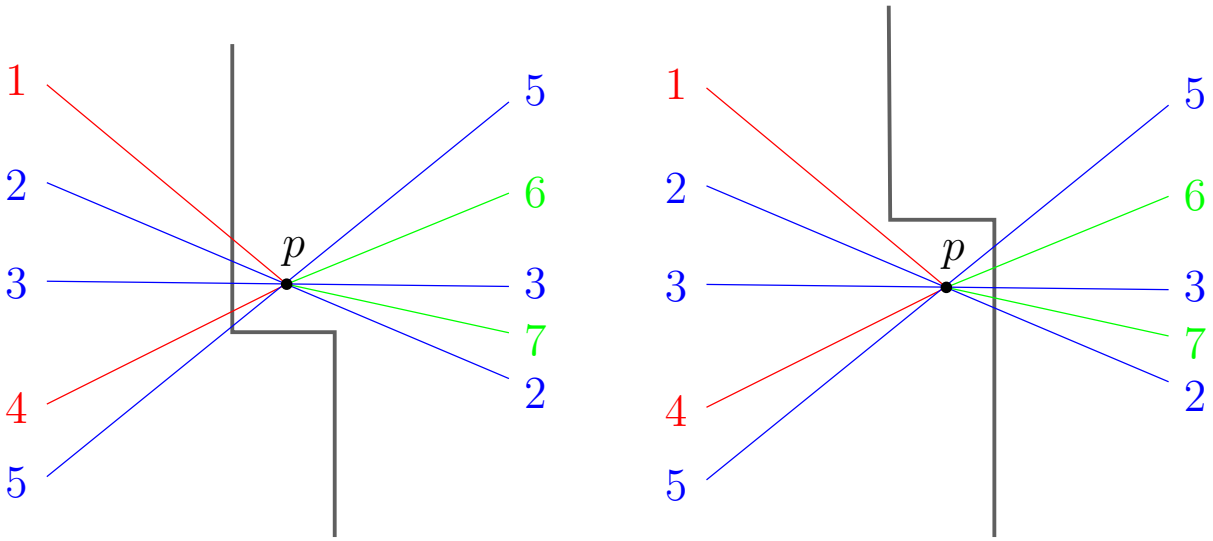
Problem 7 Given a set I of n intervals $[l_i, r_i] \subset \mathbb{R}$, $1 \leq i \leq n$, compute all intersecting pairs.

Theorem 1
Problem 7 can be solved in $O(n \log n + k)$ time and $O(n)$ space, where k is the number of intersecting pairs.

Line Sweep

Idea. Move a line ℓ (*sweep line*) from left to right, such that at any time all intersections to the left of ℓ are known.

We do not make any general position assumption here, that is, several segments can start, end, and/or intersect at the same point. The sweep line can be imagined as infinitesimally twisted.



Sweep line status (SLS). Sequence L of segments that intersect the current sweep line, sorted by y-coordinate.

Event point (EP). Point where SLS changes when moving ℓ (discretization).

Event point schedule (EPS). Sequence E of event points to be processed (not all known in advance), sorted lexicographically.

With every EP p we store

- a list end(p) of segments ending at p;
- a list begin(p) of segments that begin at p;
- a list int(p) of segments that intersect a neighboring (in SLS) segment at p.

With every segment we store pointers to the (≤ 2) entries in int$(\cdot)$ lists and a pointer to its appearance in L.

Invariants.

i) L is the sequence of segments from S which intersect ℓ, sorted by y-coordinate ($\leq$);

ii) E contains all event points (endpoints from segments in S and all points of intersection from segments adjacent in L) that are to the right of ℓ;

iii) All intersections between segments from S that are to the left of ℓ have been reported.

Event point handling. Consider an EP p.

1) If $\mathsf{end}(\mathrm{p}) \cup \mathsf{int}(\mathrm{p}) = \emptyset$, localize p in L.

2) Report all pairs of segments from $\mathsf{end}(\mathrm{p}) \cup \mathsf{begin}(\mathrm{p}) \cup \mathsf{int}(\mathrm{p})$ as intersecting.

3) Remove all segments in $\mathsf{end}(\mathrm{p})$ from L.

4) Reverse the subsequence in L that is formed by the segments from $\mathsf{int}(\mathrm{p})$.

5) Insert segments from $\mathsf{begin}(\mathrm{p})$ into L, sorted by slope.

6) Test the topmost and bottommost segment in SLS from $\mathsf{begin}(\mathrm{p}) \cup \mathsf{int}(\mathrm{p})$ for intersection with its successor and predecessor, respectively, and update EP if necessary.

Update of EPS. Insert an EP p for intersection of segments s and t.

1) If p does not yet appear in E, insert it.

2) If s or t are contained in some int$(\cdot)$ list of some other EP q, remove them there and possibly remove q from E (if end$(\mathrm{q}) \cup$ begin$(\mathrm{q}) \cup$ int$(\mathrm{q}) = \emptyset$).

3) Insert s and t into int(p).

Sweep.

1) Insert all segment endpoints into begin$(\cdot)$ and end$(\cdot)$ lists of a corresponding EP in E.

2) As long as $\mathrm{E} \neq \emptyset$, handle the first EP and then remove it from E.

Runtime Analysis

Initialization: $O(n \log n)$.

Handling of an EP p:

$$O(\#\text{intersecting pairs} + |\text{end}(p)| \log n + |\text{int}(p)| + |\text{begin}(p)| \log n + \log n).$$

Altogether:

$$O(k + n \log n + k \log n) = O((n + k) \log n).$$

Space. Clearly $|S| \leq n$. At begin $|E| \leq 2n$ and $|S| = 0$. Never more than $2|S|$ intersection EPs, therefore linear space overall.

Theorem 2 Problem 4 and Problem 5 can be solved in $O((n + k) \log n)$ time and $O(n)$ space.

Theorem 3 Problem 1, Problem 2 and Problem 3 can be solved in $O(n \log n)$ time and $O(n)$ space.

Improvements

The presented algorithm is due to Jon Bentley and Thomas Ottmann (1979).

Bernard Chazelle and Herbert Edelsbrunner (1988) gave an (rather involved) algorithm showing that $O(n \log n + k)$ time can be achieved in Theorem 2 using $O(n + k)$ space.

Kenneth Clarkson and Peter Shor (1989) and independently Ketan Mulmuley (1988) describe randomized algorithms with expected runtime $O(n \log n + k)$ using $O(n)$ and $O(n + k)$ space, respectively.

An optimal deterministic algorithm with runtime $O(n \log n + k)$ and linear space is due to Ivan Balaban (1995).

Degree of predicates

The degree of the polynomial underlying the predicate ($\approx$ precision needed to compute it).

High degree predicates are more expensive to compute exactly and lead to robustness problems when using low precision floating point computations.

Bentley-Ottmann sweep compares intersection points (degree five).

To construct the arrangement, orientation test is needed (degree three). Even to construct an intersection we need degree three.

What can we do with degree two predicates only?

- Compare endpoints lexicographically.
- Compare endpoint relative to segment.

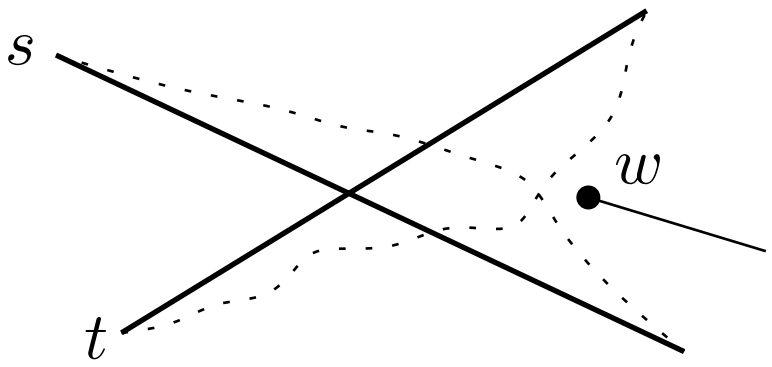
Red-Blue Intersections

Input segments form two interior-disjoint sets R and B.

Definition 4 A set S of line segments in $\mathbb{R}^{\mathrm{d}}$ is *interior-disjoint* $\iff$ $\nexists\, s, \mathrm{t} \in S$ for which $(s \setminus \mathrm{V}(s)) \cap \mathrm{t} \neq \emptyset$.

Lazy Computation. Defer handling of an intersection as long as possible, to the next segment endpoint. $\Rightarrow$ EPs known in advance.

Witness. The lexicographically smallest segment endpoint in the closed wedge bounded by two intersecting segments s and t and to the right of $s \cap \mathrm{t}$ is called the witness of $s \cap \mathrm{t}$.



Invariants

1. L is the sequence of segments from S intersecting ℓ; s appears before t in $\mathrm{L} \Longrightarrow s$ intersects ℓ above t or s intersects t and the witness of this intersection is to the right of ℓ.

2. All intersections of segments from S whose witness is to the left of ℓ, have been reported.

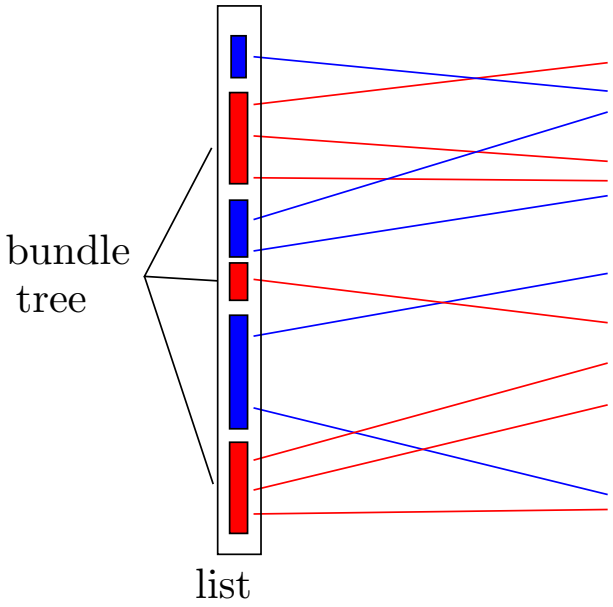
SLS Data Structure

The SLS structure consist of three levels.

1) Collect adjacent segments of the same color in *bundles*, stored as balanced search trees. For each bundle store pointers to the topmost and bottommost segment.

2) All bundles are stored in a doubly linked lists, sorted by y-coordinate.

3) All red bundles are stored in a balanced search tree (*bundle tree*).



Search tree structure should support insert, delete, split and merge in (amortized) logarithmic time each, e.g., splay trees.

Handling of an EP q

1) Localize p in bundle tree $\rightarrow \leq 2$ bundles containing p.

2) Localize p in ≤ 2 red bundles found and split them at p. All red bundles are now either above, ending, or below w.r.t. p.

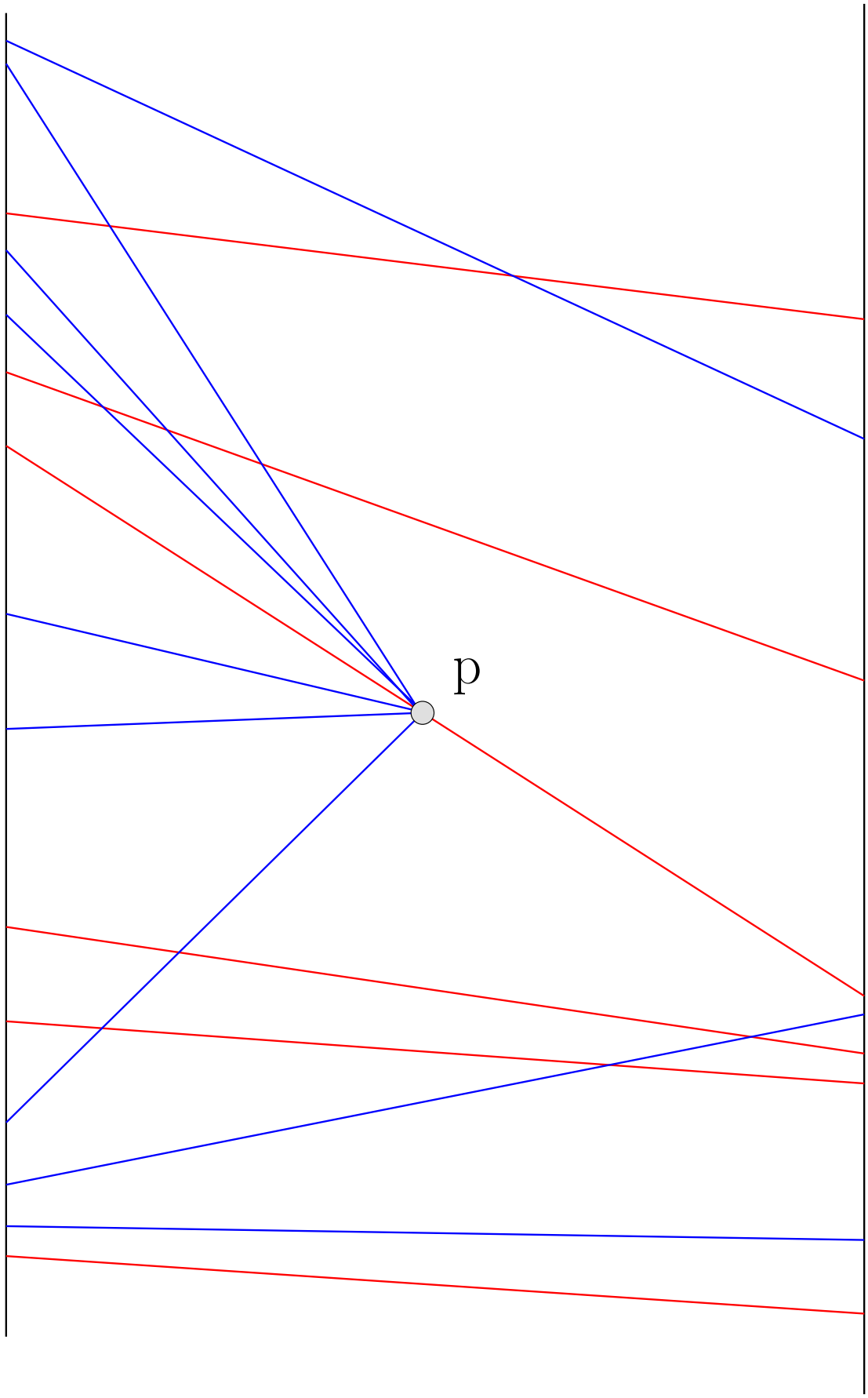
3) Localize p within the blue bundles by linear search.

4) Localize p in the ≤ 2 blue bundles found and split them at p. All bundles are now either above, ending, or below w.r.t. p.

5) Run through the list of bundles around p. Handle all adjacent pairs of bundles (A, B) that are in wrong order and report all pairs of segments as intersecting. (Exchange A and B in the bundle list and merge them with their new neighbors.)

6) Report all pairs from $\mathrm{begin}(\mathrm{p}) \times \mathrm{end}(\mathrm{p})$ as intersecting.

7) Remove ending bundles and insert starting segments, sorted by slope and bundled by color.


p

Analysis

Sorting EPs: $O(n \log n)$.

Every EP generates a constant number of tree searches and splits of $O(\log n)$ each.

Every exchange in Step 5 generates at least one intersection.

New bundles are created only by inserting a new segment or by a split. Thus $O(n)$ bundles are created and the number of merge operations is $O(n)$.

The linear search in Step 5 (beyond the ending bundles) can be charged to the subsequent merge operation.

Overall runtime is $O(n \log n + k)$ and space is linear.

Remark: If a segment should pass through an EP (at most one per color can), split it there but do not report an intersection.

Theorem 5 For two sets R and B of interior-disjoint line segments in $\mathbb{R}^2$ one can find all intersecting pairs of segments in $O(n \log n + k)$ time and linear space, using predicates of maximum degree two. Here $n = |R| + |B|$ and k is the number of intersecting pairs.

Remarks. The first optimal algorithm to construct red-blue intersections was given by Harry Mairson and Jorge Stolfi in 1988.

In 1994 Timothy Chan devised a trapezoid-sweep algorithm that used predicates of degree three only.

The version discussed above is due to Andrea Mantler and Jack Snoeyink (2000).

Axis-parallel rectangles

Efficiently represented by $2d$ coordinates. Often used as a preprocessing tool when computing intersections (bounding-box).

Problem 8 Given a set $\mathbb{R}$ of n axis-parallel rectangles in $\mathbb{R}^2$, compute all intersecting pairs.

Different from line segment intersection...

Observation 6 Two axis-parallel rectangles intersect $\Longleftrightarrow$ the left side of one intersects the other.

Leads to stabbing queries:

Problem 9
Given a set I of n intervals $[l_i, r_i] \subset \mathbb{R}$, $1 \leq i \leq n$. Build a data structure to report for a given query value $p \in \mathbb{R}$ all intervals from I containing p.

Segment trees

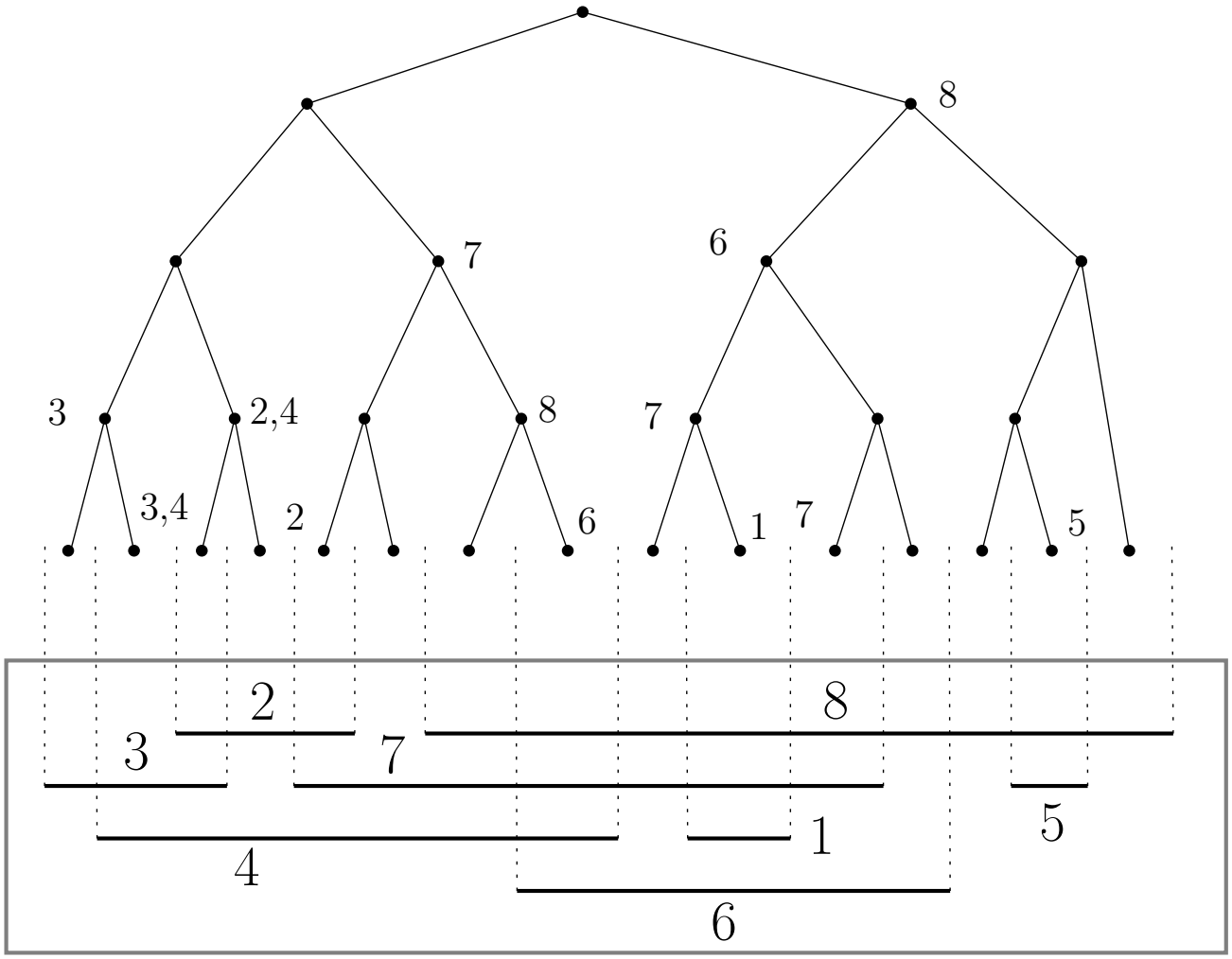
Idea: locus approach, interval endpoints divide real line into regions with identical answers (elementary intervals).

Definition 7 (Bentley 1977)
A segment tree for a set I of n real intervals is a balanced binary search tree T on the $\leq 2n+1$ elementary intervals.

To every node v of T an interval $A(v)$ is associated that is the union of all elementary intervals in the subtree with root v.

An interval $i \in I$ is stored in every node v of T for which $A(v) \subset i$ but not $A(p(v)) \subset i$, where $p(v)$ is the parent node of v.

Lemma 8
An interval is stored at $\leq 2\lceil \log 2n + 1 \rceil + 2$ *nodes of a segment tree.*


8
7
6
3
2,4
8
7
3,4
2
6
1
7
5
2
8
3
7
1
5
4
6

Theorem 9 For a set I of n real intervals one can in $O(n \log n)$ time and space build a data structure that allows to report for a given query value $p \in \mathbb{R}$ all intervals from I containing p in $O(\log n + k)$ time, where k is the size of the output. Insertion and deletion of an interval can be done in $O(\log n)$ time.

Theorem 10 For a set R of n axis-parallel rectangles in $\mathbb{R}^2$ one can report all intersecting pairs in $O(n \log n + k)$ time and space, where k is the size of the output.

Rectangle Intersection

Data structure:

1) Store the y-intervals of all rectangles intersecting the sweep line in a segment tree.

2) Stored the y-intervals of all rectangles intersecting the sweep line in a threaded balanced search tree, sorted by their smaller coordinate.

Algorithm: When a left lower corner of some rectangle r is reached, do

1) Determine all rectangles in SLS that intersect the lower left corner of r and report intersections. (Query in segment tree)

2) Localize the lower left corner of r in the search tree and traverse it linearly, reporting intersections, until the lower side of the current rectangle is above r.

3) Insert r into the SLS structures.