Chapter 3

Point Location

Keywords: computational geometry, higher dimensional searching, point location, fractional cascading, duality transform, line arrangements, randomized incremental construction (RIC), locus approach.

Searching is by no means limited to finding a query key in a set S of keys. In fact, more often than not, the domain of keys is partitioned into intervals, and the task is to find the interval which contains the query key. If the query domain is $\mathbf{R}^d$ for some $d \in \mathbf{N}$, we are often given a partition of $\mathbf{R}^d$ and we are asked to locate a query point $q \in \mathbf{R}^d$ in the partition—we are amidst the realm of *point location*. Mostly it is assumed that the partition is fixed (or subject to minor changes[1]) while there are many queries. Therefore, it pays to *preprocess* the partition in preparation of fast queries.

We will see that even higher dimensional point location problems can be resolved via 1-dimensional point location, often several of them, and sometimes interleaved in an intricate way. An alternative approach is to generate the partition in a random fashion which produces a query structure on the side, as we will see—we can view this as the higher dimensional counterparts of random search trees.

The Locus Approach. What, for example, if we are given a nonempty set S of real numbers and for a given query number q we want to find the closest number in S, i.e. the number $a \in S$ that minimizes $|a - q|$? A moment of reflection shows that there is a slight problem of ambiguity, namely it may

[1]In this chapter we concentrate on the *static* setting where we assume the partition is fixed—not that the alternative *dynamic* setting is less important, by no means!

be that this a is not unique. Take, for example, $S := \{1, 2, 4\}$ and $q = 1.5$: Both, 1 and 2 minimize the distance to 1.5.

Indeed, a set of $n \in \mathbf{N}$ real numbers

$$a_0 < a_1 < \ldots < a_{n-1}$$

partitions $\mathbf{R}$ into $n-1$ points and n open intervals two of which extend to infinity:

$$(-\infty, \frac{a_0+a_1}{2}),\ \frac{a_0+a_1}{2},\ (\frac{a_0+a_1}{2}, \frac{a_1+a_2}{2}),\ \frac{a_1+a_2}{2},\ \ldots,\ \frac{a_{n-2}+a_{n-1}}{2},\ (\frac{a_{n-2}+a_{n-1}}{2}, +\infty)$$

In each interval there is a unique closest key in S, while the points are exactly those query numbers where the closest key in S is not unique.

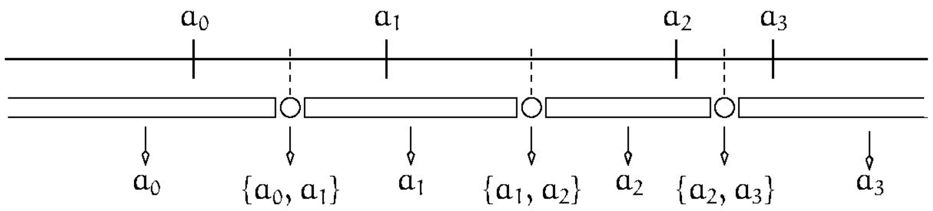

Figure 3.1: The numbers a_0, a_1, a_2, and a_3 partition $\mathbf{R}$ into regions of equal answers to "closest number queries."

The scenario derived here is quite typical, called *locus approach.* The domain is partitioned into regions of equal answers, and a query point is located among those regions which entails the answer to the query. In our example an optimal $O(n)$ space, $O(n \log n)$ preprocessing time and $O(\log n)$ query time solution evolves readily.

Exercise 3.1. Two Closest Numbers

Suppose you are given a finite set $S \subseteq \mathbf{R}$, $2 \le |S|$, *which is to be preprocessed so that for query* $q \in \mathbf{R}$ *the answer is 'the' set* $\{b_1, b_2\} \subseteq S$ *of the two closest numbers in* S *(i.e.* $\max\{|b_1-q|, |b_2-q|\} \le \min_{a \in S \setminus \{b_1, b_2\}} |a-q|$*). Follow the locus approach for the problem and describe the resulting partition of regions of equal answers (and be aware of the ambiguity issue, i.e. the 'the' has to be taken with caution).*

3.1 Point/Line Relative to a Convex Polygon

The convex hull[2] $\mathrm{conv}(P)$ of a finite set P of points in the plane is a bounded convex set. Unless $\mathrm{conv}(P)$ degenerates to the empty set, a single point, or a line segment, it is bounded by a *convex polygon*, a closed simple piecewise linear curve that separates the interior of $\mathrm{conv}(P)$ from the exterior. This polygon can be finitely described by a sequence of its vertices, in counterclockwise order, say.

This section deals with two geometric problems concerning convex polygons that turn out to be 1-dimensional queries in disguise. They prepare for the more involved structures to come.

Inside/On/Outside a Convex Polygon. A convex polygon C can be easily preprocessed for deciding whether a query point lies inside, on, or outside of C: We sort the x-coordinates of the vertices of C and prepare them for binary search (in a linear array). The intervals between two consecutive x-coordinates are associated with the two lines that carry the edges of C in the corresponding x-range of the plane. For a query point q, we first locate its x-coordinate x_q in this structure. If x_q is smaller than the smallest x-coordinate of the vertices, q is clearly outside C; same if larger than the largest x-coordinate. Otherwise, x_q lies in some interval and we can compare q with the two associated lines to decide about the answer to the query (we leave the issue, whether the intervals are open, closed, halfopen to the reader). Summing up, the structure needs $O(n)$ space and $O(\log n)$ query time. If the vertices of C are provided in sorted order along C, the structure can be built in linear time.

A Line Hitting a Convex Polygon. The question whether a query line ℓ intersects a given convex polygon C is slightly more interesting. We observe that ℓ intersects C iff it lies between (or coincides with one of) the two tangents to C parallel to ℓ. Such a tangent (parallel to ℓ) is determined by a vertex of C on t (every tangent contains one or two vertices of C).

[2] We assume some familiarity with convexity: Recall that the convex hull of a set $P \subseteq \mathbb{R}^d$ is defined as the set $\{\sum_{p\in P}\lambda_p p \mid \lambda_p \in \mathbb{R}_0^+, \sum_{p\in P}\lambda_p = 1\}$ of its convex combinations. Actually, for our purposes (P finite in the plane), an intuitive picture of a rubber band contracting around nails representing the points in P should suffice.

We prepare for the queries as follows. We direct every edge of C in the direction as we pass it moving around C in counterclockwise order. Every such directed edge e has an angle α_e in the range $[0, 2\pi)$ with the x-axis. Note that if e and the next edge e' (in counterclockwise order) share vertex v, then all directed tangents which have C to their left and have an angle in the range from α_e to $\alpha_{e'}$ touch C in vertex v (some care is needed if e is the edge of largest angle).

The angles of the edges are stored in sorted order $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ in an array. For $i \in [n-1]$, the vertex incident to the edges of angles α_{i-1} and α_i, is associated with the interval $[\alpha_{i-1}, \alpha_i)$; the vertex incident to edges of angles α_0 and α_{n-1} is associated with both $[0, \alpha_0)$ and $[\alpha_{n-1}, 2\pi)$.

Given a query line ℓ, we get two angles β and β'—depending on which direction we assign to ℓ—which we locate in our array. The vertices v and v' associated to the found intervals determine the tangents parallel to ℓ. ℓ misses C iff both v and v' lie on the same side of ℓ.

Again, we have an optimal[3] structure with linear storage and preprocessing time, and with logarithmic query time.

Exercise 3.2. Locally vs. Globally Convex

We are given a sequence of points $p_0, p_1, \ldots, p_{n-1}$ in the plane, such that for all $i \in \{0..n-1\}$, the sequence p_i, p_{i+1}, p_{i+2} are vertices of a triangle in counterclockwise order (indices are understood modulo n). Is this necessarily the sequence of vertices (in counterclockwise order) of a convex polygon?

Exercise 3.3. Convex Hull of Two Convex Polygons

Two convex polygons are given by their respective lists of vertices in counterclockwise order. Show that the convex hull of these two polygons can be computed in $O(n+m)$ time, where n and m are the number of vertices of the polygons.

HINT: Imagine two parallel directed tangents, one for each polygon which has the polygon to its left, wrapping around the polygons. The respective "outer" tangent is the tangent of the convex hull of both polygons.

Exercise 3.4. Finding a Key vs. Line Hitting Convex Polygon

Given a sorted sequence $a_0 < a_1 < \ldots < a_{n-1}$ of n real numbers, we consider the convex polygon C with vertices $\left((a_i, a_i^2)\right)_{i=0}^{n-1}$. For $k \in \mathbf{R}$,

[3] See Exercise 3.4 in support of this claim of optimality.

show that the line with equation $y = 2kx - k^2$ *intersects* C *iff* $k \in \{a_0, a_1, \ldots, a_{n-1}\}$.

REMARK: This exercise is supposed to exhibit that deciding whether a line intersects a convex polygon cannot be easier than deciding whether a query key is in a given set of keys.

3.2 Line Relative to Point Set

Suppose we are given a set P of points and we want to decide whether a query line has all of the points on one side. To this end we can compute the convex hull of P, or rather the convex polygon C bounding $\mathrm{conv}(P)$, and then decide whether the line intersects C; thus, we are back in known territory. Computing the convex hull of n points can be done in $O(n \log n)$ time[4].

Reporting the Points Below a Query Line. Next we want to preprocess P such that the points in P below a non-vertical query line can be reported quickly, each point exactly once[5]. Note that in an optimal solution, we definitely have to allow time $\Omega(k)$, k the number of points reported, and we have to allow time $\Omega(\log n)$, $n := |P|$, since we decide whether the convex hull of P is intersected. Hence, a structure with $O(k+\log n)$ query time is optimal[6].

If P is the vertex set of a convex polygon C, a structure with optimal query time $O(k + \log n)$ is relatively easy to derive given previously established knowledge: In $O(\log n)$ time we find a vertex p of C which is contained in the tangent which has C above and is parallel to the query line ℓ. If p is above or on ℓ we are done, since no point in P can be below ℓ. Otherwise, starting from p we first move in clockwise order through the vertices of C and report them (as 'below ℓ') until we either have exhausted

[4]You may remember this from a previous course (*Informatik 2*). Otherwise, you will have to accept this as a fact of life or discover it yourself—it's not hopeless.

[5]If you are in need or desire of motivation or illustration: A medical doctor has a database which stores each of her patients with their respective heights h and weights w; the pairs (h, w) are points in the plane. She wants to query all of her patients with above-'ideal' weight. Ideal weight is defined by a linear function $w = \lambda h + \mu$ with λ and μ chosen according to fashion trends and most recent findings about the ideal human body (oh yes, separate structures for female and male patients are needed). As λ and μ change with intention and over time, she has to be ready for ever changing queries ...

[6]Note that $\max\{k, \log n\} = \Theta(k + \log n)$, since for $a, b \in \mathbb{R}_0^+$, $\frac{a+b}{2} \leq \max\{a, b\} \leq a + b$.

all vertices or we meet the first vertex not below ℓ. In the latter case, we have to repeat the same procedure in counterclockwise order. Assuming that the vertices of C are represented in a doubly linked list, the reporting part (after discovery of p) can be performed in time $O(k)$. Altogether $O(n)$ storage and $O(k + \log n)$ query time suffices.

In order to handle general sets P, we first define the *onion* of P as the sequence

$$(R_0, V_0), (R_1, V_1), \ldots, (R_t, V_t)$$

obtained as follows. If P is empty, the sequence is empty. Otherwise, let $R_0 := \mathrm{conv}(P)$ and let V_0 be the vertex set of R_0. $(R_1, V_1), (R_2, V_2), \ldots, (R_t, V_t)$ is recursively defined as the onion of $P \setminus V_0$. We observe that

- $R_0 \supset R_1 \supset \ldots \supset R_t$; hence, if R_i is completely above or on a line ℓ, then all R_j, $j \in \{i..t\}$ are above or on the line ℓ, and so are all V_j, $j \in \{i..t\}$.
- $(V_0, V_1, \ldots, V_t)$ is a partition of P into nonempty sets.

Each V_i is either the vertex set of a convex polygon or $|V_i| \leq 2$. In either case we can preprocess V_i so that the points in V_i below a query line can be reported in time $O(k_i + \log n_i)$, k_i the number of points reported and $n_i = |V_i|$. Having established the structures for all $i \in \{0..t\}$ we start a query with a line ℓ in the structure for V_0, report all points in V_0 below ℓ, then proceed to the structure for V_1, etc. until either (i) we have reached a V_j that is completely above or on ℓ or (ii) we get to the situation that we have exhausted all sets (i.e. have reached V_t). In case (i), this takes time

$$O(\sum_{i=0}^{j-1} k_i + \sum_{i=0}^{j} \log n_i) = O(k + (j+1)\log n) = O((k+1)\log n) \ ,$$

for $k := \sum_{i=0}^{j-1} k_i$, the number of points in P below ℓ. We use here $j \leq k$: Since each k_i, $i \in \{0..j-1\}$, is at least 1 we have

$$k = \sum_{i=0}^{j-1} k_i \geq \sum_{i=0}^{j-1} 1 = j \ .$$

The case (ii) takes time $O(\sum_{i=0}^{t} k_i + \sum_{i=0}^{t} \log n_i)$ which is bounded by $O((k+1)\log n)$ as well.

The $O((k+1)\log n)$ bound we proved falls short of the bound $O(k+\log n)$. For example, if $k=\lceil \log n \rceil$ our bound reads $O((\log n)^2)$ instaed[7] of $O(\log n)$.

Searching in Many Lists. For an improvement of the structure we have a closer look at where we lose time. Whenever we handle a new convex polygon V_i in the structure, we locate a real number β (the angle of the query line ℓ) in a linear array storing a set of numbers, call it S_i. And we throw away all the information from V_i when we locate the same number in the next array for V_{i+1}. We could take one array for all angles occurring in any of the structures for the V_i's, and then have for each interval $t+1$ pointers to the respective intervals in the structures. Since there are roughly n intervals in the overall array, this yields roughly $t \cdot n$ pointers and thus spoils our linear bound on the storage.

For a better solution we first enhance the sets S_i with extra numbers resulting in new sets $\overline{S}_i$. The 'last' set S_t is left untouched, so $\overline{S}_t := S_t$. For $i = t-1, t-2, \ldots, 0$, to obtain $\overline{S}_i$ we add to S_i every other number of $\overline{S}_{i+1}$ in its sorted order starting with the second number. Hence $|\overline{S}_i| \leq |S_i| + \frac{|\overline{S}_{i+1}|}{2}$. Each of the $\overline{S}_i$'s is again stored in a linear array. For $0 \leq i \leq t-1$, each interval I of $\overline{S}_i$ is either completely contained in an interval I' of $\overline{S}_{i+1}$ or it contains exactly one number $a \in \overline{S}_{i+1}$ in its interior. In the former case we add a pointer from I to I', in the latter a pointer from I to a; comparison of the query number β with a determines the interval of $\overline{S}_{i+1}$ containing β. Hence, in this enhanced structure, we can locate β in constant time in $\overline{S}_{i+1}$, given we have it already located in $\overline{S}_i$. Summing up, if $\overline{S}_j$ is the last structure we have to search in (i.e. $j=t$ or no point in V_j is below query line ℓ), the overall query time went down to

$$O(k + \underbrace{\log n}_{\text{location in the first array}} + j) = O(k + \log n)$$

which is optimal.

What is the overall size of the structure? It is easy to see that it is bounded by $O(\sum_{i=0}^{t} \overline{n}_i)$, $\overline{n}_i := |\overline{S}_i|$.

Lemma 3.1. $\sum_{i=0}^{t} \overline{n}_i \leq 2n$.

[7] I recently read a short article cliaming that for us to be able to read and absorb wodrs, all that matters is that the frist and last letter of a word are in palce, othwreise any perumttaoin will do.

Proof. Set $N_j := \sum_{i=j}^{t} n_i$ and $\overline{N}_j := \sum_{i=j}^{t} \overline{n}_i$. We prove $\overline{N}_j + \overline{n}_j \leq 2N_j$. Since $\overline{N}_0 = \sum_{i=0}^{t} \overline{n}_i$ and $N_0 = n$, this yields the assertion of the lemma. We proceed by induction on j, for $j = t$ down to $j = 0$. The induction basis is given by $\overline{N}_t + \overline{n}_t = 2n_t = 2N_t$. For the induction step, let $j < t$. We use $\overline{n}_j \leq n_j + \frac{\overline{n}_{j+1}}{2}$ and induction hypothesis for

$$\overline{N}_j + \overline{n}_j = \overline{N}_{j+1} + 2\overline{n}_j \leq \overline{N}_{j+1} + 2n_j + \overline{n}_{j+1} \leq 2N_{j+1} + 2n_j = 2N_j$$

For an alternative proof[8] we sum up the inequalities $\overline{n}_j \leq n_j + \frac{\overline{n}_{j+1}}{2}$, $j \in \{0..t-1\}$ and $\overline{n}_t = n_t$ to obtain $\overline{N}_0 \leq N_0 + \frac{\overline{N}_0 - \overline{n}_0}{2}$. Since $\overline{n}_0 \geq 0$, we have $\overline{N}_0 \leq N_0 + \frac{\overline{N}_0}{2}$ which readily gives the claimed inequality (recall $N_0 = n$).

This proof is not only shorter but also develops in a more natural fashion. The first version may still be convenient when we extend to tree structures (instead of the linear structure here) at a later point. $\boxed{}$

So linear storage and optimal query time is established. For preprocessing, we have to compute at most n convex hulls of at most n points each for construction of the onion. This can be done in $O(n^2 \log n)$. Better solutions with time $O(n \log n)$ for onion construction are known but will not be treated here.

Theorem 3.2. *A set P of n points in the plane can be preprocessed in time $O(n^2 \log n)$ and linear storage so that the set of points below a query line can be reported in time $O(k + \log n)$, k the number of points below the query line.*

The method of cascading some of the elements of one list to another list is called *fractional cascading*[9] which has numerous applications.

What if we want to count the number of points below a query line? Clearly, the above solution can be employed, but we cannot really justify spending time k for delivering a number 'k'. Ideally, we would expect here an answer in time $O(\log n)$. Towards such an achievement, we introduce a simple but fertile concept from geometry.

Duality. A point in the plane is given by a pair $(a, b) \in \mathbf{R}^2$. A non-vertical line in the plane can be described by an equation $y = ax + b$, so it's again

[8] Provided by Philipp Zumstein who attended the course in summer 2004.
[9] Fractional cascading: *teilweises Herunterstürzen.*

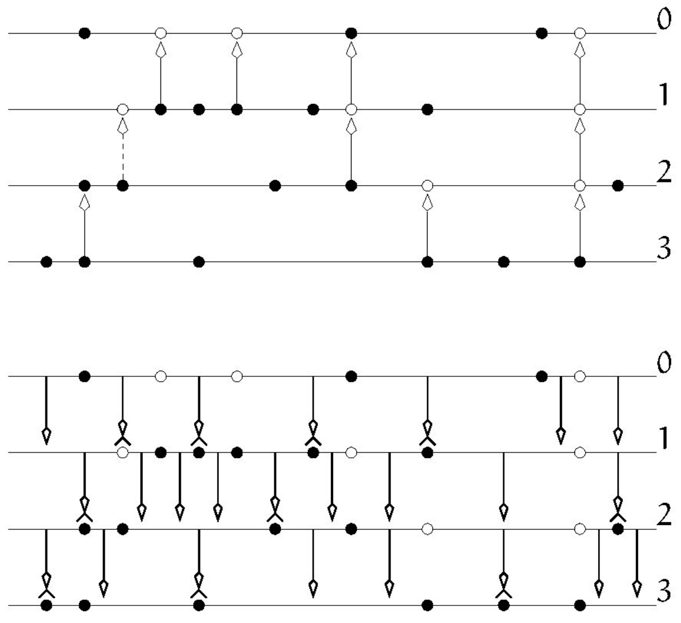

Figure 3.2: Fractional cascading: In the upper picture, black bullets are the original entries in the lists, while the white bullets are inherited from the lists of higher index. In the lower picture we see the pointers added in the structure. They point from an interval either to a containing interval (in the next list) or to a point in its interior (in the next list).

specified by a pair $(a, b) \in \mathbf{R}^2$. So lines and points in the plane are just different interpretations of the same 'thing', namely $\mathbf{R}^2$. Duality exploits this fact—it maps points to lines and lines to points with interesting implications.

Let * be the mapping that maps points to non-vertical lines, and non-vertical lines to points by

$$\begin{aligned} \text{point } p = (a, b) &\mapsto \text{ line } p^* : \; y = ax - b \\ \text{line } \ell : \; y = ax + b &\mapsto \text{ point } \ell^* = (a, -b) \end{aligned}$$

Note that point $p = (a, b)$ lies on line $\ell : \; y = cx + d$ iff

$$b = ca + d \iff -d = ac - b$$

and thus iff point $\ell^* = (c, -d)$ lies on line $p^* : \; y = ax - b$. We say that the duality * preserves *incidences*. It also preserves relative position, which will be useful for our purposes.

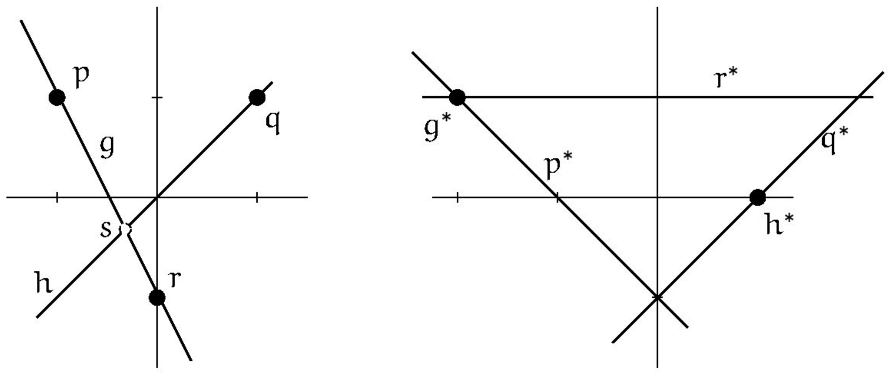

Figure 3.3: Three points and two lines and their duals. What is the dual of the intersection s of lines h and g?

Observation 3.3. *Let p be a point and ℓ a non-vertical line in $\mathbb{R}^2$. Then (i) $(p^*)^* = p$ and $(\ell^*)^* = \ell$, (ii) $p \in \ell$ iff $\ell^* \in p^*$, and (iii) p lies above ℓ iff ℓ^* lies above p^*.*

Exercise 3.5. Lines Intersecting Segment
What is the shape of the set of duals of all non-vertical lines that intersect a given line segment s?

Exercise 3.6. The Dual of a Parabola
What is the set of duals of all tangents to the parabola $y = x^2$?

Exercise 3.7. Points and Intervals to Points and Lines
A real number $c \in \mathbb{R}$ is mapped to point $\hat{c} := (c, c^2) \in \mathbb{R}^2$ and an interval $I = (a, b) \subseteq \mathbb{R}$, $a \leq b$, is mapped to the line $\hat{I}: \ y = (a+b)x - ab$. Show that $c \in I$ iff $\hat{c}$ below $\hat{I}$.

Exercise 3.8. Alternative Duality
Consider the following duality $^\circ$ between points in the plane and lines disjoint from the origin $(0, 0)$.

$$\begin{aligned} \textit{point } p = (a, b) \quad &\mapsto \quad \textit{line } p^\circ: \ ax + by = 1 \\ \textit{line } \ell: \ ax + by = 1 \quad &\mapsto \quad \textit{point } \ell^\circ = (a, b) \end{aligned}$$

What is the shape of the set of duals of all lines intersecting a given convex polygon C with $(0, 0)$ inside C?

Counting the Points Below a Query Line. Given a set P of n points in $\mathbb{R}^2$ and a query line ℓ we observed that the points $p \in P$ below ℓ are exactly those for which ℓ^* is below p^*. In other words, our original problem of counting the points from some given set P below a non-vertical query line ℓ can be transformed into a problem of counting lines from a given set L of non-vertical lines above a query point q. That, on the other hand, is a *point* location problem: The plane is partitioned into regions of answers i, $i \in \{0..n\}$, and what is left is to locate q in these regions. How does the region of points with i lines above look like?

For $k \in [n]$, the k-*level*, Λ_k, of a set L of n non-vertical lines is the set of all points in $\mathbb{R}^2$ which have at most $k-1$ lines above and at most $n-k$ lines below. Hence, a point on the k-level is disjoint from at most $n-1$ lines in L and thus it must lie on at least one of the lines in L. Moreover, it is easily seen that every vertical line intersects the k-level in exactly one point (the lowest point on this vertical line which has at most $k-1$ lines above it). That is, the k-level is an x-monotone piecewise linear curve in the plane.

Given a point q and a bi-infinite x-monotone curve C, we write $C \succ q$ ($q \succ C$), if q lies below (above, resp.) the curve C. And we write $C \succeq q$, if $C \succ q$ or $q \in C$; and similarly for $q \succeq C$.

Observation 3.4. *For $k \in [n]$, the k-level Λ_k is an x-monotone bi-infinite curve in the plane. If $1 \le i \le j \le n$, every point on the i-level is either on or above the j-level.*

The number of lines in L above a point q is $\min\{k \mid q \succeq \Lambda_k\} - 1$, with the convention that Λ_{n+1} is a symbolic curve with $q \succeq \Lambda_{n+1}$ for all $q \in \mathbb{R}^2$.

So the question of how many lines are above a query point q can be resolved by locating q among the Λ_k's, which can be done by binary search with $O(\log n)$ comparisons of q with these curves. Remains the issue of (i) 'How efficiently can we compare q with such a curve?' and (ii) 'How much space is needed to store the k-levels for such comparisons?'.

The Complexity of a Line Arrangement. A set L of n lines in the plane partitions $\mathbb{R}^2$ into areas of various dimensions: *vertices* (of dimension 0), *edges* (dimension 1), and *cells* (dimension 2). The cells are the connected components of $\mathbb{R}^2$ after removal of all lines in L; so the cells are open convex

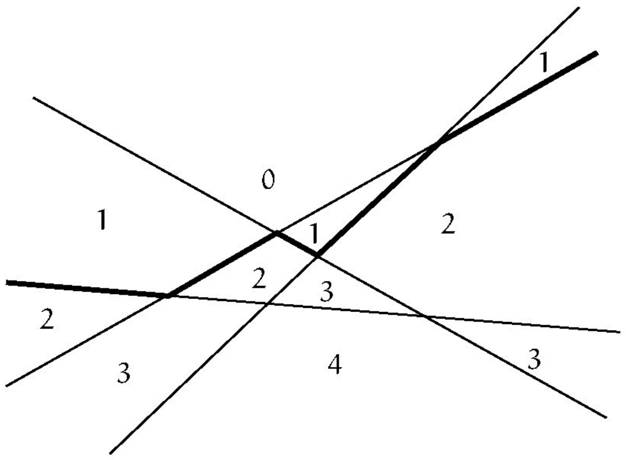

Figure 3.4: An arrangement of four lines, with the 2-level emphasized. Cells are marked with the number of lines above.

regions disjoint from the lines in L. The edges are the connected components on the lines after removal of the intersections with the respective other lines; so edges are contained in the lines, and since each line in L has at most $n-1$ intersections with the other lines, there are at most n edges contained in each line—makes at most n^2 edges altogether (some of which may extend to infinity). Finally, the vertices are what is left, i.e. the points in the plane that are contained in at least two lines. Therefore there are at most $\binom{n}{2}$ of them. The vertices, edges, and cells, they are called the *faces*, with their incidence structure are called the *line arrangement* (of L). Two faces are called *incident* if one is contained in the relative closure[10] of the other.

Lemma 3.5. $n \in \mathbb{N}_0$. *An arrangement of n lines has at most $\binom{n}{2}$ vertices, at most n^2 edges, and at most $\binom{n+1}{2}+1$ cells. If no three lines intersect in a common point and no two lines are parallel, all of these bounds are attained.*

(The proof of the cell-bound is left as an exercise.)

Recall that a k-level is contained in the union of the lines in L, hence it is composed of edges and vertices of the arrangement of L. The edges are the

[10]In perhaps more intuitive terms, 'one lies on the boundary of the other.' For example, an edge is incident to its endpoints and to two cells; a cell is incident to its bounding edges and their vertices.

linear pieces which connect the vertices, except for two edges that extend to infinity. We define the complexity m_k of Λ_k as the number of edges from the arrangement of L in Λ_k; hence, the number of vertices incident to these edges is $m_k - 1$. Clearly, the k-level can be stored with $O(m_k)$ space so that the relative position of a query point with respect to the level can be determined in time $O(\log(m_k + 1)) = O(\log n)$ (note that $m_k \leq n^2$ for sure). Note, moreover, that every edge of the arrangement is contained in exactly one level, hence $\sum_{k=1}^{n} m_k \leq n^2$.

We have obtained as an intermediate result: *A set of n lines in the plane can be stored in $O(n^2)$ space, so that the number of lines above a query point can be determined in time $O((\log n)^2)$.*

For an improvement of the query time we call for fractional cascading again (we will have to live with the quadratic space). Let us store (at least conceptually) the levels of an arrangement in a balanced tree for locating q among them, with leaves holding the number of lines above a point q whose search ends there. In every inner node v of the tree we have to find for x_q (the x-coordinate of the query point q) the interval I among the set S_v of x-coordinates of the respective level with $I \ni x_q$ (then we can compare q with the line holding the edge in the span of the interval). Depending on the outcome of the comparison, we proceed to the left or to the right child, and—unless this is a leaf—have to locate x_q again, etc. At present, we do the location anew from scratch again and again.

Instead, we now enhance the sets S_v by extra values inherited from its descendants in the tree, resulting in new sets $\overline{S}_v$. If a node v has no child which is an inner node, we leave the set unchanged: $\overline{S}_v := S_v$. Otherwise, we add to S_v every other value from each of the sets $\overline{S}_u$, u non-leaf child of v. As before, we start with the respective second elements of the lists, so that

$$|\overline{S}_v| \leq |S_v| + \sum_{u \text{ non-leaf child of } v} \frac{|\overline{S}_u|}{2} .$$

After this bottom-up generation of the sets in the nodes of the tree, we proceed top-down and add for each node v pointers from each interval I determined by $\overline{S}_v$, one pointer for each non-leaf child u of v. This pointer either directly points at an interval determined by $\overline{S}_u$ contained in I, or to the unique number of $\overline{S}_u$ that is contained in the interior of I. In this way, we can again proceed in constant time from an interval in a node to the interval in its child. Only in the root of the tree we will have to spend logarithmic time for an initial location of x_q.

With an induction proof analogous to that of Lemma 3.1 we get

Lemma 3.6.

$$\sum_{v \text{ inner node}} |\overline{S}_v| \leq 2 \cdot \sum_{v \text{ inner node}} |S_v| \leq 2n^2$$

That is, we were able to cut down the query time to $O(\log n)$ without asymptotic increase in space. We state our result in the original primal setting.

Theorem 3.7. *A set P of n points in the plane can be preprocessed with storage $O(n^2)$ so that the number of points below a non-vertical query line can be computed in time $O(\log n)$.*

Remains the issue of how much time preprocessing takes. Time $O(n^2 \log n)$ is relatively easy to derive, but optimal $O(n^2)$ is possible. We will not further elaborate on this.

Exercise 3.9. Below a Line, in Convex Position
Design a data structure that stores a set P of n points in convex position in the plane (i.e. these are the vertices of some convex polygon) in $O(n)$ space so that the number of points below a query line can be obtained in $O(\log n)$ time.

REMARK: This should serve as a warning that the complexity of the partition into regions of equal answers is by no means necessarily a lower bound for the space of a data structure supporting the respective queries—even for optimal query time.

Exercise 3.10. Low Complexity Arrangements
What are the minimum numbers of vertices, edges, and cells in an arrangement of n lines?

Exercise 3.11. Number of Cells in Arrangements
Prove that an arrangement of n lines in the plane has at most $\binom{n+1}{2} + 1 = \binom{n}{2} + \binom{n}{1} + \binom{n}{0}$ cells.

HINT: What is the maximum increase in the number of cells if we add a new line to an arrangement of $n-1$ lines?

Exercise 3.12. *Prove Lemma 3.6.*

Exercise 3.13. Stabbing Line Segments
Design an efficient data structure that stores a set of n line segments in the plane, so that the number of segments intersected by a query line can be computed in $O(\log n)$ time.

HINT: Prior treatment of Exercise 3.5 is recommended.

Exercise 3.14. Moving Points
We are given n moving points on the real line $\mathbb{R}$. Each point $i \in [n]$ has a starting position $s_i \in \mathbb{R}$ and a velocity $v \in \mathbb{R}$: Location of point i at time $t \in \mathbb{R}_0^+$ is $s_i + t \cdot v_i$. A query consists of a location $p \in \mathbb{R}$ and time $t \in \mathbb{R}_0^+$, and we want to know the set of points that pass location p after time t. Describe a data structure that supports such queries after preprocessing the set of moving points.

3.3 Planar Point Location—More Examples

We have seen that the problem of counting the points (from a given set) below a query line can be translated to the problem of locating a point in a subdivision (partition) of the plane. The latter appears again and again. An obvious occurrence is that of locating a point in a map.

Point Relative to Convex Polytope. Suppose we want to decide for a given convex polytope $\mathcal{P} \subseteq \mathbb{R}^3$ with $n \geq 4$ vertices whether it contains a query point $q \in \mathbb{R}^3$. The boundary of such a polytope decomposes into the n vertices, at most $3n - 6$ edges, and at most $2n - 4$ facets[11]. We can store the planes carrying the facets, and compare a query point with each of them to decide whether it lies in the polytope or not. This takes time and space $O(n)$. Point location after preprocessing allows improvement to $O(\log n)$ query time and linear space.

To this end, we let the boundary of $\mathcal{P}$ split into an upper part (seen from far above) and a lower part (seen from far below). For a formal definition, any vertical line that meets the polytope intersects it in a vertical line segment (possibly a single point). The collection of topmost points of these segments forms the *upper boundary*, and the bottommost points form the

[11]This nice linear relationship between the number of faces of various dimensions in a 3-dimensional polytope breaks down as we leave 'our' 3-space. For example, a 4-dimensional polytope with n vertices may have $\Theta(n^2)$ edges!

lower boundary; note that there are parts of the boundary that appear both on the upper and lower boundary, and in degenerate cases, there may be parts that appear in neither one (if edges or facets are vertical).

The vertical projection of facets, edges, and vertices of the lower boundary gives a subdivision of a convex polygon in the plane. If a query point q projects to some point outside this polygon, it lies outside the polytope $\mathcal{P}$. Otherwise it lies in the projection of some facet of the lower boundary of $\mathcal{P}$. If this facet is determined by point location in the projection, we can compare q with this plane: If q is below the plane, q is outside. Otherwise, we still have to decide whether it is also below the upper boundary of $\mathcal{P}$ in an analogous fashion.

Closest Point in the Plane—the Post Office Problem. We want to preprocess a set S of n points in the plane so that for a query point q, the point in S closest to q is delivered. (In fact, this point need not be unique; if so, we may ask for all closest points or just one of them.)

We follow the locus approach for this problem, i.e. we ask for $p \in S$: What are the query points q for which p is the closest point? Note that for two points p and p', the bisector[12] b of p and p' separates between points closer to p (p's side of b), points at equal distance to both (the line b itself), and points closer to p'. Let us denote by $h(p, p')$ the open halfplane determined by b containing p. Then the set of points $\mathcal{V}_S(p)$ closer to p than to any other point in S can be written as

$$\mathcal{V}_S(p) := \bigcap_{p' \in S \setminus \{p\}} h(p, p') \ .$$

It is called the *Voronoi cell of* p (w.r.t. S). From this characterization it follows that a Voronoi cell is a convex set with a piecewise linear boundary.

The collection of all $\mathcal{V}_S(p)$, $p \in S$, with their edges and vertices is a subdivision of the plane, the *Voronoi diagram of* S. Point location in this structure solves closest neighbor queries.

What is the complexity of the subdivision? It has n cells, but some of these cells may require many (up to $n-1$) edges to store their boundary. In order to get a bound on the overall number of edges, we consider the

[12]The bisector of two points p and p' is the line orthogonal to the segment $\text{conv}\{p, p'\}$ through its midpoint $\frac{p+p'}{2}$.

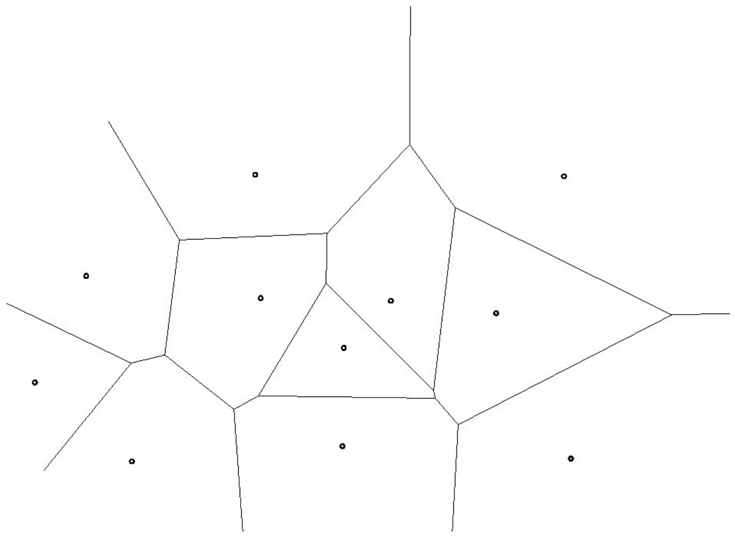

Figure 3.5: Voronoi diagram of eleven points.

dual graph[13] D of the Voronoi diagram: Its vertices are the points in S. Two points p and p' are connected by a ("combinatorial") edge $\{p, p'\}$, if their Voronoi cells share a common (geometric) edge e; e is called the *edge dual to* $\{p, p'\}$. This graph has the same number of edges as there are edges[14] in the Voronoi diagram. Moreover, this graph is planar; embed the graph D by leaving the points where they are, and an edge $\{p, p'\}$ of D is drawn by connecting p with some inner point of the edge dual to $\{p, p'\}$ by a straight line segment, and then continue with a straight line segment to p'. Note that here edges are not realized by straight line segments, but that is not required in a plane embedding of a graph.

It follows that D has at most $3n - 6$ edges, and so the Voronoi diagram has at most $3n - 6$ edges. Therefore, the number of vertices of the Voronoi diagram is linear in n, since their number cannot be more than twice the number of edges. (In fact, since the vertices of the Voronoi diagram are in correspondence to the regions of the dual graph, their number cannot exceed $2n - 4$; actually, $2n - 5$, why?)

[13]The straight line embedding of this graph is called the *Delaunay diagram* of the point set, a versatile structure with many applications from finite element methods in numerics to geometric modeling.

[14]This claim presumes that any two Voronoi cells cannot share two or more edges, which is true, since they are convex.

Summing up, the number of cells, edges, and vertices of the Voronoi diagram of n points is $O(n)$.

We have never formally defined what a planar subdivision is. In the examples we have seen that the regions of interest were bounded by straight line segments, the edges of the subdivision. If for a query point q we know the edge of the subdivision above[15] it, then the region containing q is identified. That should clearly establish the link between the point location problems identified here, and the problem solved in the section below.

3.4 Trapezoidal Decomposition

Before we proceed to the problem of the section, we get acquainted with the method by retreating to a simplified 1-dimensional scenario once more.

Warm-up. A set S of n real numbers partitions $\mathbf{R}$ into $n+1$ intervals. For a concrete setting, if $a_1 < a_2 < \ldots < a_n$ is the sorted sequence of the numbers in S, then we let these intervals be $[a_j, a_{j+1})$, $j = 0, 1, \ldots, n$, with the convention $a_0 := -\infty$ and $a_{n+1} := +\infty$; so, in fact, it's a subdivision of $[-\infty, \infty) = \mathbf{R} \cup \{-\infty\}$. Given S (not in sorted order), we want to prepare the subdivision for point location queries of the following type: For query $q \in \mathbf{R}$, the leftmost number in the interval containing q is requested (the number a_j with $q \in [a_j, a_{j+1})$); this clearly identifies the interval containing q. Sorting S in $O(n \log n)$ time and then storing it in an array allows queries to be performed in $O(\log n)$ time. We describe here a different method to prepare for generalization to $\mathbf{R}^2$.

The approach takes a random order $s_1, s_2, \ldots, s_n$ (u.a.r. from all permutations) of the numbers in S, and 'constructs' the subdivision by starting with the subdivision for the empty set (this is the single interval $I_0 := [-\infty, \infty)$). In this initial trivial situation, point location is particularly simple: The answer is $-\infty$ for any query.

In general, when a new number s_i is inserted, an interval among those defined by $S_{i-1} := \{s_1, s_2, \ldots, s_{i-1}\}$ disappears and two new intervals appear: The interval $I = [a, b)$ containing s_i disappears and the new intervals are $I' := [a, s_i)$ and $I'' := [s_i, b)$. That is, the replacement is easy to perform, as soon as we get our hands on I.

[15] Vertically above it, in y-direction.

The trick is to never really destroy an interval I whenever it disappears. Instead we mark it as 'destroyed', with two pointers to the intervals I' and I'' it got replaced by. This produces a directed graph, with the initial interval I_0 as the source (vertex with no ingoing pointer). Vertices marked as 'destroyed' are those intervals which have at some point appeared and then disappeared. The remaining intervals in the structure are those that actually make up the current subdivision; they are the sinks of the structure (i.e. without outgoing pointers). The graph is called the *history graph* for obvious reasons.

A new number s_i can easily be located in the structure generated for S_{i-1}. We start in I_0. If it is marked 'destroyed' (this is definitely the case, unless $i = 1$), we can decide with one comparison which of the intervals that replaced I_0 contains s_i; this interval is either an interval of the current subdivision, or is marked 'destroyed' and has two pointers, etc. *And after insertion of the last number s_n, we have a point location structure for the whole set handy!*

How much time does it take to locate a query point q in this structure? This is proportional to the number of times the interval containing q changes in the process, which depends on the order in which we have inserted the elements in S. Let X_i, $i \in [n]$, be the indicator variable for the event that the interval of S_{i-1} containing q differs from the interval of S_i containing q. The expected number of intervals containing q ever generated is $\mathbf{E}[X]$, $X := 1 + X_1 + X_2 + \ldots + X_n$, where the expectation is relative to the random choice of the order $s_1, s_2, \ldots, s_n$.

The analysis of $\mathbf{E}[X_i]$ is surprisingly simple, assuming the right view of the problem (see also Exercise 2.29): We build the random permutation of S in backwards order, starting with s_n, s_{n-1}, ... As we choose s_i u.a.r. in $S \setminus \{s_{i+1}, s_{i+2}, \ldots, s_n\}$ (note, this is S_i), the interval of S_i containing q differs from the one in S_{i-1} iff s_i is one of the two endpoints of this interval. This can happen with probability at most $\frac{2}{i}$, since s_i is chosen u.a.r. from i elements, and an interval has two endpoints (which may be $-\infty$ or ∞, that's why the probability is at most $\frac{2}{i}$ and, in general, not exactly $\frac{2}{i}$). Therefore, $\mathbf{E}[X_i] \leq \frac{2}{i}$. We have $\mathbf{E}[X_1] = 1$ and thus $\mathbf{E}[X] \leq 1 + 1 + \sum_{i=2}^{n} \frac{2}{i} = 2H_n = O(\log n)$ for $n \geq 1$.

We have shown that for any $q \in S$, the expected query time for q is $O(\log n)$. It also shows that inserting s_i in the structure so far (for S_{i-1}) is done in expected time $O(\log i) = O(\log n)$, and thus the whole structure for S is built in expected time $O(n \log n)$.

Have we discovered a new data structure for searching among n keys (or for sorting them)? Of course not—we leave it to the reader to realize that we encountered just another description of random search trees (e.g. the history graph is a tree here, which we did not emphasize, since it is not the case in the generalization to come).

The Segment Above—the Set-Up. A *segment* s in the plane is the convex hull of two points p_1 and p_2 in $\mathbf{R}^2$. The segment s deprived of its *endpoints* p_1 and p_2 is called the *relative interior* of s.

A set S of segments is called *non-crossing*, if every segment in S is disjoint from the relative interiors of the other segments. So segments may intersect only in their respective endpoints. $P(S)$ is the set of endpoints of the segments in S; we have $|P(S)| \leq 2|S|$. We call S *in general position*, if no two endpoints in $P(S)$ have the same x-coordinate; this excludes vertical segments, in particular. The set of relative interiors of the segments in S is denoted by S°; if S is non-crossing, for every point p in $\bigcup_{s \in S} s$ there is a unique element f in $P(S) \cup S^\circ$ with $f \ni p$.

Furtheron we assume that S is a set of n non-crossing segments in general position in the plane. A vertical upward ray emanating at a point $q \in \mathbf{R}^2$ is either disjoint from S, or there is a first point p where it meets $\bigcup_{s \in S} s$ (that could be q itself, if q lies on some of the segments in S). In the former case we define $\mathrm{above}(q) := \top$ (representing a symbolic segment above everything), or in the latter case $\mathrm{above}(q)$ is the unique f with $p \in f \in P(S) \cup S^\circ$.

Our goal is to preprocess S so that for any query point $q \in \mathbf{R}^2$ the endpoint or segment $\mathrm{above}(q)$ above q can be computed quickly.

We make a *simplifying assumption* for ease of exposition: The query point q lies on none of the segments and shares its x-coordinate with none of the endpoints in $P(S)$ (in particular, q never has an endpoint above it). It is easy to extend to these cases, but it distracts from the essentials at a first encounter of the method.

Trapezoidal Decomposition. For the sake of establishing the data structure, we actually refine the subdivision obtained by partitioning $\mathbf{R}^2$ into regions of equal answer to our queries.

For every endpoint p in $P(S)$ we consider two vertical extensions, one from p upward until the first segment in S° is met, and one downward until the first segment in S° is met; in either case, if no segment is met, we let it extend to infinity. The set of connected components of $\mathbb{R}^2$ without all segments in S and without all vertical (upward and downward) extensions from points in $P(S)$ is called *trapezoidal decomposition* of S, denoted by $\mathcal{T}(S)$ (see Figure 3.6). Note that every region in $\mathcal{T}(S)$ is a trapezoid (including artifacts as triangles, infinite trapezoidal slabs, halfplanes, or even the whole plane $\mathbb{R}^2$ if $S = \emptyset$). Every point in such a trapezoid has the same segment in S above it; according to our simplifying assumption, a query point has to lie in one of the trapezoids. Therefore, locating a point in this decomposition will solve our task.

Trapezoids are convenient since they have constant size descriptions. Do we have to pay for this with an increase in the number of regions? Not more than a constant factor, as the next lemma establishes.

Lemma 3.8. *For a set S of n non-crossing segments in general position in $\mathbb{R}^2$ we have $|\mathcal{T}(S)| = O(n)$.*

Proof. We associate the trapezoids of $\mathcal{T}(S)$ with the regions of a planar graph with $N = O(n)$ vertices; this allows us to conclude that the number of regions is at most $2N - 4 = O(n)$.

To this end we enclose S by a big circle, where we cut all the vertical extensions that would otherwise extend to infinity. In this way we get a plane drawing of a graph—hence, a planar graph. Vertices are $P(S)$ and the endpoints of the vertical extensions, including those on the enclosing circle; hence at most $3|P(S)| \leq 6n$ of them. Edges are the sections of segments, vertical extensions, and the circle between such vertices; hence, at most[16] $3(6n) - 6$, since the graph is planar. Finally, the number of regions is at most $2(6n) - 4$. This also bounds the number of trapezoids, since every trapezoid in $\mathcal{T}(S)$ is either a region of the graph, or, if it is infinite, it contains a region. $\square$

In fact, it can be shown that there are at most $3n + 1$ trapezoids in $\mathcal{T}(S)$.

[16]Recall that a planar graph with $N \geq 3$ vertices has at most $3N - 6$ edges, and in a plane embedding, the number of regions is at most $2N - 4$.

The History Graph. For some ordering $\sigma = s_1, s_2, \ldots, s_n$ of S let $S_i := \{s_1, s_2, \ldots, s_i\}$. The history graph of S (w.r.t. σ) has vertex set $\bigcup_{i=0}^{n} \mathcal{T}(S_i)$. There is a directed edge from T to T' if there is an $i \in [n]$ such that $T \in \mathcal{T}(S_{i-1}) \setminus \mathcal{T}(S_i)$, $T' \in \mathcal{T}(S_i) \setminus \mathcal{T}(S_{i-1})$, and $T \cap T' \neq \emptyset$. The source of this graph is $T_0 := \mathbf{R}^2$, and the sinks are the trapezoids in $\mathcal{T}(S)$.

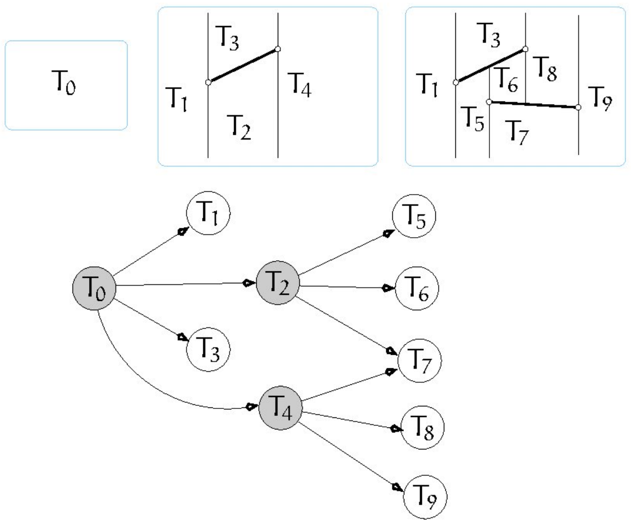

Figure 3.6: Two segments, development of their trapezoidal decomposition, and the history graph.

The idea behind this definition follows the pattern we have encountered in the previously seen 1-dimensional counterpart. We generate $\mathcal{T}(S)$ incrementally, adding one segment after the other. Whenever a trapezoid T disappears, we mark it as 'destroyed', and equip it with pointers to those new trapezoids that intersect the area of T. Location of a point q starts in the source, and as long as we are in a destroyed trapezoid, we proceed to the unique successor which contains q. In the end we have reached a sink of the structure, which corresponds to the trapezoid of $\mathcal{T}(S)$ containing q. Three issues determine the quality of the structure (sweeping the issue of preprocessing under the rug).

(i) The size of the structure, i.e. the number of vertices and edges of the history graph.

(ii) For a query point q, the length of the path followed by q from T_0 to its trapezoid in $\mathcal{T}(S)$. Or equivalently, the number of trapezoids in

$\bigcup_{i=1}^{n} \mathcal{T}(S_i)$ containing q.

(iii) For a query point q, the time it takes to proceed from a 'destroyed' trapezoid $T \ni q$ to its successor containing q.

The proof of the following crucial fact is left as an exercise (see Exercise 3.16).

Observation 3.9. *Given $T \in \mathcal{T}(S_{i-1}) \setminus \mathcal{T}(S_i)$, the number of trapezoids in $\mathcal{T}(S_i)$ overlapping with T is at most 4.*

That is, for $q \in T$, we can proceed in constant time to the successor trapezoid containing q. And the size of the history graph structure is linear in the number of its vertices (i.e. trapezoids), since there are at most four times as many edges.

Roughly speaking, a trapezoid in $\mathcal{T}(S)$ is determined by four segments in S, two responsible for the upper and lower boundary, resp., and two segments responsible for the left and right, resp., delimiting vertical extension. Clearly, the same segment may serve several purposes, or some of the boundary elements may be missing. So more precisely:

Observation 3.10. *Given $T \in \mathcal{T}(S)$, there is a set $S' \subseteq S$ (not necessarily unique) of at most 4 segments, such that $T \in \mathcal{T}(S')$.*

That is, if $T \in \mathcal{T}(S)$, there are at most 4 segments in S whose removal lets T disappear (in a backwards process).

Now we assume that the ordering $\sigma = s_1, s_2, \ldots, s_n$ is chosen u.a.r. from all $n!$ permutations of S. A backwards analysis argument (along the same lines as in the 1-dimensional warm-up example) employing Observation 3.10 shows that, for a given $q \in \mathbf{R}^2$, the expected number of trapezoids in $\mathcal{T}(S)$ containing q is at most $4H_n = O(\log n)$.

Similarly, any trapezoid T in $\mathcal{T}(S_i)$ disappears (in the backwards process) because of removal of a random segment in S_i with probability at most $\frac{4}{i}$. So the expected number of trapezoids removed in the backwards process from S_i to S_{i-1} is at most $\frac{4}{i} \cdot O(i) = O(1)$ (since there are $O(i)$ trapezoids in $\mathcal{T}(S_i)$). Hence, the overall expected number of trapezoids ever removed in the backwards process (i.e. the overall expected number of trapezoids in $\mathcal{T}(S)$) is $n \cdot O(1) = O(n)$.

Theorem 3.11. *Let S be a set of n non-crossing segments in general position in the plane.*

Assume a u.a.r. random ordering of the segments in S. Then the history graph has expected size $O(n)$. For any fixed $q \in \mathbb{R}^2$, the expected time for locating q (with the history graph structure) in $\mathcal{T}(S)$ is $O(\log n)$.

The history graph can be computed in expected $O(n \log n)$. The main detail to overcome is how to determine the trapezoids being destroyed by insertion of a new segment s. For that, one endpoint p of s is located (the structure is always ready for such an operation, as we know), and then starting at p we 'walk' along s from one trapezoid to the next. All trapezoids encountered are destroyed. New ones have to be generated.

The general underlying principle, *randomized incremental construction* has numerous applications including computations of convex hulls (of point sets in any dimension) or Voronoi diagrams.

Exercise 3.15. Not too Many, not too Few

Let S be a set of $n \geq 2$ non-crossing segments in the plane. Show that the set $P(S)$ of endpoints of S satisfies

$$2 + \frac{n}{3} \leq |P(S)| \leq 2n \ .$$

Exercise 3.16. Overlapping Trapezoids

Let S be a nonempty set of non-crossing segments in general position in the plane, let $s \in S$, and let T be a trapezoid in $\mathcal{T}(S \setminus \{s\}) \setminus \mathcal{T}(S)$. Depending on the number of endpoints of s inside T (0, 1, or 2), investigate how many trapezoids overlapping with T can be created by adding s to $S \setminus \{s\}$.

Exercise 3.17. Nearest Neighbor Changes

We are given a set P of n points in $\mathbb{R}^2$ and a point q which has distinct distances to all points in P. We add the points of P in random order (starting with the empty set), and observe the nearest neighbor of q in the set of points inserted so far. What is the expected number of distinct nearest neighbors that appear during the process?