DCG, Wintersemester 2024/25 Lecture 16 (7 pages)

Trapezoidal Decomposition

Anne Driemel and Herman Haverkort updated: December 9, 2024

In the previous lecture we discussed point location in the plane and we saw a data structure that can be used for this purpose. In this lecture we will see another data structure for point location. We will use randomized incremental construction to build this data structure.

1 Trapezoidal decomposition

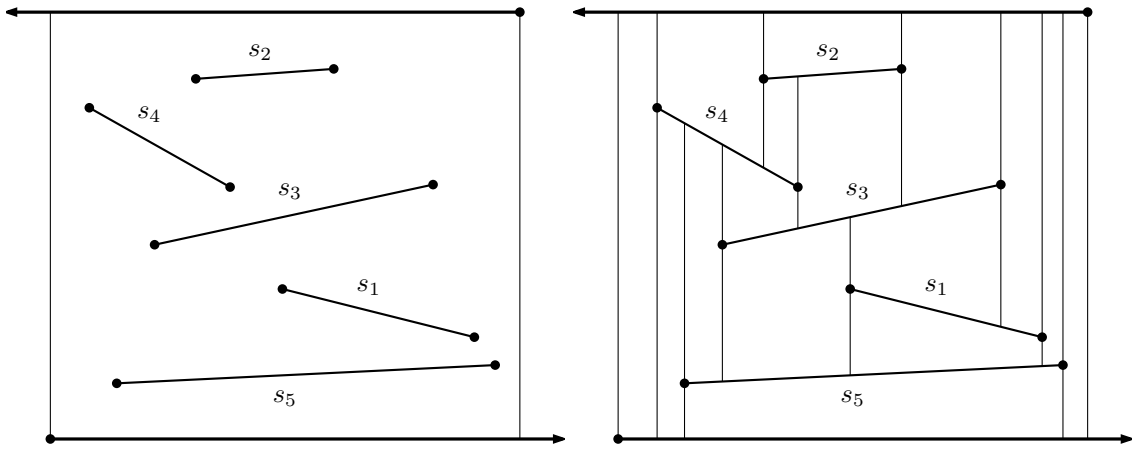
Consider a subdivision that is induced by a set of line segments that are pairwise non-crossing, under the definition below. This covers both of the applications that we have previously discussed, triangulations and convex subdivisions, and even non-convex subdivisions.

Definition 16.1 (Non-crossing)**.** *We call two line segments in the plane non-crossing if the intersection is either empty or a common endpoint.*

Assumption 16.2. *For a set of line segments in the plane we assume that no two endpoints share the same y-coordinate, unless they also have the same x-coordinate.*

Definition 16.3 (Trapezoidal decomposition)**.** *Let S be a set of n line segments that are pairwise non-crossing and such that no two different endpoints share the same y-coordinate. Let R be an axis-aligned rectangle that contains S in its interior. For every endpoint p of a line segment in S we add two vertical line segments, one ending on the line segment directly above p and one ending on the line segment directly below p. If there is no line segment directly above, then we connect p to the top edge of R. If there is no line segment directly below a, we connect p to the bottom edge of R. The arrangement that is induced by this set of line segments (including the vertical extensions) is called a trapezoidal decomposition.*

We use the same terminology of faces, edges and vertices, that we have previously introduced for arrangements of lines and hyperplanes. We will not specify the bounding rectangle from now on. We will assume that it is given with the set S of line segments. Let S' be S extended with the horizontal halfline that extends from the lower left corner of R to the right and the horizontal halfline that extends from the upper right corner of R to the left.



The faces of a trapezoidal decomposition have some special properties, which make them easy to work with. First, each face is convex, and therefore there can be at most two vertical

lines bounding any face. Any vertical edges bounding a face must originate from the vertical extensions of endpoints of segments, since no segment in S' is vertical. By our assumption that no two different endpoints of segments of S share the same x-coordinate, the vertical edges bounding the face can originate from at most two different endpoints.

As a result, each face f is bounded by two line segments $top(f)$ and $bottom(f)$ from S' and at most four vertical edges. The shape of a face is completely determined by $top(f)$ and $bottom(f)$ and the x-coordinates from the two unique endpoints $leftp(f)$ and $rightp(f)$ on its left side and on its right side. Each of these endpoints is either a common endpoint of $top(f)$ and $bottom(f)$, or it is some segment's endpoint whose vertical extension(s) bound(s) f from the left or from the right. The figure below shows examples of what a face can look like.

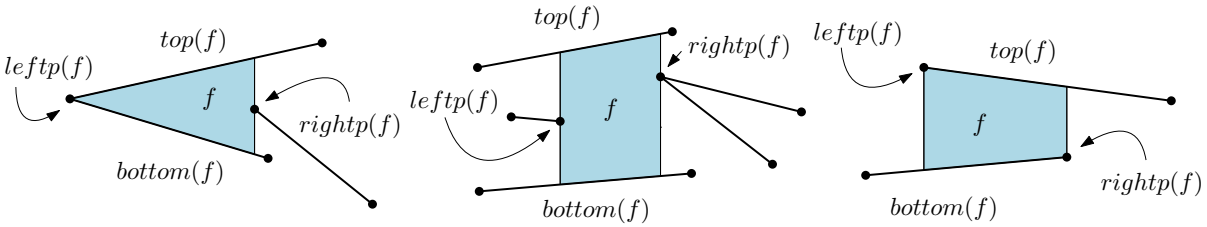


Figure 1: Three examples of what a face f may look like.

Lemma 16.4. *A trapezoidal decomposition of a set S of n line segments in the plane with a bounding rectangle R has at most $3n + 1$ faces (counting only the faces inside R) and at most $6n + 4$ vertices.*

Proof. There are at most $2n$ different endpoints of segments of S. Each of these endpoints introduces 2 new vertices by its vertical extensions. In addition, there are 4 vertices of the bounding rectangle. Together with the at most $2n$ endpoints themselves, we get at most $6n+4$.

To show the bound on the number of faces, we consider for each face the unique edge that bounds it from above. This could be a segment from S', or a part of a segment between two vertical extensions. We count the number of such top edges as follows. There are n segments in S and one top edge from R. Each upward vertical extension from one of the at most $2n$ endpoints splits one top edge into two. Therefore, each vertical extension to the top increases the number of top edges by at most one. Thus, in total we get $3n + 1$ top edges and this also bounds the number of faces. □

2 Algorithm

We describe a randomized incremental algorithm to compute the trapezoidal decomposition. The pseudocode below gives an overview of the algorithm. The algorithm maintains a graph G that is used for executing the point location query among the faces of the current trapezoidal decomposition T. This search structure is defined in more detail in Section 2.2.

2.1 Representation of the decomposition

We could represent the trapezoidal decomposition using a DCEL. However, in a DCEL, the representation of the boundary of a trapezoid f would have non-constant size, since the line segments $top(f)$ and $bottom(f)$ that bound f may be cut up in into many halfedges due to the vertical extensions that reach $top(f)$ and $bottom(f)$ from outside f. In our algorithms, we would like to avoid having to traverse all these pieces. Therefore we use a simpler data structure

Algorithm 16.1

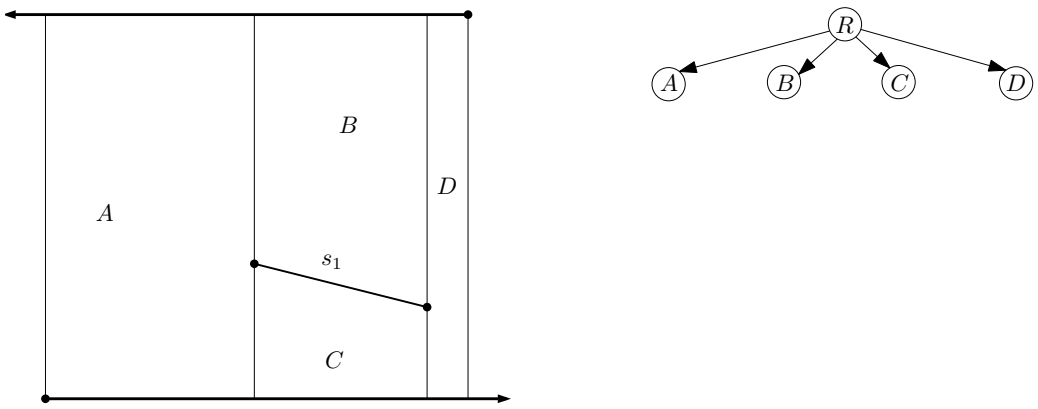
```
1: procedure TRAPEZOIDAL-DECOMPOSITION(Line segments S ⊆ ℝ² × ℝ², Rectangle R)
2:     Let s_1, ..., s_n be a random permutation of the line segments of S
3:     Initialize a planar subdivision T with the segments of R
4:     Initialize a graph G with a root node that corresponds to R
5:     for i ← 1 to n do
6:         Using G, find the face f_1 that contains the left endpoint of s_i
7:         Starting from f_1, find the other faces f_2, ..., f_k of T that properly intersect s_i
8:         Remove f_1, ..., f_k from T and insert the new faces resulting from the insertion of s_i
9:         Update the graph G with respect to the faces in T
10:    end for
11:    return T
12: end procedure
```

that sacrifices some DCEL features which we do not need, but allows us to access and maintain the boundaries of trapezoids in $O(1)$ time per trapezoid.

The data structure will have records for all line segments and endpoints of S' and for all faces of the trapezoidal composition. For each face f, we store pointers to the records for $top(f)$, $bottom(f)$, $leftp(f)$, and $rightp(f)$, and to the faces that are adjacent to f along its (at most) four vertical edges.

2.2 Search structure for point location

We now further specify the search structure that is maintained by the algorithm. The graph is a directed acyclic graph. We call a node of the graph a leaf node if it has no outgoing edges, and we call a node a root node if it has no incoming edges. Each node of the graph is associated with a face that was created in the trapezoidal decomposition at some point during the algorithm. We keep pointers from the leaf nodes to the corresponding faces in the subdivision and back. In addition, each node stores the geometry of its associated face, and can answer in constant time if a given point is contained inside this face. The graph has a root node that is associated with the bounding rectangle, which is present before the first segment is added. When there is only one segment in the trapezoidal decomposition, the graph has a root node and four leaf nodes, each of them associated with a face of the trapezoidal decomposition.



When inserting the segment s_2, the faces A and B are removed and the new faces E, F, G, H, I, are created in the trapezoidal decomposition, see Figure 2. In the search structure, we

create a node for every new face and we add edges to these new nodes. Specifically, we add an edge to a newly created node from a node whose associated face was removed when inserting s_2, if the intersection of the associated faces is non-empty.

In the same manner, we insert all segments of S one by one. After each insertion, only leaf nodes have valid pointers to faces of T; non-leaf nodes always correspond to faces that were formerly in T but have since been removed.

A query with a point q in the search structure starts in the root node and follows a path along the directed edges. In each node it checks, for every child of this node, if q is contained in that child's associated face. If yes, then the query continues with that child. When the query arrives in a leaf, it outputs the face associated with this leaf as the answer to the query.

2.3 Update step

When inserting a segment s_i into the trapezoidal decomposition, the algorithm

(i) first uses the search structure to find the face that contains the left endpoint of s_i,

(ii) and then needs to find the other faces that are intersected by s_i.

First suppose that the left endpoint of s_i lies in the interior of a face. Let $f_1, \ldots, f_k$ be the faces intersected by s_i, ordered from left to right along their intersection with s_i. Once the algorithm has found the face f_1, it can find the other faces from left to right by following the pointers to its neighbours, as follows. Recall that s_i cannot leave any face f across $\mathit{top}(f)$, across $\mathit{bottom}(f)$, or through $\mathit{rightp}(f)$, since the segments are pairwise non-crossing. Therefore we simply compare the x-coordinate of the right endpoint of s_i with the x-coordinate of $\mathit{rightp}(f)$ to determine whether the right endpoint of s_i is contained in the current face. If not, we can determine the next face f_j in $O(1)$ time, since it must be adjacent to f_{j-1} along one of the vertical edges incident on $\mathit{rightp}(f)$. In total, finding the faces only takes time linear in the number of faces that properly intersect s_i.

Now suppose that the left endpoint of s_i lies on the boundary of a face. How could this happen? Since the segments are pairwise non-crossing and no two different endpoints have the same x-coordinate, s_i cannot lie in the interior of $\mathit{top}(f)$, $\mathit{bottom}(f)$, or one of the (at most) four vertical edges of any face f. But the left endpoint of s_i could be an endpoint of one or more of the segments that were inserted before. In that case, the procedure described above will retrieve $f_1, \ldots, f_k$ correctly, provided we start with the correct face f_1. This can be achieved as follows. The query algorithm is also given the other endpoint of s_i and handles boundary cases in the following way: the left endpoint of s_i is considered to lie inside a face f if and only if the left endpoint of s_i lies in the closure of f and the interior of s_i intersects the interior of f.

Once $f_1, \ldots, f_k$ have been found, we can retrieve the corresponding nodes in the search structure. Let $v_1, \ldots, v_k$ be those nodes. The algorithm then removes the faces $f_1, \ldots, f_k$ from the trapezoidal decomposition and replaces them with new faces $g_1, \ldots, g_{k'}$. Each face f_i is split into at most four new faces, and some faces have to be merged. The algorithm creates a new node in the graph for every new face g_j. Let $w_1, \ldots, w_{k'}$ be those nodes. The algorithm inserts an edge from v_i to w_j if the corresponding faces f_i and g_j have a common point in their interior. In total this takes $O(k)$ time.

Example 16.5. *In Figure 2, when inserting s_4, the algorithm removes faces J, K and M and replaces them with new faces Q, T, U, V, W, X. In the search structure, which is shown to the right of the trapezoidal decomposition in the figure, the algorithm add edges between the nodes corresponding to these faces. The following edges are inserted into the search structure:*

(i) J is connected to Q, T, U

(ii) K is connected to U and V

(iii) M is connected to V, W and X

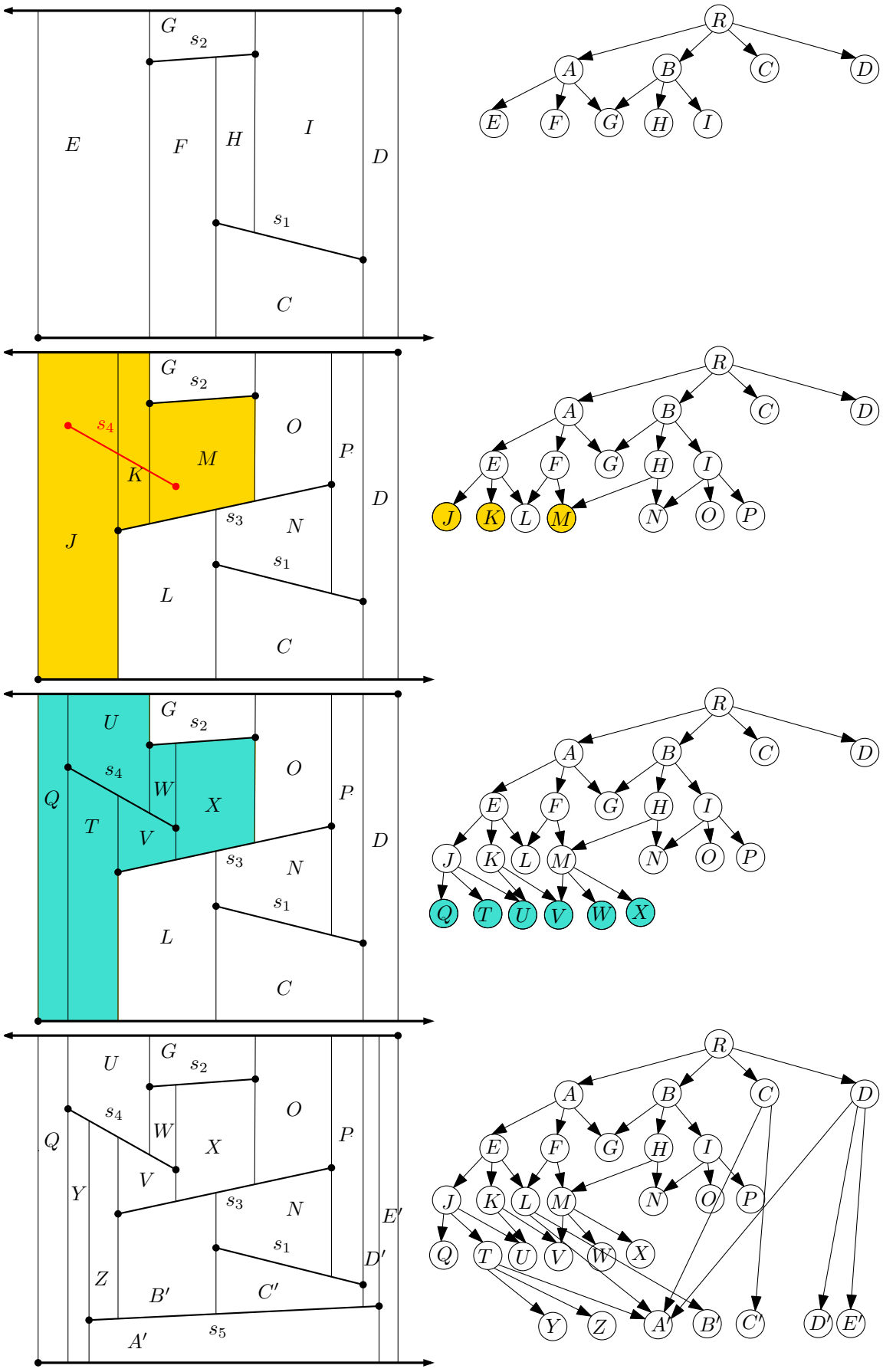


Figure 2: Sequence of trapedoizal decompositions during the course of the algorithm. The search structure with nodes corresponding to the faces is shown on the right.

3 Analysis

Theorem 16.6. *Given a set of n line segments in the plane and a bounding rectangle R, such that the segments of S are pairwise non-crossing and no two endpoints share the same x-coordinates, one can compute a trapezoidal decomposition and a search structure for point location in $O(n \log n)$ expected time. The expected size of the search structure is $O(n)$ and for any query point $q \in \mathbb{R}^2$ the expected query time is $O(\log n)$.*

Proof. We claim that for any fixed i for $1 \leq i \leq n$, the expected number of faces that are newly created in the trapezoidal decomposition, when s_i is added, is constant. This can be proven using backwards analysis within the framework of the randomized incremental construction (→ Exercise). From the claim it follows that the expected size of the data structure is in $O(n)$.

We now analyze the expected query time. Let $q \in \mathbb{R}^2$ be fixed. The query time for a query with q is linear in the length of the path that the algorithm traverses in the search structure. Let L denote this length. Indeed, the number of outgoing edges of a node in the data structure is constant and it takes constant time to check whether q is contained in the face associated with a node. Every node on the path was created at some point during the algorithm. Let X_i, for $1 \leq i \leq n$, be a random variable that is equal to 1 if there is a node on the path that was created when inserting segment s_i, and equal to 0 if this is not the case. Let A_i be the corresponding event. Since every node in the search structure, except for the root node, is created during some insertion of a segment s_i for $1 \leq i \leq n$ and since every insertion can add at most one node to the path, we can express the expected length of the path as

$$\mathbf{E}[L] = \mathbf{E}\left[1 + \sum_{i=1}^{n} X_i\right] = 1 + \sum_{i=1}^{n} \mathbf{E}[X_i] = 1 + \sum_{i=1}^{n} \mathbf{Pr}[A_i]$$

where the equality follows from the linearity of expectation and the additive term of 1 accounts for the root node.

We now want to analyze the probability of A_i occurring. Fix some value of i and consider the trapezoidal decomposition T before inserting the segment s_i. The event A_i occurs if and only if the face of T that contains q is removed during the insertion of s_i. This probability is not so easy to analyze, so instead consider the trapezoidal decomposition T' *after* inserting s_i. Let f be the face of T' that contains q. The event A_i can only occur if s_i is one of the segments defining the face f. More precisely, A_i can only occur if s_i is one of the following segments:

1. $top(f)$;
2. $bottom(f)$;
3. a segment that has $leftp(f)$ as an endpoint, and which is, in fact, the *only* segment out of $s_1, ..., s_i$ with that endpoint (if multiple segments out of $s_1, ..., s_i$ have $leftp(f)$ as an endpoint, then s_i cannot be the first of them to be inserted, so $leftp(f)$ is introduced into T before the insertion of s_i already);
4. a segment that has $rightp(f)$ as an endpoint, and which is, in fact, the *only* segment out of $s_1, ..., s_i$ with that endpoint.

Note that this list comprises at most four different segments, but possibly less.

Now we can use backwards analysis again to analyze the probability that A_i occurs. Fix a subset $Z \subset S$ of cardinality i and assume that the random permutation puts the set Z at the positions $\{1, \ldots, i\}$ and the set $S \setminus Z$ in the positions $\{i+1, \ldots, n\}$. Now, the trapezoidal decomposition T', after inserting segment s_i is fixed, but we do not know which of the elements of Z was inserted last (i.e., which of them was s_i). Each of the elements of Z has the same probability of being chosen at position i. Also, since T' is fixed and q is fixed, the face f of T' that contains q is also fixed. As analyzed above, in order for A_i to occur, it must be that s_i

is one of at most four segments that define f. Since each of them has probability $\frac{1}{i}$ of being chosen at position i, the probability of A_i occurring is at most $\frac{4}{i}$ and thus

$$\mathbf{E}[X_i] \leq \frac{4}{i}$$

Since this upper bound holds for any fixed set Z, it also holds as an upper bound on the expectation over all possible choices of Z. Thus, we have for the expectation

$$\mathbf{E}[L] \leq 1 + \sum_{i=1}^{n} \frac{4}{i} \leq 1 + 4H_n$$

where H_n denotes the nth Harmonic number. Thus, the expected query time is $O(\log n)$.

It remains to analyze the construction time. The algorithm performs n iterations of the main for-loop, in each iteration it performs one point location in the data structure and then finds the other faces that are intersected by the new line segment and updates the trapezoidal decomposition and the search structure. The point location takes expected $O(\log n)$ time and once the first face is found, finding the other faces and performing the update is linear in the number of faces newly created. By our initial claim, the expected number of faces that are added in a fixed iteration is constant. Therefore, the expected total construction time is bounded by $O(n \log n)$. □

References

- Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars. Chapter 6. Computational Geometry— Algorithms and Applications. Third Edition. Springer.