Professor Diane Souvaine
Scribe (2003): Heather Wong
Editor: Amy Bui

1 Topological sweep: $O(n^2)$ time, $O(n)$ space

Let H be a set of n lines in the plane. Assume that no three lines intersect at a point and that none of the lines is vertical. Each line intersects $n-1$ other lines and thus is divided into n edges. The regions, edges and vertices partition the plane into a subdivision known as arrangement, or $\mathscr{A}(H)$. If we use a vertical sweep line, we need to sort n^2 intersection points. Whether it is possible to sort the n^2 intersection points determined by n lines in $o(n^2 \log n)$ is still an open problem. In topological sweep we compromise the straightness of the sweepline to acheive better time and space complexities than vertical line sweep. The idea of topological sweep is to use a curved line (*topological line*) with some special properties to simulate a vertical line. Using a topological line to sweep the arrangement, we need only $O(n^2)$ time and $O(n)$ space.

A topological line (cut) is a monotonic line in y-direction which intersects every other line exactly once. It is specified by a sequence of edges $(c_1, c_2, , , , , c_n)$, each contains an intersection point of the cut with a different line in the arrangement. Notice that a vertical sweep line runs from $-\infty$ to ∞ in the y-direction and intersects each line in the arrangement exactly once. A cut has the same properties by definition.

The sweep will be implemented by starting with leftmost cut which includes all semi-infinite edges and pushing it to the right till it becomes the rightmost cut, in a series of elementary steps.

An elementary step is performed when the topological line sweeps past a vertex of the arrangement. To keep the sweep line a topological line, we can only past a vertex which is the intersection point of 2 consecutive edges in the current cut. (Otherwise, it will intersect some line more than once.) Do we always have such a vertex during the process of sweeping? That is, will the topological sweep get stuck?

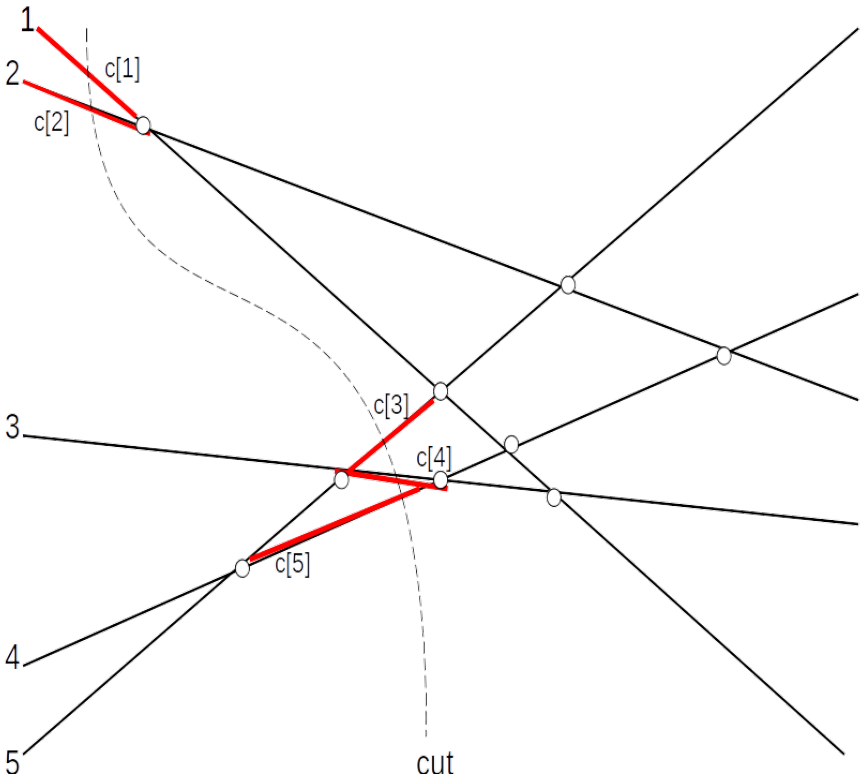

Figure 1: Example of a topological line with associated cuts, modified from Edelsbrunner, Guibas (1989), and using their labelling convention. Notice that cut `c[4]` is on line l_3, etc.

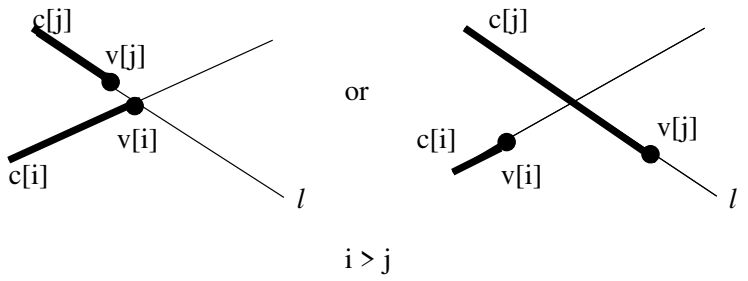

Figure 2: The right endpoint of an edge of the cut.

Lemma 1.1 *There always exist two consecutive edges of the cut with a common right endpoint, unless we are considering the rightmost cut.*

Proof: Assume that there are vertices still unprocessed but there is no pair of cut edges c_k, c_{k+1} which share a common right endpoint. Let c_i be the cut edge with the leftmost right endpoint. Let c_j be the edge of the cut on l which cuts c_i at its right endpoint v_i. c_js right endpoint (v_j) is either to the right or to the left of c_is right endpoint (v_i). See figure 2.

If v_j is to the right of v_i, then the topological line cuts l more than once. So, this can not be the case.

If v_j is to the left of v_i, then v_j is the leftmost right endpoint. Contradiction.

Therefore $v_i = v_j$. And the leftmost right endpoint is always an elementary step, i.e. it is "ready".

2 Data Structure

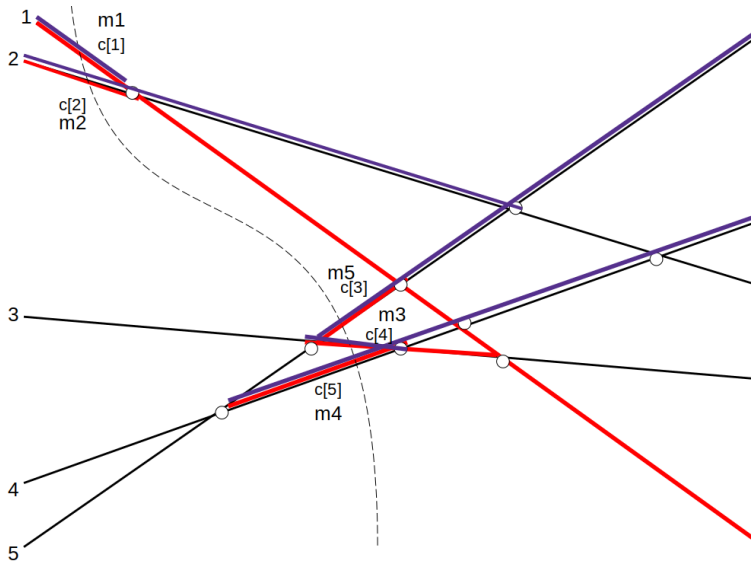

Figure 3: The same topological cut in Fig. 1, but with an overlay of the upper (purple) and lower (red) horizon tree, modified from Edelsbrunner, Guibas (1989). The cuts `c[i]` are ordered according to the article and is just important for how the authors explain the **I** data structure. `c[i]` doesn't necessarily belong to line l_i in the figure.

E$\big[$**1:n**$\big]$ is the array of line equations. E[i] $= (a_i, b_i)$ if the i^{th} line l_i of $\mathscr{A}(H)$ is $y = a_i x + b_i$. This is a static data structure, as we sort it by slopes once and don't change it again.

HTU$\big[$**1:n**$\big]$ is an array representing the upper horizon tree. HTU[i] is a pair (λ_i, ρ_i) (for "left" and "right") of indices indicating the lines that delimit the **segment** of l_i in **upper horizon tree** (purple) to the left and to the right, respectively. If this segment is the leftmost on line l_i, we set $\lambda_i = -1$; if it is the rightmist l_i, we set $\rho_i = 0$. For example, in Fig. 1 and 3, for line l_3, its associated cut is `c[4]` in the picture, it is deliminted by l_5 on the left and by l_4 on the right; so HTU[i] $= (5, 4)$.

HTL$\big[$**1:n**$\big]$ represents the lower horizon tree and is defined similarly.

I is a set of integers, represented as a stack. If i is in I, then c_i and c_{i+1} share a common right endpoint, i.e. it is "ready". In Fig. 3, $I = [4, 1]$, because `c[4]` (and `c[5]`) and `c[1]` (and `c[2]`) are both "ready".

M[1:n] is an array holding the current sequence of indices that from the lines m_1, m_2, , , , , m_n of the cut. In Fig. 3, from top to bottom, $M = [1, 2, 5, 3, 4]$.

N[1:n] is a list of pairs of indices indicating the lines delimiting each edge of the **cut** at l_i. For example, in Fig. 3, the cut on l_5 is delimited by l_3 and l_1, so $N[5] = (3, 1)$.

All the data structures are defined below for current cut of the arragement in Figure 1 and 3.

E:	(a_1, b_1)	HTU:	$(-1, 2)$	HTL:	$(-1, 0)$
	(a_2, b_2)		$(-1, 5)$		$(-1, 1)$
	(a_3, b_3)		$(5, 4)$		$(5, 1)$
	(a_4, b_4)		$(5, 0)$		$(5, 3)$
	(a_5, b_5)		$(3, 0)$		$(3, 1)$

I:	4	M:	1	N:	$(-1, 2)$
	1		2		$(-1, 1)$
			5		$(5, 4)$
			3		$(5, 3)$
			4		$(3, 1)$

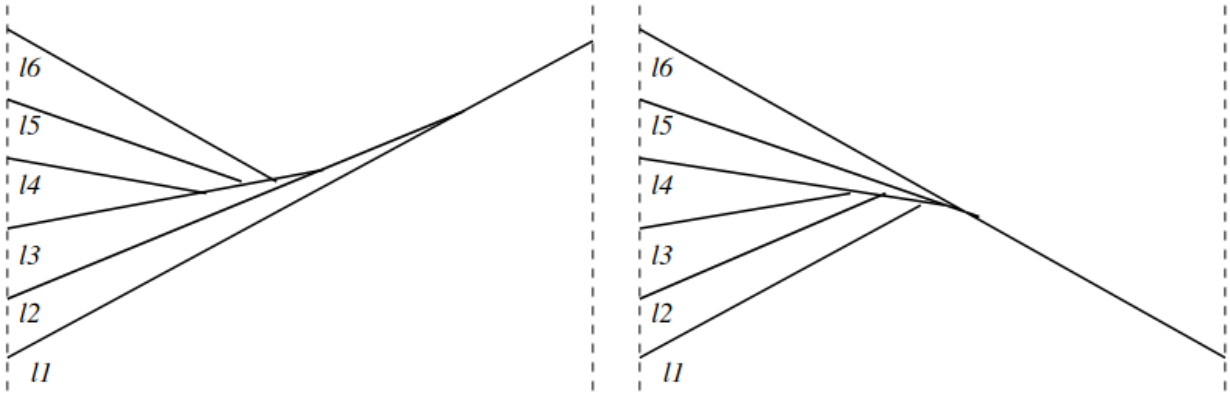

Figure 4: Broad example of Upper and Lower Horizon Trees

The *upper horizon tree* of a cut is constructed by starting with the cut-edges and extending them to the right. When two edges come together at an intersection point, only the one of higher slope continues to the right.

The *Lower horizon tree* is constructed similarly. The difference is that when two edges intersect, only the one of lower slope continues to the right. See figure 4.

Given HTU and HTL, the right endpoint of the edge on l_i is identified by the closer of HTU[i] and HTL[i].

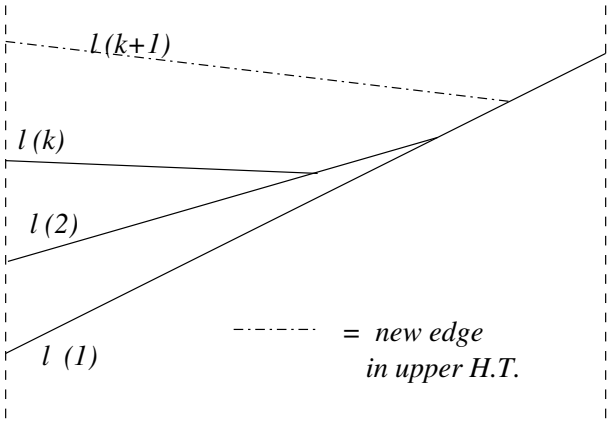

Figure 5: Creating a hammock.

3 The algorithm

Initialization:

1. Sort the lines of the arrangement by slope. This also initializes E and M. $O(n \log n)$.

2. Find the leftmost and the rightmost intersection point of the lines. Let the two points be (x_l, y_l) and (x_r, y_r). $O(n)$.

3. Create vertical lines $x = x_l - 1, x = x_r + 1$ as left boundary and right boundary. $O(1)$. Determine the intersection points of lines $l_1, , , l_n$ with the boundaries. $O(n)$.

4. Create upper horizon tree ($O(n)$):
   Insert $l_1, , , l_n$ in order to make a "hammock":
   Assume $l_1, , , l_k$ have been inserted. These lines form an *upper bay* as shown in figure 5. To insert l_{k+1}, begin at its endpoint on the left boundary. Walk in counter clockwise order around the bay till we find the intersection point of l_{k+1} with an edge.

5. Create lower horizon tree ($O(n)$) similarly by starting the travers at endpoints on the right boundary.

6. Initialize N ($O(n)$): Let HTU[i] = (-1, r] and HTL[i] = (-1, s]. If l_r intersects l_i to the left of the intersection point of l_s and l_i then the right delimiting line of e_i is r. Otherwise, the right delimiting line of e_i is s.

   USing l_1 in Fig. 3 as an example, because it is still the same position as it was when the arrangment was initialized, $HTU[1] = (-1, 2)$ and $HTL[1] = (-1, 0)$, where l_0 is that imaginary rightmost vertical spreader bar in the picture. Since l_2 intersects l_1 to the left of where l_0 intersects l_1, then the right delimiting line is 2, i.e. $N[1] = (-1, 2)$.

7. Initialize I by scanning N. $O(n)$.

Runtime Explanation:

Step 1 takes $O(n \log n)$ time. Step 2 could be done in linear time since the leftmost and rightmost intersection points must be the intersection points of two consective lines in the sorted order. Step 3,6,7 take linear time.

Now lets look at step 4. When inserting line l_i, each edge on the bay will be traversed once. There are $O(n)$ edges on the bay and there are n lines to be inserted. So we have $O(n^2)$. Could we find a tighter bound? Notice that our lines are inserted in decreasing order of slopes. Edges on the bay that fail to intersect with line l_i will not appear on the new bay formed after we insert line l_i. Each line can only fail once, and therefore be traversed once. So, HTU can be created in $O(n)$ time.

By similar argument, step 5 also takes $O(n)$. Hence, the initialization can be done in $O(n)$ after sorting of lines.

Elementary Step

After initialization, a series of elementary steps has to be executed. Notice that when passing a valid vertex (stored in I), only the two cut edges involved get changes, all other edges remain the same.

While I$\neq \Lambda$

1. Pop i from I
2. Swap M[i], M[$i+1$] /*lines are going to cross, after the elementary step*/
3. N[i].$\lambda \leftarrow$ N[$i+1$].ρ
   N[$i+1$].$\lambda \leftarrow$ N[i].ρ /*the point of elementary step becomes the left endpoint of the two new cut edges */
4. Update HTU, HTL.
5. N[i].$\rho \leftarrow$ closer HTU[M(i)].ρ,HTL[M(i)].ρ
   N[$i+1$].$\rho \leftarrow$ closer HTU[M($i+1$)].ρ,HTL[M($i+1$)].ρ
   /* find the new right endpoints */
6. If N[$i+1$].ρ = M[$i+2$] then push $i+1$ into I.
   If N[i].ρ = M[$i-1$] then push $i-1$ into I.
   /* push valid vertices formed after sweeping into I if there is any */

The steps 1,2,3,5,6 require just O(1) time. Lets take a look at step 5.

Update HTU

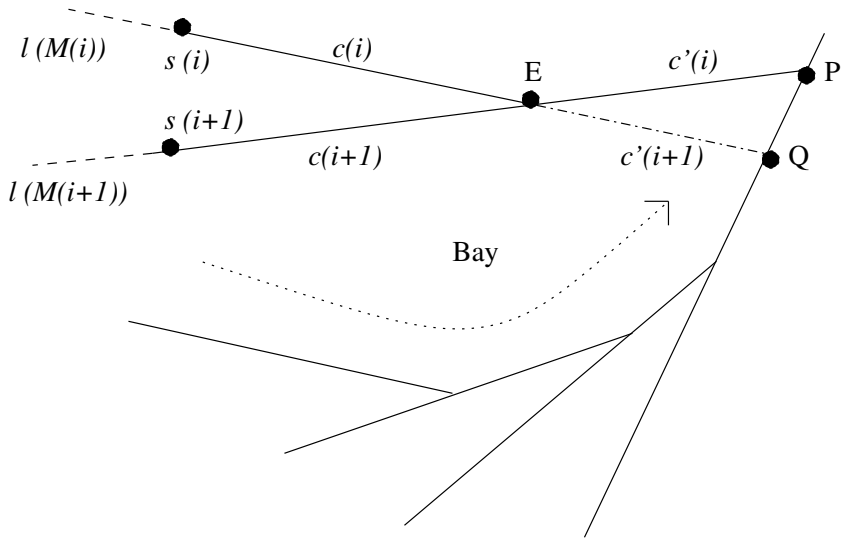

Figure 6: Updating the Horizon Tree

Assume that an elementary step is done after passing point E, which is the common right endpoint of edge c_i and c_{i+1}. Let s_i and s_{i+1} be the left endpoints of c_i and c_{i+1} respectively. At first, HTU[M[i]] contains the segment $\overline{s_iE}$ and HTU[M[$i+1$]] conatins the segment $\overline{s_{i+1}P}$. After sweeping over E, HTU[M[i]] should contain $\overline{EP}$ and HTU[M[$i+1$]] should contain $\overline{EQ}$. All other entries of HTU remain the same. Updating HTU[M[i]] takes only constant time since we already have E and P. To update HTU[M[$i+1$]], we need to traverse the bay till line $l_{M[i]}$ hits the bay. See figure 6.

To see the time complexity, consider the following charging system: For each edge traversed, we charge a unit cost to the node x corresponding to the elementary step. If some where later, an elementary step at node z makes the edge that charges x invisible from x, then we'll transfer the one unit charge to node z.

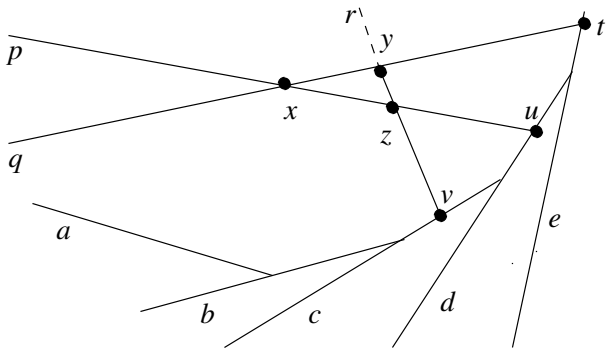

Figure 7: The elementary step at z creates edge zv, making x invisible from d.

For example, consider the arrangement in figure 7. First there is an elementary step at x, which modified the corresponding HTU entries from $\overline{qt}$ and $\overline{px}$ to $\overline{xt}$ and $\overline{xu}$ respectively. So, each of edge a, b, c and d charge one unit to x. Next elementary step occurs at point y, which changes the involving HTU entries from $\overline{xt}$ and $\overline{ry}$ to $\overline{yt}$ and $\overline{yz}$. Finally, the elementary step at point z changes HTU entries from $\overline{xu}$, $\overline{yz}$ to $\overline{zu}$ and $\overline{zv}$. The newly created edge $\overline{z}v$ makes edge d invisible from node x, the charge due to traversing d is transferred from node x to node z.

Lemma 3.1 *At the end of the algorithm each vertex is charged at most once for every edge on an incident region for the HTU (HTL) computation.*

Proof: A vertex v is charged only during the excecution of the elementary step at v. It gets one charge for every edge traversed, each of which is currently visible from it. It also gets one charge for each edge of the same region that is seperated by the current edge from its old vertex. If sometime later, an elementary step makes the edge invisible from v, then this edge will not be in the same region with v and the charge of the edge will be transferred to some other vertex. So, at the end of the algorithm, only the edges that is in the incident region of v will charge v and each of them can charge v at most once.

Lemma 3.2 $\sum_{R \in l} \mid R \mid = O(n)$.

Proof: The idea of this proof is to charge each edge of the faces that touch l to vertices on l and bound the number of charges of each vertex. Since the number of vertices on l is $O(n)$, we have $\sum_{R \in l} \mid R \mid = O(c * n) = O(n)$.

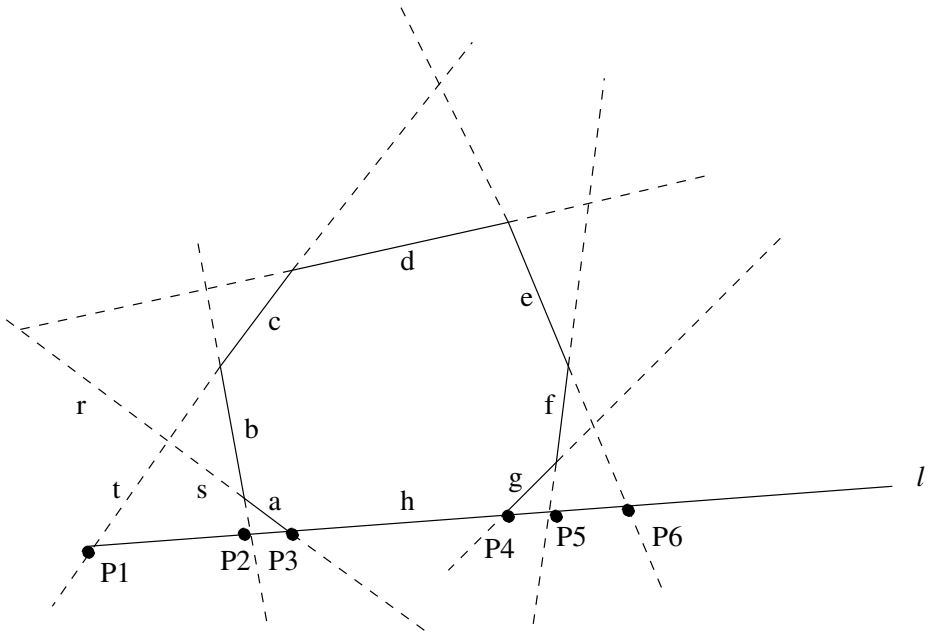

Figure 8: Edge Charges

Let $l(e_i)$ denotes the line that contains edge e_i. Consider the following charging system:

For each face F adjacent to line l and above l
For each edge e_i arranged in counter clockwise order except the ones touching l
If $l(e_i)$ intersects l to the left of F
then charge one cost to the intersection point of $l(e_{i-1})$ with l
else charge one cost to the intersection point of $l(e_{i+1})$ with l

For example, in figure 8, P_1 gets 2 charges, one from edge d, one from edge r. P_2 gets 2 charges from c and s. P_3 gets one charge from b, etc.

Similarly, for each face below l, we charge the vertices on l in the similar way. For each vertex of l, the number of charges it gets is less than or equal to 4 - 2 from edges of the face above l, and 2 from those below l. Therefore, there are $O(n)$ edges that get charges.

Now, let's see the edges that don't get any charge: For each vertex v on l, there are two edges that have v as its right or left endpoint and don't get any charge (for example, edges a and g in the figure). Also, the edges that lie on l do not get any charge either. There are $O(2n + n) = O(n)$ such edges altogether.

So, $\sum_{R \in l} \mid R \mid = O(4n + 2n + n) = O(n)$.

Lemma 3.3 $\sum_{R \in H} \mid R \mid^2 = O(n^2)$.

Proof: $\sum_{l \in H} \sum_{R \in l} \mid R \mid = O(n^2)$ and
$O(n^2) = \sum_{R \in H} \mid R \mid^2$ from 3.2.

Theorem 3.4 *The total cost of updating HTU (or HTL) through all the elementary steps is $O(n^2)$.*

Proof: From lemma 3.1 we know that an edge e charges a vertex only if they are in the same region and may do so only once. Consider region R in the arrangement. Each of its vertices can get at most $\mid R \mid$ charges, and there are $\mid R \mid$ vertices in R. So, there are at most $\mid R \mid^2$ charges associate with R. Summing this over all regions we get $\sum_{R \in H} \mid R \mid^2 = O(n^2)$ by lemma 3.3.

Hence all the topological sweep can be carried out in $O(n^2)$ time. As for the space requirement, all the data structures maintained (6 arrays) are of linear size. So, space requirement is $O(n)$.

References

[1] H. Edelsbrunner and L.J. Guibas, Topologically Sweeping an Arrangement. *J. of Computer and System Sciences*, 38:165-194, 1989

[2] D. Dobkin and D. L. Souvaine, "Computational Geometry - A User's Guide", Chapter 2 of *Algorithmic and Geometric Aspects of Robotics*, J.T.Schwartz and C.K.Yap.

[3] Class Notes of Computational Geometry, Spring 1990.

[4] Class Notes of Computational Geometry, Spring 2003.