Uno de los problemas básicos en Geometría Computacional es el de encontrar intersecciones de objetos.

→ DCEL.

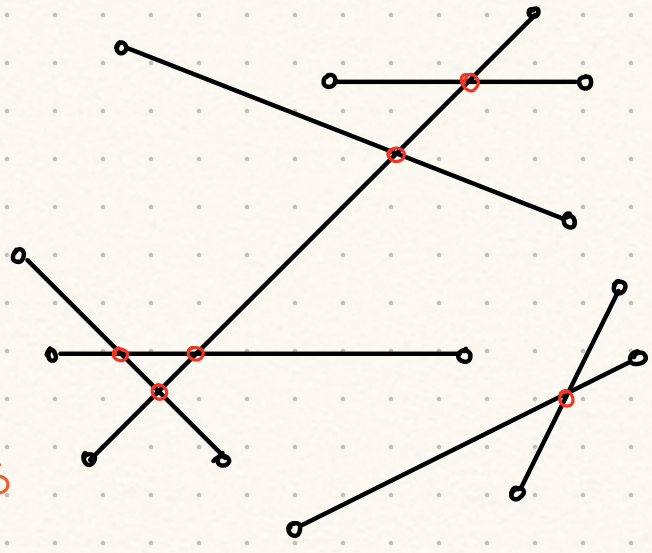
Recordemos el problema:

Dado: Un conjunto $S = \{s_1, \ldots, s_n\}$ de segmentos en el plano

Deseamos: Reportar las parejas de segmentos que se intersectan. → los puntos de intersección.

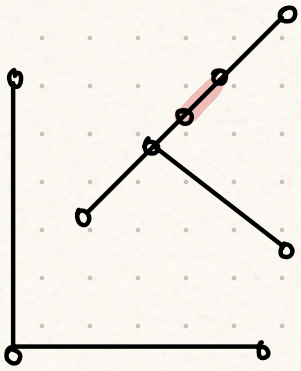
Cada s_i está representado por sus extremos. + $O(k)$ espacio.

Supondremos posición general.



$k = 6$

posición general si esto no ocurre.



no hay geometría

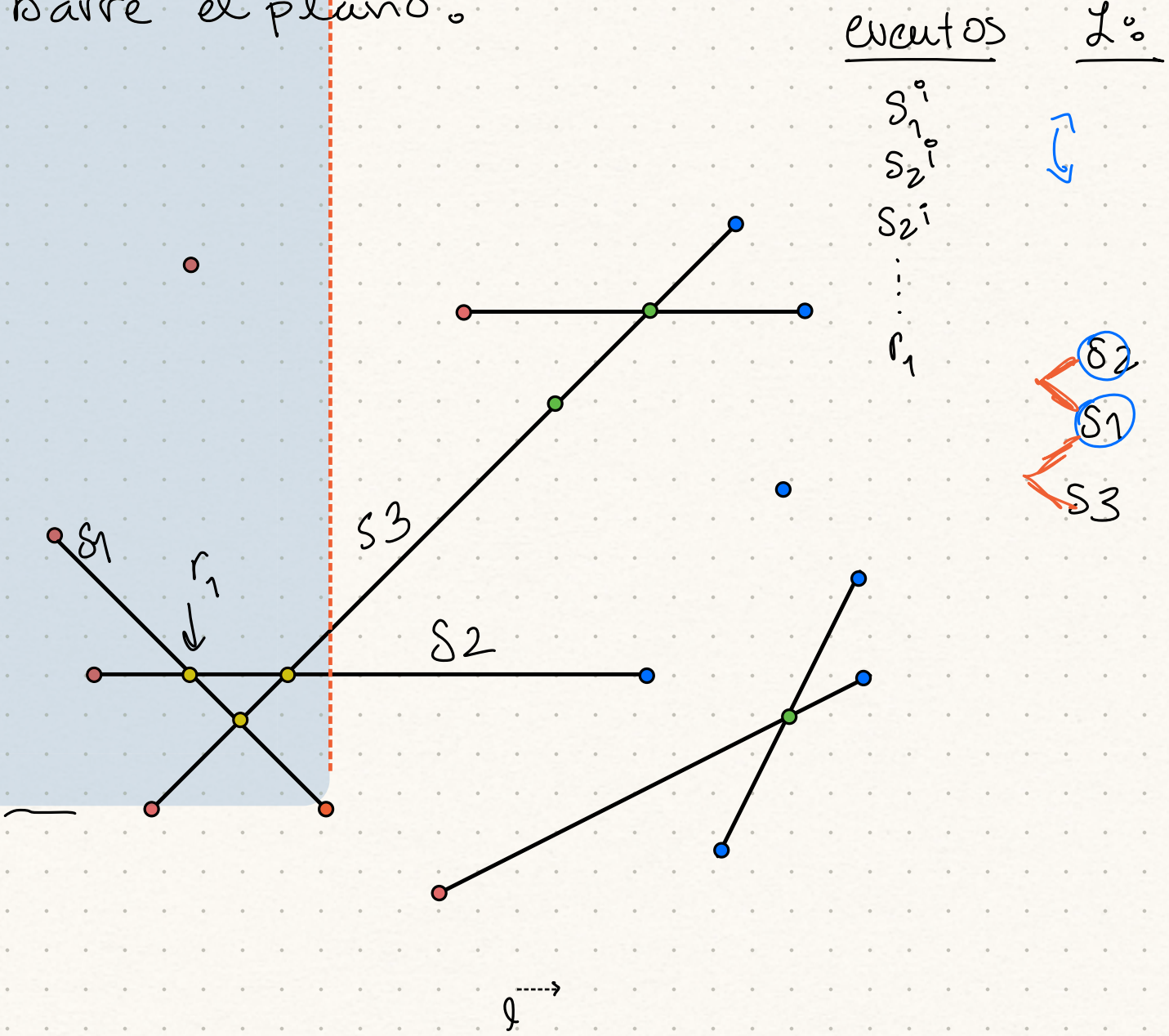
algoritmo combinatorio.

Sea k el número de intersecciones: $0 \leq k \leq \binom{n}{2}$.

Deseamos dar un algoritmo output sensitive: su complejidad depende del tamaño de la entrada y del tamaño de la salida. $T(n,k)$

Barrido de recta.

Simulamos el comportamiento de una recta que barre el plano.

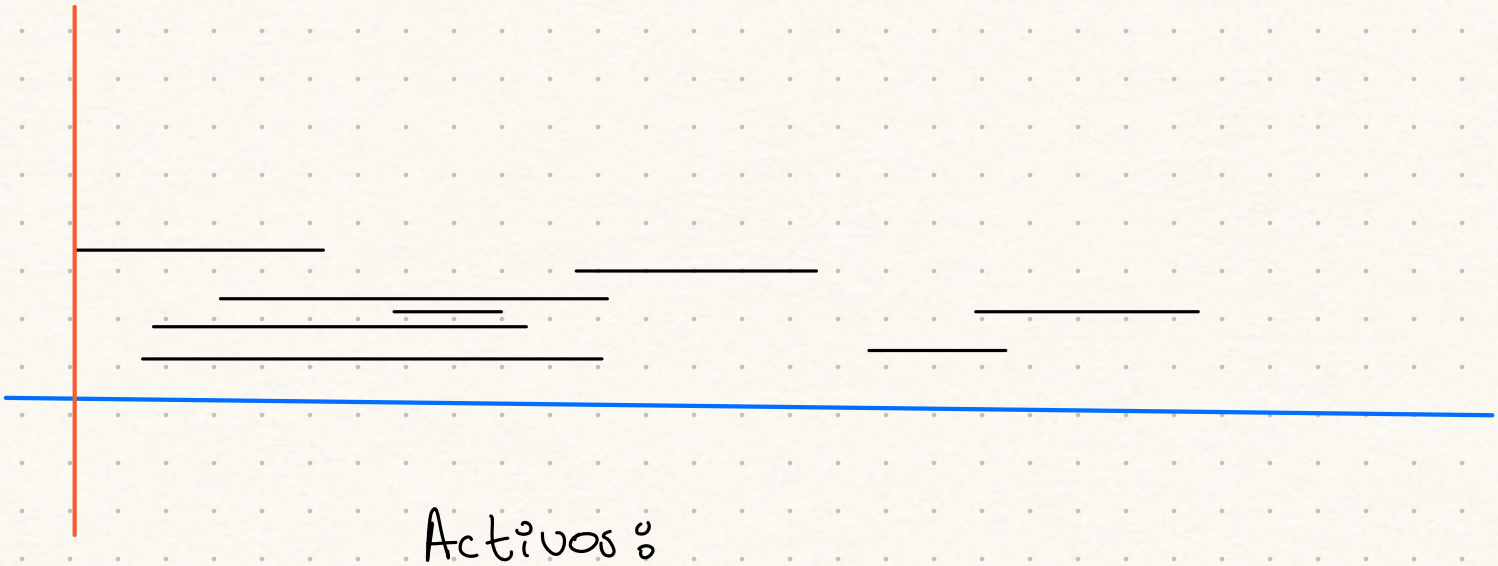


Basta con almacenar el orden de intersección de l con el conjunto de segmentos.

Eventos:

- extremos (punto inicial, punto final) — anteman o
- puntos de intersección. — "on the fly"

Pensemos por un momento el caso 1D:



x_0

Activos:

↓ $S1, S2, S3, S4,$

~~$S1$~~$, S2, S3, S4,$

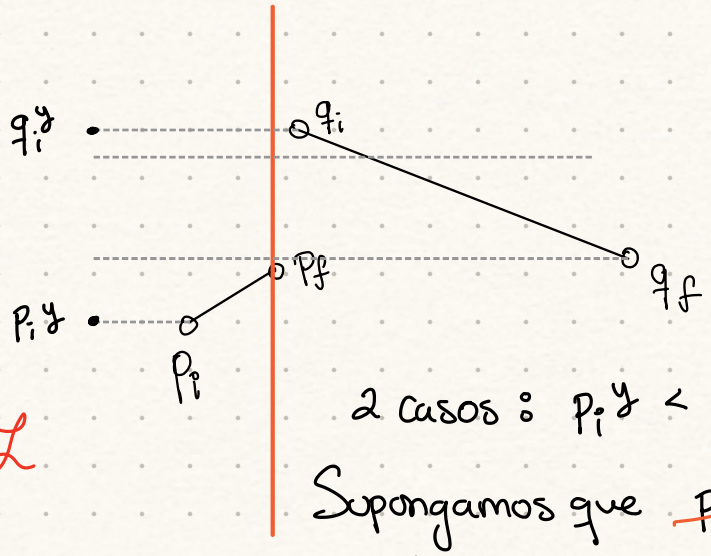
~~$S1$~~$, S2, S3, S4, S_5$

$S2, S5$
$S3, S5$
$S4, S2$
} out

1D:

Condición de intersección: ambos segmentos estén activos al mismo tiempo

2D:



$q_i \neq p_i \neq q_f \neq p_f$

2 casos: $p_i^y < q_i^y$ ó $q_i^y < p_i^y$

Supongamos que ~~$p_i^y < q_i^y$~~. $p_i^x < q_i^x$

Si los segmentos no están activos almismo tiempo en L, entonces: $p_f^x < q_i^x$

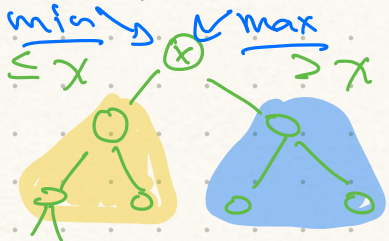
Si esto sucede entonces no se intersectan.

Estructuras de datos.

① Eventos: Puntos $(x,y) \rightarrow x,y \in \mathbb{R}$

① punto inicial
② punto final
③ punto de intersección

Priority queue Q. (¿Qué es? – heap ó – arbol binario balanceado)



Operaciones:

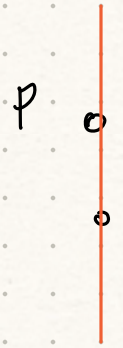
$O(\lg n)$
- extraer
- insertar
- verificar si el elemento está en Q.

Orden de prioridad:

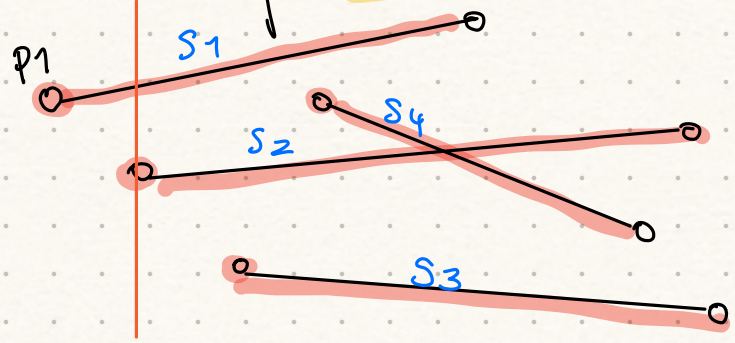
Si p y q son dos eventos, tenemos:

$p \prec q$ sii $p_x < q_x$ (p tiene prioridad sobre q.)

ó $p_x = q_x$ y $p_y > q_y$

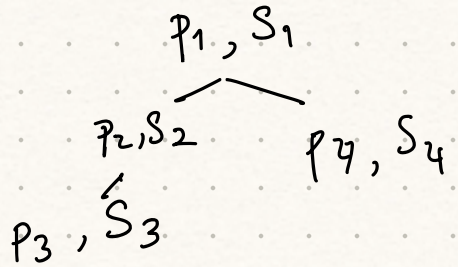


Con cada evento p almacenamos también los segmentos que inician en p.



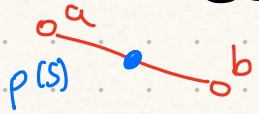
p=(x,y)
s=(p,q)

menor ó igual ↙ ○ ↘ mayor



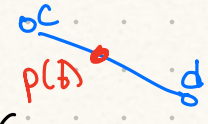
¿Cómo calculamos el punto de intersección de dos segmentos?

Recordemos que un punto sobre un segmento $\overline{ab}$ se escribe como:



$$p(s) = (1-s)a + sb, \text{ con } 0 \le s \le 1$$

Un punto en $\overline{cd}$ se escribe como:

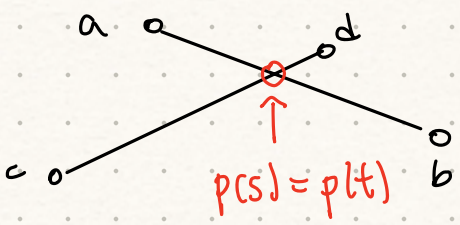


$$q(t) = (1-t)a + tb, \text{ con } 0 \le t \le 1$$

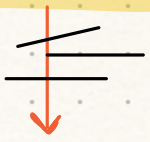
Entonces, $\overline{ab}$ y $\overline{cd}$ se intersectan si $\exists$ s y t tq:

Resolver en $\mathbb{R}^2$.

$$\begin{cases} (1-s)a_x + sb_x = (1-t)c_x + td_x & (1) \\ (1-s)a_y + sb_y = (1-t)c_y + td_y & \\ 0 \le s \le 1, \quad 0 \le t \le 1 & (2) \end{cases}$$



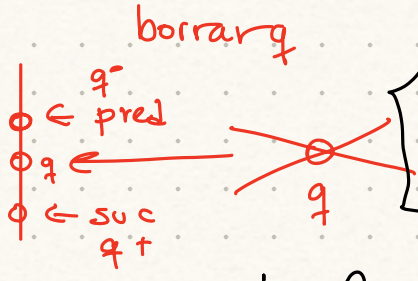
② Estatus (recta): los segmentos que intersectan a la recta, ordenados de arriba a abajo.



No vamos a almacenar el punto de intersección sino el orden.

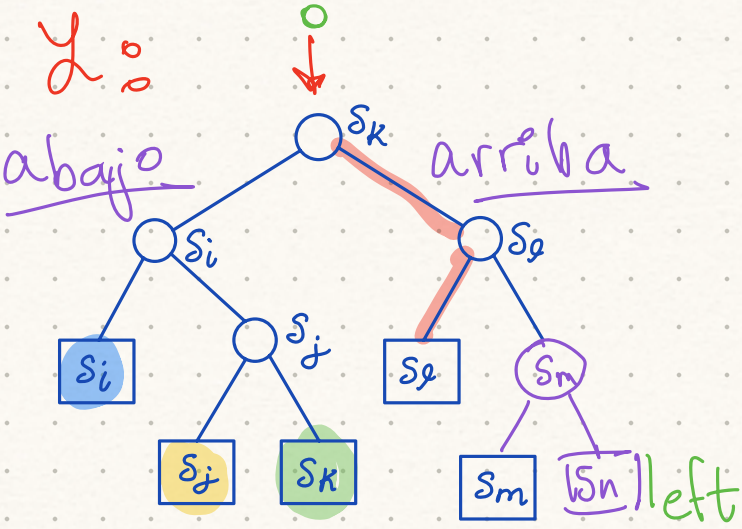
- Denotemos la estructura como $\mathcal{L}$.
- la estructura debe ser dinámica
- Hay un orden bien establecido en los segmentos

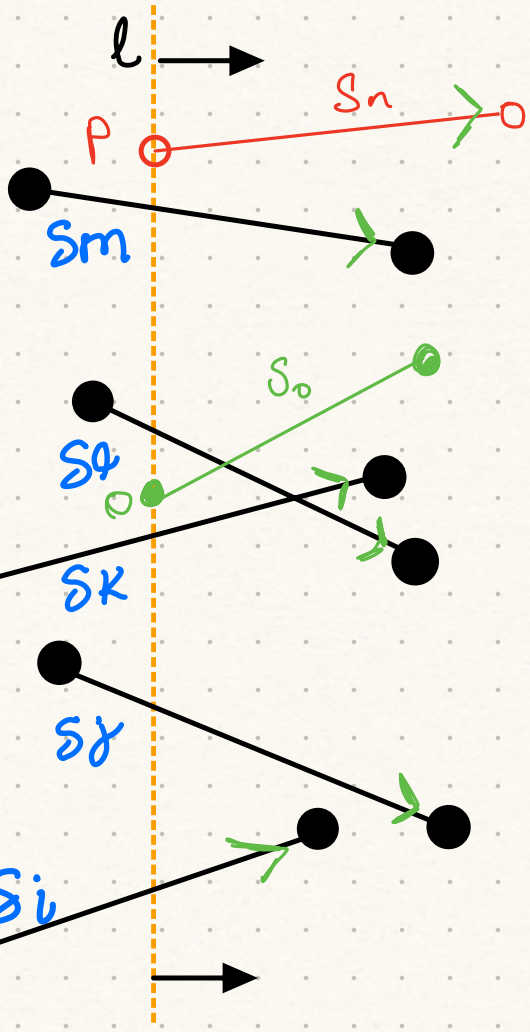
Operaciones:
- insertar
- eliminar
- swap → Corner.



el intercambio únicamente ocurre entre el predecesor y el sucesor de q

⇒ Árbol binario balanceado





s^- → predecesor

s^+ → sucesor

sucesor

$s_n, s_m, s_l, s_o, s_k, s_j, s_i$

Insertar p en Y:

Complejidad

de extremos

Eventos: $2n + k$ ← # de intersecciones

$\Rightarrow O(\lg n(2n+k))$

$= O(n \lg n + k \lg n)$ no es óptimo

¿Espacio?

$\mathcal{L}$: guarda cada segmento solo una vez $\Rightarrow O(n)$

Q: $O(n+k)$

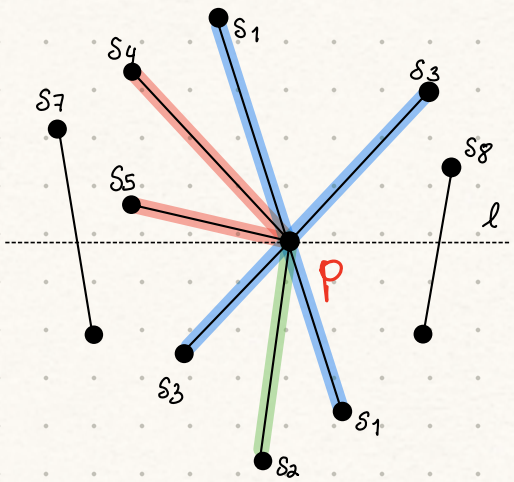
→ Se puede mejorar si únicamente almacenamos los puntos de intersección de segmentos que son adyacentes actualmente.

$\Rightarrow O(n)$

Casos degenerados

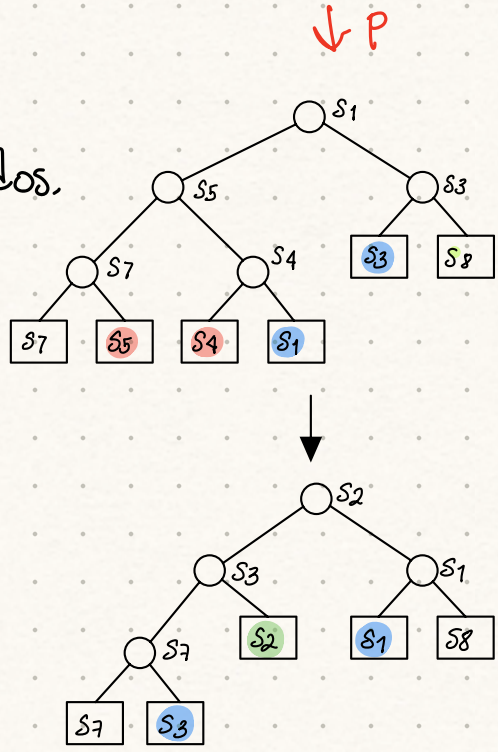
p es un extremo final de s_6 & s_4
es un extremo inicial de s_2

p es un punto de intersección de todos.



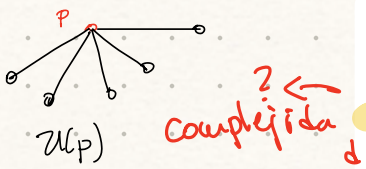
$L(p) = \{s_5, s_4\}$
$U(p) = \{s_2\}$
$C(p) = \{s_7, s_3\}$

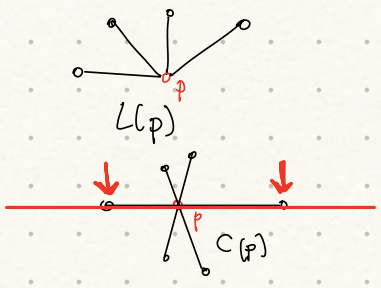
Estatus ($\mathcal{L}$):



Algorithm FINDINTERSECTIONS(S)
Input. A set S of line segments in the plane.
Output. The set of intersection points among the segments in S, with for each intersection point the segments that contain it.

1. Initialize an empty event queue $\mathcal{Q}$. Next, insert the segment endpoints into $\mathcal{Q}$; when an upper endpoint is inserted, the corresponding segment should be stored with it.
2. Initialize an empty status structure $\mathcal{T}$.
3. **while** $\mathcal{Q}$ is not empty
4. **do** Determine the next event point p in $\mathcal{Q}$ and delete it.
5. HANDLEEVENTPOINT(p)





HANDLEEVENTPOINT(p)
1. Let $U(p)$ be the set of segments whose upper endpoint is p; these segments are stored with the event point p. (For horizontal segments, the upper endpoint is by definition the left endpoint.)
2. Find all segments stored in $\mathcal{T}$ that contain p; they are adjacent in $\mathcal{T}$. Let $L(p)$ denote the subset of segments found whose lower endpoint is p, and let $C(p)$ denote the subset of segments found that contain p in their interior.
3. **if** $L(p) \cup U(p) \cup C(p)$ contains more than one segment
4. **then** Report p as an intersection, together with $L(p)$, $U(p)$, and $C(p)$.
5. Delete the segments in $L(p) \cup C(p)$ from $\mathcal{T}$.
6. Insert the segments in $U(p) \cup C(p)$ into $\mathcal{T}$. The order of the segments in $\mathcal{T}$ should correspond to the order in which they are intersected by a sweep line just below p. If there is a horizontal segment, it comes last among all segments containing p.
7. (∗ Deleting and re-inserting the segments of $C(p)$ reverses their order. ∗)
8. **if** $U(p) \cup C(p) = \emptyset$
9. **then** Let s_l and s_r be the left and right neighbors of p in $\mathcal{T}$.
10. FINDNEWEVENT(s_l, s_r, p)
11. **else** Let s' be the leftmost segment of $U(p) \cup C(p)$ in $\mathcal{T}$.
12. Let s_l be the left neighbor of s' in $\mathcal{T}$.
13. FINDNEWEVENT(s_l, s', p)
14. Let s'' be the rightmost segment of $U(p) \cup C(p)$ in $\mathcal{T}$.
15. Let s_r be the right neighbor of s'' in $\mathcal{T}$.
16. FINDNEWEVENT(s'', s_r, p)