Algorithms
Chapter 33
Computational Geometry

Associate Professor: Ching-Chi Lin
林清池 副教授

chingchi.lin@gmail.com

Department of Computer Science and Engineering
National Taiwan Ocean University

Outline

- **Line-segment properties** 線段的性質
- Determining whether any pair of segments intersects 判断平面上線段是否相交
- Finding the convex hull 找凸包
- Finding the closest pair of points 平面上找距離最近兩点

Overview

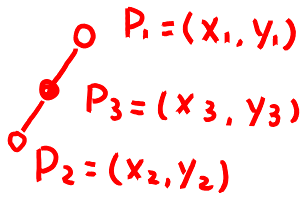
- **Computational geometry**: study algorithms for solving geometric problems such as 計算几何学解决一些几何問題:
  - computer graphics, 电腦图学
  - robotics, 机器人学
  - VLSI design, and 超大積体电路設計
  - computer aided design. 电腦輔助設計

- In this chapter, each input object is represented as a set of points $\{p_1, p_2, p_3, \ldots\}$, where each $p_i = (x_i, y_i)$ and $x_i, y_i \in \mathbf{R}$.
  - For example, an n-vertex polygon $P = \langle p_0, p_1, p_2, \ldots, p_{n-1} \rangle$.

    假設有n個点,每一個点 $P_i = (x_i, y_i)$ 為平面上一点

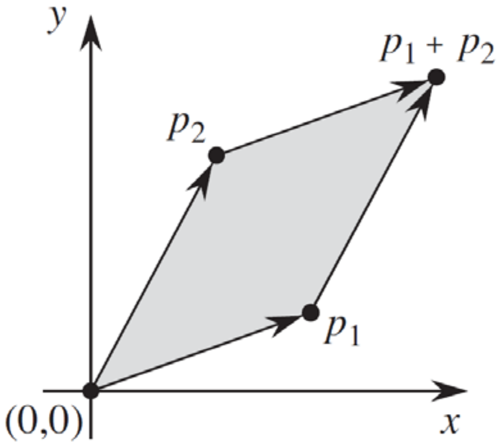
Line-segment properties

- A **convex combination** of two distinct points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is any point $p_3 = (x_3, y_3)$ such that for some α in the range $0 \le \alpha \le 1$, we have
  - $x_3 = \alpha x_1 + (1 - \alpha)x_2$, and
  - $y_3 = \alpha y_1 + (1 - \alpha)y_2$.



- We also write that $p_3 = \alpha p_1 + (1 - \alpha)p_2$.
- The **line segment** $\overline{p_1 p_2}$ is the set of convex combinations of p_1 and p_2.
- We call p_1 and p_2 the **endpoints of** segment $\overline{p_1 p_2}$.
- If p_1 is the **origin** (0, 0), then we can treat the **directed segment** $\overrightarrow{p_1 p_2}$ as the **vector** p_2. P_1如果在原点, $\overrightarrow{P_1P_2}$可視為向量 P_2

Cross products



$|P_1 \times P_2$ 外積$|$

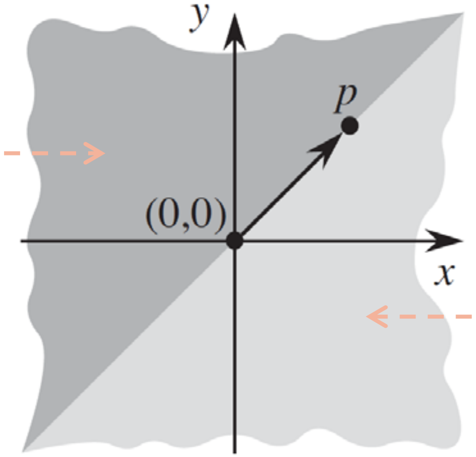
$=$所張開平行四辺形面積

- Consider vectors p_1 and p_2. The **cross product** $p_1 \times p_2$ of p_1 and p_2 is the signed area of the parallelogram formed by the points (0, 0), p_1, p_2, and $p_1 + p_2 = (x_1 + x_2, y_1 + y_2)$.
- An equivalent definition:

$$\begin{aligned} p_1 \times p_2 &= \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \\ &= x_1 y_2 - x_2 y_1 \\ &= -p_2 \times p_1 . \end{aligned}$$

P_1到P_2是順時鐘或逆時鐘

Clockwise, counterclockwise, or collinear ?



counterclockwise

clockwise

$P_1 \times P_2$:

① < 0 ⇒ P_1到P_2順時鐘

② > 0 ⇒ P_1到P_2逆時鐘

③ = 0 ⇒ 共線

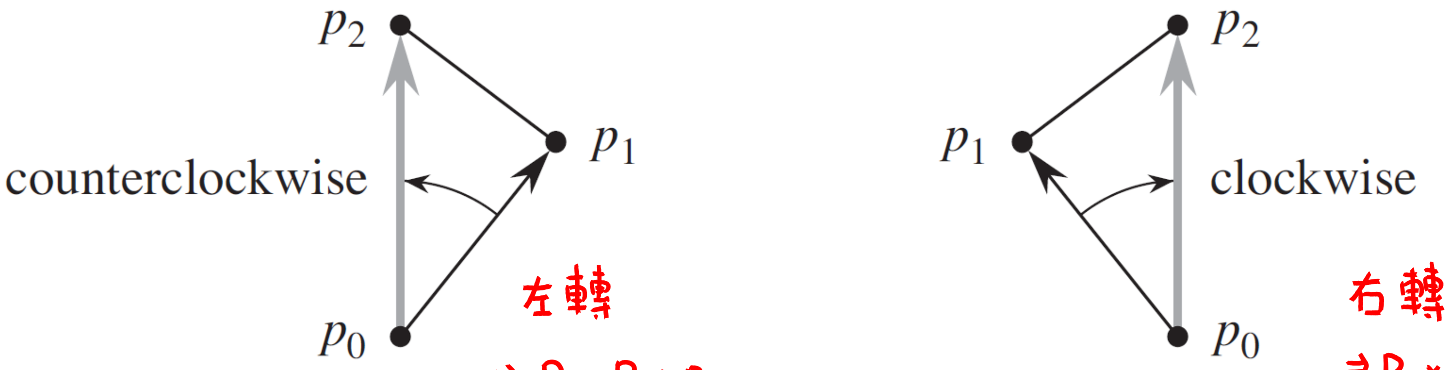
- **Question 1**: Given two vectors p_1 and p_2, is p_1 clockwise from p_2 with respect to their common endpoint p_0? If $p_1 \times p_2$ is
  - **positive**, then p_1 is clockwise from p_2.
  - **negative**, then p_1 is counterclockwise from p_2.
  - **0**, then the vectors are **collinear**, pointing in either the same or opposite directions.

Turn left or right ?

$P_1 \times P_2$:

① < 0 ⇒ 順時鐘　③ = 0 ⇒ 共線

② > 0 ⇒ 逆時鐘



左轉

⇒ $P_1 \times P_2 > 0$

右轉

⇒ $P_1 \times P_2 < 0$

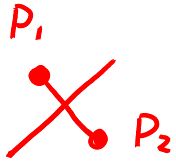
- **Question 2**: Given two line segments $\overline{p_0p_1}$ and $\overline{p_1p_2}$, if we traverse $\overline{p_0p_1}$ and then $\overline{p_1p_2}$, do we make a left turn at point *p*1?
  - Check whether $\overrightarrow{p_0p_2}$ is clockwise or counterclockwise relative to $\overrightarrow{p_0p_1}$.
  - If **counterclockwise**, the points make a left turn.
  - If **clockwise**, they make a right turn.

Whether two line segments intersect ?

- **Question 3**: Do line segments $\overline{p_0p_1}$ and $\overline{p_1p_2}$ intersect ?

穿越

- A segment $\overline{p_1p_2}$ **straddles** a line if point p_1 lies on one side of the line and point p_2 lies on the other side.
  - A boundary case arises if p_1 or p_2 lies directly on the line.

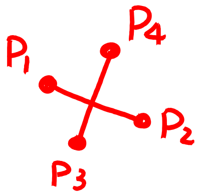


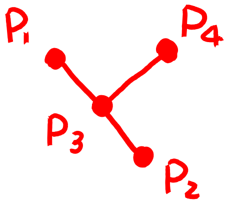
特例: P1或P2在線上

straddle: P1和P2在線兩端

- Two line segments intersect if and only if either (or both) of the following conditions holds:
  - Each segment straddles the line containing the other.
  - An endpoint of one segment lies on the other segment. (This condition comes from the boundary case.)

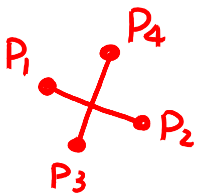
線段相交: ① 發生 straddle
② 線段A端點落在B線段上





判斷相交:
O(1)時間可完成

Pseudocode



穿越成立: ①和②成立

① P1和P2分別在 $\overline{P_3P_4}$ 兩邊

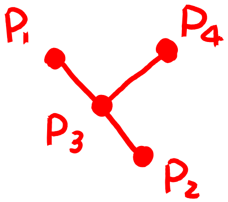
② P3和P4分別在 $\overline{P_1P_2}$ 兩邊

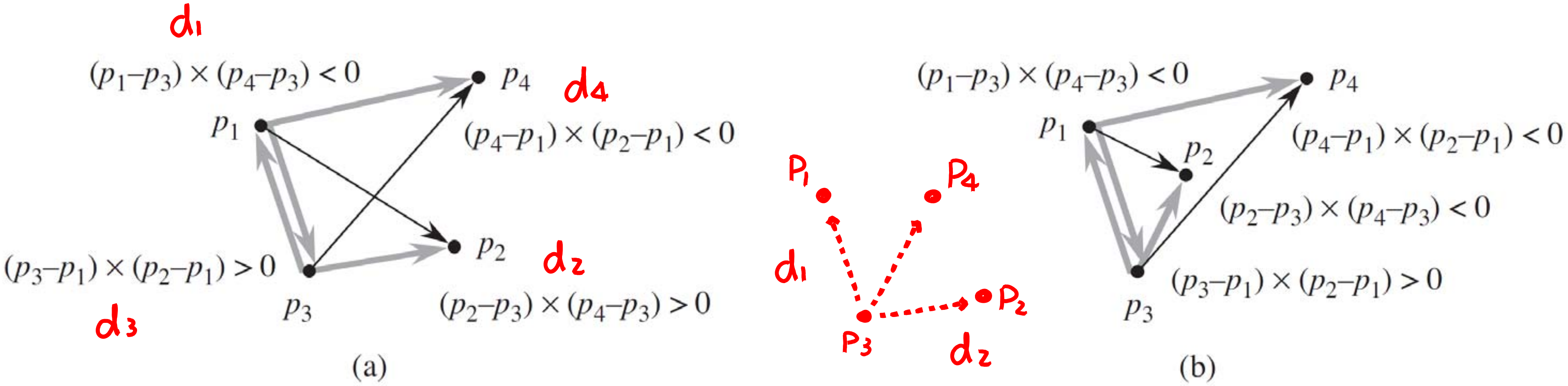
```
SEGMENTS-INTERSECT($p_1, p_2, p_3, p_4$)
    $d_1 \leftarrow$ DIRECTION($p_3, p_4, p_1$)
    $d_2 \leftarrow$ DIRECTION($p_3, p_4, p_2$)
    $d_3 \leftarrow$ DIRECTION($p_1, p_2, p_3$)
    $d_4 \leftarrow$ DIRECTION($p_1, p_2, p_4$)
穿越 if (($d_1 > 0$ and $d_2 < 0$) or ($d_1 < 0$ and $d_2 > 0$))① and      ⇒ $d_1 \times d_2 < 0$
        (($d_3 > 0$ and $d_4 < 0$) or ($d_3 < 0$ and $d_4 > 0$))②      ⇒ $d_3 \times d_4 < 0$
        return TRUE
共線 elseif $d_1 = 0$ and ON-SEGMENT($p_3, p_4, p_1$)
        return TRUE
    elseif $d_2 = 0$ and ON-SEGMENT($p_3, p_4, p_2$)
        return TRUE
    elseif $d_3 = 0$ and ON-SEGMENT($p_1, p_2, p_3$)
        return TRUE
    elseif $d_4 = 0$ and ON-SEGMENT($p_1, p_2, p_4$)
        return TRUE
    else return FALSE
```

```
DIRECTION($p_i, p_j, p_k$)
    return $(p_k - p_i) \times (p_j - p_i)$
```

```
ON-SEGMENT ($p_i, p_j, p_k$)
    if $\min(x_i, x_j) \le x_k \le \max(x_i, x_j)$ and
       $\min(y_i, y_j) \le y_k \le \max(y_i, y_j)$
        return TRUE
    else return FALSE
```

xy座標在線段上



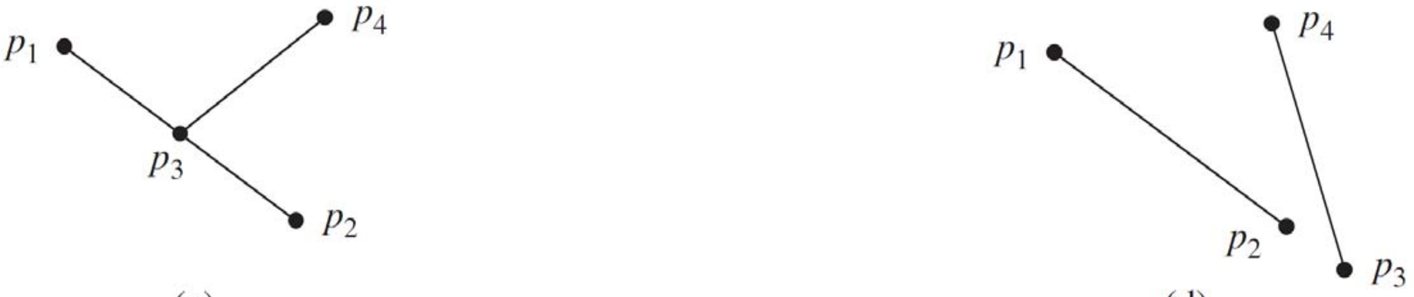


d_1
$(p_1-p_3) \times (p_4-p_3) < 0$
p_4
d_4
p_1
$(p_4-p_1) \times (p_2-p_1) < 0$
p_2
d_2
$(p_3-p_1) \times (p_2-p_1) > 0$
p_3
$(p_2-p_3) \times (p_4-p_3) > 0$
d_3
P_1
P_4
d_1
P_2
P_3
d_2
$(p_1-p_3) \times (p_4-p_3) < 0$
p_4
p_1
p_2
$(p_4-p_1) \times (p_2-p_1) < 0$
$(p_2-p_3) \times (p_4-p_3) < 0$
$(p_3-p_1) \times (p_2-p_1) > 0$
p_3

(a) (b)

① P_1 和 P_2 分别在 $\overline{P_3P_4}$ 兩边 $\Rightarrow d_1 \times d_2 < 0$

② P_3 和 P_4 分别在 $\overline{P_1P_2}$ 兩边 $\Rightarrow d_3 \times d_4 < 0$



(c) (d)

- Two line segments intersect if and only if conditions (a) or (c) holds.
- In (b), segment $\overline{p_3p_4}$ straddles the line containing $\overline{p_1p_2}$, but segment $\overline{p_1p_2}$ does not straddle the line containing $\overline{p_3p_4}$.
- In (d), p_3 is collinear with $\overline{p_1p_2}$, but it is not between p_1 and p_2. The segments do not intersect.

Outline

- Line-segment properties
- **Determining whether any pair of segments intersects**

  判断平面上線段是否相交
- Finding the convex hull
- Finding the closest pair of points

Determining if two line segments intersect ?

- This section presents an algorithm for determining whether any two line segments in a set of segments intersect.
- The algorithm uses a technique known as **sweeping**. 掃描法
- The algorithm runs in $O(n \lg n)$ time, where n is the number of segments we are given. n：線段的個數

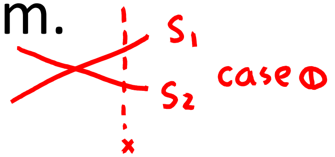
想像有一條掃描線由左至右掃

- In **sweeping**, an imaginary vertical **sweep line** passes through the given set of geometric objects, usually from left to right.
- We assume that 假設
  - no input segment is vertical; and 沒有垂直線
  - no three input segments intersect at a single point. 不會發生三條線相交於一點

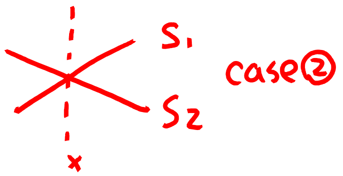
$s_1 \geq_x s_2$ if case ① 或 case ②

Ordering segments & Moving the sweep line$_{1/2}$

- Two segments s_1 and s_2, are **comparable** at x if the vertical sweep line with x-coordinate x intersects both of them.



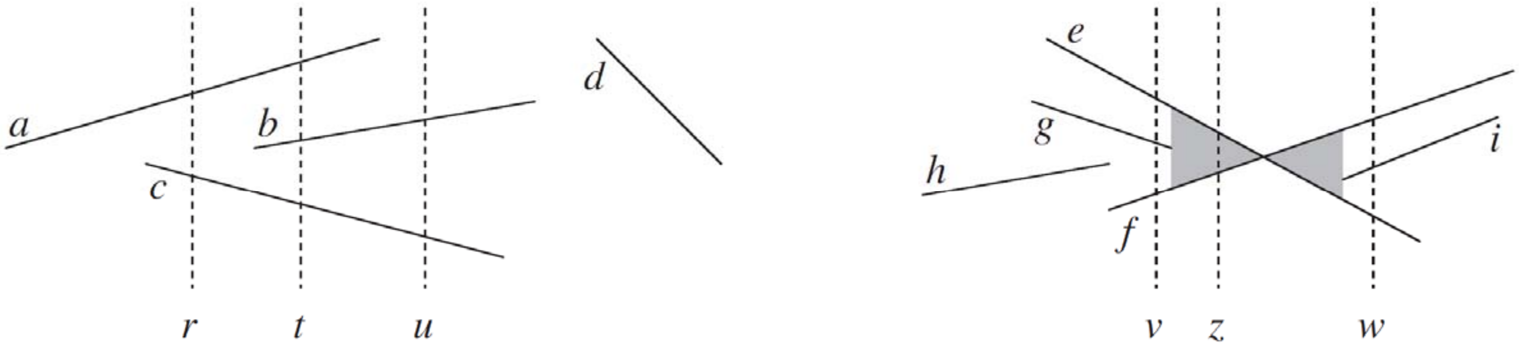
- We say that s_1 is **above** s_2 at x, written $s_1 \geq_x s_2$, if
  - the intersection of s_1 with the sweep line at x is higher than the intersection of s_2 with the same sweep line; or
  - if s_1 and s_2 intersect at the sweep line.



維護兩個資料結構

- Sweeping algorithms typically manage two sets of data:
  - The **sweep-line status** gives the relationships among the objects intersected by the sweep line. sweep-line status : 線段的上下關係
  - The **event-point schedule** is a sequence of points, called **event point**, ordered from left to right, that defines the halting positions of the sweep line. event-point schedule : 將事件點由左至右排序

Ordering segments & Moving the sweep line$_{2/2}$



(a) (b)

- In (a), we have
  - $a \geq_r c$, $a \geq_t b$, $b \geq_t c$, $a \geq_t c$, and $b \geq_u c$.
  - segment *d* is comparable with no other segment shown.

d線段和其他線段不能相比

- In (b), one can see that
  - when segments *e* and *f* intersect, their orders are reversed: we have $e \geq_v f$ but $f \geq_w e$.

當e和f線段相交，他們的順序会变

端点就是事件点

Event-point schedule & Sweep-line status

- Event-point schedule: 將端点依 x 座標由左至右排序
  - Each segment endpoint is an event point.
  - We sort the segment endpoints by increasing *x*-coordinate and proceed from left to right.

遇到 左端点 ⇒ 將線段加入 sweep-line Status
右端点 ⇒ 將線段移除 sweep-line Status

- When we encounter a segment’s
  - Left endpoint: insert the segment into the sweep-line status;
  - Right endpoint: delete the segment into the sweep-line status.
- Whenever two segments first become consecutive, we check whether they intersect.

當線段第一次彼此相鄰
⇒ 檢測是否相交

Operations for sweep-line status

- We require the following operations for sweep-line status *T*:
  - INSERT(*T*, *s*): insert segment *s* into *T*.
  - DELETE(*T*, *s*): delete segment *s* from *T*.

    回傳 s 的上一個
  - ABOVE(*T*, *s*): return the segment immediately above segment *s* in *T*.
  - BELOW(*T*, *s*): return the segment immediately below segment *s* in *T*.

    回傳 s 的下一個
- Each of the above operations can be performed in $O(\lg n)$ time using red-black trees. 以上每一個动作都只需要 O(lgn)時間
- Recall that the red-black-tree operations in Chapter 13 involve comparing keys.
  - We can replace the key comparisons by comparisons that use cross products to determine the relative ordering of two segments (see Exercise 33.2-2). 原本 key 有大小關係，這裡用 cross product 來判定線段的上下關係

Segment-intersection pseudocode

```
ANY-SEGMENTS-INTERSECT($S$)    將端点排序
    $T \leftarrow \emptyset$
    sort the endpoints of the segments in $S$ from left to right,
        breaking ties by putting left endpoints before right endpoints
        and breaking further ties by putting points with lower
        $y$-coordinates first
    for each point $p$ in the sorted list of endpoints
        if $p$ is the left endpoint of a segment $s$
            INSERT($T, s$)
            if (ABOVE($T, s$) exists and intersects $s$)
                or (BELOW($T, s$) exists and intersects $s$)
                return TRUE
        if $p$ is the right endpoint of a segment $s$
            if both ABOVE($T, s$) and BELOW($T, s$) exist
                and ABOVE($T, s$) intersects BELOW($T, s$)
                return TRUE
            DELETE($T, s$)
    return FALSE
```

$O(n\log n)$

左端点：① insert (T, s)
② 檢查是否和 above(T,s)
和 below(T,s) 相交

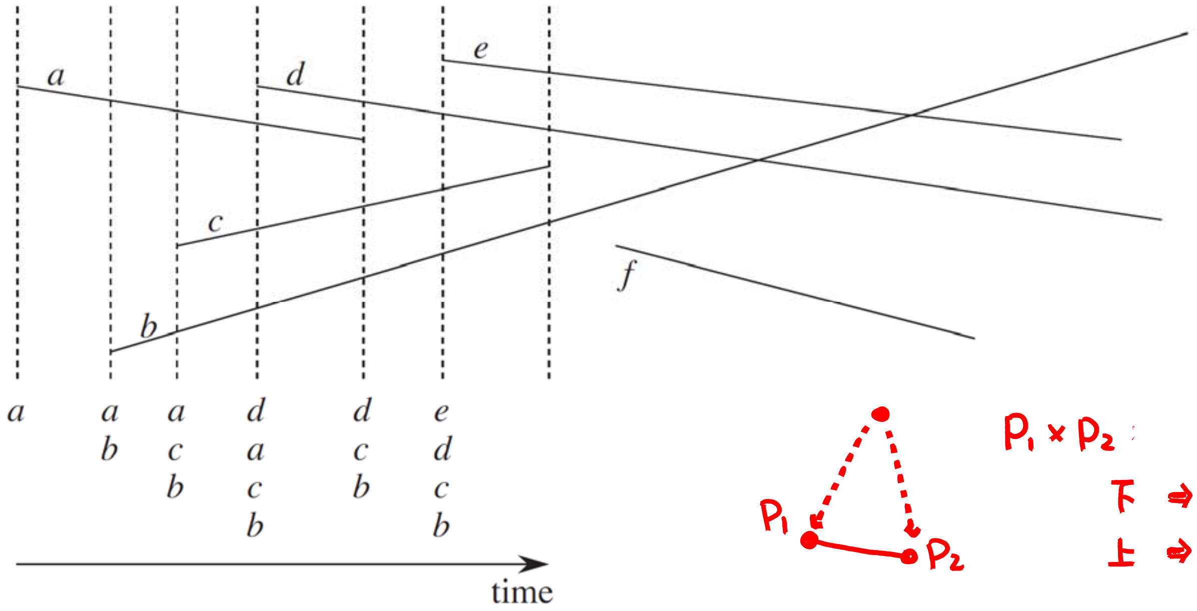
$2n\cdot(O(\log n)+O(1))$

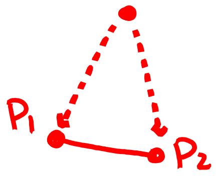
右端点：① 檢查 above(T,s)和
below(T,s)是否相交
② delete (T, s)

Time complexity: $O(n\log n)$

當線段第一次彼此相鄰
⇒檢測是否相交

The execution of ANY-SEGMENTS-INTERSECT





$P_1 \times P_2$:

下 ⇒ 顺時鐘

上 ⇒ 逆時鐘

- Each dashed line is the sweep line at an event point.
- The intersection of segments *d* and *b* is found when segment *c* is deleted. c的右端点delete時發現bd相交

Outline

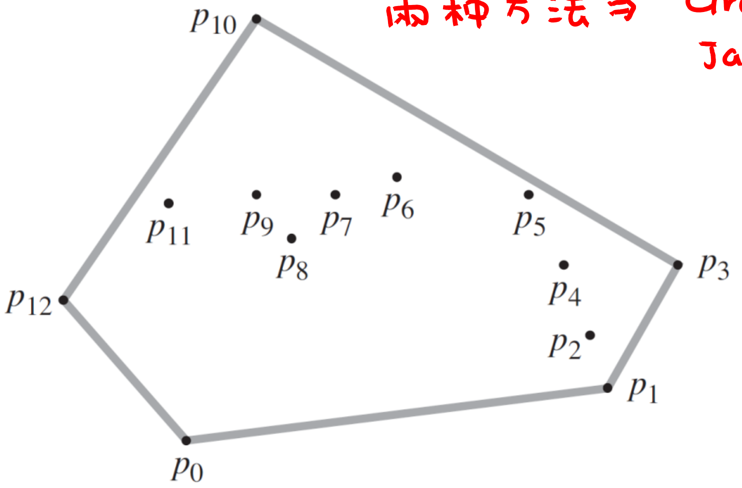
- Line-segment properties
- Determining whether any pair of segments intersects
- **Finding the convex hull**
- Finding the closest pair of points

Finding the convex hull

凸包：最小凸多边形可將
全部東西包進去

- The **convex hull** of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior.

兩种方法 ⇒ Graham's scan : O(nlgn)
Jarvis's march : O(nh)



- Two algorithms:
  - Graham's scan, runs in $O(n \lg n)$ time, n is the number of points.
  - Jarvis's march, runs in $O(nh)$ time, where h is the number of vertices of the convex hull.

Graham's scan

- Both Graham's scan and Jarvis's march use a technique called **rotational sweep**, processing vertices in the order of the polar angles. Graham's scan 和 Jarvis's march 都使用 rotational sweep 這個技巧 (旋轉式掃描)
- Graham's scan :
  - By maintaining a stack *S* of candidate points.
  - Each point of the input set *Q* is pushed once onto the stack.
  - The points that are not vertices of CH(*Q*) are eventually popped from the stack.
  - When the algorithm terminates, stack *S* contains exactly the vertices of CH(*Q*).

Graham's scan pseudocode

```
GRAHAM-SCAN(Q)
    let $p_0$ be the point in Q with the minimum y-coordinate,
        or the leftmost such point in case of a tie
    let $(p_1, p_2, \ldots, p_m)$ be the remaining points in Q,
        sorted by polar angle in counterclockwise order around $p_0$
        (if more than one point has the same angle, remove all but
        the one that is farthest from $p_0$)
    let S be an empty stack
    PUSH($p_0$, S)
    PUSH($p_1$, S)
    PUSH($p_2$, S)
    for $i \leftarrow 3$ to m
        while the angle formed by points NEXT-TO-TOP(S), TOP(S),
                and $p_i$ makes a nonleft turn
            POP(S)
        PUSH($p_i$, S)
    return S
```

$O(n)$ 找 y 軸最小的點 P_0

$O(n \log n)$

將 P_1, P_2 … P_m 以和 P_0 的角度由小到大排

一次加入一點，檢查最上面是否為左轉

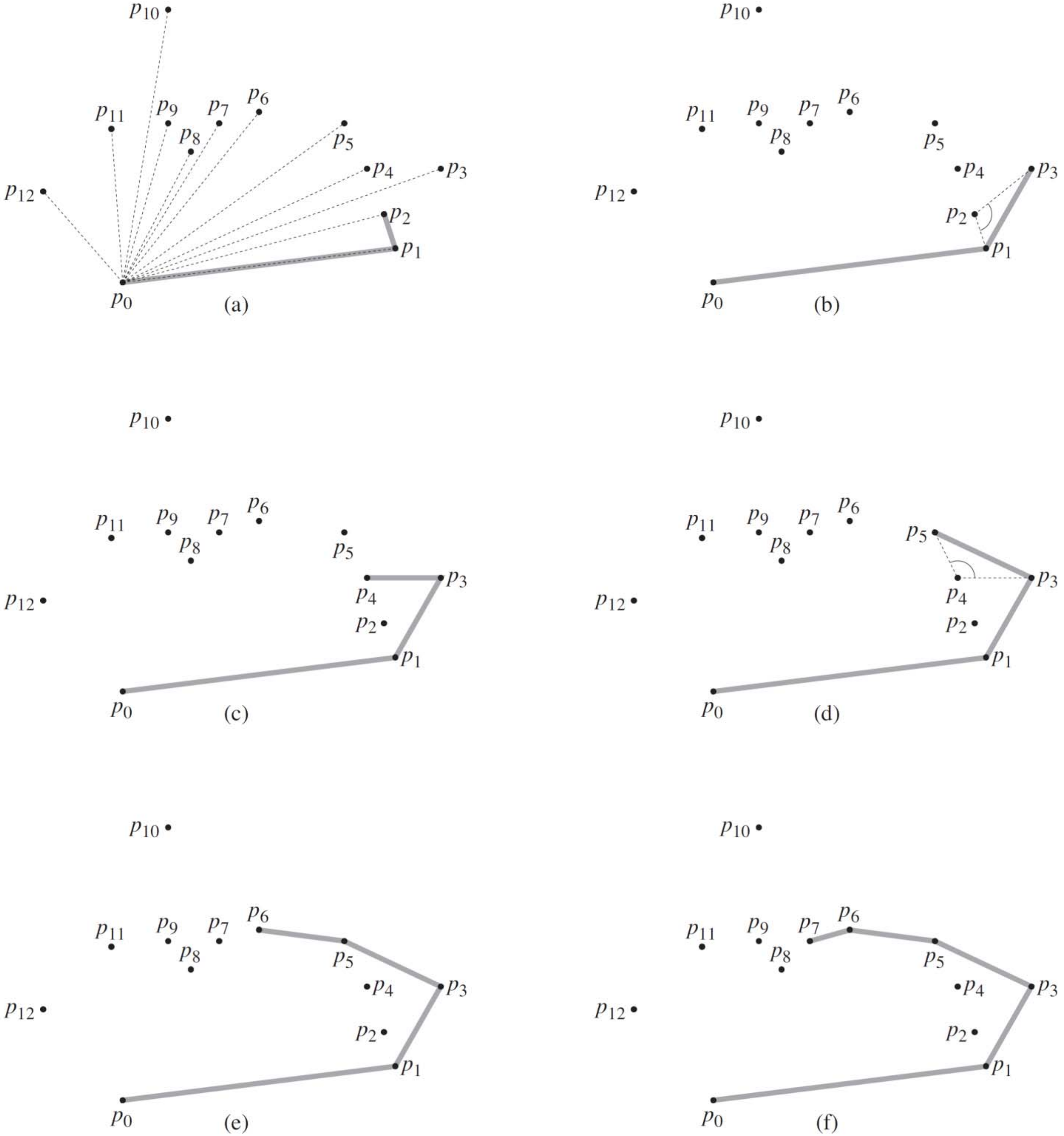
是 ⇒ OK

否 ⇒ 將第二點 delete 直至最上面三點為左轉

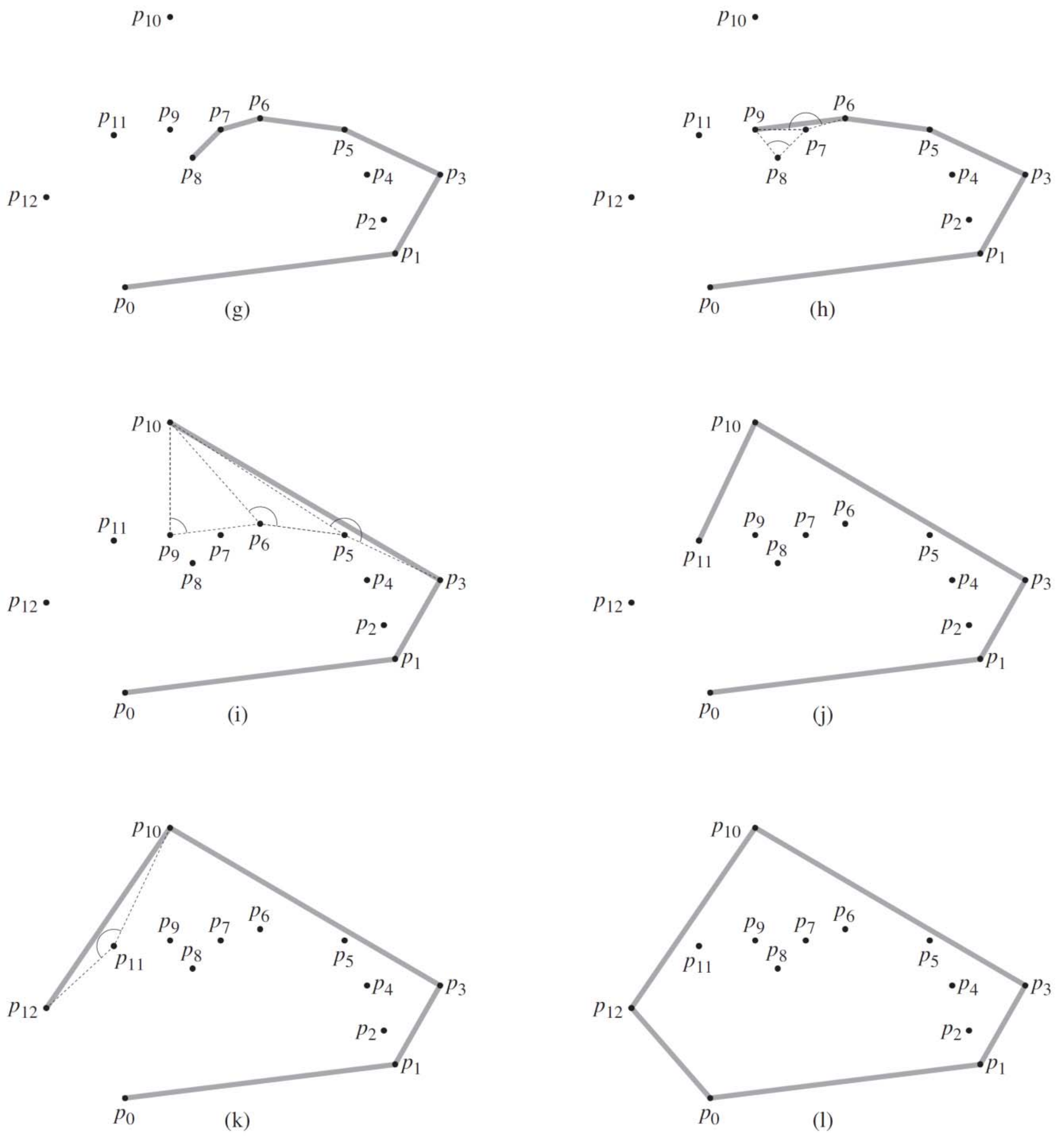
$O(1)$

$O(n)$

Time complexity: $O(n \log n)$



(a) (b) (c) (d) (e) (f)

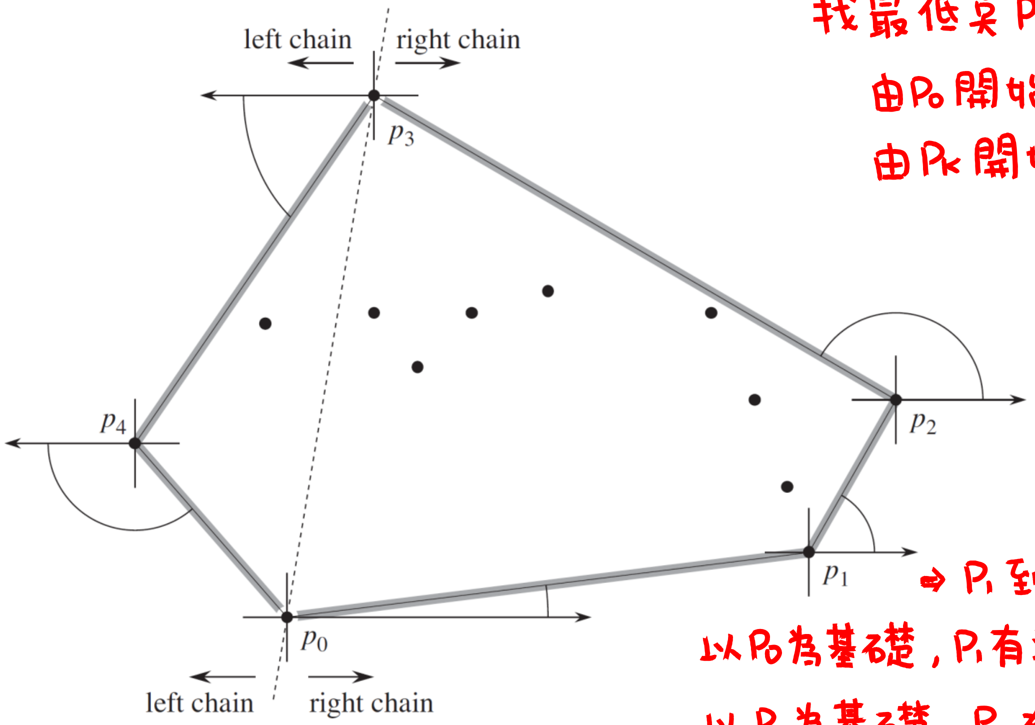


In (h), the right turn at angle $\angle p_7 p_8 p_9$ causes p_8 to be popped, and then the right turn at angle $\angle p_6 p_7 p_9$ causes p_7 to be popped.

Jarvis's march$_{1/2}$ 礼物包装法

- Jarvis's march computes the convex hull of a set Q of points by a technique known as **package wrapping** (or **gift wrapping**).
- Jarvis's march :
  - Find the lowest point p_0 and the highest point p_k. 找最低点 P_0 和最高点 P_k
  - Construct the **right chain** of CH(Q). 由 P_0 開始找 right chain
    - We start with p_0, the next convex hull vertex p_1 has the smallest polar angle with respect to p_0. 以 P_0 為基礎，P_1 有最小角度
    - Similarly, p_2 has the smallest polar angle with respect to p_1, and so on.
    - When we reach the highest vertex p_k, we have constructed the right chain of CH(Q). 以 P_1 為基礎，P_2 有最小角度…依此類推
  - Construct the **left chain** of CH(Q) similarly. 由 P_k 開始找 left chain

Jarvis's march$_{2/2}$



找最低点 P_0 和最高点 P_k

由 P_0 開始找 right chain

由 P_k 開始找 left chain

⇒ P_1 到其他点都是逆時鐘

以 P_0 為基礎，P_1 有最小角度

以 P_1 為基礎，P_2 有最小角度 ... 依此類推

- Time complexity: $O(nh)$, where h is the # of vertices of CH(Q).
  - Each comparison between polar angles takes $O(1)$ time.

Outline

- Line-segment properties
- Determining whether any pair of segments intersects
- Finding the convex hull
- **Finding the closest pair of points** 平面上 找距離最近兩點

平面上找距離最近兩点

Finding the closest pair of points

- Consider the problem of finding the closest pair of points in a set Q of $n \geq 2$ points.
  - **"Closest"** refers to the usual euclidean distance: the distance between points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}\,.$$

- A **brute-force algorithm** simply looks at all the $\binom{n}{2}$ pairs of points. 暴力法：$O(n^2)$
- In this section, we shall describe a divide-and-conquer algorithm whose running time is described by the familiar recurrence $T(n) = 2T(n/2) + O(n)$. 使用 divide and conquer：$O(n\lg n)$
- Thus, this algorithm uses only $O(n \lg n)$ time.

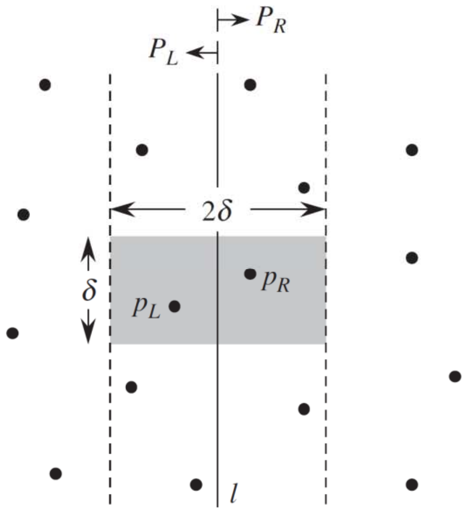
The divide-and-conquer algorithm$_{1/3}$

- The input of each recursive: x: 將P的点以x軸座標由小到大排 y: 將P的点以y軸座標由小到大排
  - $P \subseteq Q$. P: 部份点
  - X : contains all the points in P and the points is sorted by monotonically increasing x-coordinates.
  - Y : contains all the points in P and the points is sorted by monotonically increasing y-coordinates.
- If $|P| \leq 3$, perform the brute-force method. 如果 $|P| \leq 3$, 用暴力法
- If $|P| > 3$, recursive invocation carries out the divide-and-conquer paradigm as follows. 如果 $|P| > 3$, 方法如下

The divide-and-conquer algorithm$_{2/3}$

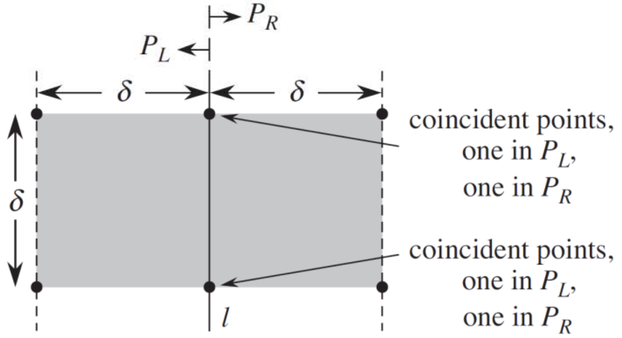
- **Divide:** 將P切成 P_L 和 P_R，以ℓ為切割線
  - Find a vertical line ℓ that bisects the point set P into two sets P_L and P_R such that $|P_L| = \lceil |P|/2 \rceil$, $|P_R| = \lfloor |P|/2 \rfloor$.
  - Divide X into arrays X_L and X_R. 將x分成 x_L 和 x_R
  - Divide Y into arrays Y_L and Y_R. 將y分成 y_L 和 y_R
- **Conquer:** 分別解 P_L 和 P_R，$\delta = \min(\delta_L, \delta_R)$
  - Let the closest-pair distances returned for P_L and P_R be δ_L and δ_R, respectively, and let $\delta = \min(\delta_L, \delta_R)$.
- **Combine:** 最小可能性 ① δ ② 一點在 P_L，一點在 P_R
  - The closest pair is either the pair with distance δ, or one point in P_L and the other in P_R whose distance is less than δ.
  - If the latter happens, both points of the pair must be within δ units of line ℓ. 如果是情形②，兩點距離一定 $< \delta$
  - To find such a pair, if one exists, the algorithm does the following:

The divide-and-conquer algorithm$_{3/3}$



(a)

对於P点而言，只有y'中
P点之後的那7点，距離有可能在δ之內，
故每一点只需檢測7次


P_R
P_L
δ
δ
δ
coincident points, one in P_L, one in P_R
coincident points, one in P_L, one in P_R
l

(b)

1. It creates an array Y', which is the array Y with all points not in the 2δ-wide vertical strip removed. y'：將y中不在2δ-wide中的移除

2. For each point p in the array Y', try to find points in Y' that are within δ units of p. (Only the **7** points in Y' that follow p need to be considered.)

3. Suppose δ' is closest-pair distance found over all pairs of points in Y'. If $\delta' < \delta$, then return δ'. Otherwise, return δ.

Implementation$_{1/2}$

- Main difficulty:
  - Ensure that arrays X_L, X_R, Y_L, and Y_R, which are passed to recursive calls, are sorted by the proper coordinate.
  - Ensure that array Y' is sorted by y-coordinate.

困難点：资結 x_L, x_R, y_L, y_R, y' 的維護

Implementation$_{2/2}$

- Method:
  - Presort the pints in *Q* by the proper coordinate to get *X* and *Y* before the first recursive call.
  - In each recursive call:
    - Divide *P* into P_L and P_R ➔ $O(n)$ time. 將y切割成yL和yR的方法
    - The following pseudocode gives the idea to get Y_L, and Y_R from *Y*.

```
$length[Y_L] \leftarrow length[Y_R] \leftarrow 0$
for $i \leftarrow 1$ to $length[Y]$    对y上每個点作檢測
    if $Y[i] \in P_L$
        then $length[Y_L] \leftarrow length[Y_L] + 1$    屬於PL放到yL
            $Y_L\,[length[Y_L]] \leftarrow Y[i]$
        else $length[Y_R] \leftarrow length[Y_R] + 1$    屬於PR放到yR
            $Y_R\,[length[Y_R]] \leftarrow Y[i]$
```

    - Similar pseudocode works for forming arrays X_L, X_R, and Y'.

Running time

將Q用x軸和y軸排序的時間

⇑

- We get $T'(n) = T(n) + O(n \lg n)$. T(n): recursive step 的時間
  - $T(n)$: the running time of each recursive step.
  - $T'(n)$: the running time of the entire algorithm.

- We can rewrite the recurrence as

$$T(n) = \begin{cases} 2T(n/2) + O(n) & \text{if } n > 3, \\ O(1) & \text{if } n \le 1. \end{cases}$$

- Thus, $T(n) = O(n \lg n)$ and $T'(n) = O(n \lg n)$.