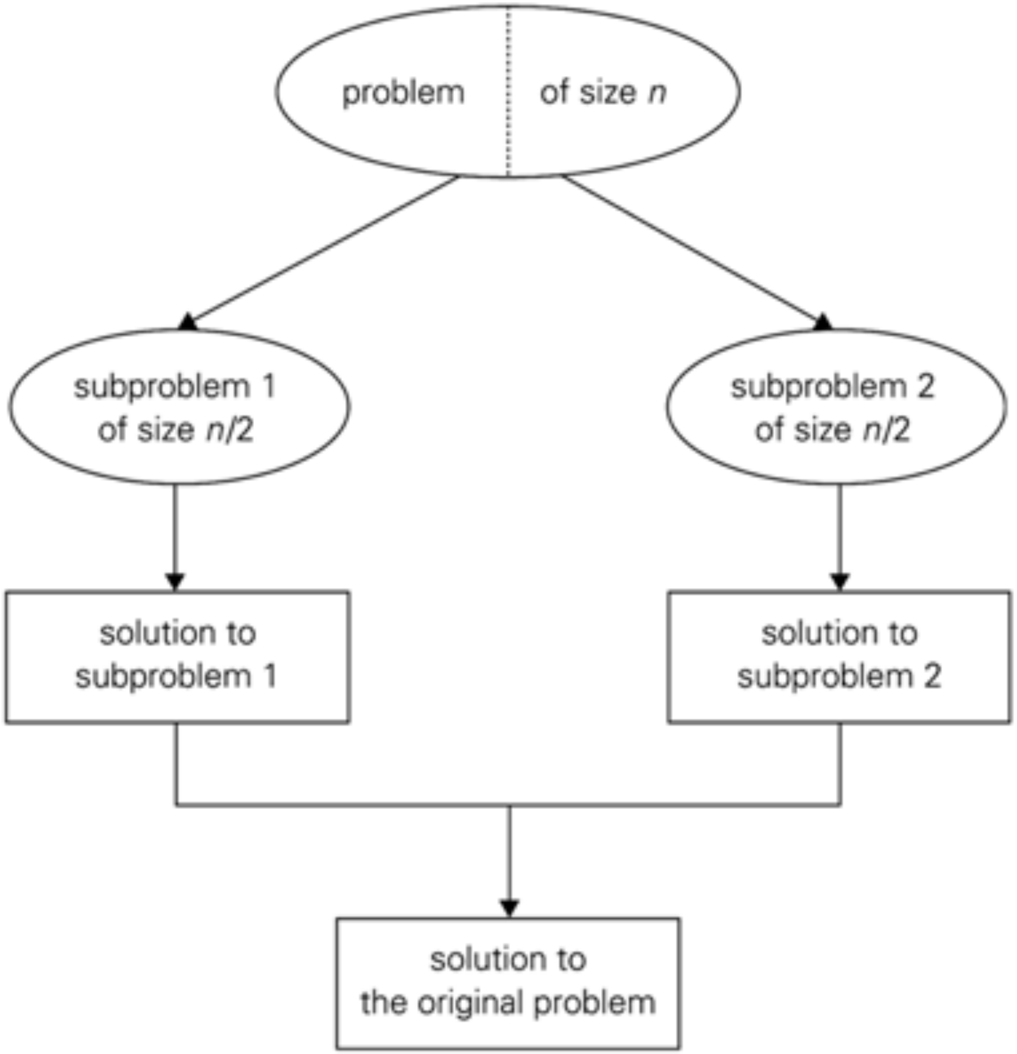
# Divide and Conquer
## Chapter 4.1 – 4.2

Mei-Chen Yeh

# General plan

1. **Divide** into several smaller instances of the same problem →把大測資切成小測資.
2. **Solve** the smaller instances →解決每個小問題.
3. **Combine** the solutions →結合小問題的答案.

# Example: a typical case



# Example 1: binary search

- Find an element in a *sorted* array:

1. **Divide**: Check middle element.
2. **Conquer**: Recursively search 1 sub-array.
3. **Combine**: Trivial.

Example: Find 9

| | | | | | | |
|---|---|---|---|---|---|---|
| 3 | 5 | 7 | 8 | 9 | 12 | 15 |

# Recurrence for binary search

的時間複雜度

1. 先寫出遞迴式

$n^{\log_b a}$

a b

$$T(n) = 1T(n/2) + \Theta(1)$$

*# sub-problems*

剩下的問題數

*sub-problem size*

把其中半邊丟掉

*work dividing and combining*

side information:
lg$n$ = log$_e n$

$$T(n) = \Theta(\lg n)$$

↳ $n^{\log_b a} \times \lg n = 1 \times \lg n = \Theta(\lg n)$

2. 比較 $n^{\log_b a}$ 和 $f(n)$ 的最高次項之大小

$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \Rightarrow case\ 2$$

↳ $n^{\log_b a}$ 和 $f(n)$ 算出來相等，所以要乘上 $\lg n$.

# Example 2: powering a number

- Compute $a^n$, where $n \in \mathbb{N}$.
- Naïve algorithm: $\Theta(n)$
- Divide-and-conquer algorithm: 勝

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$

剩下的問題數

$$T(n) = 1T(n/2) + \Theta(1) = \Theta(\lg n)$$

矩陣乘法

# Example 3: matrix multiplication

- Input: A = [$a_{ij}$], B = [$b_{ij}$]
- Output: C = [$c_{ij}$] = AB

$\Big\}$ $i, j = 1, 2, \ldots, n$

$$\begin{bmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

要找一個C要做n次
然後這邊總共有n^2個C
所以總共要做n^3次
所以複雜度是O(n^3)。

# Naïve algorithm

Naïve
矩陣運算

**for** $i \leftarrow 1$ **to** $n$ **do**

**for** $j \leftarrow 1$ **to** $n$ **do**

$c_{ij} = 0$

**for** $k \leftarrow 1$ **to** $n$ **do**

$c_{ij} = c_{ij} + a_{ik} b_{kj}$

$$T(n) = \Theta(n^3)$$

↳ 每次都要算 1~n.

要找一個C要做n次
然後這邊總共有n^2個C
所以總共要做n^3次
所以複雜度是O(n^3)。

# Divide-and-conquer algorithm

邊長

- *nxn* matrix = 2x2 matrix of $\frac{n}{2} \times \frac{n}{2}$ sub-matrices

先只考慮能平分
4等份的情況

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

$$C \quad = \quad A \quad \cdot \quad B$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned}\right\}$$

8回のかけ算と4回の足し算.

**8** mults of $\frac{n}{2} \times \frac{n}{2}$ sub-matrices

**4** adds of $\frac{n}{2} \times \frac{n}{2}$ sub-matrices $\frac{n^2}{4}$個加法

做矩陣加法的

$n$是邊長

測資邊長為 $n$, size為 $n \times n$ 的大矩陣

# A divide-and-conquer algorithm

$T(n)$

用遞迴的方式計算一個矩陣乘法.

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

$$C \quad = \quad A \quad \cdot \quad B$$

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
    n = A.rows
    Let C be a new n x n matrix
    if n == 1
        c11 = a11·b11
    else partition A, B, and C
    r = SQUARE-MATRIX-MULTIPLY-RECURSIVE(a, e)      T(n/2)
        + SQUARE-MATRIX-MULTIPLY-RECURSIVE(b, g)    T(n/2)    n²/4
    s = SQUARE-MATRIX-MULTIPLY-RECURSIVE(a, f)      T(n/2)
        + SQUARE-MATRIX-MULTIPLY-RECURSIVE(b, h)    T(n/2)    n²/4
    t = SQUARE-MATRIX-MULTIPLY-RECURSIVE(c, e)      T(n/2)
        + SQUARE-MATRIX-MULTIPLY-RECURSIVE(d, g)    T(n/2)    n²/4
    u = SQUARE-MATRIX-MULTIPLY-RECURSIVE(c, f)      T(n/2)
        + SQUARE-MATRIX-MULTIPLY-RECURSIVE(d, h)    T(n/2)    n²/4
    return C
```

每次的加法會有 $\frac{n^2}{4}$ 的時間複雜度

回傳一個大矩陣C.

$C = A \times B$

$$T(n) = 8T(n/2) + \Theta(n^2)$$

想知道大小是n*n的矩陣
我必須要先知道大小字是(n/2)*(n /2)的矩陣的結果

# Analysis

每次矩陣對半切,
邊長為n的矩陣邊長會變 $\frac{n}{2}$

$$T(n) = 8T(n/2) + \Theta(n^2)$$

就是這個8,
造成時間複雜度
卡在 $\theta(n^3)$,
因為 $n^{\log_b a} = n^{\log_2 8} = n^3$

*# sub-matrices*

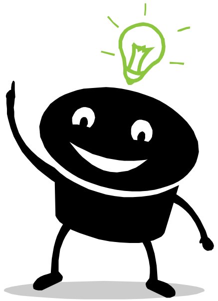
*sub-matrix size*

*work adding sub-matrices*

$$n^{\log_b a} = n^{\log_2 8} = n^3 \Rightarrow case\ 1$$

$$T(n) = \Theta(n^3)$$

no better than the naïve algorithm ☹



沒有那麼直覺的Divide & conquer

# Strassen’s algorithm

為了降低時間複雜度

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C \quad = \quad A \quad \cdot \quad B$$

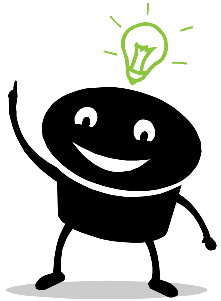
Multiply 2x2 matrices with only **7** recursive mults

$$P_1 = a \cdot (f - h)$$
$$P_2 = (a + b) \cdot h$$
$$P_3 = (c + d) \cdot e$$
$$P_4 = d \cdot (g - e)$$
$$P_5 = (a + d) \cdot (e + h)$$
$$P_6 = (b - d) \cdot (g + h)$$
$$P_7 = (a - c) \cdot (e + f)$$

$$\boxed{r = P_5 + P_4 - P_2 + P_6}$$
$$s = P_1 + P_2$$
$$t = P_3 + P_4$$
$$u = P_5 + P_1 - P_3 - P_7$$

7 mults

18 adds/subs



# Strassen’s algorithm

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

$$C \quad = \quad A \quad \cdot \quad B$$

Multiply 2x2 matrices with only **7** recursive mults

$$P_1 = a \cdot (f - h)$$
$$P_2 = (a + b) \cdot h$$
$$P_3 = (c + d) \cdot e$$
$$P_4 = d \cdot (g - e)$$
$$P_5 = (a + d) \cdot (e + h)$$
$$P_6 = (b - d) \cdot (g + h)$$
$$P_7 = (a - c) \cdot (e + f)$$

$$\begin{aligned} r &= P_5 + P_4 - P_2 + P_6 \\ &= (a + d)(e + h) \\ &\quad + d(g - e) - (a + b)h \\ &\quad + (b - d)(g + h) \\ &= ae + ah + de + dh \\ &\quad + dg - de - ah - bh \\ &\quad + bg + bh - dg - dh \\ &= ae + bg \end{aligned}$$

# Strassen's algorithm

一樣對半切

- 1. **Divide**: Partition A and B into $\frac{n}{2} \times \frac{n}{2}$ sub-matrices. Form terms to be multiplied using + and –.

  只有7個乘法

  2. **Conquer**: Perform **7** multiplications of $\frac{n}{2} \times \frac{n}{2}$ sub-matrices recursively.

  雖然要做18次加法、
  但矩陣加法的時間複雜度都在$n^2$的範圍裡.

  3. **Combine**: Form C using + and – on $\frac{n}{2} \times \frac{n}{2}$ sub-matrices.

$$T(n) = 7T(n/2) + \Theta(n^2)$$

# Analysis

$$T(n) = 7T(n/2) + \Theta(n^2)$$

$$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \Rightarrow case\,1 \quad T(n) = \Theta(n^{2.81})$$

The number 2.81 may not seem much smaller than 3, but because the difference is in the exponent, the impact on running time is significant.

In fact, Strassen's algorithm beats the naïve algorithm for $n \geq 32$ or so.

Best to date (or theoretical interest): $\Theta(n^{2.376\ldots})$

# Two more examples

3/14 (二) 談

# Closest-Pair Problem

- Find two closest points in a set of *n* points
- Assumptions
  - Points are in a plane. $P_i = (x_i, y_i)$
  - The standard Euclidean distance is used to measure distances between points.

$$d(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Example: $P_1 = (5, 3)$, $P_2 = (2, 8)$

$d(P_1, P_2) = \sqrt{3^2 + 5^2} = 5.831$

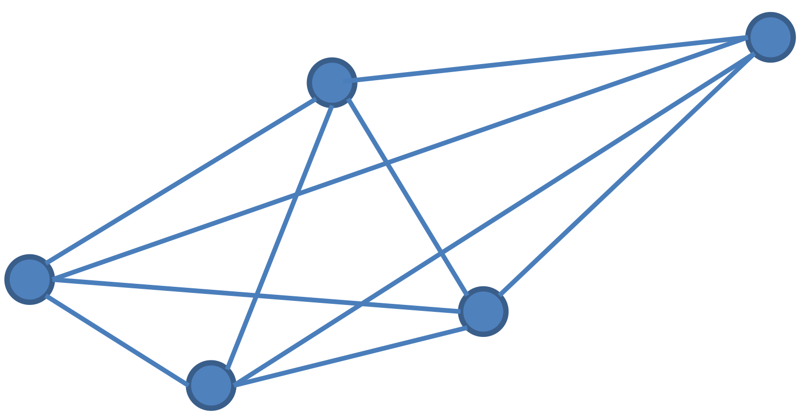
# Brute-Force Algorithm

暴力解：

任2點都要算距離.

Compute the distance between each pair of distinct points and return the pair with the smallest distance!

*$C(n, 2)$ pairs!*



# Brute-Force Algorithm

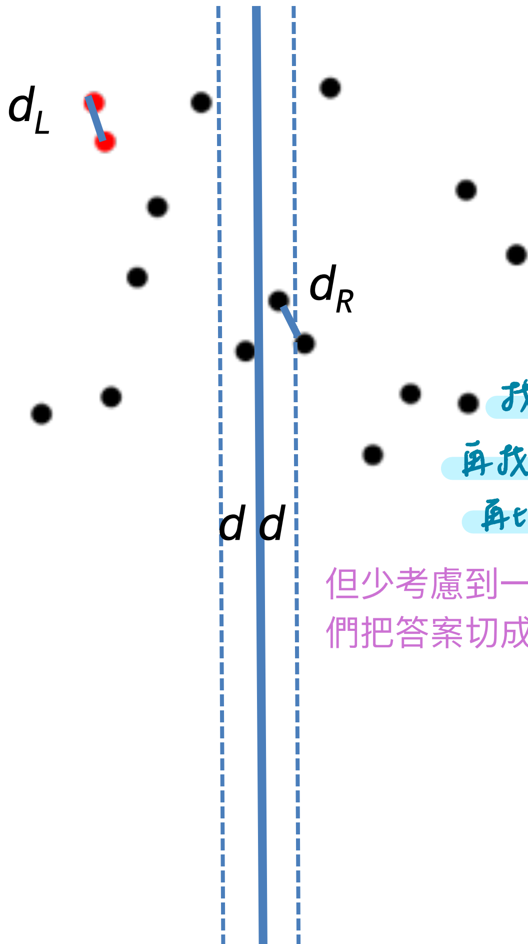
Compute the distance between each pair of distinct points and return the pair with the smallest distance! $C(n, 2)$ *pairs!*

**ALGORITHM** *BruteForceClosestPoints(P)*

//Finds two closest points in the plane by brute force
//Input: A list $P$ of $n$ ($n \geq 2$) points $P_1 = (x_1, y_1), \ldots, P_n = (x_n, y_n)$
//Output: Indices $index1$ and $index2$ of the closest pair of points
$dmin \leftarrow \infty$
**for** $i \leftarrow 1$ **to** $n - 1$ **do** $i \sim n-1$
$i+1 \sim n$ **for** $j \leftarrow i + 1$ **to** $n$ **do**
$d \leftarrow sqrt((x_i - x_j)^2 + (y_i - y_j)^2)$ //$sqrt$ is the square root function
**if** $d < dmin$
$dmin \leftarrow d$; $index1 \leftarrow i$; $index2 \leftarrow j$ $\therefore \Theta(n^2)_{\#}$
**return** $index1, index2$

$$C(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 = (n-1) + (n-2) + \ldots + 1 = \frac{(n-1)n}{2} \in \Theta(n^2)$$

# Divide-and-Conquer Algorithm




找出$P_L$裡最近
再找出$P_R$裡最近
再比2個的大小.

但少考慮到一個情況是我們把答案切成一點一邊了

1. Divide: Bisect the point set into two sets $P_L$ and $P_R$ with same sizes → 針對x座標排序, 再對y座標排序. ，一直對半切, 切成很多份
2. Conquer: Make two recursive calls—one to find the closest pair in $P_L$ and the other to find the closest pair in $P_R$. Let $d = \min(d_L, d_R)$.

$$T(n) = 2T(n/2) + \ldots$$

每次分半

分成2個小問題

3. Combine: Choose either $d$ or a pair of points with one in $P_L$ and the other in $P_R$

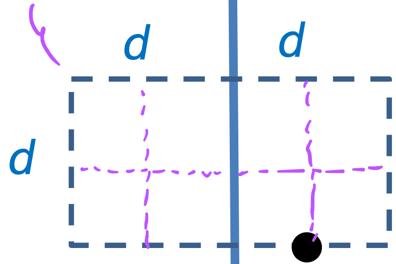
for loop只能寫一層，要檢查分界線附近有沒有很靠近的兩點，時間複雜度要維持線性。

**$O(n)$ using pre-sorted lists**

讓時間複雜度維持在O(n). 讓他掃一次就好。

# Divide-and-Conquer Algorithm

再把d格子切4分.
每個格子裡最多只能一個點.




根據y座標的方向，往後看
『固定個數』的點就夠了。
只要跟n無關，是個常數就可以。

*at most 1 point can reside in each d/2*d/2 square! check the following 7 points!*

1. Divide: Bisect the point set into two sets $P_L$ and $P_R$ with same sizes
2. Conquer: Make two recursive calls—one to find the closest pair in $P_L$ and the other to find the closest pair in $P_R$. Let $d = \min(d_L, d_R)$.

**$T(n) = 2T(n/2) + \ldots$**

3. Combine: Choose either $d$ or a pair of points with one in $P_L$ and the other in $P_R$

**$O(n)$ using pre-sorted lists**

這就是f(n)

# Divide-and-Conquer Algorithm

- $T(n) = 2T(n/2) + O(n)$ $\to f(n) = n$ $\log_b a = \log_2 2$ $n^{\log_b a} = n$ $\therefore \Theta(n \log n)_{\#}$
  - $T(n) =$ $\color{red}{\mathbf{O(nlogn)}}$ $\color{red}{<}$ $\color{red}{\mathbf{O(n^2)}}$ *What are eliminated?*
  - Master theorem: case 2!

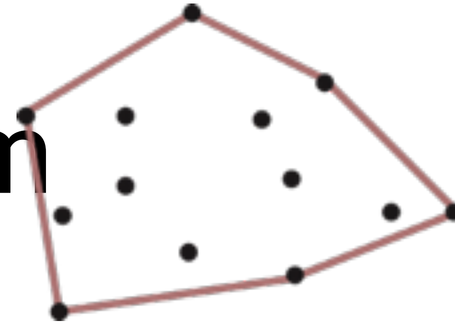
Aside:
Sort a sequence of $n$ elements: $O(n\log n)$

$\Theta(n \log n)$

如果今天有2點，很明顯離得很遠，在Divide & Conquer的做法不會去比較這兩點（在無數對半切的過程裡就被篩選掉了），但暴力法會去比較『任意兩點』，所以時間複雜度層級比較高。

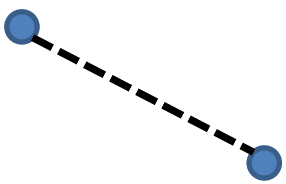
$\Theta(n^2)$

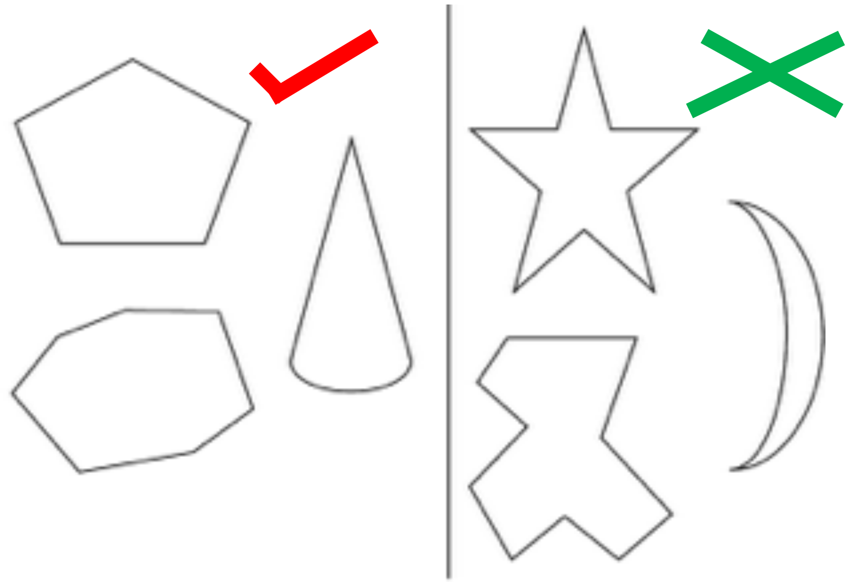
# Convex-Hull Problem



- Find the ***convex hull*** of a set of *n* points
- Convex set

在形狀上任挑兩點連線，線在
圖形內部的就稱為Convex set.





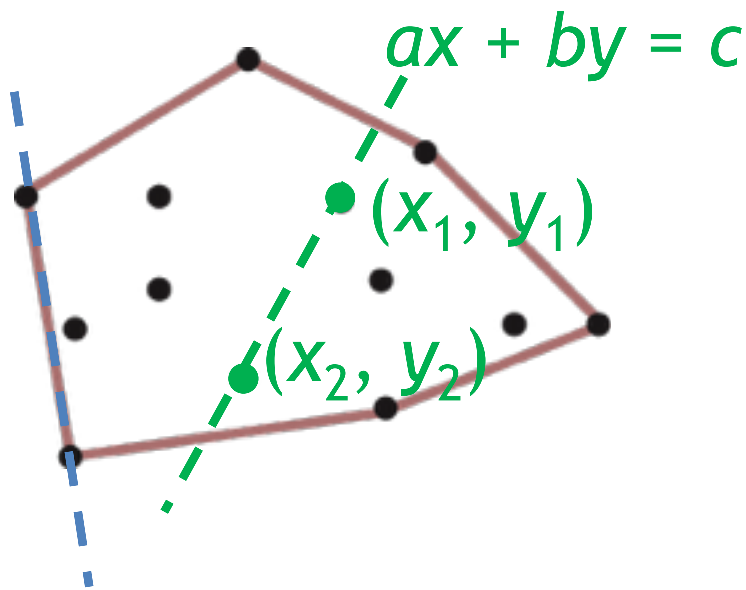
- Convex hull

在散佈的點中，找出面積最小的Convex set

The ***smallest*** convex set that contains all points

# Brute-Force Algorithm

他的概念是，如果今天取的線
是在convex的圖形上，
那麼其他點一定會在這條線的單側。




$a = y_2 - y_1$
$b = x_1 - x_2$
$c = x_1y_2 - y_1x_2$

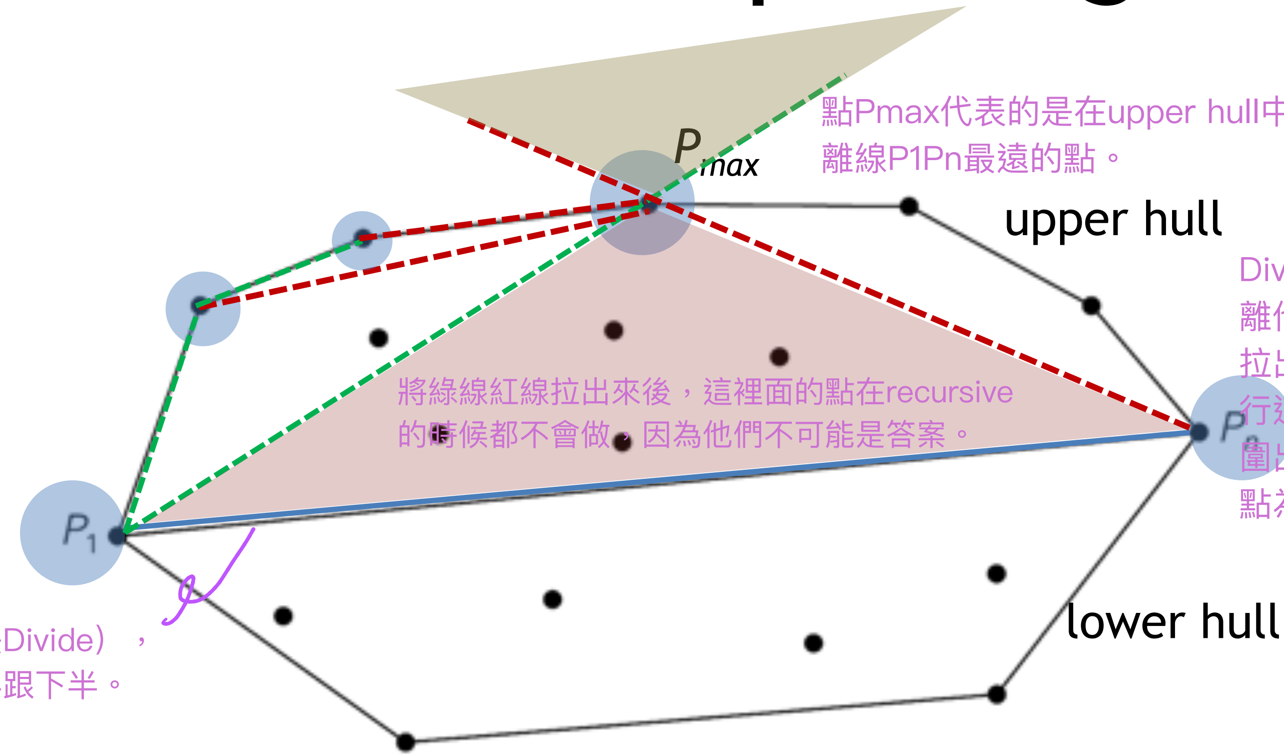
先寫兩層for loop，
每次兩點拉線，寫第三層for loop，
去檢查其他的點是不是都在線的單側。

- For each of $n(n-1)/2$ pairs of distinct points
  - Check the sign of $ax+by-c$ for each of the other $n$-2 points

$O(n^3)$

但這個做法的O是n^3…

# Divide-and-Conquer Algorithm




點Pmax代表的是在upper hull中，
離線P1Pn最遠的點。

Divide：定義一條線，以
離他最遠的一點為頂點，
拉出一個三角形。重複執
行這個動作來切，做到你
圍出來的三角形裡面沒有
點為止。

將綠線紅線拉出來後，這裡面的點在recursive
的時候都不會做，因為他們不可能是答案。

前處理（並不是Divide），
把圖形分成上半跟下半。

$$T(n) = 2T(n/2) + O(n) = \boldsymbol{O(n\log n)}$$

O(n)發生在每次要把點Pmax挖出來的時候。
就是找這個點的時間複雜度。

# Conclusion

第4章總結

- Divide and conquer is just one of several powerful techniques for algorithm design.
- Divide-and-conquer algorithms can be analyzed using recurrences and the master method.
- The divide-and-conquer strategy often leads to efficient algorithms.

# Coming up

- Sorting (Chapter 7-9)