

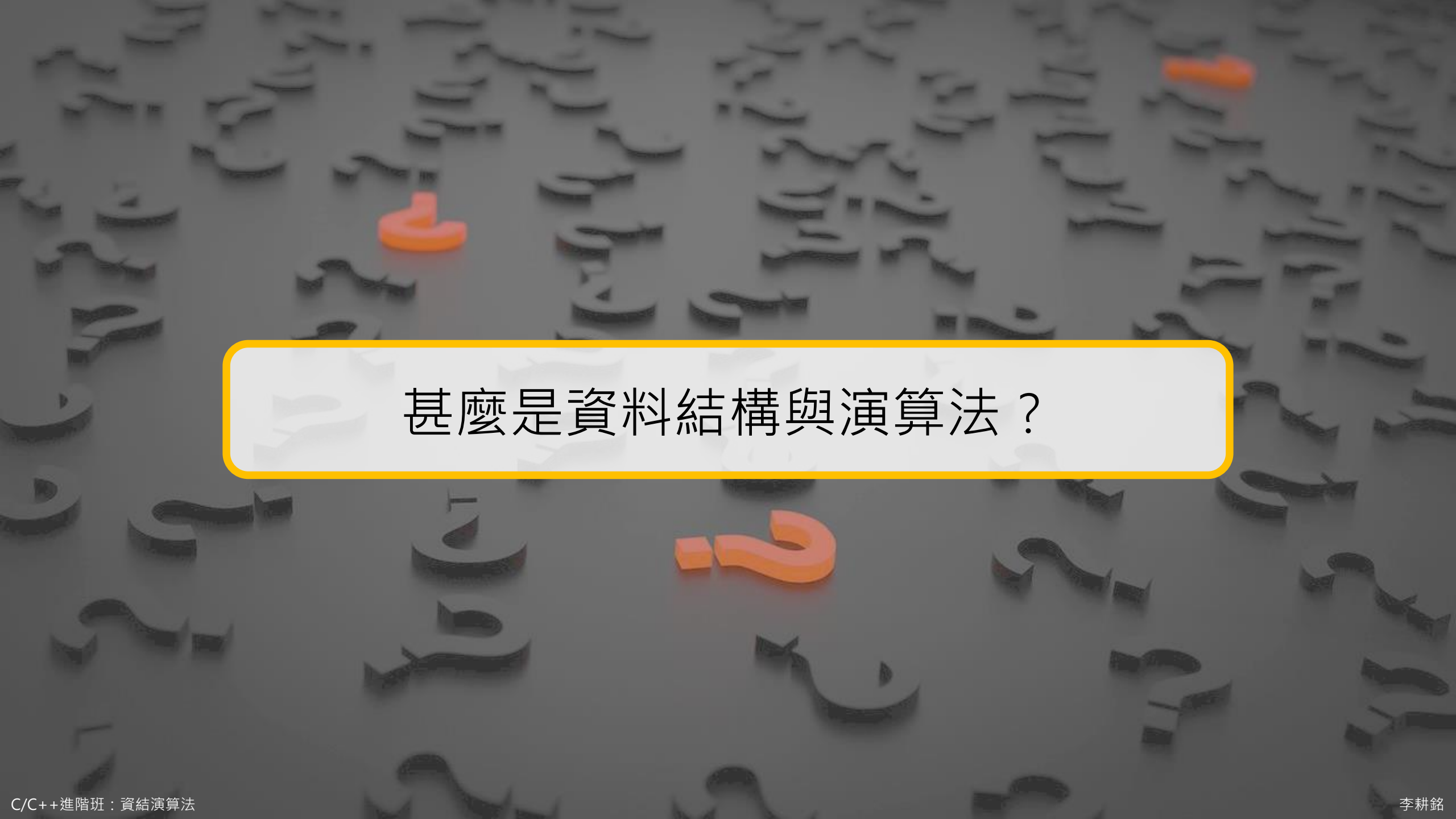
# C/C++ 進階班 演算法

## 資料結構演算法入門

李耕銘

# 課程大綱

- 甚麼是資料結構與演算法？
- 為什麼要學資料結構與演算法？
- 資料結構與演算法有關係嗎？
- 有哪些常見的資料結構或演算法問題？
- 如何評估演算法的好壞？
- 程式碼的寫法直接決定了效能嗎？
- 為什麼上課會以C/C++為主？
- 面試/競試時有哪些要注意的？

The background is a dark gray surface covered with numerous 3D mathematical symbols and question marks. Some symbols are in a lighter gray, while others, including several question marks, are in a bright orange color. The symbols are scattered across the entire frame, creating a textured, intellectual atmosphere.

甚麼是資料結構與演算法？

# 資料結構

- 資料間的內容與關聯

資料結構 = 資料內容 + 關聯

- 在電腦中儲存資料的方式
- 之前學過最基礎的資料結構：陣列

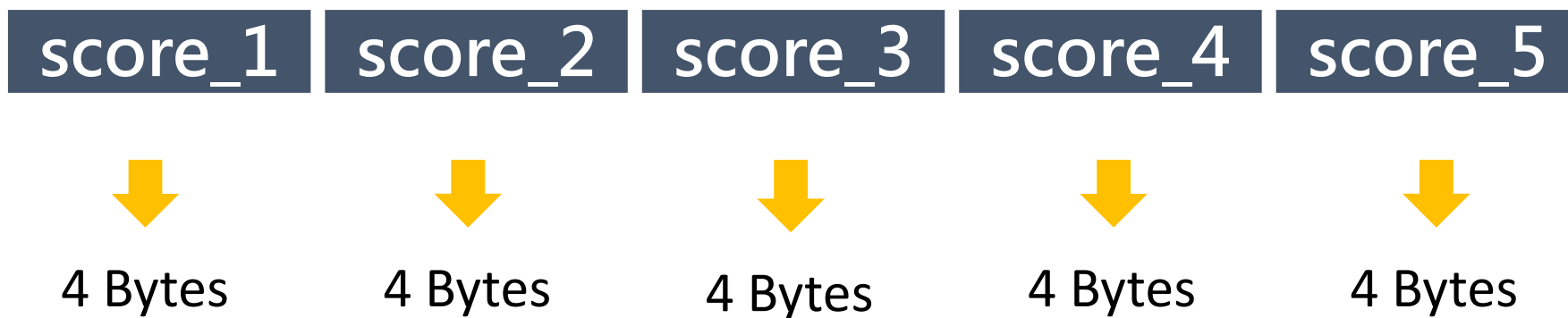


V.S.



# 陣列 (Array)

在記憶體開出一塊**連續的空間**來儲存**相同資料型態**的變數





# 演算法

- 一步步解決問題的方式
  - 必須遵守規則
  - 用何種程式語言都可以
- 演算法解決該問題的所有狀況
  - 此演算法解決(solve)了該問題

## Algorithm

*A finite sequence of well-defined, computer-implementable instructions.*

# 演算法

*Programming*

=

*Data structures + Algorithm*

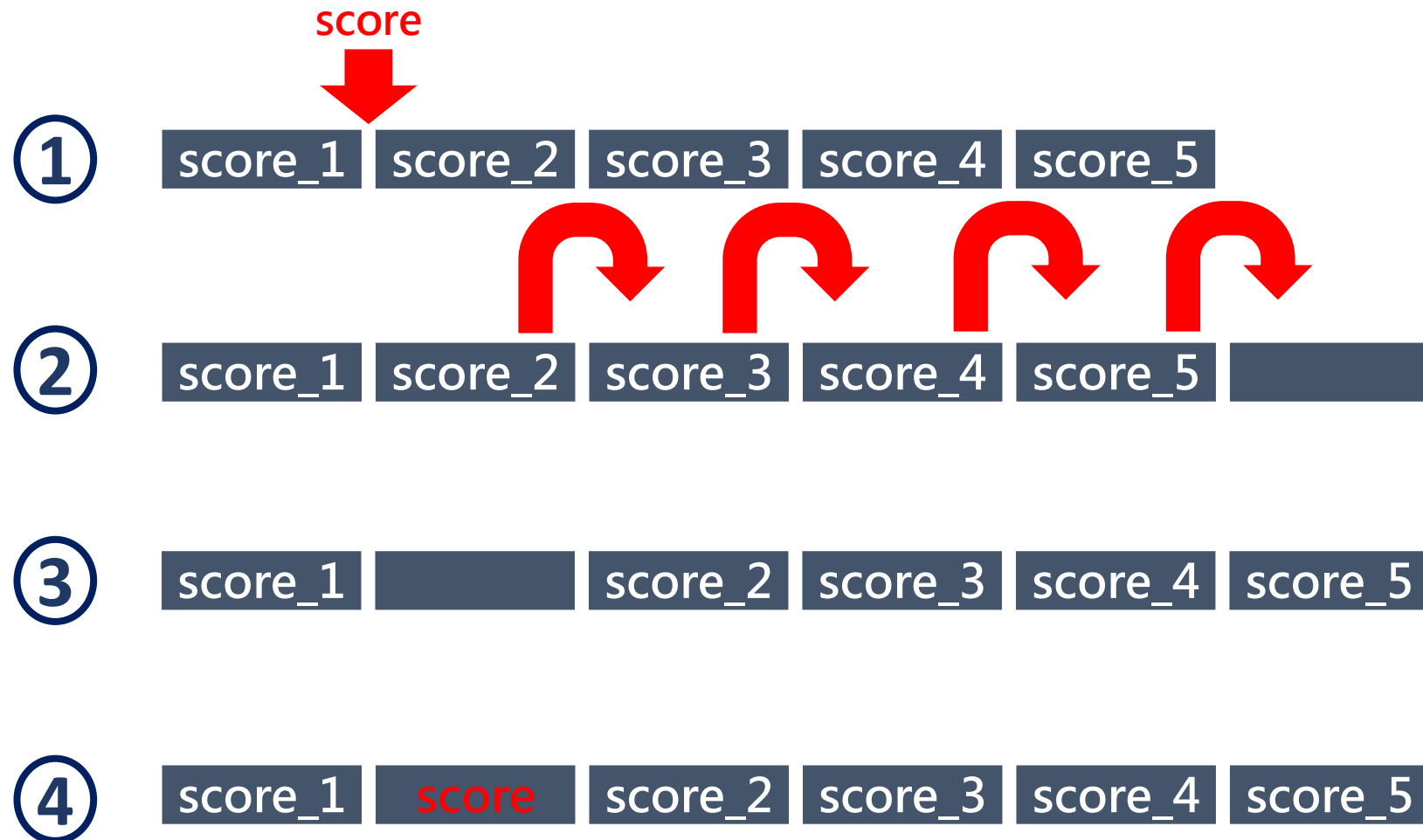
It's all about **efficiency** !

# 為什麼要學資料結構與演算法？



# 資料結構

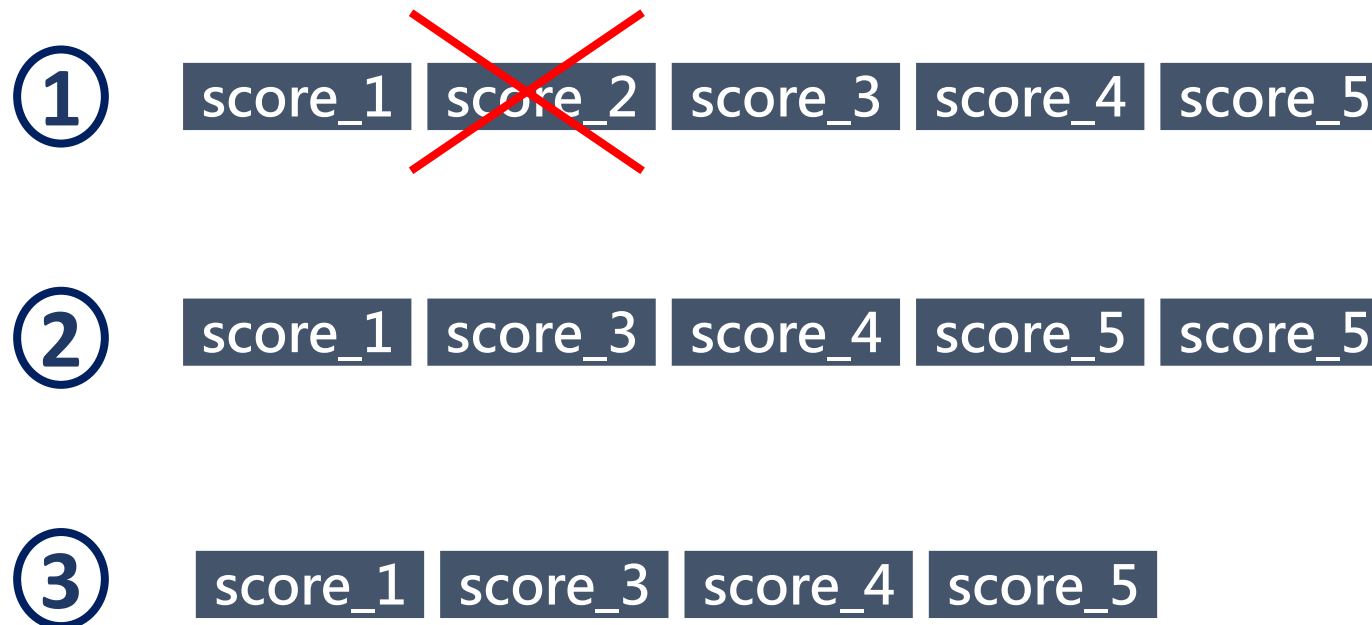
- 陣列有甚麼好處？
  - 空間最小化
- 陣列有甚麼問題？
  - 新增
  - 刪除
  - 搜尋



為了新增一筆資料，要付出多少代價？

# 資料結構

- 陣列有甚麼好處？
  - 空間最小化
- 陣列有甚麼問題？
  - 新增
  - 刪除
  - 搜尋



為了刪除一筆資料，要付出多少代價？

# 資料結構

- 陣列有甚麼好處？
  - 空間最小化
- 陣列有甚麼問題？
  - 新增
  - 刪除
  - 搜尋



為了搜尋一筆資料，要付出多少代價？

# 資料結構

- 偏偏這些操作又很常見！

- 註冊
- 登入
- 搜尋
- 排序

電子郵件地址或手機號碼

密碼

登入

[忘記密碼?](#)

建立新帳號

為名人、團體或公司企業建立粉絲專頁

# 資料結構

## 登入時發生甚麼事情？



# 資料結構

## 1. 搜尋



帳號1	帳號2	帳號3	帳號4	帳號5	帳號6	帳號7
密碼1	密碼2	密碼3	密碼4	密碼5	密碼6	密碼7

## 2. 取出



## 3. 比對



# 資料結構

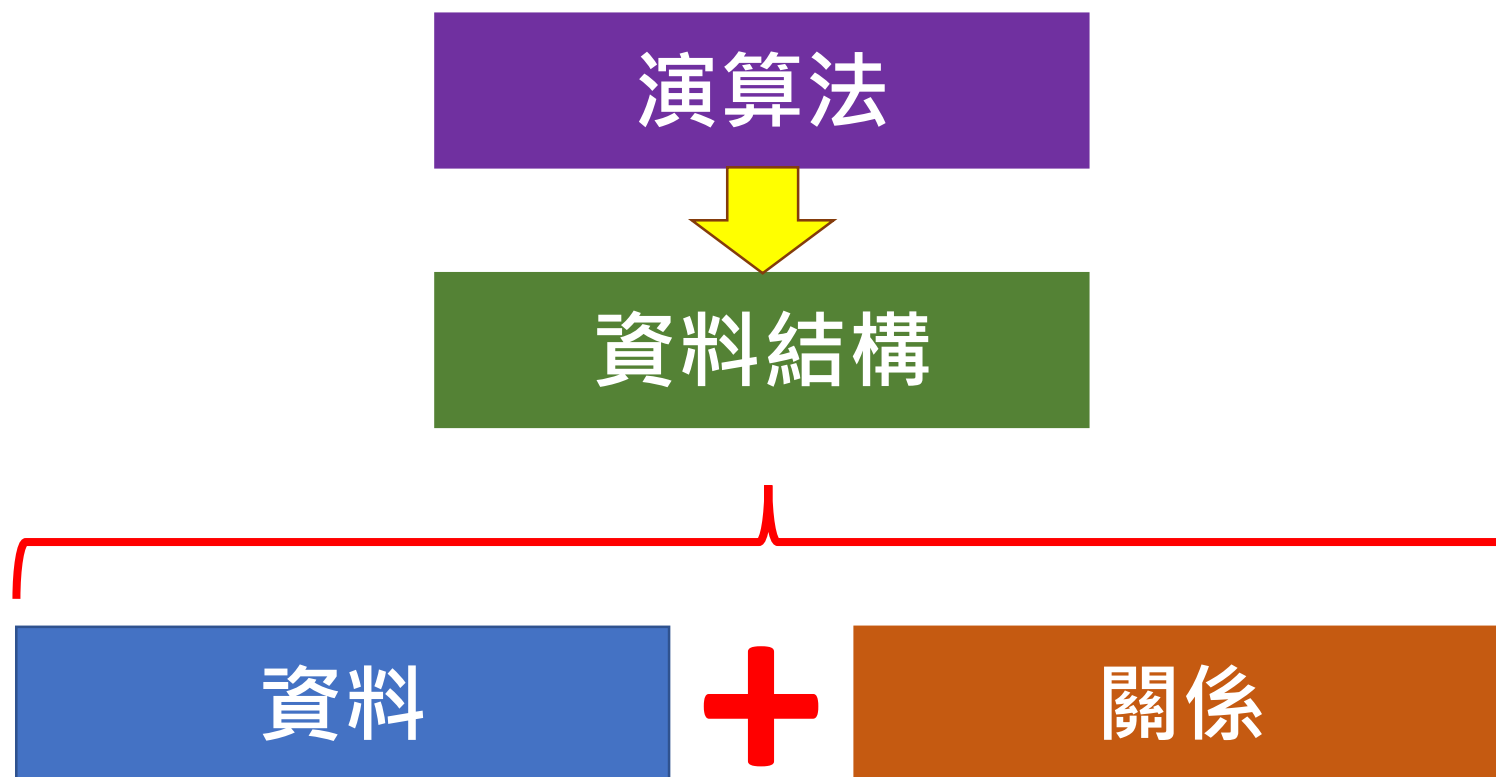
- 資料結構是在電腦中儲存資料的方式
- 針對不同的用途、算法給予不同的資料結構
- 不同的資料結構與算法擁有不同的
  - 空間複雜度
  - 時間複雜度
  - Coding複雜度

資料結構與演算法有關係嗎？

# 資料結構與演算法

有適合的資料結構，演算法才能施展。

換個方式說：演算法可以看做是使用資料的方式。



# 資料結構與演算法

- 資料結構決定了資料的使用方式(演算法)
- 透過改變資料的儲存，改變資料的使用方式
- 設計出適合相對應演算法的資料型態
- 但通常不會自己刻資料結構
  - 針對不同情境、算法，選擇相對應的資料結構



score\_1

score\_2

score\_3

score\_4

score\_5

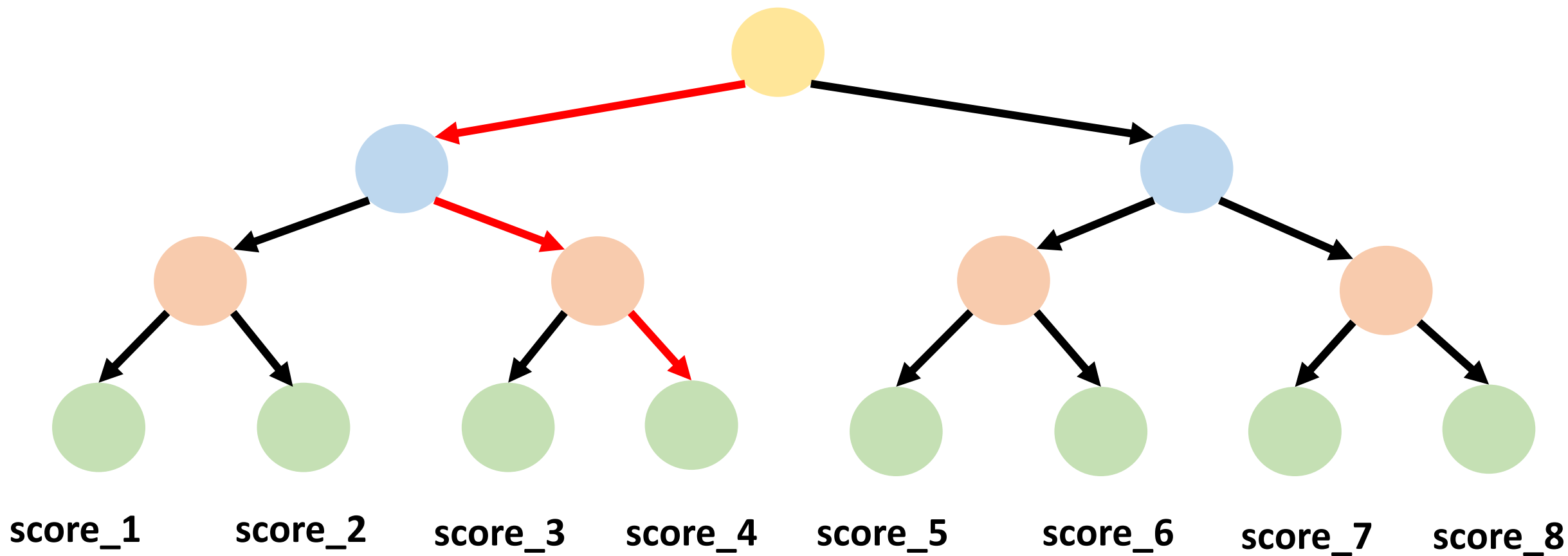
score\_6

score\_7

score\_8

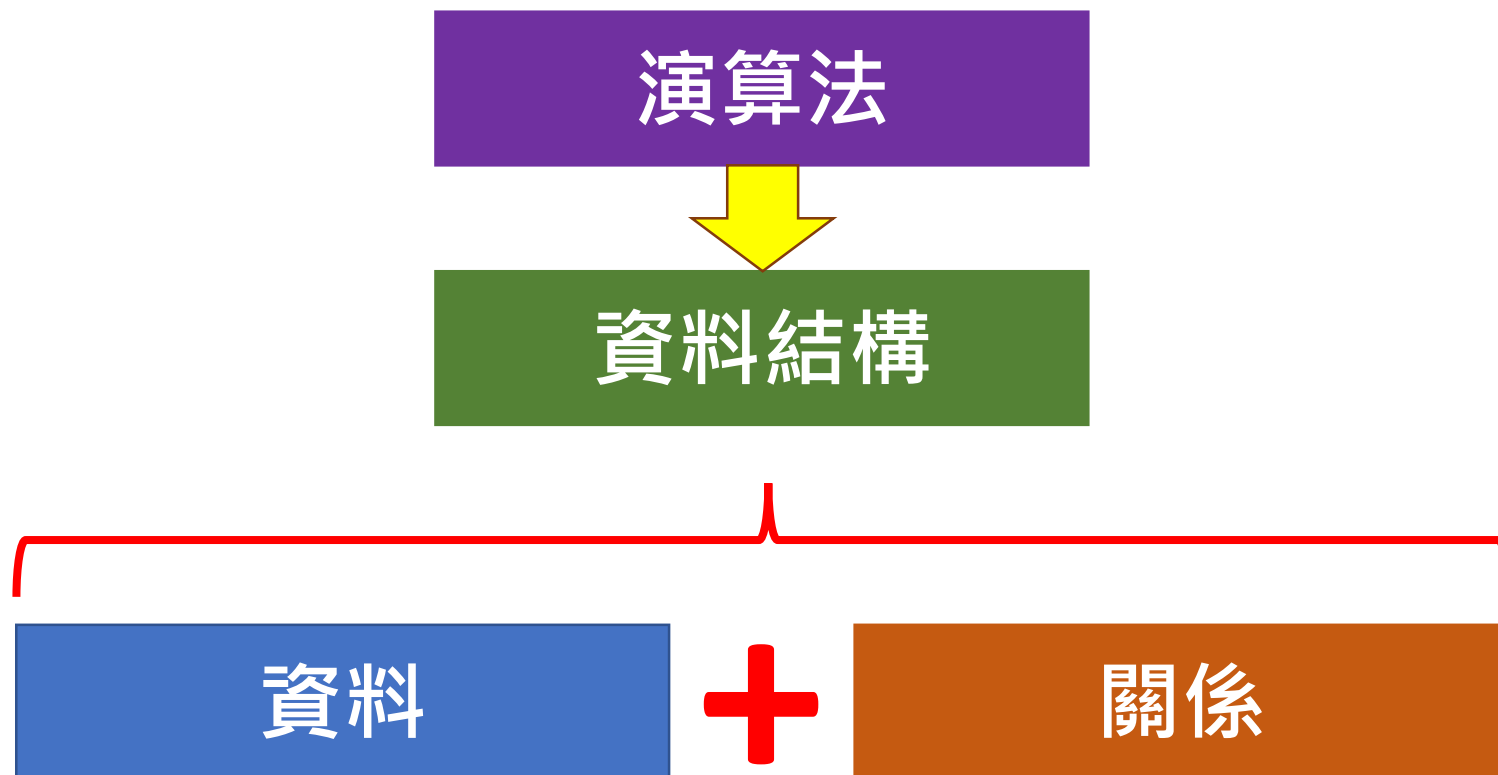
# 資料結構與演算法

換個方式儲存資料的話呢？

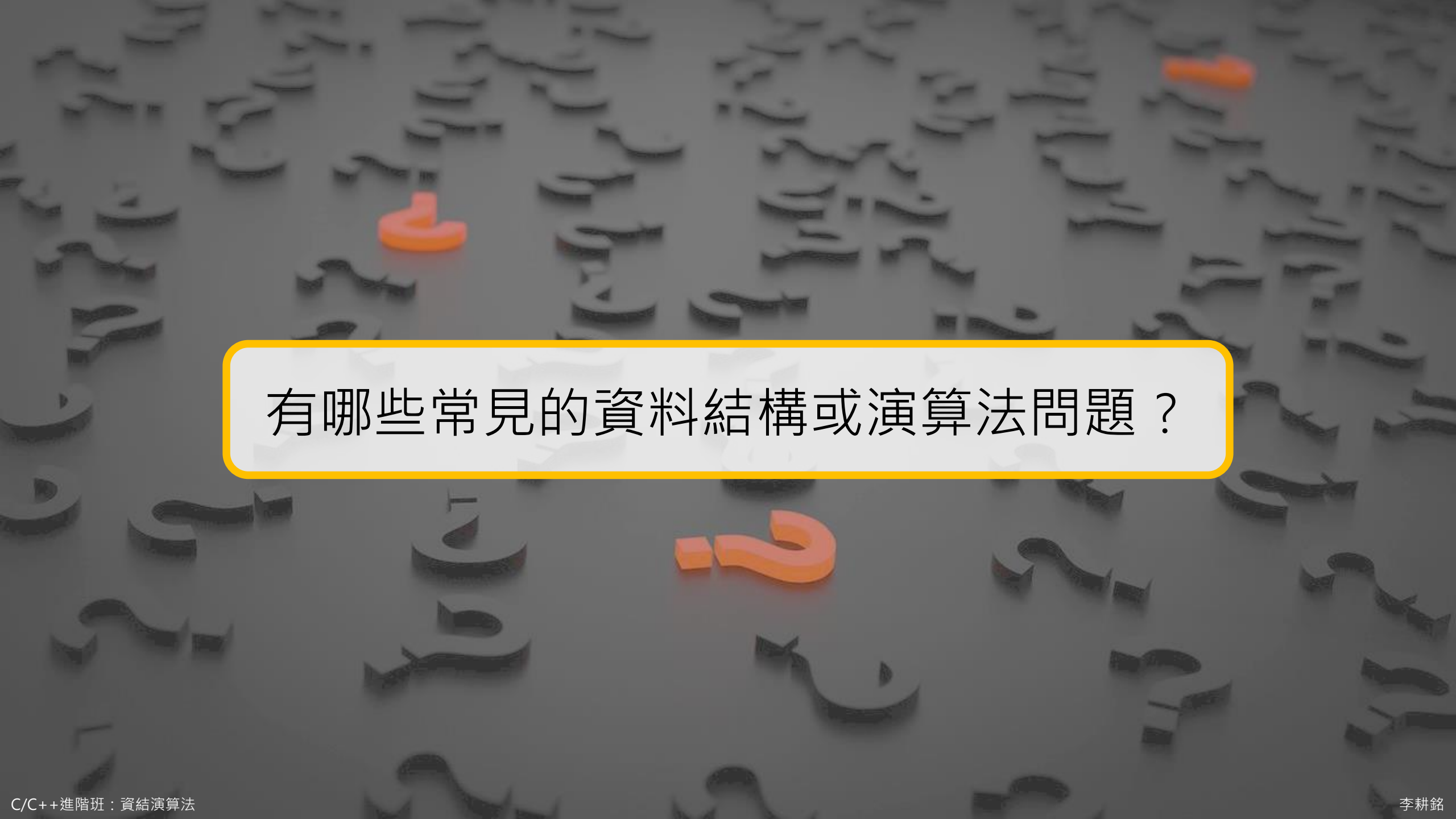


# 資料結構與演算法

所以**資料結構**、**演算法**通常會一起學





The background is a dark gray surface covered with numerous 3D mathematical symbols such as  $\{$ ,  $\}$ ,  $\infty$ ,  $\pi$ ,  $\sigma$ ,  $\omega$ ,  $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\delta$ ,  $\epsilon$ ,  $\zeta$ ,  $\eta$ ,  $\theta$ ,  $\iota$ ,  $\kappa$ ,  $\lambda$ ,  $\mu$ ,  $\nu$ ,  $\xi$ ,  $\omicron$ ,  $\pi$ ,  $\rho$ ,  $\sigma$ ,  $\tau$ ,  $\upsilon$ ,  $\phi$ ,  $\chi$ ,  $\psi$ ,  $\omega$ ,  $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\delta$ ,  $\epsilon$ ,  $\zeta$ ,  $\eta$ ,  $\theta$ ,  $\iota$ ,  $\kappa$ ,  $\lambda$ ,  $\mu$ ,  $\nu$ ,  $\xi$ ,  $\omicron$ ,  $\pi$ ,  $\rho$ ,  $\sigma$ ,  $\tau$ ,  $\upsilon$ ,  $\phi$ ,  $\chi$ ,  $\psi$ ,  $\omega$ . Scattered across this surface are several bright orange 3D question marks.

有哪些常見的資料結構或演算法問題？

# 常見的資料操作

- sort 排序
- search 搜尋
- delete 刪除特定元素
- insert 插入特定元素
- push 放入一個元素
- pop 取出一個元素
- reversal 反轉
- query 查詢



無時無刻都在用

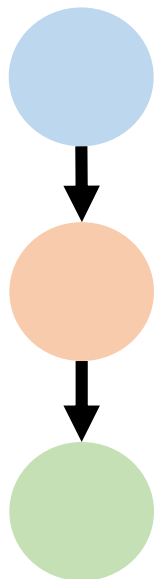
如何**加速**這些操作？

# 常見的資料結構

- Array 陣列
- Linked list 鏈結串列
- Stack 堆疊
- Queue 佇列
- Binary tree 二元樹
- Undirected graph 無向圖
- Directed graph 有向圖

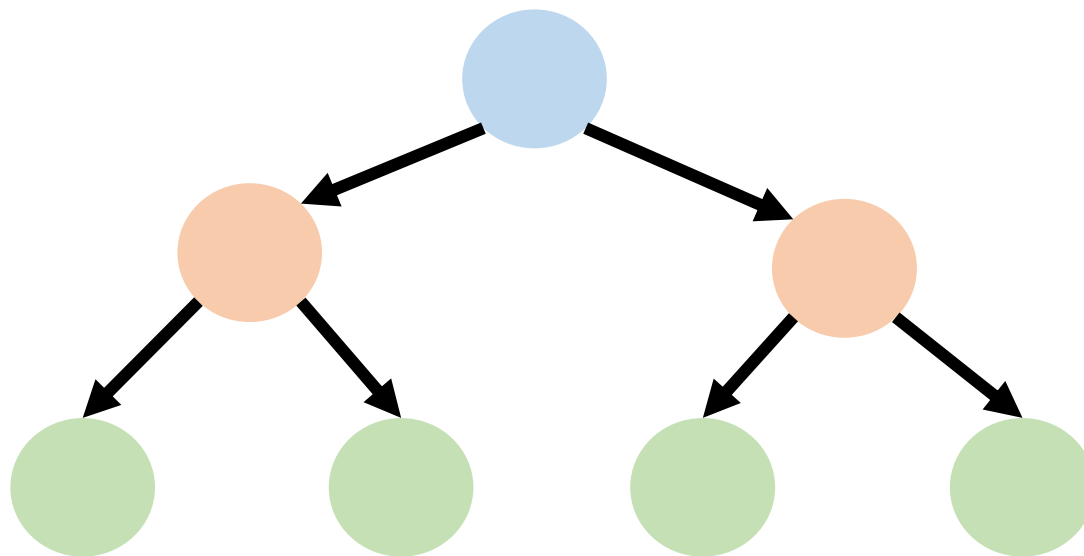
# 常見的資料結構

## 線性關係



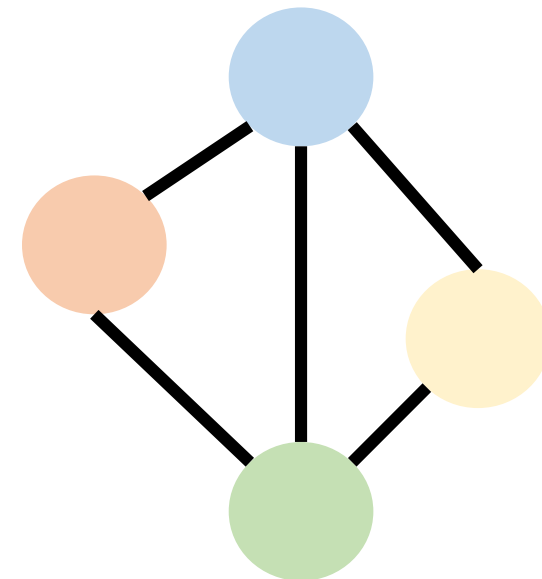
陣列  
鏈結串列  
堆疊  
佇列

## 階層關係



樹

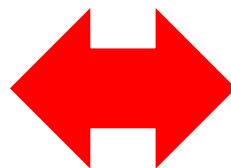
## 相鄰關係



圖論

# 常見的資料結構與演算法

- 讀取
- 搜尋
- 遞迴
- 二分搜尋
- 排序
- 排序
- 動態規劃
- 貪婪演算法



- Array 陣列
- Linked list 鏈結串列
- Stack 堆疊
- Queue 佇列
- Binary tree 二元樹
- Undirected graph 無向圖
- Directed graph 有向圖

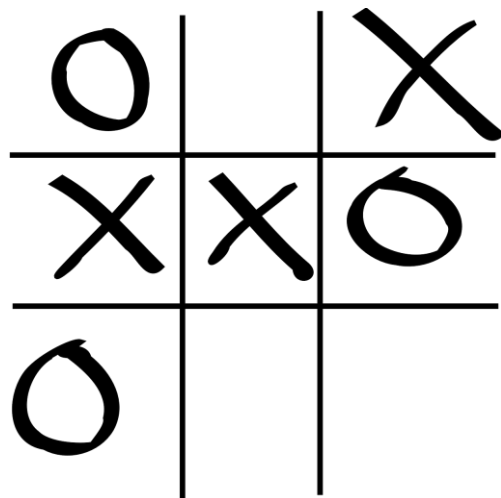
## 如何評估演算法的好壞



# 為什麼要評估？

- 電腦很快，但並非無限快
- 記憶體很大，但仍有容量上限

**Resource limitation**



**5478**



**$10^{171}$**

# 評估演算法

- 耗用的資源？
- 運算所需的次數/時間？
- Coding的難度？
- 理解的難度？
- **平衡**很重要
  - 以人口抽查代替普查



# 複雜度

- 時間複雜度 v.s. 空間複雜度
- 通常CPU運算資源比較珍貴
  - 多半我們在意的是**時間複雜度**

```
int a = 5, b = 3;
```

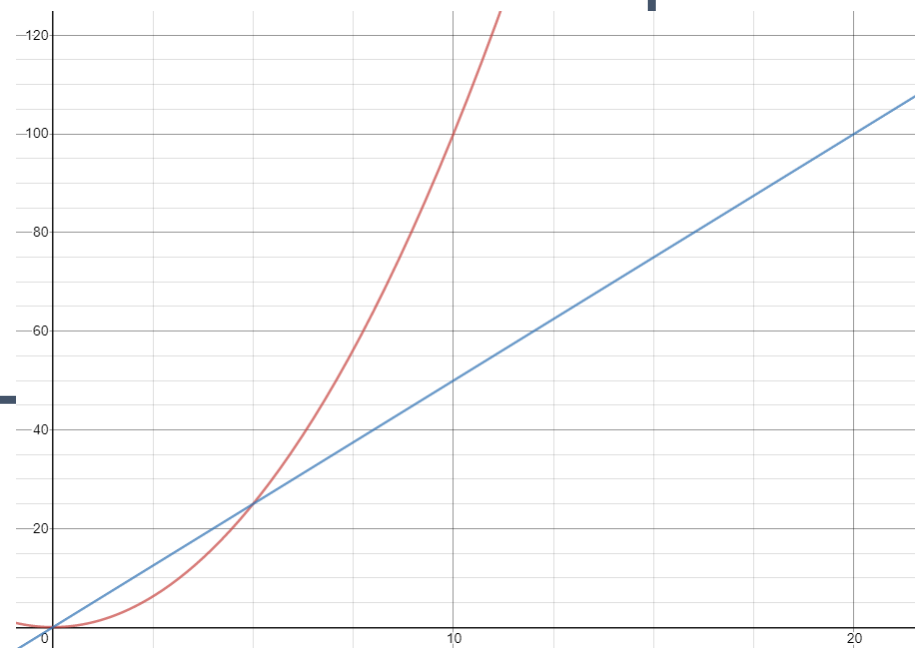
```
a += b;  
b = a - b;  
a = a - b;
```

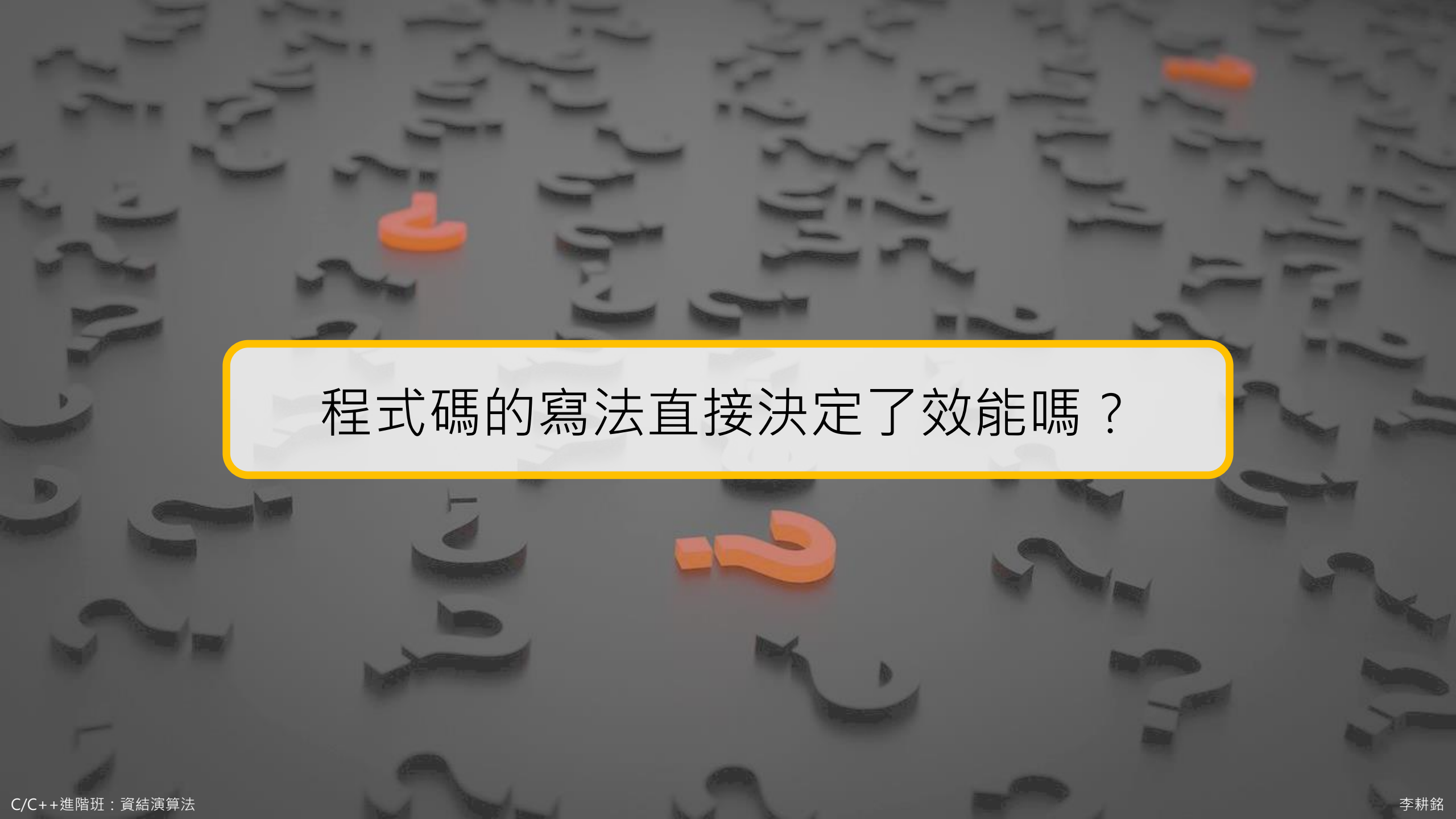
```
int a = 5, b = 3, tmp;
```

```
tmp = a;  
a = b;  
b = tmp;
```

# 估算複雜度的限制

- 每種運算所需時間略有差異
  - 乘除比加減法慢
- 通常資料量很小的時候，時間差異不大
  - 在意資料量級很大的時候
- 小時候胖不是胖
  - $x^2 < 5x$  for  $x < 5$



The background is a dark gray surface covered with numerous 3D mathematical symbols and question marks. Some symbols are in a lighter gray, while others, including several question marks, are in a bright orange color. The symbols are scattered across the entire frame, creating a textured, intellectual atmosphere.

程式碼的寫法直接決定了效能嗎？

# 效能還跟甚麼有關？

- 通常數字小時，基本運算可以在常數個指令內完成
- 但CPU 單次能運算的位元為有限長度，數字過大，就必須改用軟體來完成
- 程式碼越短不等於執行越快
  - 還是得回到組合語言去看
- 編譯器也會默默幫你做很多事



# 編譯器優化

```
for (int i=0; i<strlen(str);i++) {  
    if (str[i]=='a') counts++;  
}
```

- `int i = 0;`
  - check if `i < strlen(str)`, if yes, repeat
    - check if `(str[i] == 'a')` , if yes
      - `counts++;`
    - `i++;`
- for loop → 雙重迴圈 →  $n^2$**
- Why not ?*

為什麼上課會以C/C++為主？

# Why C/C++ ?

- Pros
  - STL 有把常見的資料結構與演算法封裝進去
  - 有指標可以用
  - 效能優越：Python→TLE(Time Limit Exceeded)
- Cons
  - Hard to learn ☹

The background of the slide features a dark gray surface with numerous 3D question marks scattered across it. Some question marks are orange and stand out, while others are dark gray and blend into the background. In the background, there are also faint, 3D Chinese characters, possibly related to the topic of the slide.

面試/競試時有哪些要注意的？

# 提交結果

- Accepted (AC)
- Wrong Answer (WA)
- Time Limit Exceed (TLE)
- Runtime Error (RE)
- Compile Error (CE)
- Memory Limit Exceed (MLE)

# 優化方式

- 輸入優化

```
std::ios::sync_with_stdio(false);  
std::cin.tie(0);
```

- 讀檔檢查

```
freopen("test.in", "r", stdin);  
freopen("test.out", "w", stdout);
```

# Take Home Message

- 資料結構是甚麼？演算法是甚麼？
- 資料結構與演算法間有甚麼關係？
- 有哪些評估程式效能的方式？
- 程式碼越短，效能一定越好嗎？
- 除了寫法外，還有哪些東西會影響效能？