第一章

資料結構導論

Introduction

本章內容

1-1 資料結構在學什麼

1-2 資料與資訊的意義

1-3 演算法的定義與表示法

流程圖 虛擬碼

1-4 程式的分析

正確達到目的 可維護性高

效率高 記憶體需求低

1-5 迴圈的頻率計數

1-6 Big-O 符號

想一想

為什麼電腦圍棋程式AlphaGo能夠打敗天下無敵手，讓世上所有的圍棋高段俯首稱臣？沒錯！就是「人工智慧」(Artificial Intelligence, AI)。那到底什麼是人工智慧呢？

人工智慧就是程式，是由「演算法」合作而成，配合適當的「資料結構」，形成完美的搭配組合。

1-1 資料結構在學什麼

◎ **資料結構**這門課中探討計算機系統所儲存以及處理的**資料**，並且學習如何組織這些資料，以及處理這些資料的**方法**。

◎ 資料結構可以應用在**導航系統**

各個地點的位置及道路的「圖形資料」都已經被組織安排過了，只要再配合一些方法算出「最短路徑」，就可以幫助駕駛人解決認路或選擇路徑的問題。

◎ 資料結構可以應用在**搜尋引擎**

搜尋引擎事先日夜不斷的蒐集網頁資料，依照關鍵字等線索在電腦主機中建構成「索引結構」(index)，等網路使用者下達關鍵字進行搜尋時，就很快的到索引結構中找出網頁的鏈結並回應給使用者。

◎ 資料結構可以應用在**社群網站**

社群網站將每一個人視為「圖形結構」上的一個點，兩個人若是朋友則相對的兩個點就會有一條連接線。如果A與B都有共同的朋友C，而且A與B還不是朋友，社群網站就會向A推薦B且向B推薦A。當任一方接受推薦並且獲得另一方確認時，社群網站就牽線成功，在A點與B點之間加上連接線。

1-2 資料與資訊的意義

◎ **資料** (**d**ata) ：用具體符號表示，而能夠被計算機處理的資訊 。

◎ **資訊**(**i**nformation) ：資料所呈現出來，可經人們分析而理解的訊息。
(資料強調符號本身，所以資料較為具體，而資訊較為抽象。)

◎ **知識** (**k**nowledge) 對現有資訊的學習、歸納或推導而來，是人類（或電腦？）學習與理解資訊所得的結果。

◎ **智慧** (intelligence, **w**isdom) 則是知識的系統化結果，是對於知識的整體洞察。

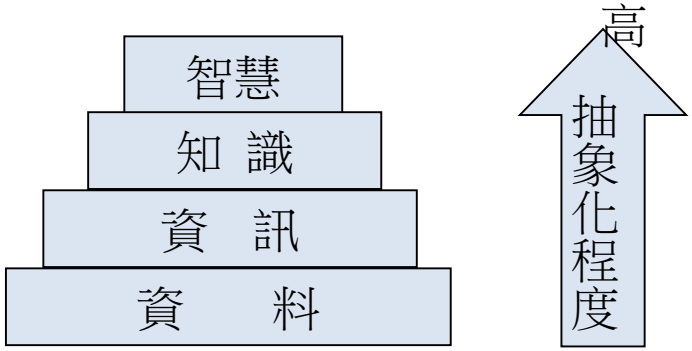


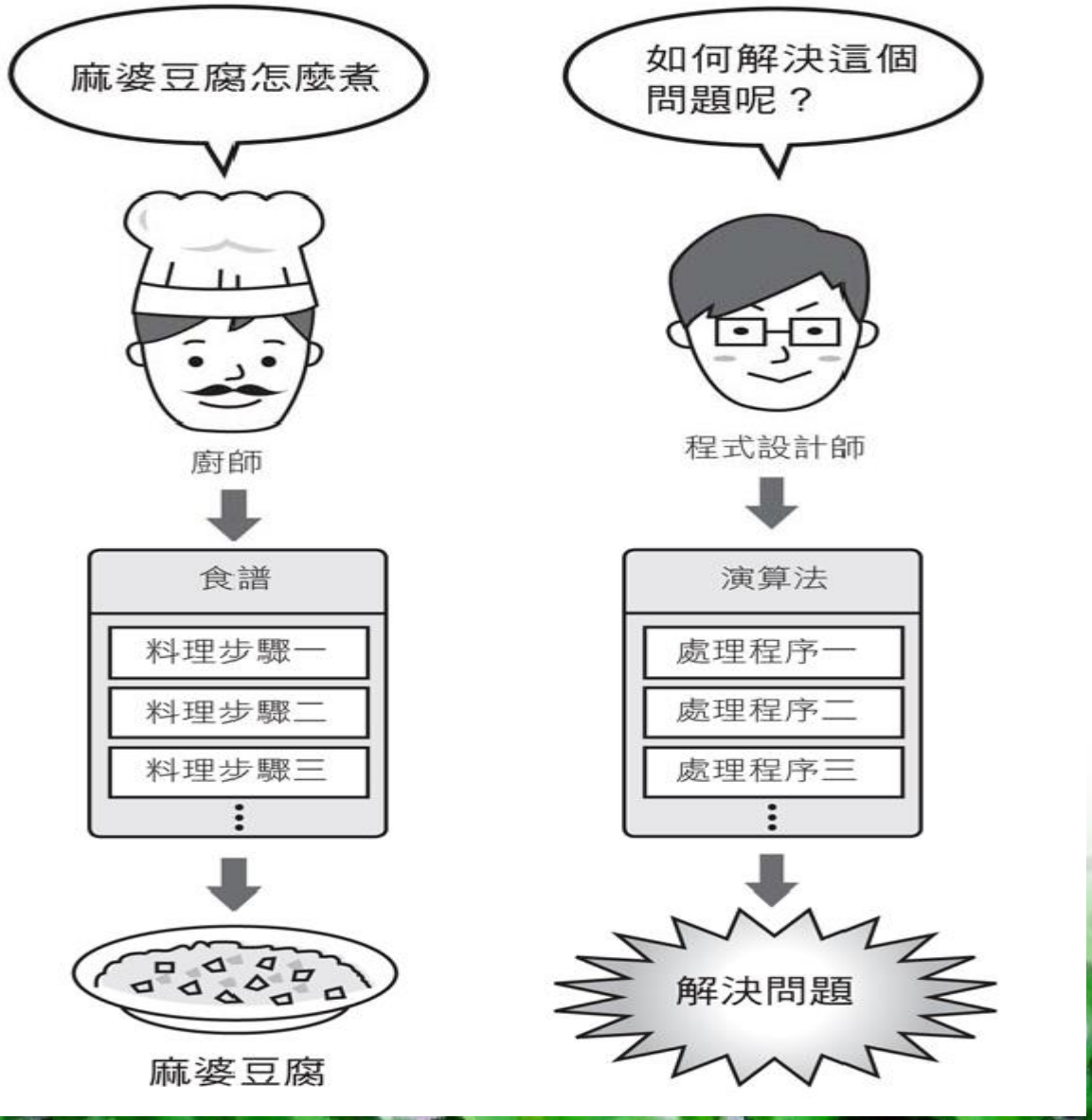
圖1.1 DIKW架構

在近代被發現的「甲骨文」

- ✓屬於人類遺產中的重要**資料**
- ✓歷史學家可以從中獲得中國商朝時代的許多**資訊**，如政治制度的運作與宗教信仰的活動等
- ✓他們更研究整理了這些資訊，形成一門獨特的**知識**，稱為「甲骨學」
- ✓通達甲骨學的學者，能洞察人類文化的本質，就具有相當的**智慧**了

圖片來源：維基百科

1-3 演算法的定義與表示法



1-3 演算法的定義與表示法

◎資料結構＋演算法＝程式

◎ **演算法** (algorithm) 的定義：在有限步驟內解決數學問題的程序。在計算機科學的領域中，演算法泛指 “適合被實作為計算機程式的解題方法”

◎ 演算法通常具有以下五個特性：

一·輸入 (Input)

二·明確性 (Definiteness)

三·正確性 (Correctness)或是
有效性 (Effectiveness)

四·有限性 (Finiteness)

五·輸出 (Output)

例1.4 歐幾里得演算法 計算兩個自然數的最大公因數 (Greatest Common Divisor, GCD)

- ✓ 歐幾里得(Euclid)是古希臘的數學家(西元前325年—西元前265年)
- ✓ 歐幾里得演算法被公認為是歷史上第一個演算法
- ✓ 在現代的密碼學中，它是在電子商務安全機制中被廣泛使用的「公鑰加密演算法」的重要元件

演算法描述如下：

敘述1. 輸入兩個自然數 A , B

敘述2. A 除以B，餘數為 R

敘述3. 如果 R 為零，則跳至敘述 5

敘述4. A←B，B←R，跳至敘述 2

敘述5. B 即為 GCD

例1.4 歐幾里得演算法 計算兩個自然數的最大公因數 (Greatest Common Divisor, GCD)

敘述1. 輸入兩個自然數 A , B
A = 18, B = 12

敘述2. A 除以B，餘數為 R
R = 18 MOD 12 → R =6

敘述3. 如果 R 為零，則跳至敘述 5
R ≠ 0

敘述4. A←B，B←R，跳至敘述 2
A = 12 (A←B), B = 6 (B←R)

敘述2. A 除以B，餘數為 R
R = 12 MOD 6 → R = 0

敘述3. 如果 R 為零，則跳至敘述 5
R = 0

敘述5. B 即為 GCD (B=6)

A	B	R
18	12	
18	12	6
12	6	0

以上敘述符合演算法的特性

一·**輸入** (Input) ：兩個自然數A, B

二·**明確性** (Definiteness) ：以逐步追蹤法 一步一步執行敘述，不造成混淆

三·**正確性** (Correctness) ：以兩組(A>B及A<B)輸入執行均得到正確 結果

四·**有限性** (Finiteness) ：R愈來愈小，R終會等於零

五·**輸出** (Output) ：A, B 的最大公因數

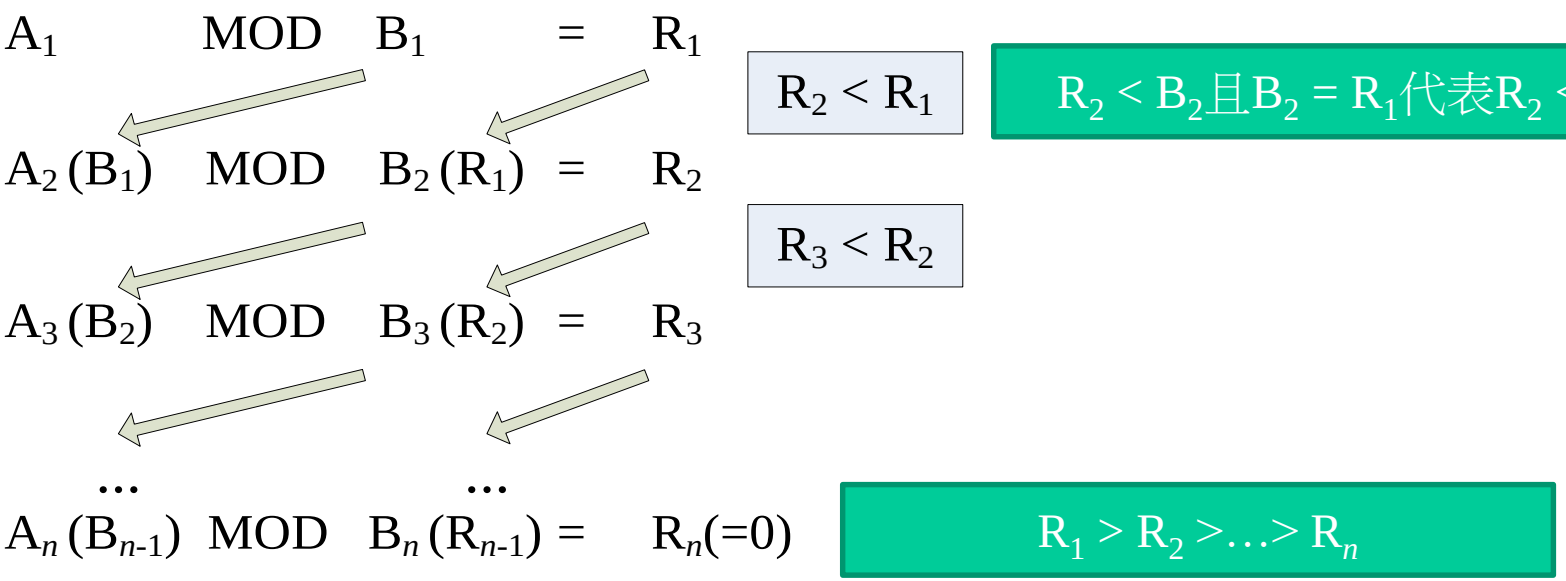
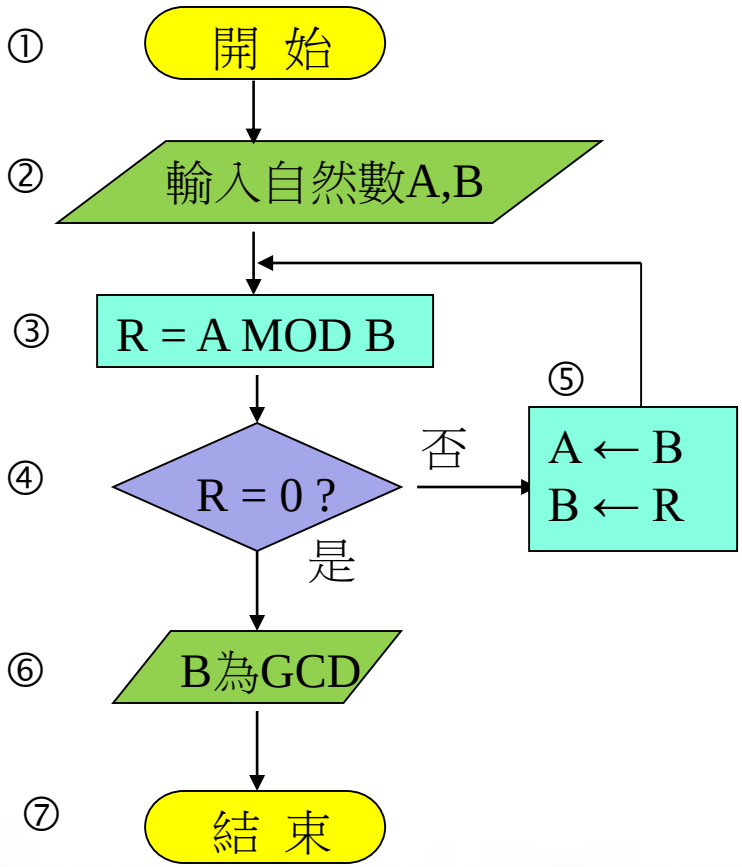


圖1.2 輾轉相除餘數終會為零

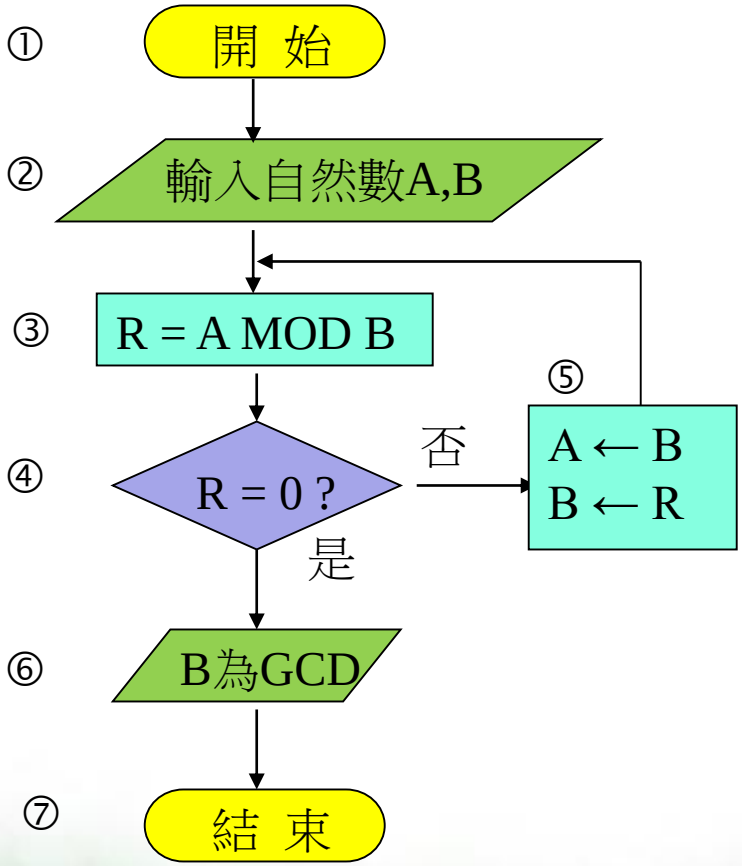
用流程圖來描述這個演算法



- 「開始」及「結束」通常使用「膠囊」形狀的符號如①與⑦
- 「平行四邊形」通常是指「輸入」或「輸出」，如②與⑥
- 「矩形」則是指「處理」或「運算」，如③與⑤的算術運算式與指定式。
- 「菱形」是「決策分支」或「判斷」，在執行時根據條件決定走適當的分支，如④中，條件成立時（R=0時）往下走到⑥，條件不成立時往右走到⑤。

你可以模仿上面的流程圖,
描述你在吃到飽的「包肥」
(buffet) 餐廳用餐 的流程嗎？

用虛擬碼來描述演算法



演算法：計算兩個自然數的最大公因數

輸入自然數A與B

當 R (R=A MOD B)不為0 時，重複迴圈

A←B（把B的值給A）

B←R（把R的值給B）

迴圈結束

輸出B為結果

演算法結束

如果將虛擬碼丟給 C 語言的編譯器編譯, 你猜會有什麼結果？

演算法：計算兩個自然數的最大公因數

輸入自然數A與B

當 R (R=A MOD B)不為0 時，重複迴圈

A←B（把B的值給A）

B←R（把R的值給B）

迴圈結束

輸出B為結果

演算法結束

??

1-4 程式的分析

一個好的程式，通常必須滿足下列的條件：

一·**正確達到目的**：通常以具有代表性的多組輸入來測試驗證

二·**可維護性高**：牽涉到編寫程式的方法和風格，以下是幾項檢驗項目：

檢驗項目一：編寫程式是否符合**模組化**的原則，提供由上而下 (Top-Down)的理解思路？

檢驗項目二：所有的常數變數及函式 (function) **名稱是否有意義**？

檢驗項目三：所有函式的輸入、輸出及功能是否被**明確定義**？

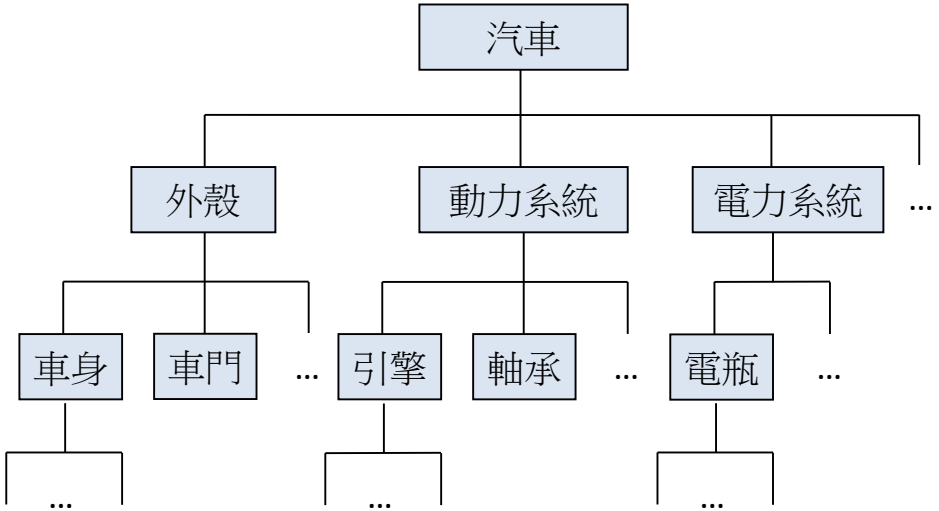
檢驗項目四：是否在適當的地方加上**註解**？

檢驗項目五：**說明文件**是否詳實完備？

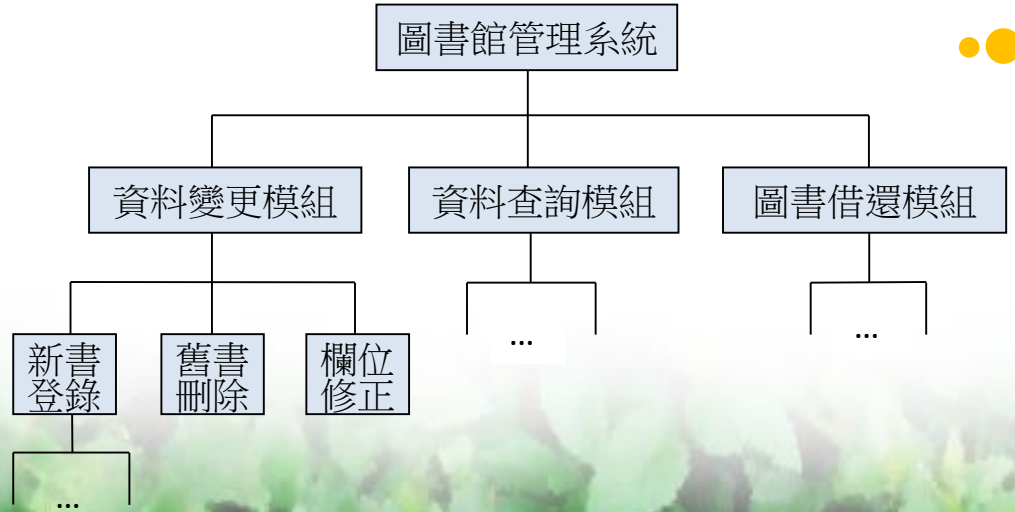
三·**效率高**：計算程式的時間複雜度來分析效率並尋求改良之道

四·**記憶體需求低**：在不影響效率的情況下，需求愈低愈好

可維護性—模組化



汽車的模組化架構圖



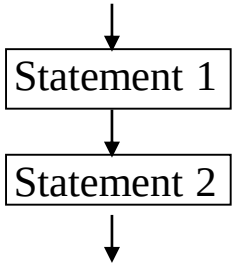
應用軟體的模組化架構圖

程式的效率

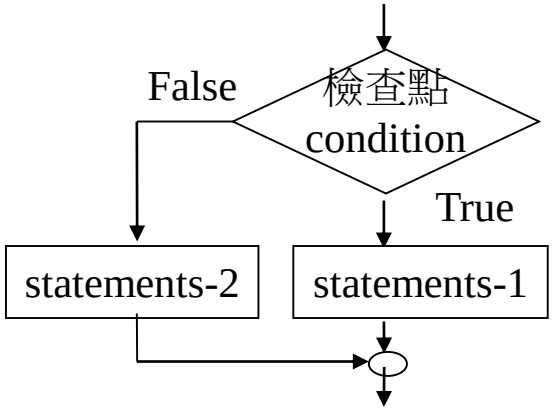
- ✓程式的執行**效率**反映在程式執行所花的**時間長短**。
- ✓測量程式的執行效率最直接的方式，就是**直接量時間**。
  - 在程式的開頭和結尾各記錄一次時間，兩個時間差就是執行的時間。
- ✓同樣的程式，經過不同的編譯程式編譯，或在不同的硬體或作業系統上執行，都可能量測到不同的執行時間。
- ✓只看程式碼也可以評估效率，我們可以計算程式中所有敘述被執行的總次數，也就是「**頻率計數**」(frequency count)，並且用頻率計數來比較程式的效率，因為頻率計數愈大，執行的時間也應該愈長。

結構化程式的效率三種結構

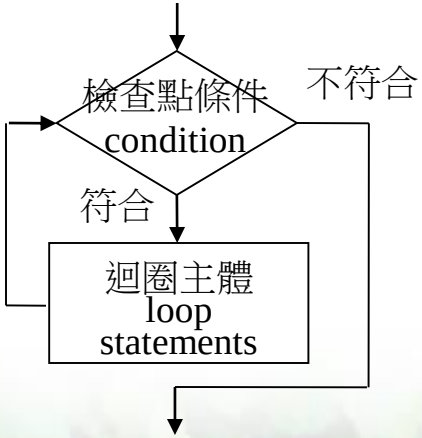
- 循序敘述的頻率計數



- 決策分支敘述的頻率計數



- 迴圈敘述的頻率計數



頻率計數的計算

循序敘述的頻率計數

計算循序敘述片段的頻率計數，只要將敘述的行數加總即可。
下列的程式片段，計算整數 *m* 除以整數 *n* 的商數以及餘數：

```
1.    Q = m / n ;
2.    R = m % n ;
3.    printf("%d ÷ %d = %d … %d", m, n, Q, R);
      // cout << m << " ÷ " << n << " = " << Q << "…" << R;
```

這個程式片段的頻率計數為 **3**，因為不管*m*和*n*的值為何，這3個敘述都會被執行。

決策分支敘述的頻率計數

取各個條件**分支中總行數的最大值**加上比較敘述的次數。下列的程式片段是用來計算整數 m 除以整數 n 的商數以及餘數，但是會先對除數的值加以判斷：

```
1.    if ( n == 0 )
2.              printf("Error ! Divisor cannot be Zero !");
                  //cout << "Error ! Divisor cannot be Zero !";
3.    else
4.    {         Q = m / n ;
5.              R = m % n ;
6.              printf("%d ÷ %d = %d … %d", m, n, Q, R);
                //cout <<m <<"÷"<<n << " = " << Q << "…" << R;
7.    }
```

頻率計數為 **4** (= 3 + 1)，因為在兩個可能的分支中，最大的區塊有3個敘述（第4行至第6行），加上一個比較 (if) 是一定要執行的。

迴圈的頻率計數

✓先根據「迴圈計數器」的範圍算出迴圈重複的次數

✓再乘上迴圈主體的行數。

✓下列的程式片段，計算10組整數除法的商數及餘數

```
1.    for ( i = 0; i < 10 ; i++)
2.    {          Q = m[i] / n[i] ;
3.               R = m[i] % n[i] ;
4.               printf("\n%d ÷ %d = %d … %d", m[i], n[i], Q, R);
                 //cout<<m[i]<<"÷"<<n[i]<<"=" << Q << "…" << R;
5.    }
```

- 迴圈重複10次 (i 從0~9)，迴圈內有3個敘述，迴圈的主體（第2行到第4行）的頻率計數為**30** (= 10 × 3)。
- 第1行for迴圈的頭會執行**11**次，前10次造成迴圈主體的重複執行，第11次發現條件不符而離開迴圈。
- 整個迴圈（迴圈的頭加上迴圈的主體）的頻率計數為 **41** (= 11 + 30)。

for迴圈的計數

單層迴圈

```
1       for ( i = 1 ; i <= n ; i++)
2           result = result + 1 ;
```

迴圈計數器 *i* 的範圍是1… *n*，因此迴圈共重複 *n* 次

簡單的數學式：

$$\text{總次數} = \sum_{i=1}^{n} 1 = n$$

雙層迴圈、內圈固定次數

```
1       for ( i = 1 ; i < = n ; i++)
2           for ( j = 1 ; j < n ; j++)
3                   result = result + 1 ;
```

- 外圈迴圈計數器 i 的範圍是 1… *n*，內圈迴圈計數器 j 的範圍是 1… *n*-1，
- 外圈每執行1圈，內圈就會執行 *n*-1 圈。而外圈會執行 *n* 圈，因此第3行敘述共將執行 *n* (*n*-1) 次。

數學式：

$$\text{總次數} = \sum_{i=1}^{n}\sum_{j=1}^{n-1} 1 = \sum_{i=1}^{n}(n-1) = n\,(n\text{-}1)$$

雙層迴圈、內圈不固定次數

```
1       for ( i = 1 ; i < = n ; i++)
2           for ( j = i+1 ; j <=  n ; j++)
3               result = result + 1 ;
```

由於內圈迴圈計數器 j 的範圍，是**隨著外圈迴圈計數器 i 的範圍改變**的，因此我們針對迴圈計數器 i 的變化來分析：

i 的值	j 的範圍	第3行執行次數
1	2 …… n	n-1
2	3 …… n	n-2
3	4 …… n	n-3
…	…	
n-1	n …… n	1
n	n+1 …… n	0
總次數 ＝ (n-1) + (n-2) + … + 1 = n * (n-1) / 2		

數學式：

總次數 ＝ $\sum_{i=1}^{n}\sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n}(n-(i+1)+1) = \sum_{i=1}^{n}(n-i)$

$= n^2 - \frac{n(n+1)}{2} = \frac{n(n-1)}{2}$

計算*n*階乘（ *n*! ）的函式：

```
   int      factor( int n)
   {        int       result, i ;
1            result = 1 ;
2            i = 1 ;
3            while ( i < = n )
4            {
5                  result = result * i ;
6                   i = i + 1 ;
7            }
8            return result;
   }
```

多1次為最後一次比較條件不成立

計算每一行敘述被執行的次數：

行號	*n*>0 時	*n* <= 0 時
1	1	1
2	1	1
3	*n*+1	1
5	*n*	0
6	*n*	0
8	1	1
總次數	3*n*+4	4

如果 *n*>0，頻率計數為 3*n*+4次，
如果 *n* <= 0 ，頻率計數為 4次

一個有 100 行敘述的程式, 其頻率計數一定比有 1000 行的程式小嗎？也就是 100 行的程式一定比 1000 行的程式快嗎？為什麼？

$$100 > 1000\ ?$$

1-5 Big-O符號

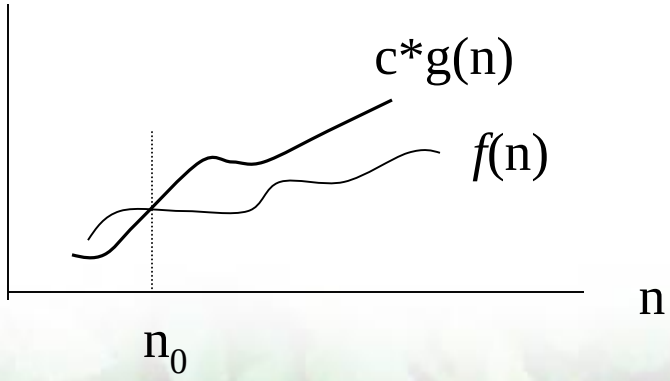
Big-O符號的**數學定義**為：若且唯若 $f(n) = O(g(n))$ 則存在大於 0 的常數 c 和 n_0，使得對所有的 n 值，當$n \geqq n_0$ 時，$f(n) \leqq c * g(n)$ 均成立。

用**數學式**表示為：

$$f(n) = O(g(n)) \Leftrightarrow \exists\ c, n_0>0\ \ni\ \forall\ n \geqq n_0, f(n) \leqq c * g(n)$$

用**口語解釋**為：$f(n)$ 取Big-O符號為O(g(n))，當 n 夠大的時候，g(n)相當於是$f(n)$的**上限**

用**圖示**可表達為：



- $5n^2 + 6n + 9 = O(n^2)$

  → $f(n) = 5n^2 + 6n + 9$, $g(n) = n^2$

  → f 函數取 Big-O 符號為 g 函數

  → $5n^2 + 6n + 9$ 取 Big-O 符號為 $O(n^2)$

- $3n \lg n + 9n + 10 = O(n \lg n)$ ，因為 $n \lg n$ 的次數比 n 大，取最高次項而不計係數時，自然取到 $n \lg n$ ($\lg n = \log_2 n$)

- $9n^2 + 6n \lg n + 9 = O(n^2)$ ，因為 n^2 的次數最大，取最高次項而不計係數時，自然取到 n^2

- $19n^3 + 9n^2 + 6n + 9 = O(n^3)$ ，因為 n^3 的次數最大，取最高次項而不計係數時，自然取到 n^3

- 常見的時間複雜度等級有：

$O(1)$	：常數級 (constant)
$O(\lg n)$	：對數級 (logarithmic)
$O(n)$	：線性級 (linear)
$O(n \lg n)$	：對數─線性級 (log linear)
$O(n^2)$	：平方級 (quadratic)
$O(n^3)$	：立方級 (cubic)
$O(2^n)$	：指數級 (exponential)
$O(n!)$	：階乘級 (factorial)

- 成長速度順序是：

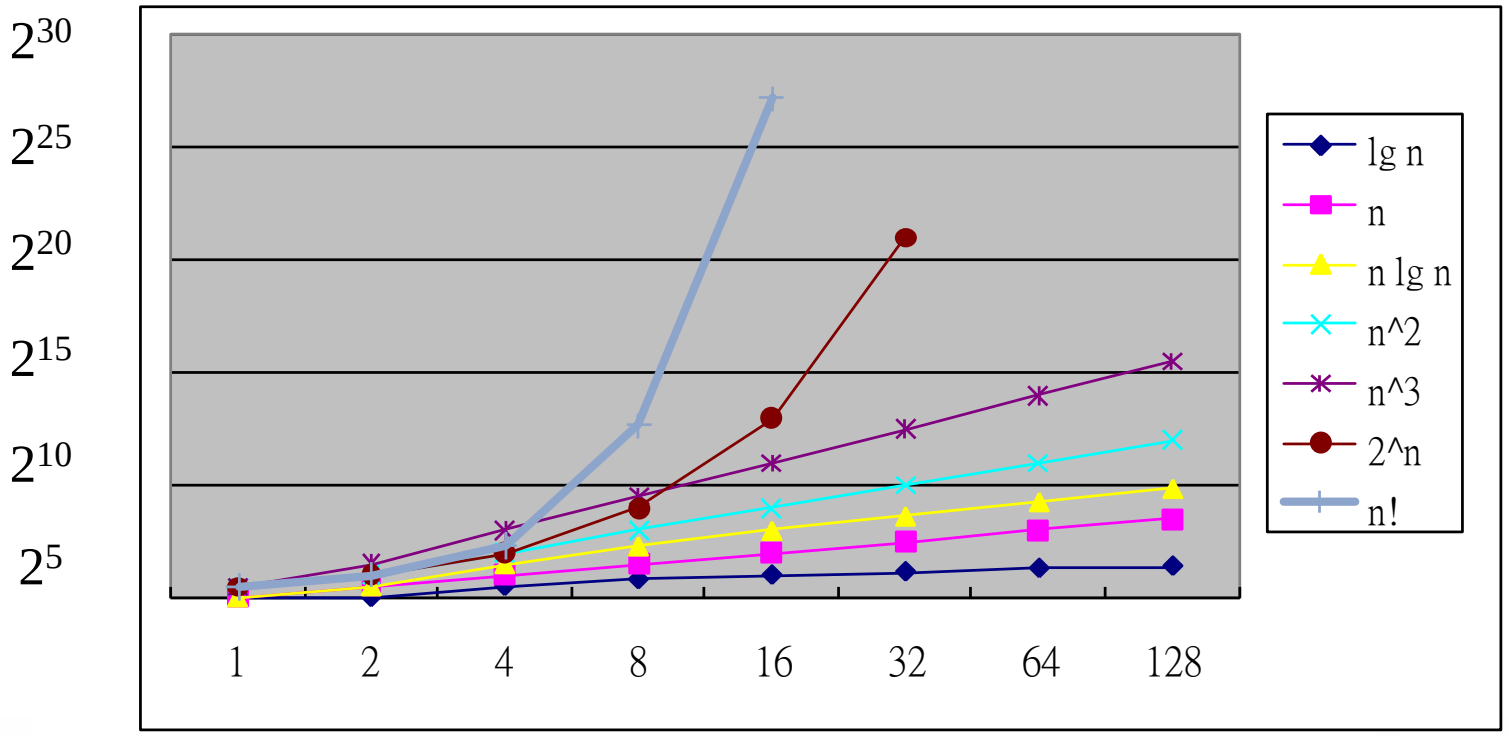
$$O(1) < O(\lg n) < O(n) < O(n \lg n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$$

- 解決同樣問題的三個程式，時間複雜度分別是 $O(n^2)$、$O(n \lg n)$、$O(n^3)$，時間複雜度為$O(n \lg n)$的程式較有效率。這就是概略分級的意義，使我們能簡單比較程式的效率。

n / g(n)	2	10	50	**100**	500	1千	1萬	1百萬
lg n	1	3.3	5.6	**6.6**	9	10	13.3	20
n	2	10	50	**100**	500	1000	10000	10^6
n lg n	2	33	280	**660**	4500	10000	1.3×10^5	2×10^7
n^2	4	100	2500	**10000**	250000	10^6	10^8	10^{12}
n^3	8	1000	125000	**10^6**	1.25×10^8	10^9	10^{12}	10^{18}
2^n	4	1024	10^{15}	**10^{30}**	10^{150}	10^{300}	10^{3000}	10^{300000}
$n!$	2	3628800	3×10^{64}	**9×10^{157}**	10^{1134}	4×10^{2567}	3×10^{35659}	太大

$$O(1) < O(\lg n) < O(n) < O(n \lg n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$$

將各種複雜度隨n值變化的結果繪成圖表：
(X軸代表n值的成長，Y軸代表$g(n)$值的成長)



華山派的武功有「劍宗」和「氣宗」之分, 你認為哪個與「資料結構與演算法」的邏輯較為對應？哪個較為對應「寫程式」的技巧？