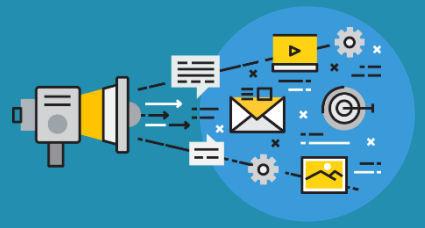
國立金門大學
理工學院

Algorithms and Data Analysis

-演算法與資料分析-

演算法簡介

授課教師：張珀銀



①演算法簡介

②演算法的歷史

③常用的排序算法

演算法簡介

演算法

- 演算法(algorithm)，在數學和電腦科學之中，指一個被定義好的、計算機可施行其指示的有限步驟或次序，常用於計算、數據處理和自動推理。
- 演算法是有效方法，包含一系列定義清晰的指令，並可於有限的時間及空間內清楚的表述出來。

資料來源：https://zh.wikipedia.org/wiki/%E7%BB%9F%E8%AE%A1%E5%AD%A6

演算法

- 演算法中的指令描述的是一個計算，它執行時從一個初始狀態和初始輸入（可能爲空）開始，經過一系列有限而清晰定義的狀態最終產生輸出並停止於一個終態。
- 一個狀態到另一個狀態的轉移不一定是確定的。包括隨機化演算法在內的一些演算法，都包含了一些隨機輸入。

資料來源：https://zh.wikipedia.org/wiki/%E7%BB%9F%E8%AE%A1%E5%AD%A6

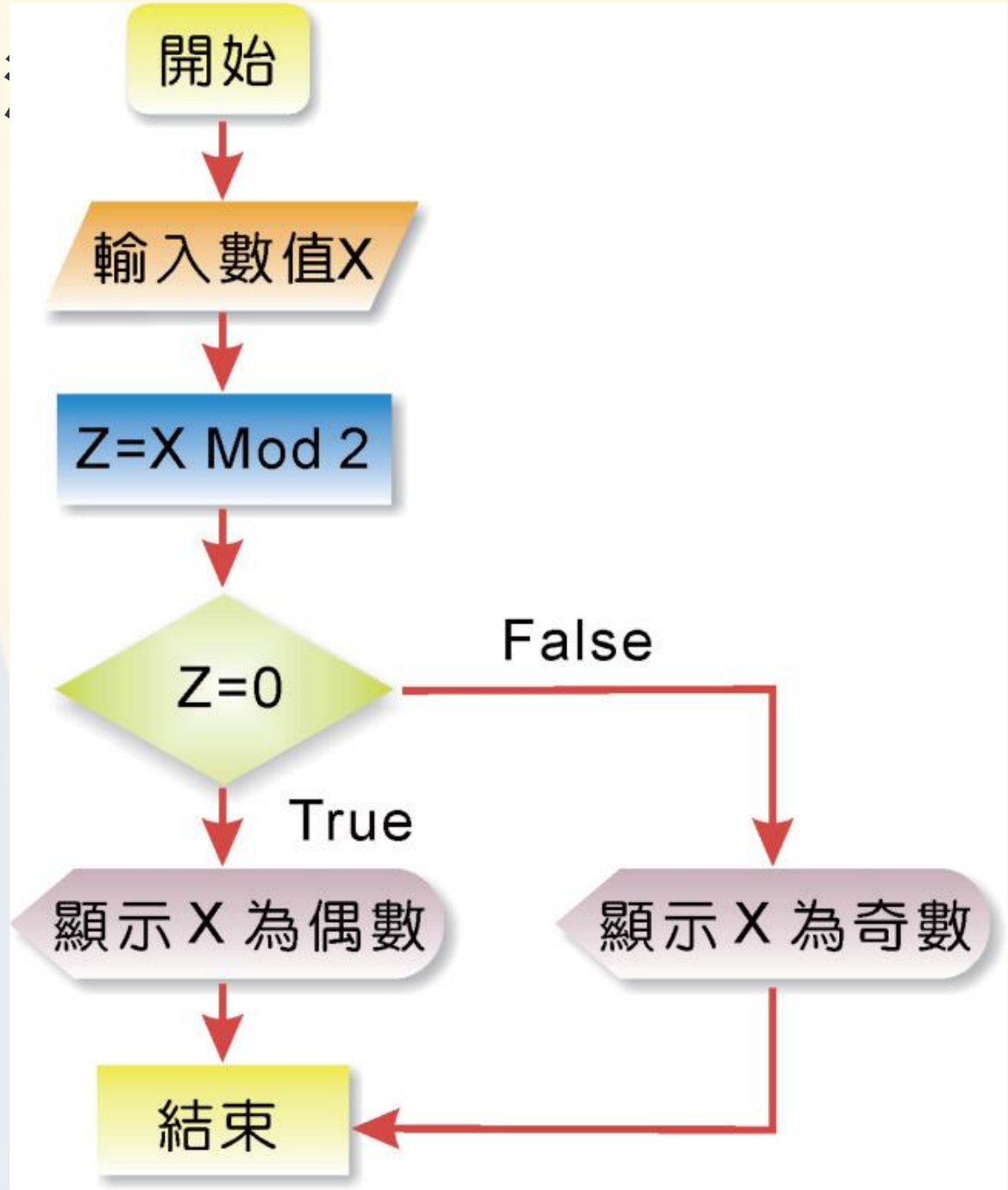
演算法的條件

演算法特性	內容與說明
輸入（Input）	0 個或多個輸入資料，這些輸入必須有清楚的描述或定義。
輸出（Output）	至少會有一個輸出結果，不可以沒有輸出結果。
明確性（Definiteness）	每一個指令或步驟必須是簡潔明確而不含糊的。
有限性（Finiteness）	在有限步驟後一定會結束，不會產生無窮迴路。
有效性（Effectiveness）	步驟清楚且可行，能讓使用者用紙筆計算而求出答案。

演算法的流程圖（Flow Diagram）

- 流程圖（Flow Diagram）也是相當通用的演算法表示法，必須使用某些圖形符號。例如請各位輸入一個數值，並判別是奇數或偶數。

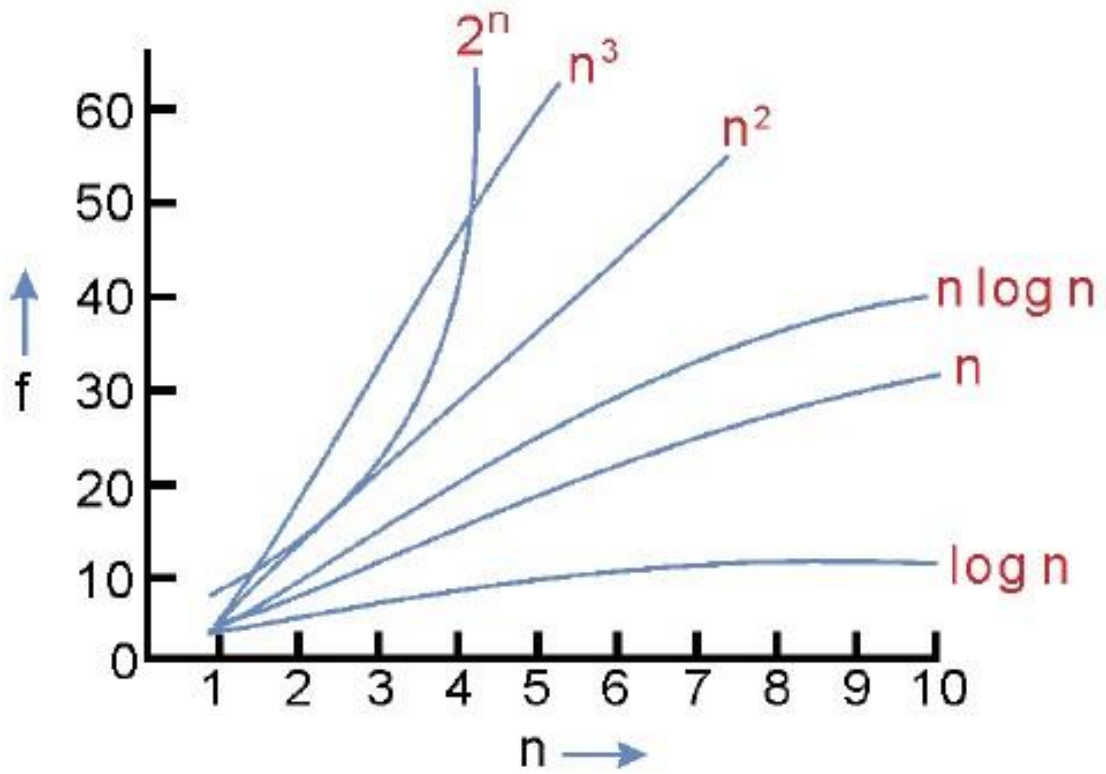
演算



時間複雜度O(f(n))

Big-oh	特色與說明
O(1)	稱為常數時間（constant time），表示演算法的執行時間是一個常數倍。
O(n)	稱為線性時間（linear time），執行的時間會隨資料集合的大小而線性成長。
$O(\log_2 n)$	稱為次線性時間（sub-linear time），成長速度比線性時間還慢，而比常數時間還快。
$O(n^2)$	稱為平方時間（quadratic time），演算法的執行時間會成二次方的成長。
$O(n^3)$	稱為立方時間（cubic time），演算法的執行時間會成三次方的成長。
(2^n)	稱為指數時間（exponential time），演算法的執行時間會成 2 的 n 次方成長。例如：解決 Nonpolynomial Problem 問題演算法的時間複雜度即為 $O(2^n)$。
$O(n\log_2 n)$	稱為線性乘對數時間，介於線性及二次方成長的中間之行為模式。

時間複雜度O(f(n))



對於 $n \geq 16$ 時，時間複雜度的優劣比較關係如下：

$$O(1) < O(\log_2 n) < O(n) < O(n\log_2 n) < O(n^2) < O(n^3) < O(2^n)$$

常見演算法簡介

分治法（Divide and conquer）

- 「分治法」（Divide and conquer）是很重要的一種演算法，核心精神在將一個難以直接解決的大問題依照相同的概念，分割成兩個或更多的子問題，以便各個擊破，分而治之。
- 由於分割問題也是遇到大問題時的解決方式，可以將子問題規模不斷縮小，直到這些子問題足夠簡單到可以解決，最後將各子問題的解合併得到原問題的解答。
- 分治法類型的演算法應用相當廣泛，如快速排序法（quick sort）、遞迴演算法（recursion）、大整數乘法。

遞迴法

- 對程式設計師的實作而言，「函數」（或稱副程式）不單只是能夠被其他函數呼叫（或引用）的程式單元，在某些語言還提供了自身引用的功能，這種功用就是所謂的「遞迴」。
- 從程式語言的角度來說，遞迴的定義是，假如一個函數或副程式，是由自身所定義或呼叫的，就稱為遞迴（Recursion），它至少要定義2 種條件，包括一個可以反覆執行的遞迴過程，與一個跳出執行過程的出口。

- 範例 ch01_01.py 請設計一個計算第n 項費伯那序列的遞迴程式。

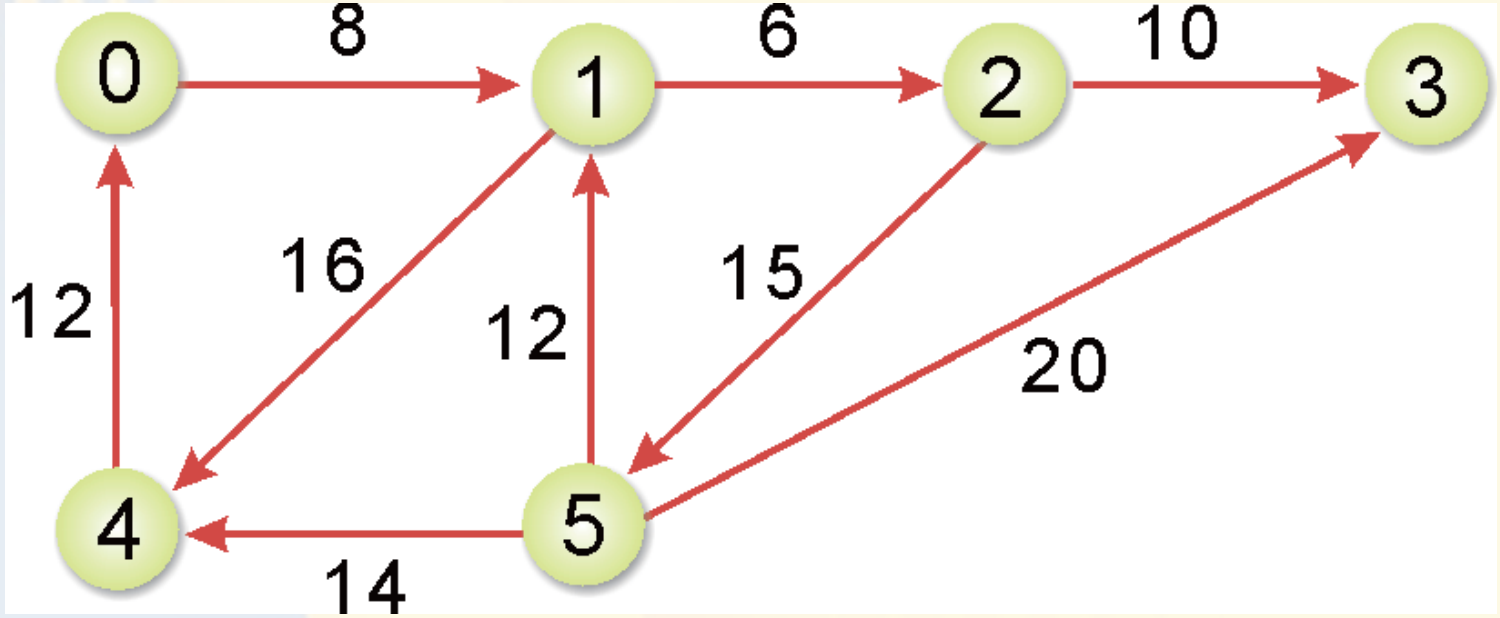
```
01  def fib(n):        # 定義函數 fib()
02      if n==0 :
03          return 0 # 如果 n=0 則傳回 0
04      elif n==1 or n==2:
05          return 1
06      else:    # 否則傳回 fib(n-1)+fib(n-2)
07          return (fib(n-1)+fib(n-2))
08
09  n=int(input('請輸入所要計算第幾個費式數列 :'))
10  for i in range(n+1):# 計算前 n 個費氏數列
11      print('fib(%d)=%d' %(i,fib(i)))
```

- 範例 ch01_01.py 請設計一個計算第n 項費伯那序列的遞迴程式。

```
請輸入所要計算第幾個費式數列:10
fib(0)=0
fib(1)=1
fib(2)=1
fib(3)=2
fib(4)=3
fib(5)=5
fib(6)=8
fib(7)=13
fib(8)=21
fib(9)=34
fib(10)=55
```

貪心法

- 又稱為**貪婪演算法**，方法是從某一起點開始，在每一個解決問題步驟時用貪心原則，採取在當前狀態下最有利或最優化的選擇，不斷的改進該解答，持續在每一步驟中選擇最佳方法，並且逐步逼近給定的目標，當達到某一步驟不能再繼續前進時，演算法停止，以盡可能快地求得更好的解。



動態規劃法

- 主要的做法是如果一個問題答案與子問題相關的話，就能將大問題拆解成各個小問題，其中與分治法最大不同的地方是可以讓每一個子問題的答案被儲存起來，以供下次求解時直接取用。
- 這樣的作法不但能減少再次需要計算的時間，並將這些解組合成大問題的解答，故使用動態規劃將可以解決重覆計算的缺點。

疊代法

- ch01_02.py 請利用for 迴圈設計一個計算1!~n! 的遞迴程式。

```
01  # 以 for 迴圈計算 n!
02  sum = 1
03  n=int(input('請輸入 n='))
04  for i in range(0,n+1):
05      for j in range(i,0,-1):
06          sum *= j      # sum=sum*j
07      print('%d!=%3d' %(i,sum))
08      sum=1
```

疊代法

```
請輸入n=10
0!=   1
1!=   1
2!=   2
3!=   6
4!=  24
5!=120
6!=720
7!=5040
8!=40320
9!=362880
10!=3628800
```

枚舉法

- 「枚舉法」又稱為窮舉法，是一種常見的數學方法，也是日常中使用到最多的一種演算法，核心思想就是：枚舉所有的可能。根據問題要求，一一枚舉問題的解答，或者為了解決問題而分為不重複、不遺漏的有限種情況，一一枚舉並加以解決，最終達到解決整個問題的目的。枚舉法這種分析問題、解決問題的方法，得到的結果總是正確的唯一的缺點就是速度太慢。

回溯法

- 也算是枚舉法中的一種， 對於某些問題而言，回溯法是可以找出所有（或一部分）解的一般性演算法，可隨時避免枚舉不正確的數值。
- 一旦發現不正確的數值，就不遞迴至下一層，而是回溯至上一層，節省時間。這種走不通就退回再走的方式，
- 主要是在搜尋過程中尋找問題的解，當發現已不滿足求解條件時，就回溯返回，嘗試別的路徑，避免無效搜索。

演算法的歷史

Euclid’s Algorithm

- 歐幾里得算法（英語：Euclidean algorithm），是求最大公因數的算法(輾轉相除法)。輾轉相除法首次出現於歐幾里得的《幾何原本》（第VII卷，命題i和ii）中，而在中國則可以追溯至東漢出現的《九章算術》。
- 兩個整數的最大公因數是能夠同時整除它們的最大的正整數。輾轉相除法基於如下原理：兩個整數的最大公因數等於其中較小的數和兩數相除餘數的最大公因數。例如，252和105的最大公因數是21
- $251 = 21 \times 12$；$105 = 21 \times 5$

Euclid’s Algorithm

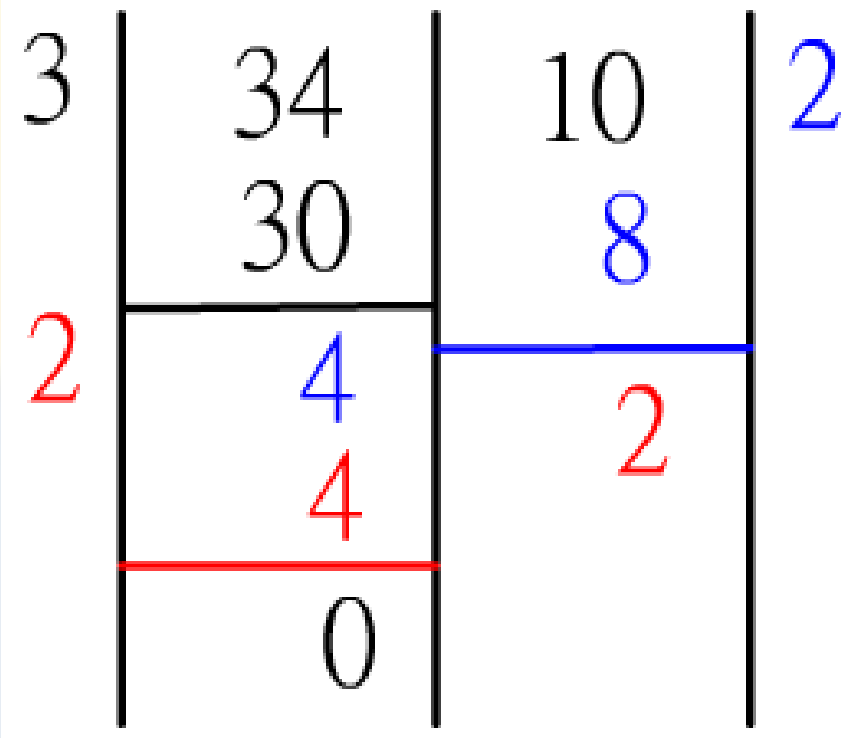
- 因為 252 − 105 = 21 × (12 − 5) = 147 ，所以147和105的最大公因數也是21。在這個過程中，較大的數縮小了，所以繼續進行同樣的計算可以不斷縮小這兩個數直至餘數為零。這時，所剩下的還沒有變成零的數就是兩數的最大公因數。

Euclid’s Algorithm

- 輾轉相除法有很多應用，它甚至可以用來生成全世界不同文化中的傳統音樂節奏。
-]在現代密碼學方面，它是RSA算法（一種在電子商務中廣泛使用的公鑰加密算法）的重要部分。
- 輾轉相除法是現代數論中的基本工具。 輾轉相除法處理大數時非常高效，如果用除法而不是減法實現，它需要的步驟不會超過較小數的位數（十進位下）的五倍。
- 拉梅(法國數學家)於1844年證明了這點，同時這也標誌著計算複雜性理論的開端。

輾轉相除法(Euclidean algorithm)

- 輾轉相除法是歷史上最著名的演算法之一，是求兩數的 最大公因數(GCD) 極快速的方法。
- 原理是兩個數字互相減來減去，最後就會剩下構成兩個數字的共通單位，也就是 最大公因數。



GCD_Euclidean

```python
nums = [int(i) for i in input('輸入數字 ( 逗號分隔 )').split(',')]  # 使用生成式將輸入的數字變成串列
nums.sort()  # 由小到大排序
result = nums[0]  # 取出最小的項目當作預設的最大公因數
while result ≠ 1:  # 如果 result 不為 1，就不斷執行迴圈內容
    for i in range(1, len(nums)):  # 使用 for 迴圈，依序將串列元素取出執行
        r = nums[i] % result  # 取得相除後的餘數
        if r ≠ 0:  # 如果相除後餘數不為 0
            nums.insert(0, r)  # 將餘數插入為串列的第一個項目
            break  # 只要遇到餘數不為 0 就跳出迴圈
    if result ≠ nums[0]:  # 如果 result 不等於串列第一個項目 ( 餘數 )
        result = nums[0]  # 將 result 改為第一個項目 ( 餘數 )，然後重新執行 while 迴圈
    else:
        break  # 如果相等，表示沒有餘數，得到最大公因數
print(result)
```

Google 的搜尋引擎演算法

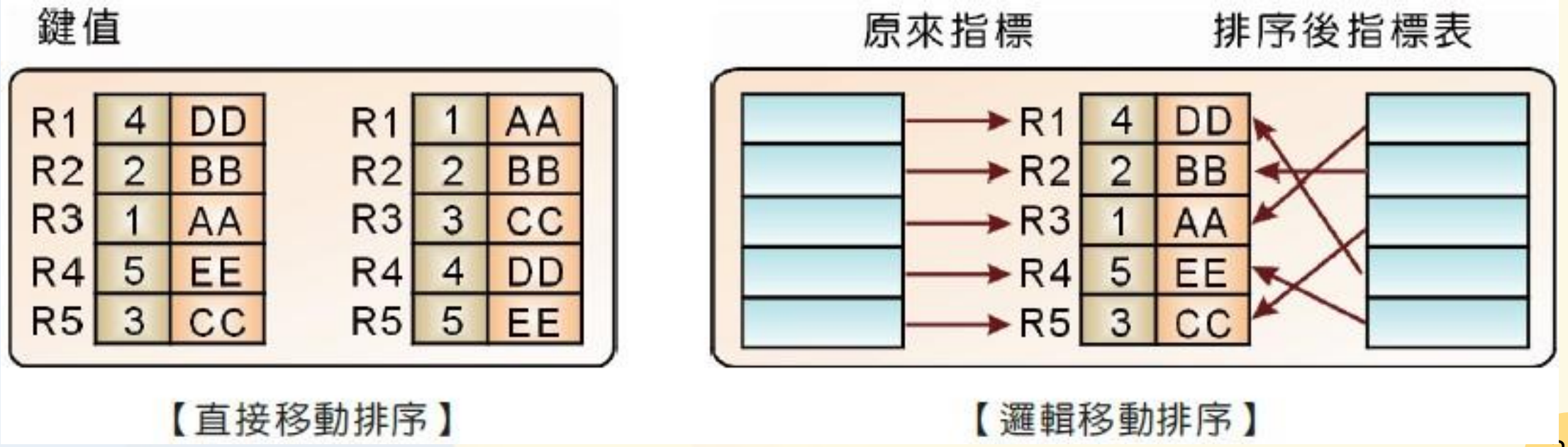
搜尋引擎最佳化（search engine optimization, SEO) SEO 相關歷年重大更新整理

- PageRank 連結排名演算法 (1997)
- Caffeine 咖啡因演算法 (2009)
- Panda 熊貓、內容農場演算法 (2011)
- Hummingbird 搜尋意圖演算法 (2013)
- RankBrain 人工智慧演算法 (2015)
- YMYL/Medic 健康財產、醫療演算法 (2018)
- Penguin 企鵝、作弊連結演算法 (2019)
- BERT 自然語意演算法 (2019)

常見的排序算法

認識排序

- 在排序的過程中，資料的移動方式可分為「直接移動」及「邏輯移動」兩種。「直接移動」是直接交換儲存資料的位置，而「邏輯移動」並不會移動資料儲存位置，僅改變指向這些資料的輔助指標的值。



認識排序

- 兩者間優劣在於：直接移動會浪費許多時間進行資料的更動，而邏輯移動只要改變輔助指標指向的位置就能輕易達到排序的目的。基本上，資料在經過排序後，會有下列三點好處：
  - 1.資料較容易閱讀。
  - 2.資料較利於統計及整理。
  - 3.可大幅減少資料搜尋的時間。

氣泡排序法

- 由小到大排序

❶ 第一次掃瞄會先拿第一個元素 55 和第二個元素 23 作比較，如果第二個元素小於第一個元素，則作交換的動作。接著拿 55 和 87 作比較，就這樣一直比較並交換，到第 4 次比較完後即可確定最大值在陣列的最後面。

第一次掃瞄:

55 23 87 62 16

互換

55 87 23 62 16

不變

87 62 23 55 16

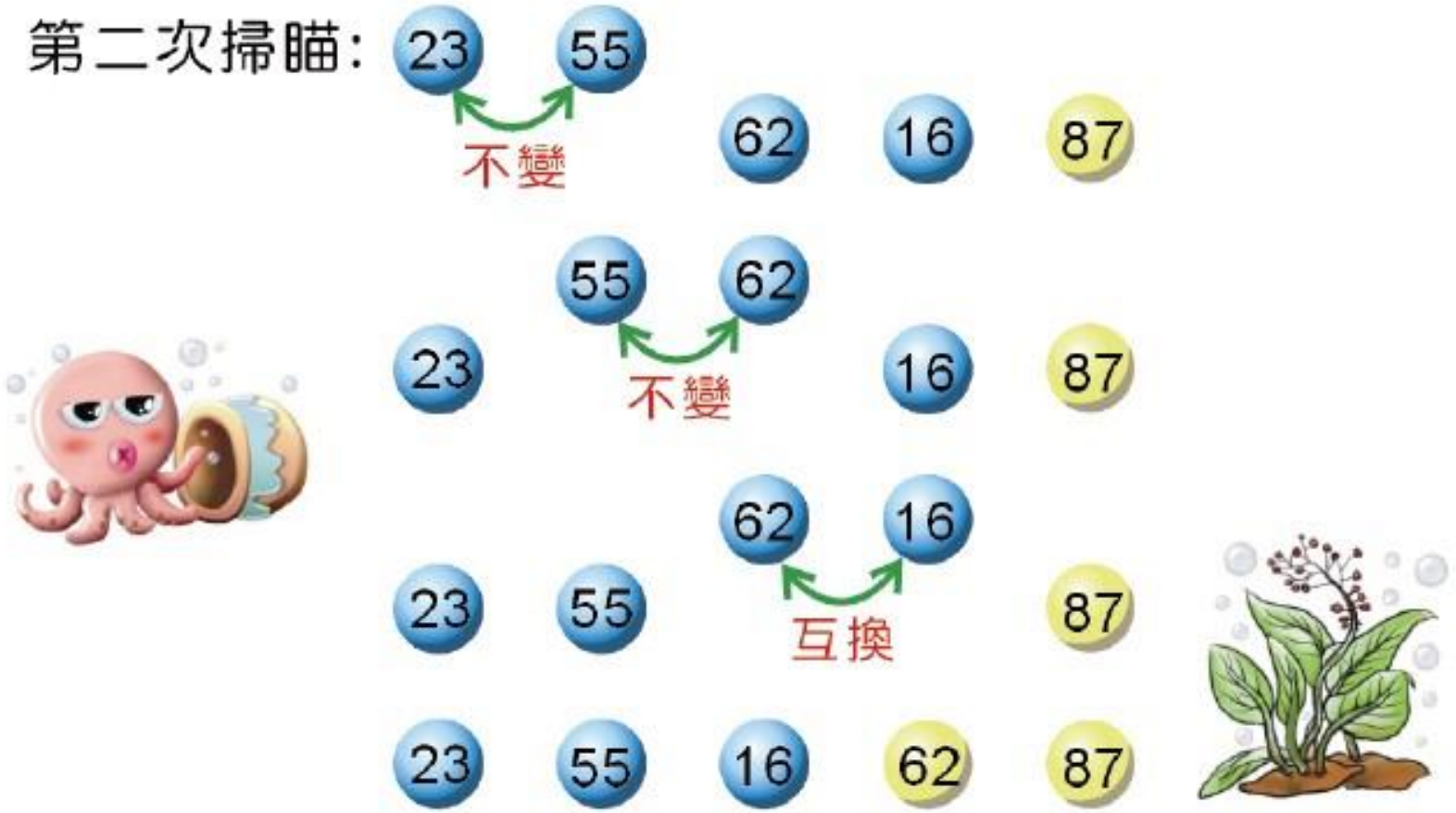
互換

87 16 23 55 62

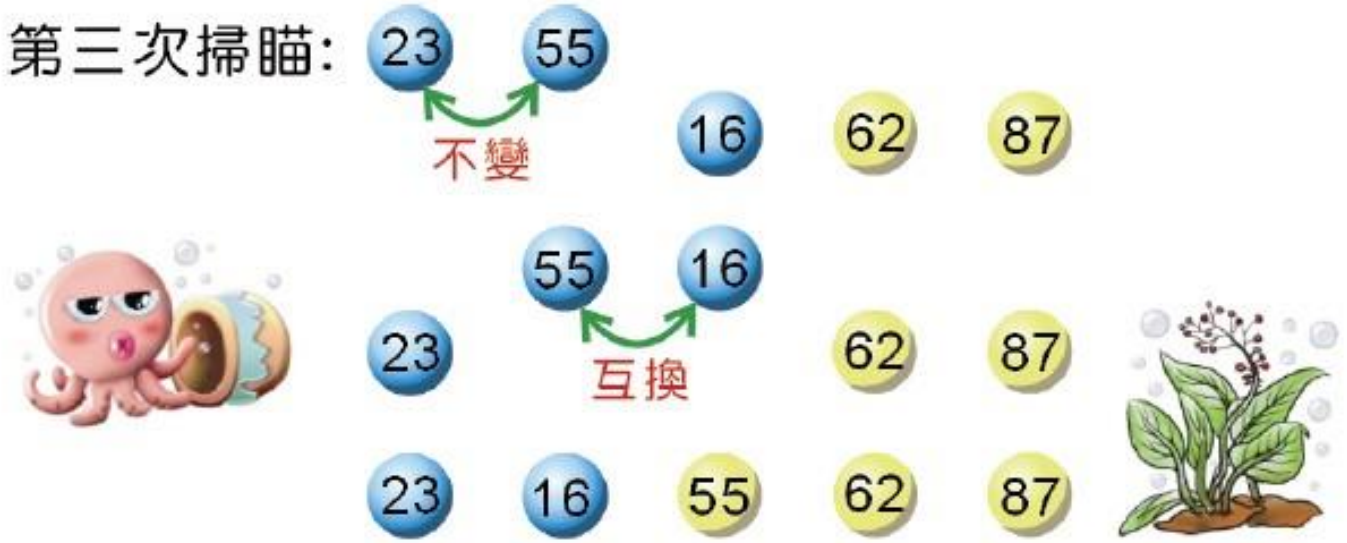
互換

23 55 62 16 87

❷ 第二次掃瞄亦從頭比較起，但因最後一個元素在第一次掃瞄就已確定是陣列最大值，故只需比較 3 次即可把剩餘陣列元素的最大值排到剩餘陣列的最後面。



❸ 第三次掃瞄完，完成三個值的排序。



❹ 第四次掃瞄完，即可完成所有排序。

氣泡排序法實作

- **範例 ch03_01.py** 請設計一Python 程式，並使用氣泡排序法來將以下的數列排序
- 16, 25, 39, 27, 12, 8, 45, 63

```
氣泡排序法：原始資料為：
 16 25 39 27 12  8 45 63
第 1 次排序後的結果是： 16 25 27 12  8 39 45 63
第 2 次排序後的結果是： 16 25 12  8 27 39 45 63
第 3 次排序後的結果是： 16 12  8 25 27 39 45 63
第 4 次排序後的結果是： 12  8 16 25 27 39 45 63
第 5 次排序後的結果是：  8 12 16 25 27 39 45 63
第 6 次排序後的結果是：  8 12 16 25 27 39 45 63
第 7 次排序後的結果是：  8 12 16 25 27 39 45 63
第 8 次排序後的結果是：  8 12 16 25 27 39 45 63
排序後結果為：
  8 12 16 25 27 39 45 63
```

```
16,25,39,27,12,8,45,63
```

```
01  data=[16,25,39,27,12,8,45,63]# 原始資料
02  print('氣泡排序法：原始資料為：')
03  for i in range(8):
04      print('%3d' %data[i],end='')
05  print()
```

```
06
07  for i in range(7,-1,-1): #掃描次數
08      for j in range(i):
09          if data[j]>data[j+1]:#比較，交換的次數
10              data[j],data[j+1]=data[j+1],data[j]#比較相鄰兩數，
                                                  如果第一數較大則交換
11      print('第 %d 次排序後的結果是：' %(8-i),end='') #把各次掃描後
                                                       的結果印出
12      for j in range(8):
13          print('%3d' %data[j],end='')
14      print()
15
16  print('排序後結果為：')
17  for j in range(8):
18      print('%3d' %data[j],end='')
19  print()
```

選擇排序法


原始值: 55 23 87 62 16
❶ 首先找到此數列中最小值後與第一個元素交換。
交換
第一次掃瞄: 55 16
23 87 62
16 23 87 62 55

❷ 從第二個值找起，找到此數列中（不包含第一個）的最小值，再和第二個值交換。

第二次掃瞄：

❸ 從第三個值找起，找到此數列中（不包含第一、二個）的最小值，再和第三個值交換。

第三次掃瞄：

❹ 從第四個值找起，找到此數列中（不包含第一、二、三個）的最小值，再和第四個值交換，則此排序完成。

第四次掃瞄：

- **範例 ch03_02.py** 請設計一Python 程式，並使用選擇排序法來將以下的數列排序：

```
16,25,39,27,12,8,45,63
```

```
01  def showdata (data):
02      for i in range(8):
03          print('%3d' %data[i],end='')
04      print()
05
06  def select (data):
07      for i in range(7):
08          for j in range(i+1,8):
09              if data[i]>data[j]: #比較第 i 及第 j 個元素
10                  data[i],data[j]=data[j],data[i]
```

```
11      print()
12
13  data=[16,25,39,27,12,8,45,63]
14  print('原始資料為：')
15  for i in range(8):
16      print('%3d' %data[i],end='')
17  print('\n-------------------------------------')
18  select(data)
19  print( “排序後資料：")
20  for i in range(8):
21      print('%3d' %data[i],end='')
22  print('')
```

```
原始資料為：
 16 25 39 27 12  8 45 63
-------------------------------------

排序後資料：
  8 12 16 25 27 39 45 63
```