

---

# Python 遞迴 Recursion

臺北科技大學資訊工程系

---

# 遞迴函式(Recursive Function)

## □ 計算 1 到 N 的加總

- 可拆解成計算 1 到 N-1 的加總，然後再加上 N
- 得出遞迴公式:  $\text{sum}(i) = i + \text{sum}(i - 1)$

$$\begin{cases} \text{sum}(1) = 1 \\ \text{sum}(n) = \text{sum}(n - 1) + n, n \geq 2 \end{cases}$$

# 遞迴函式(Recursive Function)

□ 計算  $1 + 2 + 3 + 4 + 5 + \dots + N$  ?

○ 迴圈

```
01 def add(N):  
02     sum = 0  
03     for i in range(1, N + 1):  
04         sum = sum + i  
05     return sum
```

○ 遞迴

```
01 def add(N):  
02     if N == 1:  
03         return 1  
04     else:  
05         return N + add(N - 1)
```

# 遞迴函式(Recursive Function)

## □ 函式執行中不斷自己呼叫自己，範例

### ○ 階層計算(factorial)

➤  $n! = n \times (n-1)!$

$5! = 5 \times 4!$

$= 5 \times 4 \times 3!$

$= 5 \times 4 \times 3 \times 2!$

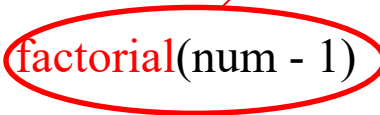
$= 5 \times 4 \times 3 \times 2 \times 1!$

$= 5 \times 4 \times 3 \times 2 \times 1$

$= 120$

$$\begin{cases} f(1) = 1 \\ f(n) = f(n-1) * n, n \geq 2 \end{cases}$$

```
01 def factorial(num):
02     if (num == 1):
03         return num
04     else:
05         return num * factorial(num - 1)
06
07 print(factorial(5))           #120
```



```
01 def factorial(num):
02     if (num == 1):
03         return num
04     else:
05         return num * factorial(num - 1)
06
07 print(factorial(5))           #120
```

# 遞迴函式(Recursive Function)

## □ 設計遞迴函式兩個重點

○ (1) **(base condition)**，結束呼叫的終止條件。

➢ 為避免函式永無止盡自我呼叫 (self-calling)，需設計一個明確終止條件

○ (2) **(general condition)**，遞迴自我呼叫的方式

➢ 問題大小減一

```
01 def factorial_loop (n):
02     factor = 1
03     for i in range(1,n+1):
04         factor *= i
05     return factor
06
07 print(factorial_loop(5)) # 120
```

### 遞迴函式

```
01 def factorial (num):
02     if (num == 1): 終止條件
03         return num
04     else:
05         return num * factorial(num - 1) 自我呼叫
06
07 print(factorial(5)) # 120
```

# Exercise

- 輸入 N，計算 1 ~ N 中，所有偶數相加的總和

遞迴函式

- `def f(N: int)`，recursive

- 觀察偶數：

- $f(6) = 6 + 4 + 2 = 6 + f(4)$

- $f(4) = 4 + 2 = 4 + f(2)$

- $f(2) = 2$

- 觀察奇數：

- $f(7) = f(6) = 6 + 4 + 2 = 6 + f(4)$

- $f(4) = 4 + 2 = 4 + f(2)$

- $f(2) = 2$

```
01 def f(N: int):
02     if (N<2): return 0
03     elif (N==2): return 2
04     else:
05         if (N%2!=0): N = N -1
06         return N + f(N-2)
07
08 def main():
09     print(f(6))    12
10     print(f(7))    12
11     print(f(3))    2
12     print(f(2))    2
13     print(f(1))    0
14
15 main()
```

# Exercise

□ 根據分析:

$$\left\{ \begin{array}{l} f(2) = 0 \\ f(n) = f(n-2) + n, n \geq 2 \end{array} \right. \quad \left\{ \begin{array}{l} f(2) = 0 \\ \text{先將輸入的 } n \text{ 變成 } n-1, \text{ 再算} \\ f(n) = f(n-2) + n, n \geq 2 \end{array} \right.$$

□ Code example:

遞迴函式

```
01 def f(N):
02     if N < 2:
03         return 0
04     else:
05         if N % 2 != 0:
06             N = N - 1
07         return N + f(N - 2)
```

非遞迴函式

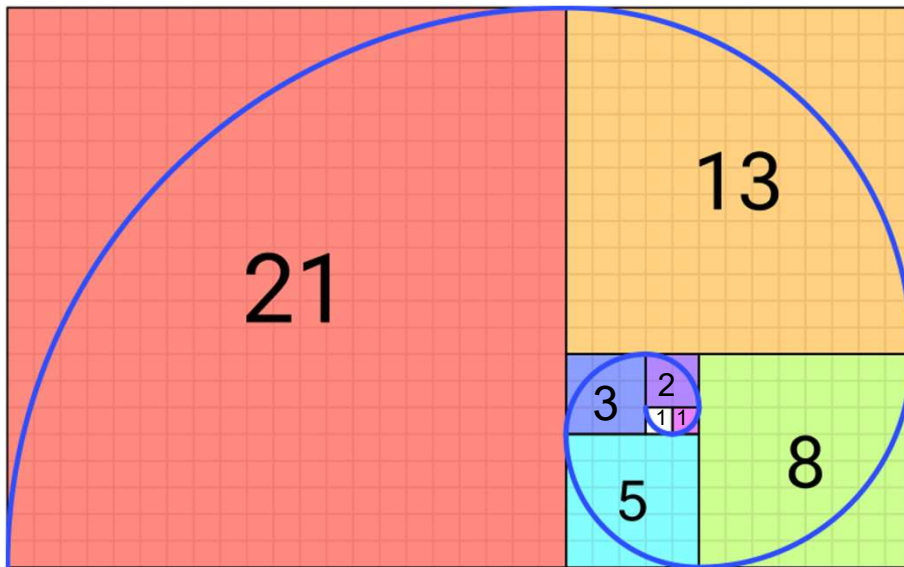
```
01 def f(N):
02     sum = 0
03     for i in range(2, N + 1, 2):
04         sum += i
05     return sum
```

# Exercise

義大利數學家

□ 費氏數列 Fibonacci sequence (0, 1, 1, 2, 3, 5, 8, 13, ...)

- 一個數列，**每一項**都等於其**前兩項**的和，
- **第  $n$  項**等於**第  $n-1$  項**以及**第  $n-2$  項**的和。



| 時間(月) | 初生兔子(對) | 成熟兔子(對) | 兔子總數(對) |
|-------|---------|---------|---------|
| 1     | 1       | 0       | 1       |
| 2     | 0       | 1       | 1       |
| 3     | 1       | 1       | 2       |
| 4     | 1       | 2       | 3       |
| 5     | 2       | 3       | 5       |
| 6     | 3       | 5       | 8       |
| 7     | 5       | 8       | 13      |
| 8     | 8       | 13      | 21      |
| 9     | 13      | 21      | 34      |
| 10    | 21      | 34      | 55      |



# Exercise

□ 費氏數列 Fibonacci sequence (0, 1, 1, 2, 3, 5, 8, 13, ...)

編號每一行程式  
劃出流程圖  
寫下執行編號順序  
寫下輸出內容

$f(0) = 0,$   
 $f(1) = 1,$   
 $f(n) = f(n-1) + f(n-2), n \geq 2$

```
01 def fibonacci(n):  
02     if n == 0:  
03         return 0  
04     elif n == 1:  
05         return 1  
06     else:  
07         return fibonacci(n-1) + fibonacci(n-2)
```

```
print(fibonacci(0))  
print(fibonacci(1))  
print(fibonacci(2))  
print(fibonacci(3))  
print(fibonacci(4))  
print(fibonacci(5))
```

0  
1  
1  
2  
3  
5

# 遞迴函式特性

## □ 遞迴優點

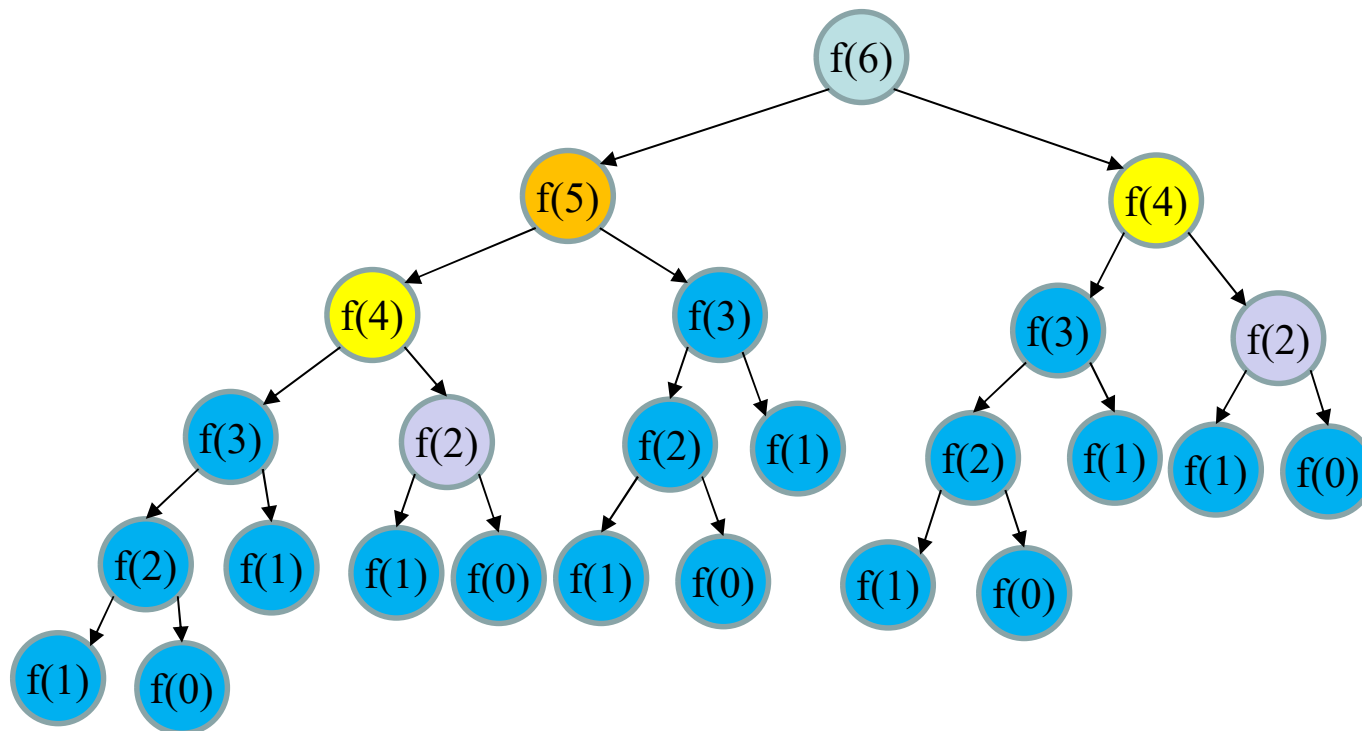
- 容易理解，
- 縮短程式碼長度

## □ 增加時間複雜度 (time complexity) 與空間複雜度 (space complexity)

- 要計算  $\text{fibonacci}(n-1) + \text{fibonacci}(n-2)$  時，要先計算並記住  $\text{fibonacci}(n-1)$  及  $\text{fibonacci}(n-2)$ ，這個「計算」需時間
  - 計算第  $n$  項的值需計算第  $n-1$  項及第  $n-2$  項的值，
  - 而第  $n-1$  項的值又來自於第  $n-2$  項及第  $n-3$  項，
  - 計算第  $n-2$  項又要用到第  $n-3$  項及第  $n-4$  項...
- 函式回傳值，存在記憶體，直到同一層函式被執行完為止，
- 因此佔用大量記憶體空間，且曾計算過的又重新計算，
- 浪費時間空間，造成程式效率不佳 (inefficiency)

# Exercise 改善遞迴效率

- 費式數列 0, 1, 1, 2, 3, 5, 8, ..
  - 使用迴圈
  - 使用遞迴
  - 使用遞迴 + 串列，避免不必要的重複計算



# Exercise 改善遞迴效率

□ 費式數列 0, 1, 1, 2, 3, 5, 8, .. (以空間 list 換取時間)

○ 遞迴版本演化

# 第一版

```
def h1(n):  
    if n==0 or n==1:  
        return n  
    else:  
        return h1(n-1)+h1(n-2)
```

# 第二版

```
def h2(n):  
    if n==0 or n==1:  
        return n  
    else:  
        v1 = h2(n-1)  
        v2 = h2(n-2)  
        return v1 + v2
```

def test():

```
    n=7  
    print(h1(n))  
    print(h2(n))
```

# 第三版

```
def h3(data, n):  
    if n==0 or n==1:  
        return n  
    else:  
        if data[n-1]>0:  
            v1=data[n-1]  
        else:  
            v1 = h3(data, n-1)  
        if data[n-2]>0:  
            v2=data[n-2]  
        else:  
            v2 = h3(data, n-2)  
        return v1 + v2
```

# 第四版

```
def h4(data, n):  
    if n < 2:  
        return data[n]  
    if data[n] == 0:  
        data[n] = h4(data, n-1) + h4(data, n-2)  
    return data[n]
```

def test():

```
    data = [0, 1] + [0 for i in range(20)]  
    print(h3(data, 7)) # 第三版  
    data = [0, 1, 1] + [0 for i in range(20)]  
    print(h4(data, 7)) # 第四版
```

# 第四版

```
[0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
[0, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
[0, 1, 1, 2, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
[0, 1, 1, 2, 3, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
[0, 1, 1, 2, 3, 5, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
[0, 1, 1, 2, 3, 5, 8, 13, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
13
```

# Exercise

□ GCD(60, 36)

○ 短除法

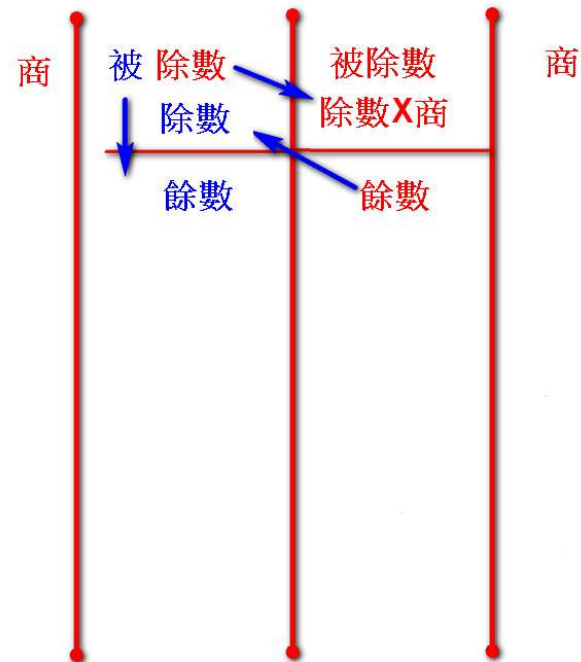
○ 輾轉相除法

□ GCD(1112, 695)

|   |     |     |   |
|---|-----|-----|---|
| 1 | 123 | 321 | 2 |
|   | 75  | 246 |   |
| 1 | 48  | 75  | 1 |
|   | 27  | 48  |   |
| 3 | 21  | 27  | 1 |
|   | 18  | 21  |   |
|   | 3   | 6   | 2 |
|   |     | 6   |   |
|   |     | 0   |   |

3  
GCD

輾轉相除法



短除法

|   |     |
|---|-----|
| 2 | 420 |
| 2 | 210 |
| 5 | 105 |
| 3 | 21  |

7  
GCD

# Exercise

## □ 計算最大公因數 GCD

```
def gcd(x, y): 輾轉相除法
    while (x>0) and (y>0):
        if (x>y):
            x = x%y
        else:
            y = y%x
    return (x if x > y else y)
```

## □ 使用遞迴函式的方式計算最大公因數 GCD

```
01 def gcd(m, n): 輾轉相除法
02     if n == 0: 終止條件
03         return m
04     else:
05         return gcd(n, m%n )
06
07 print(gcd(18, 24))
```

自我呼叫

編號每一行程式  
劃出流程圖  
寫下執行編號順序  
寫下輸出內容

# Exercise

- 一個數列 K 的前兩項是 0、1，之後每一項為  $K_n = 2 * K_{n-1} + 3 * K_{n-2}$ 
  - 使用遞迴函式計算  $K_n$ ，
  - 輸入一個正整數 N，印出該數列的第 N 項。
- 輸入一個字串，計算字串內數字的個數 (noOfDigits(n))
- 輸入一個整數，計算裡面數字的個數 (noOfDigits(n))
- 輸入一個整數，計算數字的加總 (digitSum(n))
- 輸入一字串，判斷是否 palindrome 回文 (checkPalindrome(wordPal, index))
- 輸入字串，輸出字串反轉
- 撰寫 binary search

# Exercise

```
01 #輸入一個字串，計算字串內數字的個數
02 def f(z):
03     if len(z)==0:
04         return 0
05     elif z[0].isdigit():
06         return 1 + f(z[1:])
07     return f(z[1:])
08
09 print(f('1_9f8276xe5432r!1^'))
10 print(f('9'))
```

```
11
1
```



# Exercise Solution

編號每一行程式  
劃出流程圖  
寫下執行編號順序  
寫下輸出內容

```
01 #輸入一個整數，計算數字的加總
02 def digitSum(n):
03     if(n == 0):
04         return 0
05     return ((n%10) + digitSum(n//10)) #呼叫 DigitSum 自己
06 print(digitSum(678))
```

```
01 #判斷迴文
02 def checkPalindrome(x):
03     strLen = len(x)
04     if len(x) == 0: return False
05     if x[0] != x[strLen-1]: return False
06     return checkPalindrome ( x[1: strLen -1 ] )
07     #呼叫自己
08 print(checkPalindrome('abcba'))
09 print(checkPalindrome('abba'))
10 print(checkPalindrome('abcdedcba'))
11 print(checkPalindrome('amobma'))
```

```
01 #輸入一個整數，判斷幾位數
02 def noOfDigits(n):
03     if (n!=0):
04         return 1 + noOfDigits(n//10)
05     else:
06         return 0
07 print(noOfDigits(12301))
```

# Exercise 1/2

□ permutation(排列)稱為 "arrangement number"

○ 'AB'排列 'AB', 'BA'

$$P(AB) = A P(B) + B P(A)$$

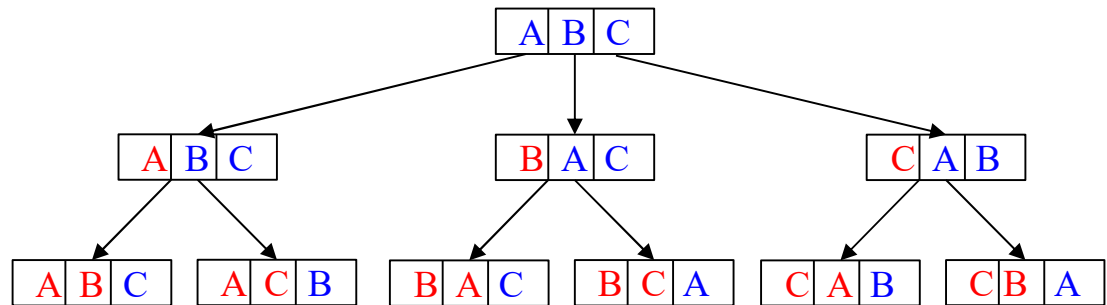
$$P(B) = ? , P(A) = ?$$

○ f('ABC')

➤ 'A'+f('BC')

➤ 'B'+f('AC')

➤ 'C'+f('AB')



# Exercise 1/2

## □ 'ABC' 切割，減小問題

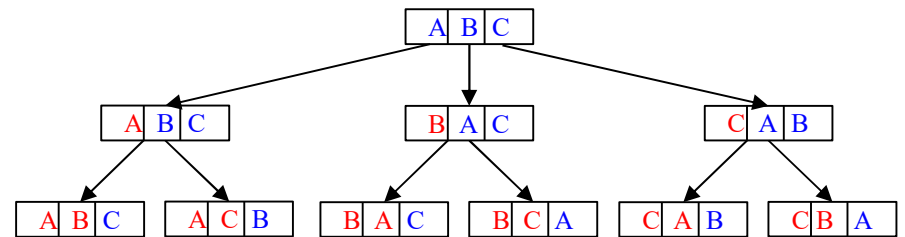
○ 'A' + 'BC'、'B' + 'AC'、'C' + 'AB'

```
01 data = 'ABC'
02
03 x=data[0]
04 y=data[:0]+data[0+1:]
05 print(x,y)
06
07 x=data[1]
08 y=data[:1]+data[1+1:]
09 print(x,y)
10
11 x=data[2]
12 y=data[:2]+data[2+1:]
13 print(x,y)
```

A BC  
B AC  
C AB

```
01 data = 'ABC'
02 for i in range(len(data)):
03     x = data[i]
04     y = data[:i] + data[i+1:]
05     print(x,y)
```

A BC  
B AC  
C AB



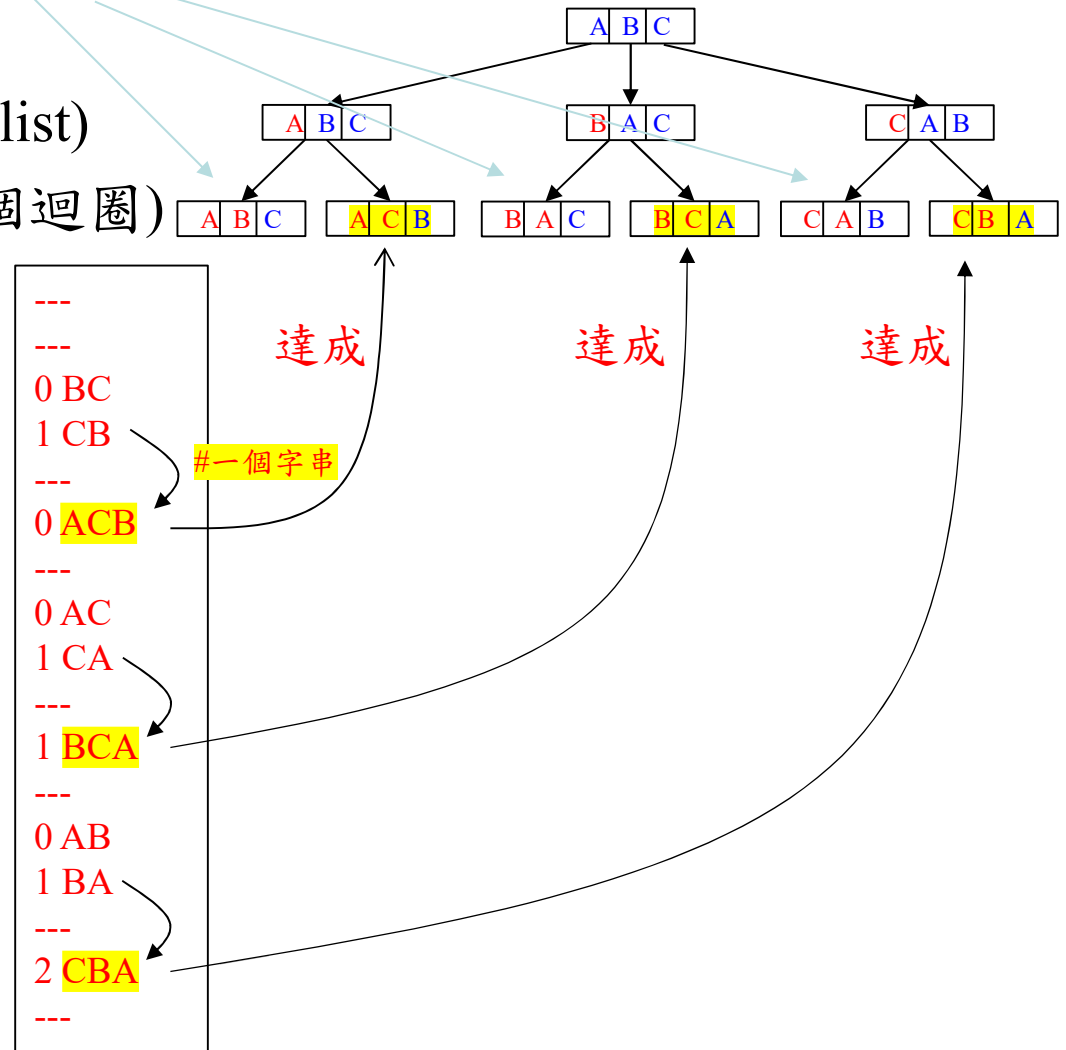
# Exercise 1/3

無法達成

## ❑ 遞迴第一版

- 回傳一個字串(不夠，用list)
- 使用一個迴圈(不夠，2個迴圈)

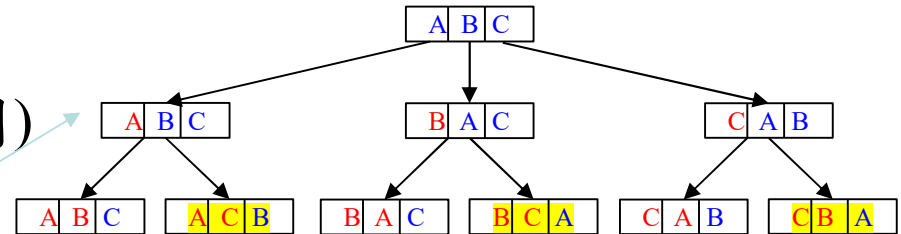
```
01 def f(data):
02     if len(data) <= 1:
03         return data
04     print('---')
05     for i in range(len(data)):
06         x=data[i]
07         y=data[0:i]+data[i+1:]
08         r = x + f(y) #一個字串
09         print(i, r)
10     print('---')
11     return r #回傳一個字串
12     # 排列結果應該不只一個
13 f('ABC')
```



# Exercise 2/3

## 遞迴第二版

- 使用一個迴圈(不夠，兩個迴圈)
- 回傳一個字串(不夠，用list)



```
01 def f(data):
02     if len(data)<=1:
03         return [data]
04     r = []
05     for i in range(len(data)):
06         x = data[i]
07         y = data[:i]+data[i+1:]
08         z = f(y) # 用一個變數存結果
09         for sub in z:
10             r = r + [x+sub] # 排列個數不只一個，要加總
11     return r
12
13 print(f('ABC'))
```

['ABC', 'ACB', 'BAC', 'BCA', 'CAB', 'CBA']

## Exercise 3/3

□ 遞迴第三版，輸入字串，輸出字串排列

```
01 def f(s):
02     if len(s) == 1:
03         return [s]
04     result = []
05     for i in range(len(s)):
06         result += [s[i] + sub for sub in f(s[:i] + s[i+1:])]
07         # 使用list產生器得到所有排列結果
08     return result
09
10 print(f('ABC'))
11 print(f('ABCD'))
```

編號每一行程式  
劃出流程圖  
寫下執行編號順序  
寫下輸出內容

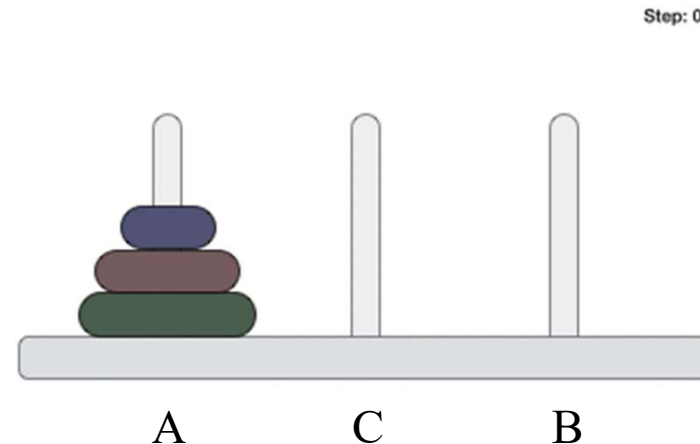
['ABC', 'ACB', 'BAC', 'BCA', 'CAB', 'CBA']

['ABCD', 'ABDC', 'ACBD', 'ACDB', 'ADBC', 'ADCB', 'BACD', 'BADDC',  
'BCAD', 'BCDA', 'BDAC', 'BDCA', 'CABD', 'CADB', 'CBAD', 'CBDA',  
'CDAB', 'CDBA', 'DABC', 'DACB', 'DBAC', 'DBCA', 'DCAB', 'DCBA']

# 河內塔

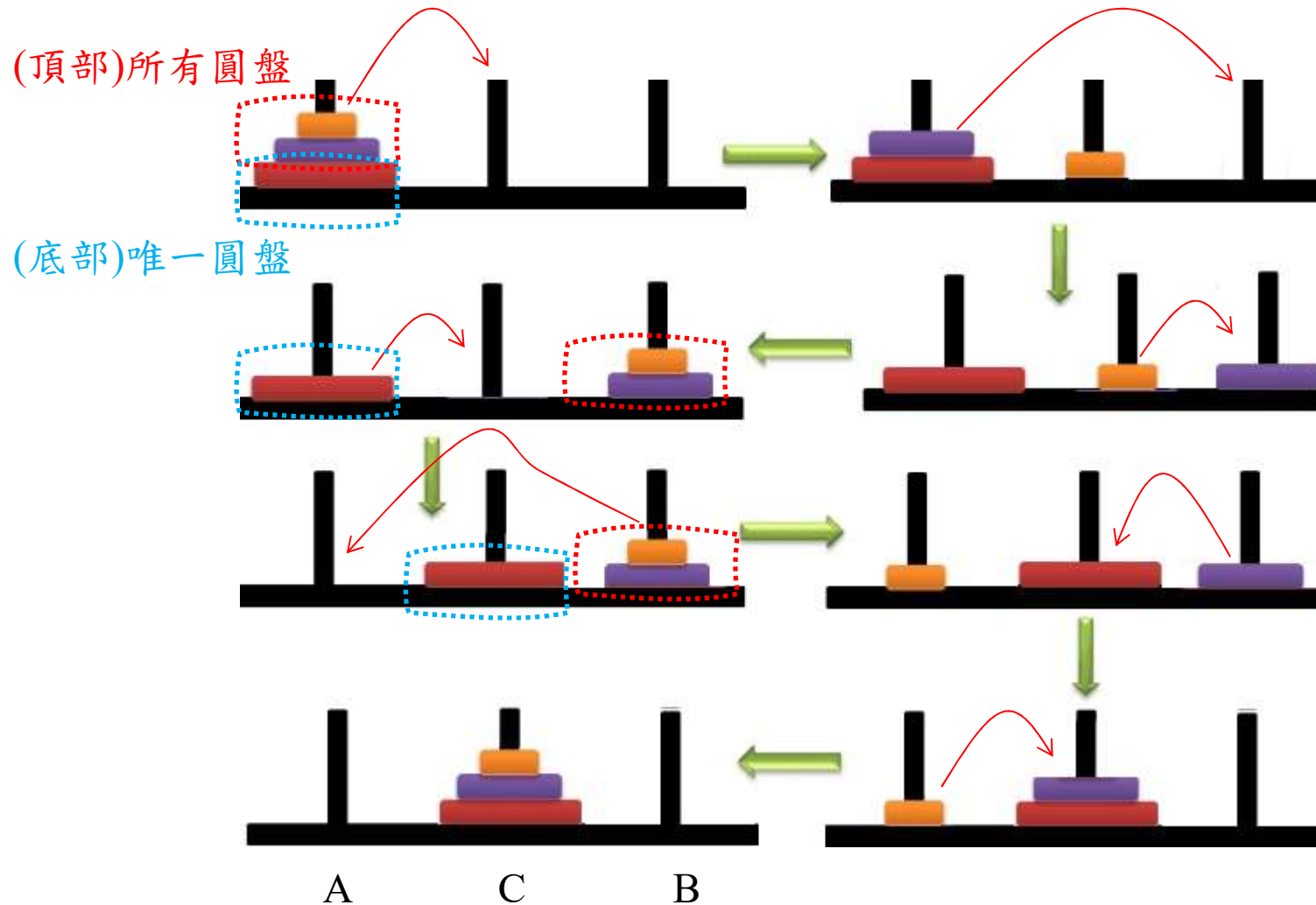
## □ 問題

- 有三根竿子A，B，C。
- A竿上有 N 個 ( $N > 1$ ) 穿孔圓盤，尺寸由下到上依次變小。
- 按下列規則將所有圓盤，從A杆移至C杆：
  - 每次只能移動一個圓盤。
  - 大盤不能疊在小盤上面。
  - 圓盤可以在任意一個竿子上。
- 可將圓盤臨時置於B竿，也可將從A竿移出的圓盤重新移回A竿，但都須遵循上述規則。



# 河內塔

□ 問題，從A杆移至C 杆

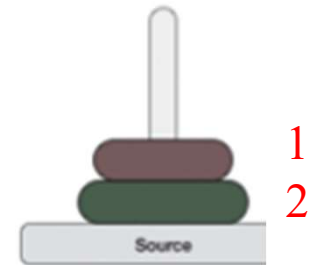




# Exercise 河內塔

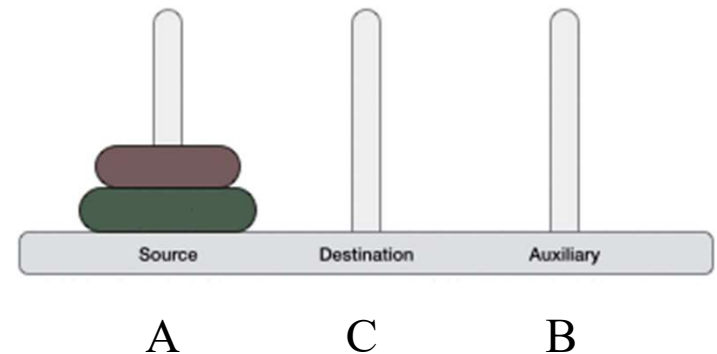
## □ 解決

- 將較小的(頂部)所有圓盤移動到輔助竿子B。
- 將較大的(底部)唯一圓盤移動到目標竿子C。
- 將較小的圓盤從輔助竿子B移動到目標竿子C。
- 請寫出執行流程與編號



```
01 def move(n, x, y):
02     print(n, ': ', x, '->', y)
03     # n層，A搬到C，透過B
04     def hanoi(n, A, C, B):
05         if(n == 1): move(n, A, C)
06         else:
07             hanoi(n-1, A, B, C)
08             move(n, A, C)
09             hanoi(n-1, B, C, A)
10     # 二層，A搬到C，透過B
11     hanoi(2, 'A', 'C', 'B')
```

1 : A -> B  
2 : A -> C  
1 : B -> C

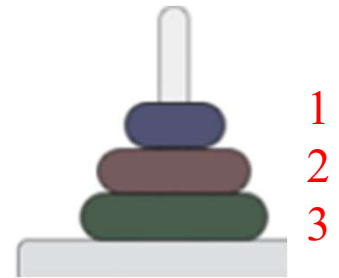


Step: 0

# Exercise 河內塔

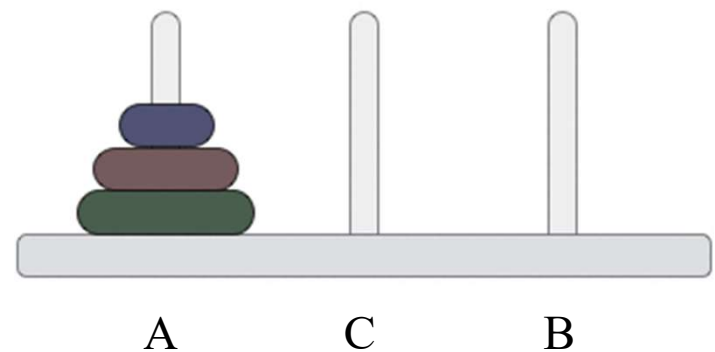
## □ 解決

- 將較小的(頂部)所有圓盤移動到輔助竿子B。
- 將較大的(底部)唯一圓盤移動到目標竿子C。
- 將較小的圓盤從輔助竿子B移動到目標竿子C。
- 請寫出執行流程與編號



```
01 def move(n, x, y):
02     print(n, ': ', x, '->', y)
03     # n層，A搬到C，透過B
04     def hanoi(n, A, C, B):
05         if(n == 1): move(n, A, C)
06         else:
07             hanoi(n-1, A, B, C)
08             move(n, A, C)
09             hanoi(n-1, B, C, A)
10     # 三層，A搬到C，透過B
11     hanoi(3, 'A', 'C', 'B')
```

```
1 : A -> C
2 : A -> B
1 : C -> B
3 : A -> C
1 : B -> A
2 : B -> C
1 : A -> C
```



# 實作搜尋(Search)串列

## □ 二元搜尋(Binary Search)



- 資料須要 **事先排序**，
- 每次都從 **範圍(left ~ right)** 的 **中間點**  $\text{mid} = (\text{left} + \text{right}) / 2$  找起
- **中間點** 太大，往 **左** 找。
- **中間點** 太小，往 **右** 找。

假設找 9

sort

| [0]      | [1] | [2] | [3]                          | [4]                  | [5]   | [6] | [7]       |
|----------|-----|-----|------------------------------|----------------------|---|-----|-----------|
| 1        | 2   | 5   | 7                            | 9                    | 14  | 23  | 26        |
| left = 0 |     |     | mid = 3<br>7 < 9<br>left = 3 |                      |   |     | right = 7 |
|          |     |     |                              |                      | mid = 5<br>(7+3) / 2<br>14 > 9<br>right = 5 |     |           |
|          |     |     |                              | mid = 4<br>(3+5) / 2 |   |     |           |

# Exercise 二元搜尋

- ❑ 請寫出執行流程與編號
- ❑ 請寫出遞迴程式

```
01 def binarySearch(data, left, right, key):
02     while left<=right:
03         mid = (left+right) // 2 #整數除法
04         if data[mid]==key: return mid
05         elif data[mid]>key: right = mid-1
06         else: left = mid+1
07     return -1
08
09 data=[1, 2, 5, 7, 9, 14, 23, 26]
10 print(binarySearch(data, 0, 7, 9)) 4
11 data=[11, 23, 49, 57, 66, 78, 84, 91]
12 print(binarySearch(data, 0, 7, 84)) 6
```

```
01 def binarySearch(data, left, right, key):
02     mid = (left+right)//2
03     if data[mid]==key: return mid
04     if left==right: return -1
05     if data[mid]>key: right = mid-1
06     else: left = mid+1
07     return binarySearch(data, left, right, key)
08
09 data=[1, 2, 5, 7, 9, 14, 23, 26]
10 print(binarySearch(data, 0, 7, 9)) 4
11 data=[11, 23, 49, 57, 66, 78, 84, 91]
12 print(binarySearch(data, 0, 7, 84)) 6
```

# Exercise 數位電路模擬

## □ 模擬一個數位電路。

- 輸入  $n$  是二進位 8 位元，輸出是二進位 4 位元。
- 輸入範圍從 00000000 到 11111111 (十進位 0 ~ 255)。
- 此數位電路內具有回饋迴路，其功能如下：
  - $C(m) = m$  if  $m = 0$  or  $m = 1$
  - $C(m) = C(m/2)$  if  $m$  is even 偶數
  - $C(m) = C((m+1)/2)$  if  $m$  is odd 奇數
- 此電路有一個紀錄器，會記錄跑過幾次回饋迴路，最後輸出為回饋電路跑過的次數。
  - 例如  $m = 00001010$  (十進位 10)，
  - 則電路內部運算回饋電路輸入依序為十進位 5, 3, 2, 1。
  - $C(10) = C(5) = C(3) = C(2) = C(1) = 1$
  - 共跑過 4 次(呼叫  $C()$  四次)。則此電路輸出為 0100 (十進位 4)。

# Exercise 數位電路模擬I

□ 模擬一個數位IC，內有回饋電路與紀錄器電路。

○ 輸入  $m$  是二進位 8 位元，輸出是二進位 4 位元。

○ 輸入範圍從 00000000 到 11111111 (十進位 0 ~ 255)。

○ 數位IC內有一個回饋電路，回饋方式：

➢  $C(m) = m$  if  $m = 0$  or  $m = 1$  (十進位)

➢  $C(m) = C(m/2)$  if  $m$  偶數 (十進位)

➢  $C(m) = C((m+1)/2)$  if  $m$  奇數 (十進位)

➢ 例如  $m = 00001010$  (十進位 10)，則電路回饋依序為十進位 5, 3, 2, 1。

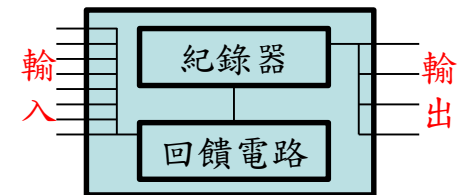
➢  $C(10) = C(5) = C(3) = C(2) = C(1) = 1$ ，共回饋 4 次(呼叫  $C()$  四次)。

○ 數位 IC 內有一個紀錄器，會記錄回饋電路的回饋次數。

➢  $R(m) = [C(m) \text{ 的回饋次數}]$ ，例如  $R(10) = 4$ 。

○ 數位 IC 的輸出為紀錄器所記錄之回饋電路的回饋次數。

➢ 若數位IC的輸入為  $m = 00001010$  (十進位 10)，因回饋電路的回饋次數為4，則此數位 IC 輸出為 0100 (十進位 4)。



# Exercise 數位電路模擬I

- 模擬一個數位IC，內有回饋電路與紀錄器電路。

輸入說明:

二進位 8 bit 位元

第一行是第一個測試案例資料

接著是一行 0 分隔測試資料

第三行是第二個測試案例資料

....

最後 -1 結束

輸出說明:

二進位 4 bit 位元

每一行是一個測試案例資料的結果

Sample Input:

00000000

0

11111111

0

00000001

0

10000000

0

00111111

-1

Sample Output:

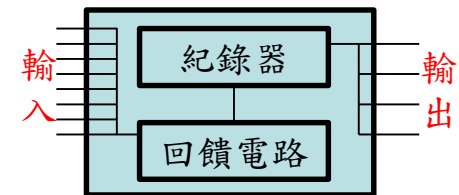
0000

1000

0000

0111

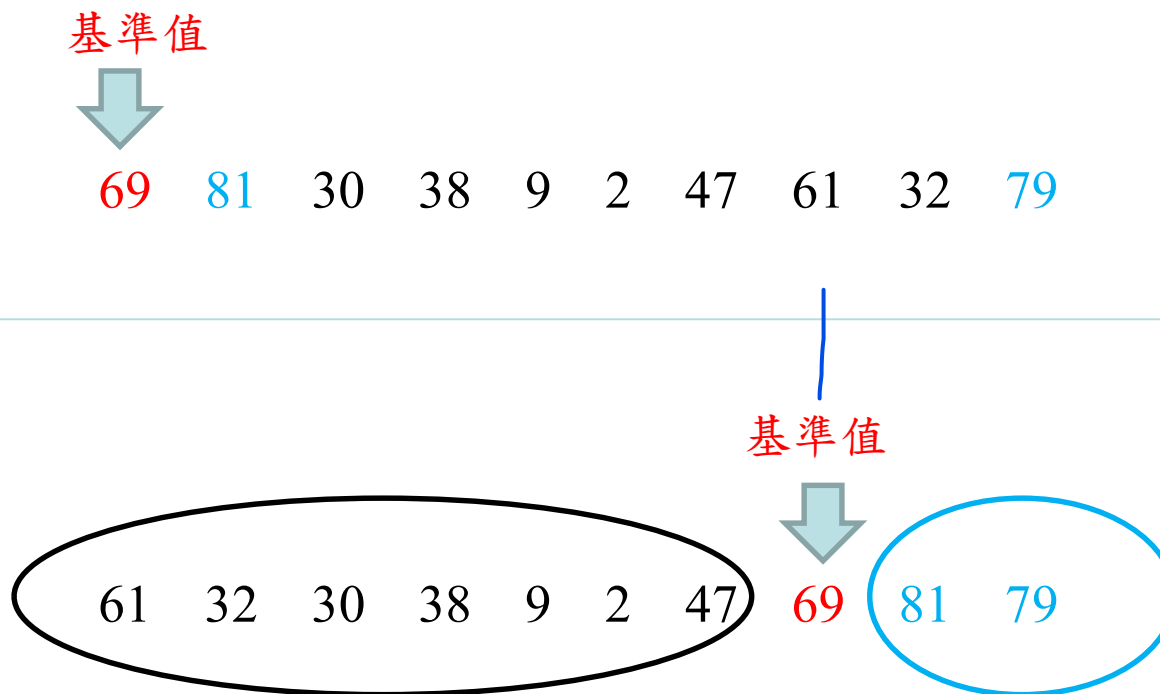
0110



# 快速排序法(Quick sort)

## □ 概念

- 選取某個元素做為**基準值**，令此**基準值**為target。
- 將所有**比 target 小的資料**，都放在 target 左邊；
- 將所有**不比 target 小的資料**，都放在 target 右邊。





# 快速排序法(Quick sort)

## □ 步驟

- 選取第 0 個元素 69 為基準
- 從最右邊往左，找比基準 69 還小的元素 32
- 從最左邊往右，找比基準 69 還大的元素 81
- 兩個元素交換

|    |    |    |    |    |   |   |    |    |    |    |
|----|----|----|----|----|---|---|----|----|----|----|
|    | ●  |    |    |    |   |   |    |    |    |    |
| 索引 | 0  | 1  | 2  | 3  | 4 | 5 | 6  | 7  | 8  | 9  |
| 數值 | 69 | 81 | 30 | 38 | 9 | 2 | 47 | 61 | 32 | 79 |

# 快速排序法(Quick sort)

## □ 概念

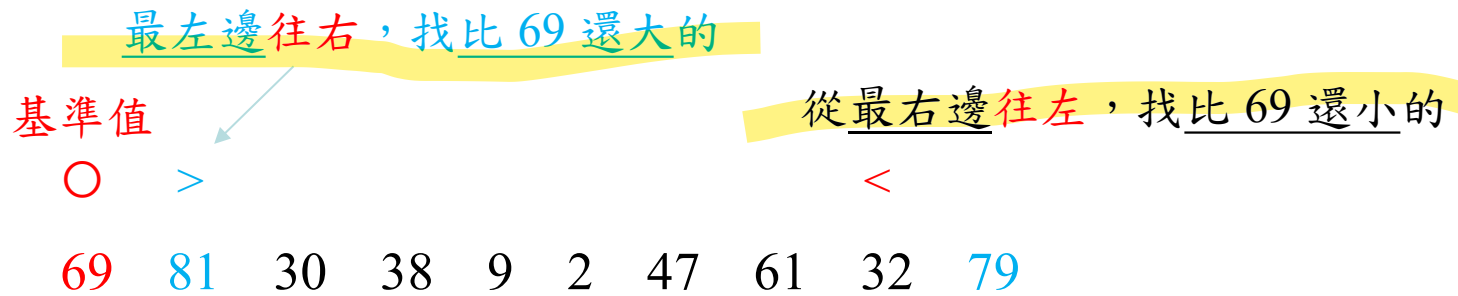
基準值

○

69 81 30 38 9 2 47 61 32 79

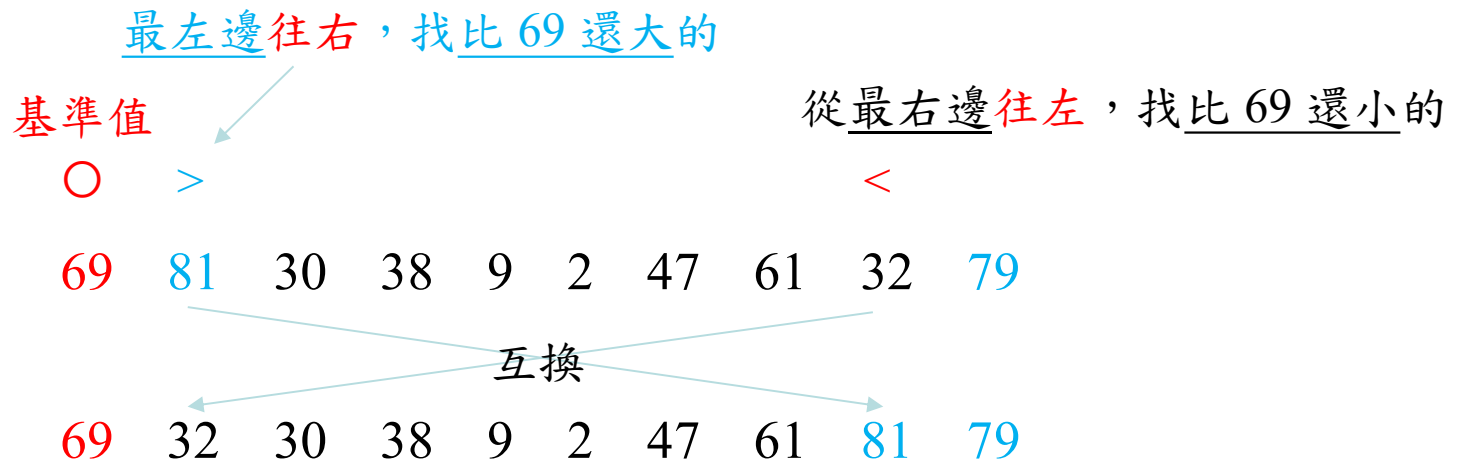
# 快速排序法(Quick sort)

## □ 概念



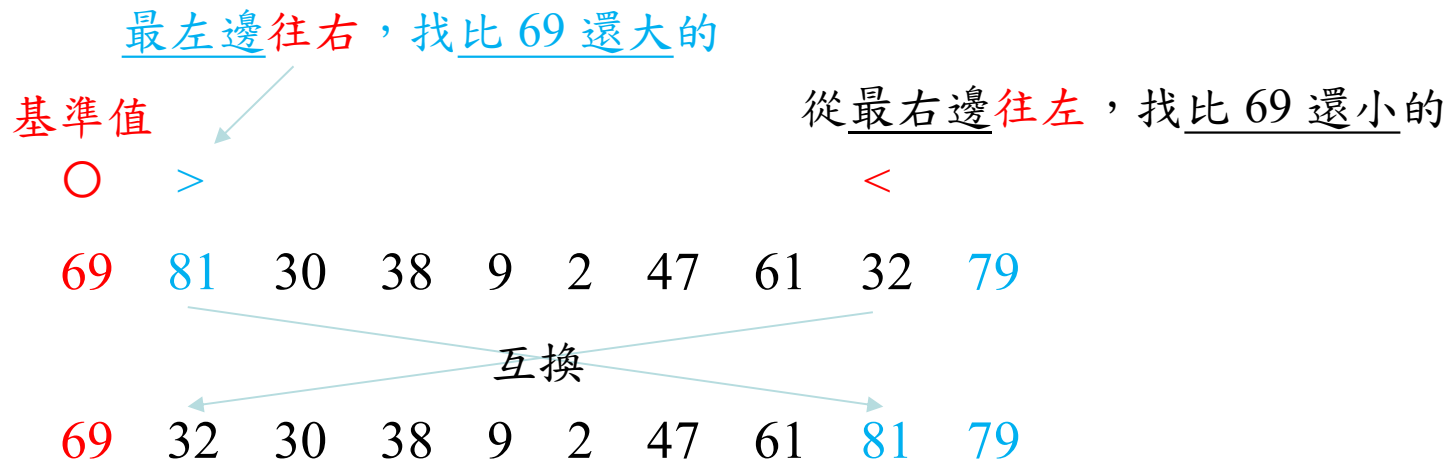
# 快速排序法(Quick sort)

## □ 概念



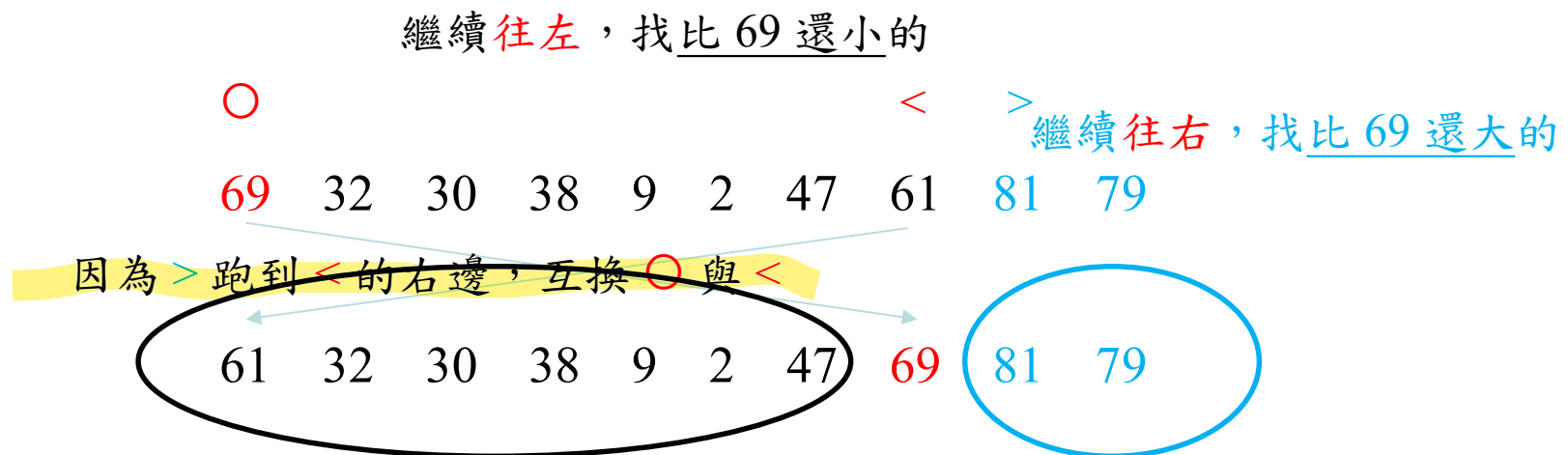
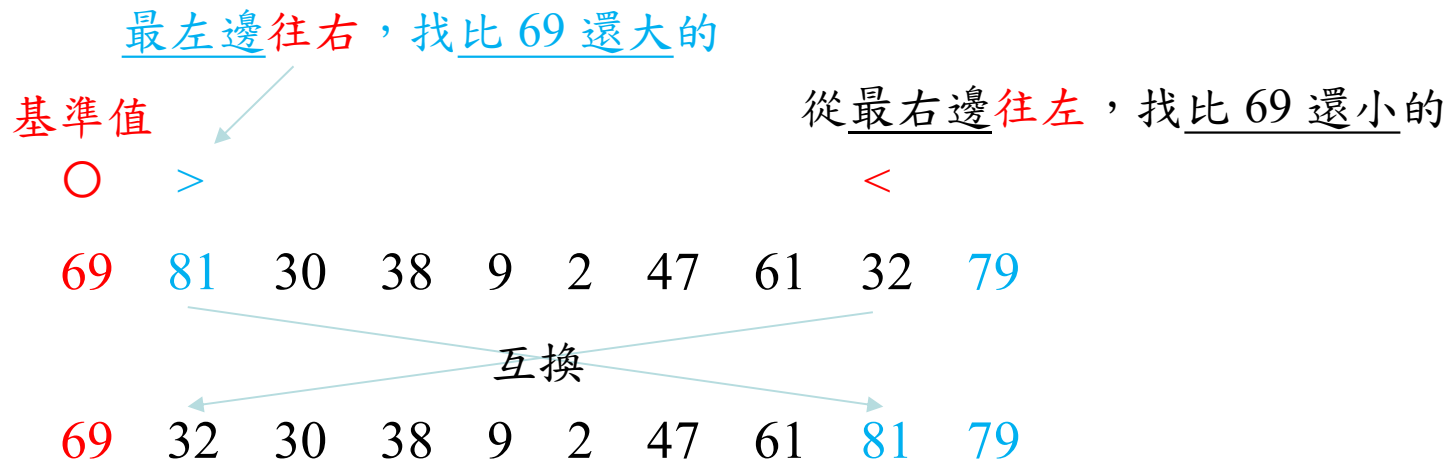
# 快速排序法(Quick sort)

## □ 概念



# 快速排序法(Quick sort)

## □ 概念



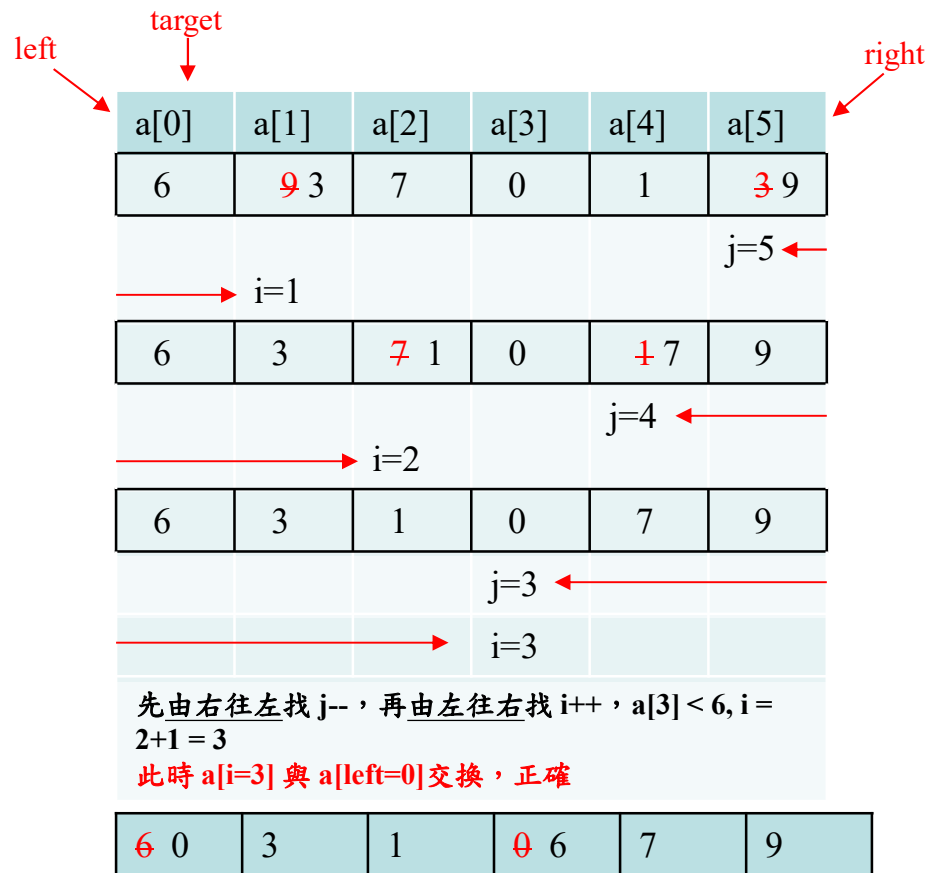
# 快速排序法(Quick sort)

```
01 def QuickSort (data, left, right):
02     if (left >= right): return data
03     i = left
04     j = right
05     target = data[left]
06     while i != j:
07         while (data[j] > target) and (i < j):
08             j = j-1 #從右邊開始找
09         while (data[i] <= target) and (i < j):
10             i = i+1 #從左邊開始找比基準點大，
11                 #如果有找到又沒與從右邊的相遇
12                 #表示 data[i]一定可以換到比較小的
13                 #否則 data[i]一定是小的最邊緣，可以跟中間值交換
14         if(i < j):
15             data[i], data[j] = data[j], data[i] #左右沒相遇則可交換
16         print(data)
17         data[left], data[i] = data[i], data[left] #i是中間值
18         data = QuickSort(data, left, i-1) #處理左半邊
19         data = QuickSort(data, i+1, right) #處理右半邊
20     return data
21
22 print(QuickSort([6, 9, 7, 0, 1, 3], 0, 5))
```

[6, 9, 7, 0, 1, 3]  
[6, 3, 7, 0, 1, 9]  
[6, 3, 1, 0, 7, 9]  
[6, 3, 1, 0, 7, 9]  
[0, 3, 1, 6, 7, 9]  
[0, 3, 1, 6, 7, 9]  
[0, 1, 3, 6, 7, 9]  
[0, 1, 3, 6, 7, 9]

# 快速排序法(Quick sort)

- 須從右邊 **right** 開始往左找  $j--$ ，找比中間點  $\leq$  的準備交換
- 之後才可以左邊 **left** 開始往右找  $i++$ ，找  $>$  中間點的交換





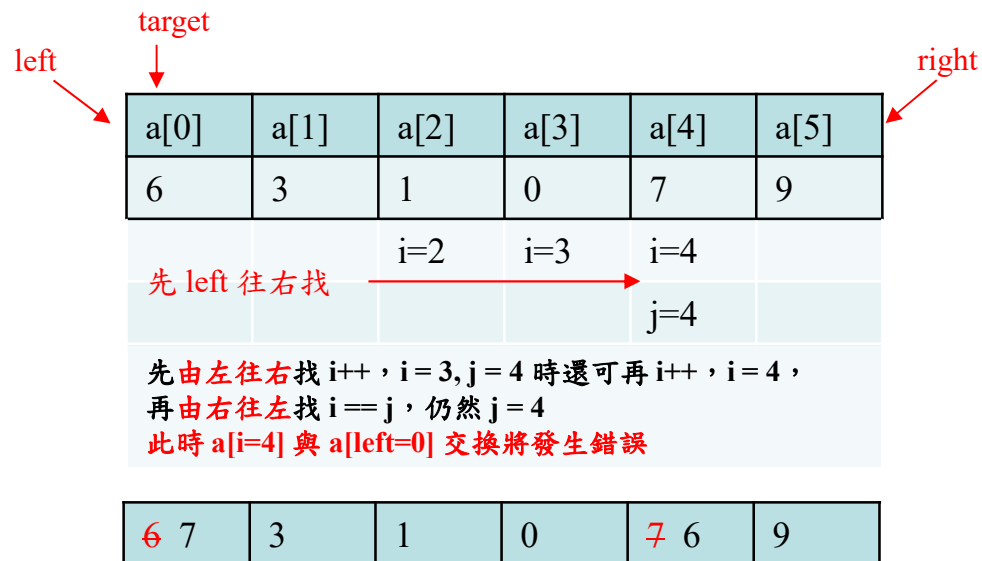
# 快速排序法(Quick sort)

- 從右邊 **right** 開始往左找  $j--$ ，找比中間點  $\leq$  的準備交換，
- 之後左邊 **left** 開始往右找  $i++$ ，找  $>$  中間點的交換



# 快速排序法(Quick sort)

- 若先 **left 往右找**，必停在比  $a[0]$  大地方 ( $i = 4$ )，
- 接著 **right 往左找**，可能停在  $i == j == 4$  ( $a[4] = 7$ )，就會把比  $a[0]$  大 (7)，換到  $a[0]$  地方，**造成錯誤**



# Exercise

- 輸入一串數字(1~9)，及一正整數 N，輸出是否有剛好 N 個總和相等的子集(需包含所有元素) \*\*請用遞迴\*\*

| Input              | Output                             |
|--------------------|------------------------------------|
| 4 3 2 3 5 2 1<br>4 | True<br>(1+4 = 3+2 = 3+2 = 5, 共4組) |
| 4 3 2 3 5 2 1<br>2 | True<br>(1+4+3+2 = 3+2+5, 共2組)     |
| 4 3 2 3 5 2 1<br>6 | False                              |

□ 基本概念

- 假設每一個總和是value，有N個 value，所有輸入數字加總sum
  - N個總和相等， $N \times \text{value} = \text{sum}$
- 若輸入 data = [4, 3, 2, 3, 5, 2, 1], N = 4，有4組，每組加總是5
  - $\text{value} = \text{sum}(\text{data})/N$ ， $(4+3+2+3+5+2+1)/4 = 5$
- 4組，都有加總 value = 5，一次檢查一組，再把這組數字從 data 移除

# Exercise

- 計算步驟(演算法)，若  $\text{data} = [4, 3, 2, 3, 5, 2, 1]$ ,  $N = 4$ 
  - 迴圈檢查四組，都要找到一組數字加總  $\text{value} = 5$ ，放進 bag
    - 一次檢查一組，找出數字存入 bag，再把這組數字(bag)從data移除
      - 第一次
        - »  $\text{data} = [4, 3, 2, 3, 5, 2, 1]$ ,  $N = 4$ ,  $\text{value} = 5$ ,  $\text{bag} = [4, 1]$
        - » data 移除 bag， $\text{data} = [3, 2, 3, 5, 2]$
      - 第二次
        - »  $\text{data} = [3, 2, 3, 5, 2]$ ,  $N = 4$ ,  $\text{value} = 5$ ,  $\text{bag} = [3, 2]$
        - » data 移除 bag， $\text{data} = [3, 5, 2]$
      - 第三次
        - »  $\text{data} = [3, 5, 2]$ ,  $N = 4$ ,  $\text{value} = 5$ ,  $\text{bag} = [3, 2]$
        - » data 移除 bag， $\text{data} = [5]$
      - 第四次
        - »  $\text{data} = [5]$ ,  $N = 4$ ,  $\text{value} = 5$ ,  $\text{bag} = [5]$

# Exercise

## □ findBag：每次檢查找出一組數字的計算步驟

○ data = [4, 3, 2, 3, 5, 2, 1], N = 4, value = 5, bag = [4, 1]

- 設置 index，從0開始，value 每次試著減掉 data[index]，最後剛好減完，成功。
  - 第1次，index=0:  $\text{value} - \text{data}[0] = 5 - 4 = 1$
  - 第2次，index=1， $\text{data}[1] = 3$ ， $\text{value} - \text{data}[1] = 1 - 3 \neq 0$ ，不符合
  - 第3, 4, 5, 6次，index=2~5，2, 3, 5, 2，都不符合
  - 第7次，index=6， $\text{data}[6] = 1$ ， $\text{value} - \text{data}[6] = 0$ ，成功
- data 移除 bag = [4, 1]後，data = [3, 2, 3, 5, 2]

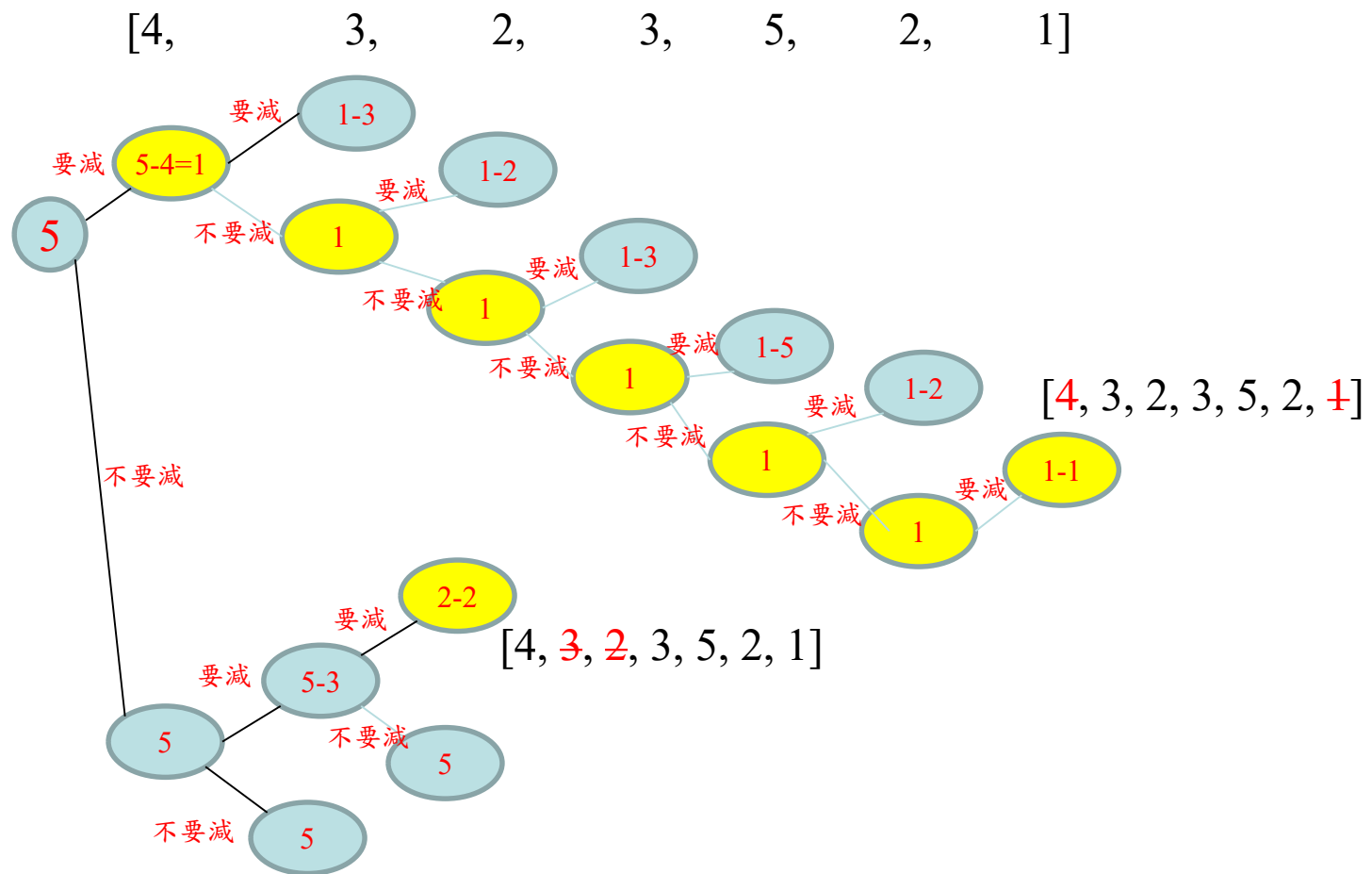
# Exercise

## □ findBag：遞迴的計算步驟

- 基本/結束條件1: data全找了(index-0~6)找不到，
  - $\text{index} > \text{len}(\text{data})$ ，False
- 基本/結束條件2:  $\text{data}[\text{index}] == \text{剩下的value}$ ，成功找到，把  $\text{data}[\text{index}]$  加入 bag，True
- 一般情況
  - 目前遇到的這個要加入bag，value- $\text{data}[\text{index}]$ ，遞迴呼叫，往下找(index+1)
    - 把 $\text{data}[\text{index}]$ 加入bag
  - 目前遇到的這個不要加入bag，value不減 $\text{data}[\text{index}]$ ，遞迴呼叫，往下找(index+1)
- 輸入參數
  - data, index, value, bag

# Exercise

□ findBag : 遞迴的演算法計算步驟



# Exercise – findBag遞迴+ compute迴圈

```
def remove(data, bag):
    for i in bag:
        data.remove(i)

def findBag(data, index, value, bag):
    if index >= len(data) :
        return False
    elif data[index] > value:
        return False
    elif data[index] == value:
        bag.append(data[index])
        return True
    elif findBag(data, index+1, value, bag) == True:
        return True
    elif findBag(data, index+1, value-data[index], bag) == True:
        bag.append(data[index])
        return True
```

```
def compute(data, N):
    if sum(data)%N != 0:
        return False
    value = sum(data)//N
    for i in range(N): #迴圈找N次
        bag = []
        if findBag(data, 0, value, bag) == False:
            return False
        print('bag=', bag, end=', ')
        remove(data, bag)
    return True

print(compute([4, 3, 2, 3, 5, 2, 1], 4))
print(compute([4, 3, 2, 3, 5, 2, 1], 2))
print(compute([1, 2, 3, 5], 2))
print(compute([1, 5, 11, 5], 2))
```

```
bag= [5], bag= [2, 3], bag= [2, 3], bag= [1, 4], True
bag= [2, 5, 3], bag= [1, 2, 3, 4], True
False
bag= [11], bag= [5, 5, 1], True
```



# Exercise – findBag遞迴+ compute遞迴

## □ compute遞迴計算步驟

### ○ 停止條件

- $N=0$ ，停止找，成功
- 找bag，找不到，失敗

### ○ 一般情況

- 找到bag，移除bag，繼續 $N-1$ 尋找

## □ findBag遞迴的計算步驟(每次找第0個，找完刪除第0個)

### ○ 停止條件

- data空的，False
- 找到，value找齊，True

### ○ 一般情況

- 找到目前第0個，第0個加入bag，刪除第0個，往後找True
- 找到目前第0個，第0個不加入bag，刪除第0個，往後找True

### ○ 輸入參數: data, value, bag

# Exercise – 遞迴版2

## □ compute遞迴演算步驟

```
def remove(data, bag):
    for e in bag:
        data.remove(e)

def findBag(data, value, bag):
    if data[0]== value :
        bag.append(data[0])
        return True
    elif len(data)==0 or value <0:
        return False
    if findBag(data[1:], value-data[0], bag)==True:
        bag.append(data[0])
        return True
    if findBag(data[1:], value, bag)==True:
        return True
```

```
bag= [1, 4], bag= [2, 3], bag= [2, 3], bag= [5], True
bag= [1, 2, 3, 4], bag= [2, 5, 3], True
False
bag= [5, 5, 1], bag= [11], True
```

```
def compute(data, N, value):
    if N==0: return True
    bag = []
    if findBag(data, value, bag)==False:
        return False
    print('bag=',bag, end=', ')
    remove(data, bag)
    return compute(data, N-1, value)
```

```
def process(data, N):
    if sum(data)%N!=0:
        return False
    value = sum(data)//N
    return compute(data, N, value)
```

```
print(process([4, 3, 2, 3, 5, 2, 1], 4))
print(process([4, 3, 2, 3, 5, 2, 1], 2))
print(process([1, 2, 3, 5], 2))
print(process([1, 5, 11, 5], 2))
```

# Exercise – 2進位編碼迴圈版

□ findBag :

|                   |      |     |     |     |     |     |     |
|-------------------|------|-----|-----|-----|-----|-----|-----|
| value = 4         |      |     |     |     |     |     |     |
| i =               | [0]  | [1] | [2] | [3] | [4] | [5] | [6] |
| data =            | [ 4, | 3,  | 2,  | 3,  | 5,  | 2,  | 1]  |
| index是第i個元素的二進位編碼 |      |     |     |     |     |     |     |
| code = 0, index = | 0    | 0   | 0   | 0   | 0   | 0   | 0   |
| code = 1, index = | 0    | 0   | 0   | 0   | 0   | 0   | 1   |
| code = 2, index = | 0    | 0   | 0   | 0   | 0   | 1   | 1   |
| code = 3, index = | 0    | 0   | 0   | 0   | 1   | 0   | 0   |
| code = 4, index = | 0    | 0   | 0   | 0   | 1   | 0   | 1   |
| code = 5, index = | 0    | 0   | 0   | 0   | 1   | 1   | 0   |

# Exercise – 2進位編碼迴圈版

## □ 計算步驟

○ 找出所有可能組合，測試哪一種組合，總和是value

```
def remove(data, bag):
    for e in bag:
        data.remove(e)

def findBag(data, code, value):
    N = len(data)
    bag = []
    for i in range(N):
        index = code%2      #data第i個元素2進位編碼
        code = code//2
        if index==1:        # 1取，0不取
            value = value-data[i]
            bag.append(data[i])
    if value!=0: bag=[]
    return bag
```

```
def compute(data, N):
    if sum(data)%N!=0:
        return False
    value = sum(data)//N
    for i in range(2**len(data)):
        bag = findBag(data, i, value)
        if (len(bag)>0):
            print('bag=', bag, end=', ')
            remove(data, bag)
    return True

print(compute([4, 3, 2, 3, 5, 2, 1], 4))
print(compute([4, 3, 2, 3, 5, 2, 1], 2))
print(compute([1, 2, 3, 5], 2))
print(compute([1, 5, 11, 5], 2))
```

```
bag= [3, 2], bag= [3, 2], bag= [4, 1], bag= [5], True
bag= [4, 3, 3], bag= [2, 5, 2, 1], True
False
bag= [11], bag= [1, 5, 5], True
```

---

# END

---

