

C++資料結構與程式設計

遞迴(*Recursion*)

NTU CSIE

大綱

基本遞迴觀念

進階遞迴應用

- 河內塔遊戲 (Hanoi Tower)

遞迴

什麼是遞迴 (Recursion) ?

- 是一種在函式(Function)之中
呼叫自己的程式碼架構

範例

- 寫一個void print(int n)函式
列出整數1~n

思考:

- 要印n前
先印n-1
- ...
- 要印2前
先印1
- 印出1
- 印出2
-
- 印出n

```
#include <stdio.h>
```

```
void print(int n)
```

```
{
```

```
    if(n <= 1) // 遞迴終止條件 {
```

```
        printf("%d\n", n);
```

```
        return;
```

```
    }
```

```
    else {
```

```
        print(n-1); // 遞迴部分 ( 呼叫自己 )
```

```
        printf("%d\n", n);
```

```
    }
```

```
}
```

```
int main()
```

```
{
```

```
    print(10); // 印出1~10
```

```
    return 0;
```

```
}
```

實例與應用

遞迴的特性

- 呼叫與被呼叫的函式具有一個固定的關係，以下列的total函式為例（傳入n印出1加到n的總和）：

```
int total(int n){  
    if (n <=1)                // 遞迴終止條件  
        return 1;  
    else  
        return (n+total(n-1)); // 遞迴部分（呼叫下一層的函式）  
}
```

使用遞迴的步驟

- (1) 了解問題是否適合用遞迴的特性來解題
- (2) 決定遞迴結束條件
- (3) 決定遞迴執行部份

範例：total_n

```
int main() {  
    int x = total_n(3);  
    // 返回位址  
    print(x); 印出6  
    return 0;  
}
```

呼叫

```
int total_n(int d) // d = 3  
{  
    if (d <= 1) // 遞迴終止條件  
        return 1;  
    else  
        return (d + total_n(d-1));  
}
```

呼叫

```
int total_n(int d) // d = 2  
{  
    if (d <= 1) // 遞迴終止條件  
        return 1;  
    else  
        return (d + total_n(d-1));  
}
```

呼叫

```
int total_n(int d) // d = 1  
{  
    if (d <= 1) // 遞迴終止條件  
        return 1;  
    else  
        return (d + total_n(d-1));  
}
```

return d+3

return d+1

return 1

費伯那西數列

費伯那西數列（每個數字為前兩個數字的和）：

- 0, 1, 1, 2, 3, 5, 8, 13, 21.....
- $F(n) = F(n-1) + F(n-2)$

範例程式（迴圈寫法）：印出第N個費伯那西數字

```
int Fib(int n)
{
    int a1, a2=0, a3=1, i;
    if(n==1)
        return 0;
    else if(n<=2)
        return 1;
    for(i=3; i<=n; i++) {
        a1=a2;
        a2=a3;
        a3=a1+a2;
    }
    return a3;
}
```

費伯那西數列 – 遞迴

思考:

- 要算 $F(n)$ 前先算 $F(n-1)+F(n-2)$
- ...
- 要印 $F(3)$ 前先算 $F(2)+F(1)$
- 算出 $F(2)+F(1)$
- ...
- 算出 $F(n-1)+F(n-2)$

我們得到：

- $Fib(1) = 0, Fib(2) = 1, Fib(3) = 1, Fib(4) = 2, Fib(5) = 3 \dots$

[迴圈 / 遞迴]費伯那西數列的比較

```
int Fib(int n)
{
    if(n==1)
        return 0;
    else if(n<=2)
        return 1;
    else
        return Fib(n-1)+Fib(n-2);
}
```

	迴圈寫法	遞迴寫法
程式行數	較長	較短
執行效率	較快	較慢

小練習 (ex03_n!.c)

1.寫一個遞迴函式, 可印出 $n \sim 1$

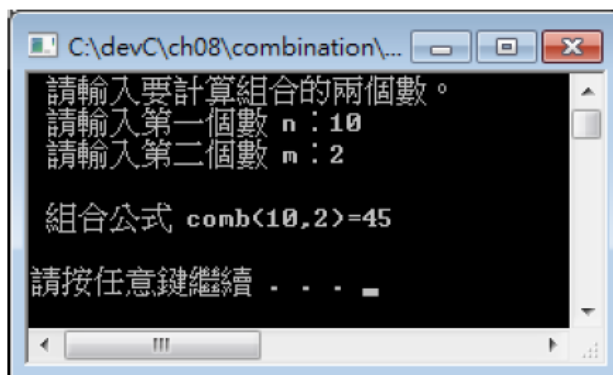
2.寫一個遞迴函式, 算出 $n!$ 之值 ($1*2*3*...*n$)

回家作業 數學組合公式求法 解答



範例：combination.c

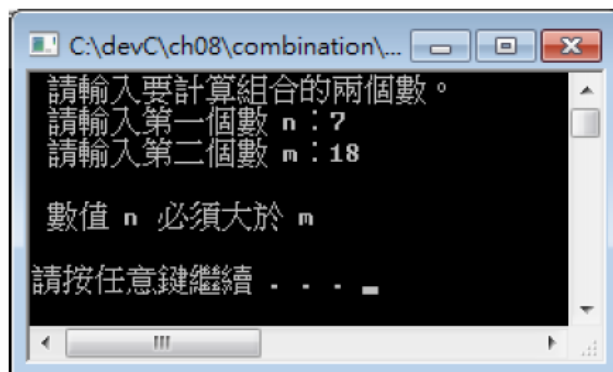
使用遞迴函式求數學組合的 C_m^n 之值。使用者可以輸入組合的第一個數 n 和第二個數 m ，程式執行結果會計算數學組合 C_m^n 之值，如下兩圖：



```
C:\devC\ch08\combination\...
請輸入要計算組合的兩個數。
請輸入第一個數 n : 10
請輸入第二個數 m : 2

組合公式 comb(10,2)=45
請按任意鍵繼續 . . .
```

正確操作



```
C:\devC\ch08\combination\...
請輸入要計算組合的兩個數。
請輸入第一個數 n : 7
請輸入第二個數 m : 18

數值 n 必須大於 m
請按任意鍵繼續 . . .
```

錯誤操作

1. 數學組合公式如下：

$$C_m^n = \begin{cases} 1 & /*\text{如果 } n=m \text{ 或 } m=0, \text{ 則傳回 } 1*/ \\ C_m^{n-1} + C_{m-1}^{n-1} & /*\text{如果 } n \geq m \text{ 則傳回此公式的結果值}*/ \end{cases}$$

2. 例如：欲求出數學組合 C_2^5 之值，則計算過程如下：

$$\begin{aligned} C_2^5 &= C_2^4 + C_1^4 \\ &= (C_2^3 + C_1^3) + (C_1^3 + C_0^3) \\ &= (C_2^2 + C_1^2) + (C_1^2 + C_0^2) + (C_1^2 + C_0^2) + 1 \\ &= 1 + (C_1^1 + C_0^1) + (C_1^1 + C_0^1) + 1 + (C_1^1 + C_0^1) + 1 + 1 \\ &= 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 \end{aligned}$$

```
04 int comb(int, int);    /*宣告 comb 函式原型*/
05
06 int main(int argc, char *argv[])
07 {
08     int numN, numM, ans;
09     printf(" 請輸入要計算組合的兩個數。\\n");
10     printf(" 請輸入第一個數 n:");
11     scanf("%d", &numN);
12     printf(" 請輸入第二個數 m:");
13     scanf("%d", &numM);
14     if(numN>=numM)
15     {
16         ans=comb(numN, numM);
17         printf("\\n 組合公式 comb(%d,%d)=%d\\n\\n",numN, numM, ans);
18     }
19     else
20     {
21         printf("\\n 數值 n 必須大於 m \\n\\n");
22     }
23     system("PAUSE");
24     return 0;
25 }
```

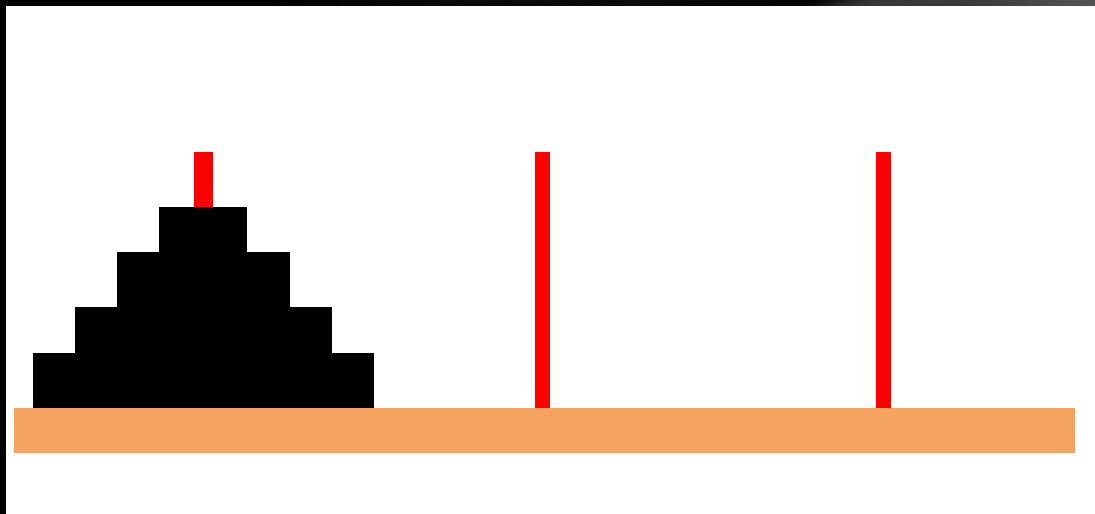
```
27 int comb(int n, int m)
28 {
29     if (n==m || m==0)
30     {
31         return 1;
32     }
33     else
34     {
35         return comb(n-1, m)+comb(n-1, m-1);
36     }
37 }
```

大綱

基本遞迴觀念

進階遞迴應用

- 河內塔遊戲 (Hanoi Tower)

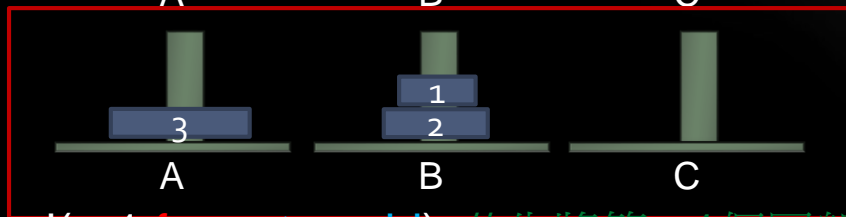
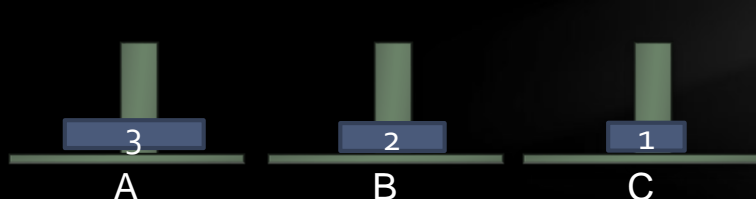
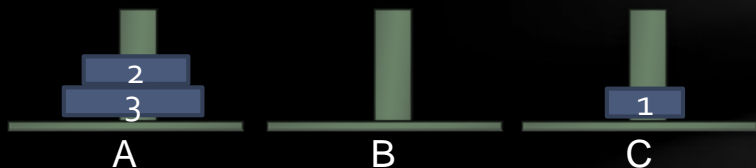


河內塔(Hanoi Tower)

河內塔問題：在A、B、C三個塔上有數個圓盤，請求出將圓盤從A塔全數搬移到C塔的順序，且大的圓盤不可以疊到小圓盤上。

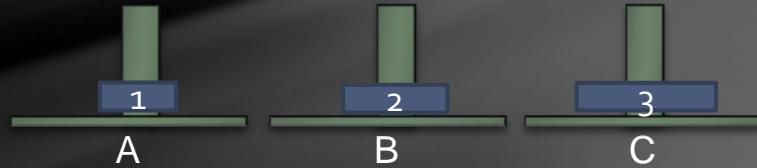
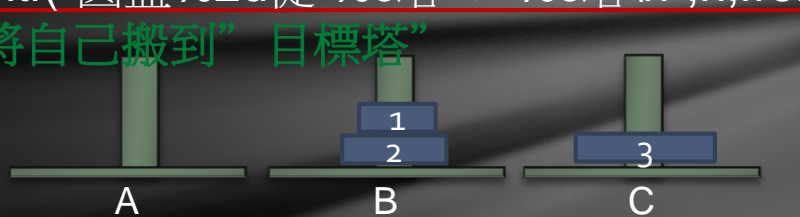
```
void hanoi (int n, char from, char mid, char to)
```

```
// 在搬動第n個圓盤時
```



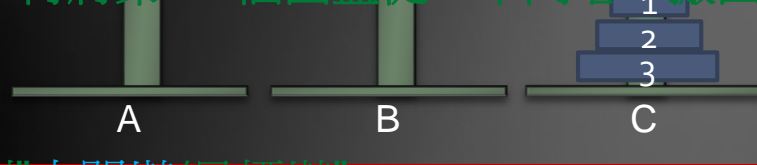
```
printf("圓盤%2d從 %c塔 -> %c塔\n",n,from,to);
```

```
// 將自己搬到"目標塔"
```



```
hanoi(n-1,mid,from,to);
```

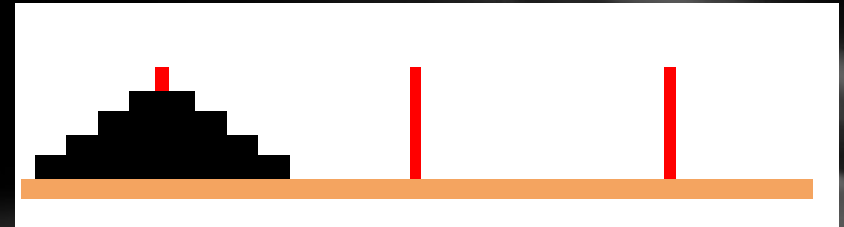
```
// 再將第n-1個圓盤從"中間塔"搬回來
```



```
hanoi(n-1,from,to,mid); // 先將第n-1個圓盤搬到"中間塔/目標塔"
```

河內塔(Hanoi Tower)

題目：印出河內塔中，編號1~n的圓盤從A塔到C塔的搬動過程。

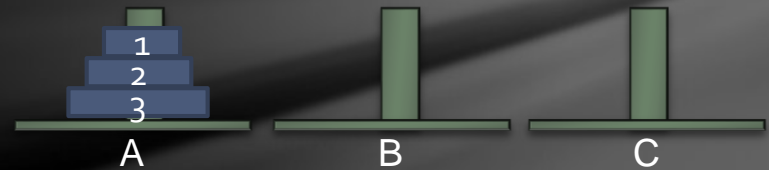


目標：

- 將編號為n的圓盤從「起點塔」搬到「目標塔」

步驟：

- 對於每個編號為n的圓盤：
 - (1) 將編號為n-1的圓盤從「起點塔」搬到「中間塔」。
 - (2) 將自己(編號為n的圓盤)搬到「目標塔」，印出過程。
 - (3) 將編號為n-1的圓盤從「中間塔」搬到「目標塔」。
 - (4) 終止條件：對於編號為n-1的圓盤時，只搬動到編號為1的圓盤為止。



河內塔範例程式碼

範例程式碼

```
printf("請輸入塔數: ");  
scanf("%d", &n);  
hanoi(n, 'A', 'B', 'C');
```

- 功能：印出河內塔中，編號1~n的圓盤從A塔到C塔的搬動過程。

```
void hanoi (int n, char from, char mid, char to)  
{ // 在搬動第n個圓盤時  
    if(n==0)  
        return ;  
    // 先將第n-1個圓盤搬到“中間塔”  
    hanoi(n-1,from, to, mid);  
    // 將自己搬到“目標塔”  
    printf("圓盤%2d從 %c塔 -> %c塔\n",n,from,to);  
    // 再將第n-1個圓盤從“中間塔”搬回來  
    hanoi(n-1,mid, from, to);  
}
```


小練習

請利用河內塔程式碼，讓使用者輸入「圓盤總數 n 」，計算各圓盤被搬動多少次(圓盤最多不超過100個)？

大綱

基本遞迴觀念

進階遞迴應用

- 河內塔遊戲 (Hanoi Tower)

分治法 (Divide and Conquer)

- 快速排序法 (Quick Sort)

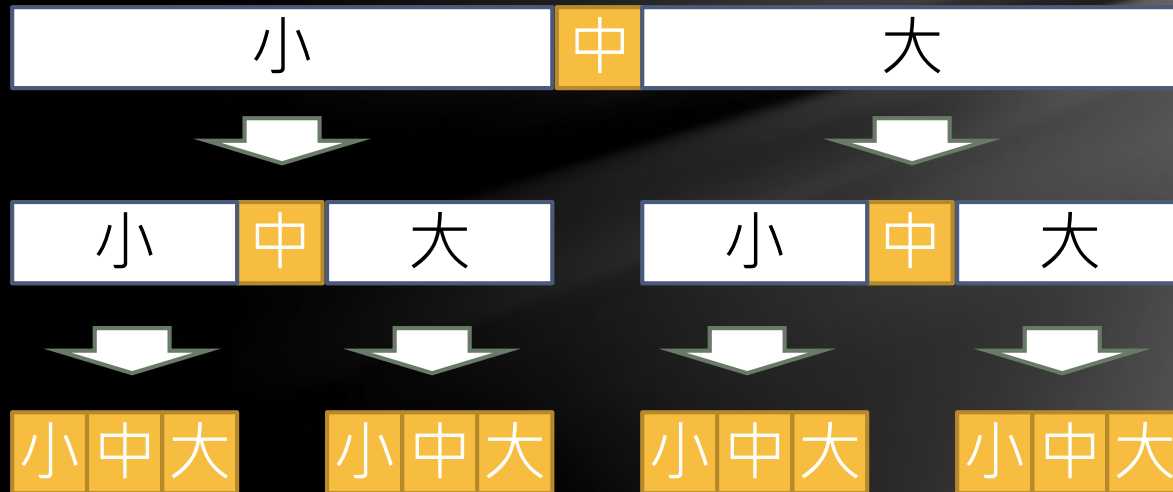
作業

分治法 (Divide and Conquer)

分而治之、各個擊破

- 是一種利用遞迴特性解題的技巧。將一個大問題，切割成許多小問題；將這些小問題解決之後，原本的大問題也就解決了。

實例: 排序



快速排序法(Quick Sort)

快速排序法

- 是一種運用”切割並各個擊破”的方式，將一群資料切割成兩個部分做排序。

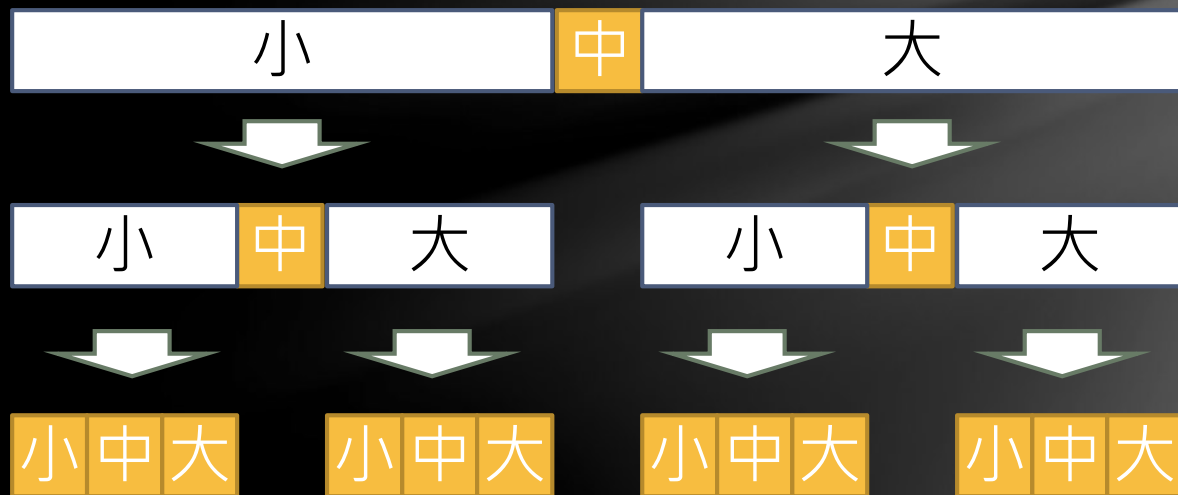
步驟說明（由小排到大）

- (1) 在一群資料中訂出一個基準值(預設為集合中的第一個)
- (2) 比基準值大的集中在右邊、比基準值小的集中在左邊
 - (a) 在這群資料中由左至右，找一個比基準值大的資料
 - (b) 在這群資料中由右至左，找一個比基準值小的資料
 - (c) 交換(a)、(b)中的資料，但如果(a)、(b)的搜尋相遇就停止
- (3) 最後將基準值放在兩者之間，得到兩群子資料A、B。
- (4) 對每群子資料做一樣的操作。

快速排序法(Quick Sort)

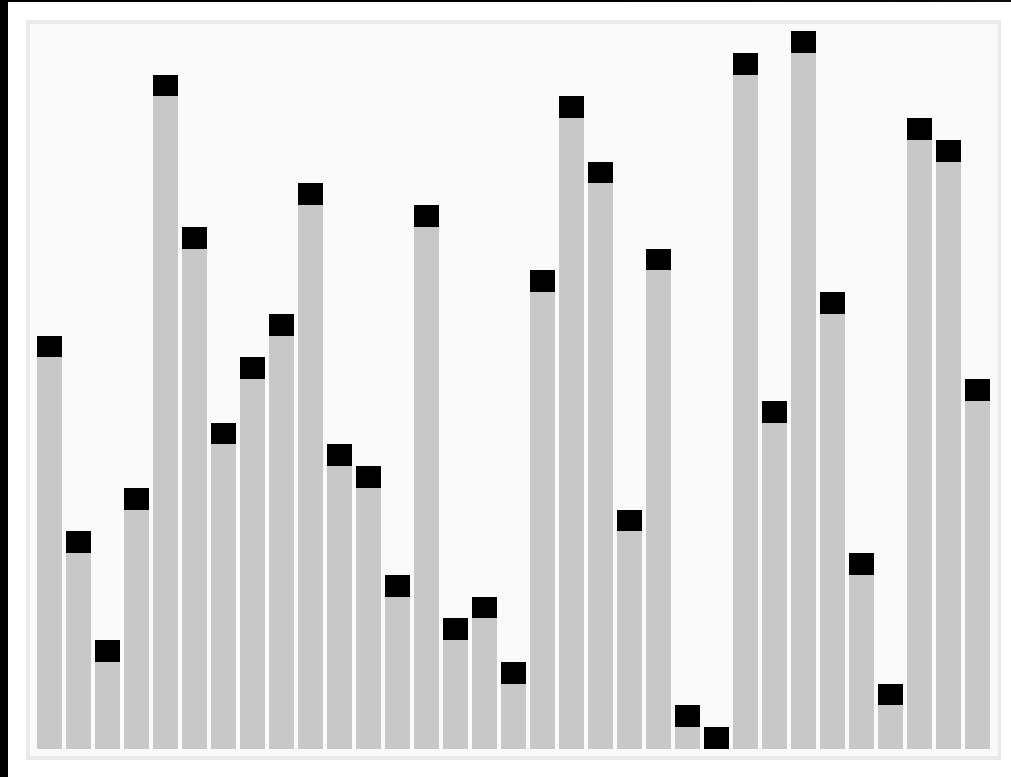
要訣

- 分組再分組 (先分小的那組)
- 直到不能分
- 結束



快速排序法(Quick Sort)

範例圖示



快速排序法圖例(由小排到大)

第一步：
訂出基準值



5 pivot

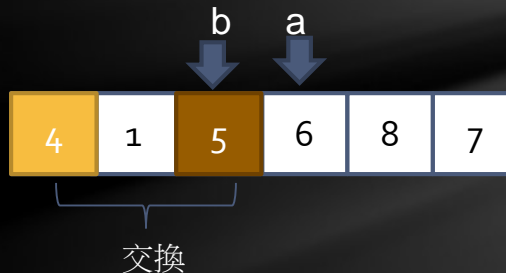
第二步：

- 1) a向右找出比基準值大的資料
 - 2) b向左找出比5小的資料
 - 3) 交換兩者
- *)若a,b交錯則跳到第3步



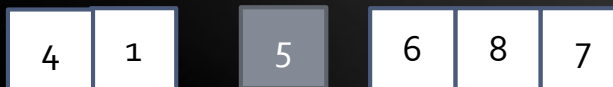
5 pivot

第三步：
將資料分為兩群，基準值放在中間



5 pivot

第四步：
將左右兩區資料個別再做“快速排序”



快速排序法程式碼 (chap04_ex2.c)

```
void QuickSort(int *numbers, int low, int high) // Quick Sort 由小排到大
{
    int a, b, pivot;

    a = low+1; // a從左邊開始找
    b = high;  // b從右邊開始找
    pivot = numbers[low]; // 基準值預設為第一個
    do{
        while(numbers[a] < pivot) // a向右找，找到比基準值大的才停下來
            a++;
        while(numbers[b] > pivot) // b向左找，找到比基準值小的才停下來
            b--;
        if(a < b)
            swap(&numbers[a], &numbers[b]);
    }while(a < b);
    numbers[low] = numbers[b]; // 把基準值擺到中間
    numbers[b] = pivot;        // 兩者交換後，把數列切成兩半

    if((b-1) > low)
        QuickSort(numbers, low, b-1); // 小的那一半繼續做QuickSort
    if((b+1) < high)
        QuickSort(numbers, b+1, high); // 大的那一半繼續做QuickSort
}
```


排序法的效益評估 (2)

排序法	最壞狀況所需時間	平均花費時間	是否屬於穩定排序	是否需交換位置	所需的額外空間	備註
氣泡排序法	$O(n^2)$	$O(n^2)$	YES	YES	$O(1)$	n小較好
選擇排序法						程式易寫
快速排序法	$O(n^2)$	$O(n^2)$	NO	YES	$O(1)$	
快速排序法	$O(n^2)$	$O(n \log_2 n)$	NO	YES	$O(\log n)$	
					$\sim O(n)$	