

# 第七章

## 資料排序

## Sorting

版權屬作者所有，非經作者  
同意不得用於教學以外用途

# 本章內容

7-1 排序及定義

7-2 基本排序法

氣泡排序法、選擇排序法、插入排序法

7-3 進階排序法

合併排序法、快速排序法、基數排序法

7-3 樹狀排序法

堆積排序法、二元樹排序法

當眾多玩家使出看家本領，追分搶分灌分後，遊戲設計者如何按照分數將各家英雄好漢定出高下，以找出冠軍得主？

是的，答案是使用「**排序**」演算法。

排序被廣泛用在各種場合，例如：

- 1.各種考試後的分發，按照成績高低順序媒合志願
- 2.使用 Excel 試算表時，按照選定的欄位將資料由小到大或由大到小排序。



## 7-1 排序及定義

※ 用某些欄位為依據來調整紀錄間的順序，這個動作稱為**排序**（sort）。  
用來排序的欄位稱為**鍵欄**（key field），鍵欄的值稱為**鍵值**（key value）



## ※ 內部排序法 ( internal sort )

待排序的資料全部在主記憶體 ( RAM ) 中，這些排序法稱為內部排序法

## ※ 外部排序法 ( external sort )

待排序的資料只有部分在主記憶體中，其他大部份資料存於外部記憶體

※ 排序法主要是比較它們排序的**效率**。通常我們以排序的資料量 (  $n$  個資料 ) 為參數，有些需時  $O(n^2)$ ，有些可達  $O(n \lg n)$

※ 排序法另一個要素是執行這些排序法所需的額外記憶體空間大小。有些完全不需額外記憶體，有些只需少數記憶體，有些需要  $O(n)$  的額外記憶體，有些甚至更多

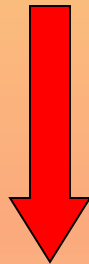
※ 在內部排序法中，影響效率的因素通常是作**比較** ( compare ) 的次數，和**資料交換 ( 或移動 )** 的次數

## ※ 排序法的穩定性 ( stability )

所謂穩定性，是指原先具有相同鍵值的那些記錄，經過排序後仍然保持原先的先後次序（鍵值不同的當然已經改變過）

18, 3, 9, 18', 10, 9', 6, 3'

兩個人  
都18歲



3, 3', 6, 9, 9', 10, 18, 18'

3, 3', 6, 9', 9, 10, 18, 18'

不具有穩定性

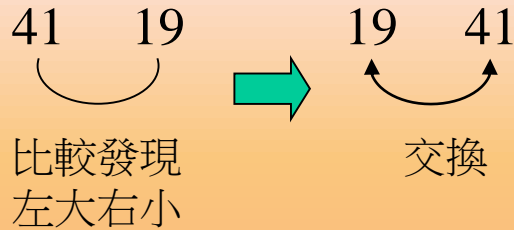
具有穩定性

## 7-2 基本排序法

## 氣泡排序法

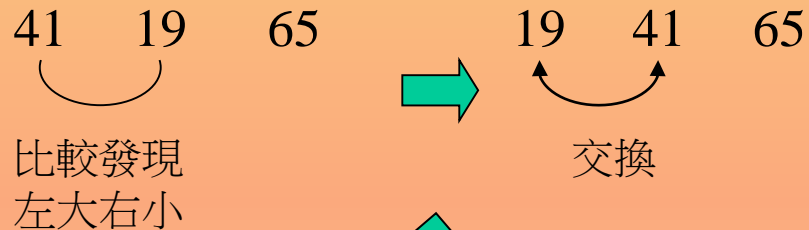
基本動作：相鄰兩個元素比較，左大右小則交換。

➤ 只有2個資料時



➤ 有3個資料時

先比較第1、  
2個資料



再比較第2、  
3個資料



## Pass 1 (第一回合)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	說明
37	41	19	81	41'	25	56	61	49	a[0]比a[1]
37	41	19	81	41'	25	56	61	49	( $37 \leq 41$ 不須交換)，接著a[1]比a[2]
37	19	41	81	41'	25	56	61	49	( $41 > 19$ 須交換)，接著a[2]比a[3]
37	19	41	81	41'	25	56	61	49	( $41 \leq 81$ 不須交換)，接著a[3]比a[4]
37	19	41	41'	81	25	56	61	49	( $81 > 41'$ 須交換)，接著a[4]比a[5]
37	19	41	41'	25	81	56	61	49	( $81 > 25$ 須交換)，接著a[5]比a[6]
37	19	41	41'	25	56	81	61	49	( $81 > 56$ 須交換)，接著a[7]比a[7]
37	19	41	41'	25	56	61	81	49	( $81 > 61$ 須交換)，接著a[7]比a[8]
37	19	41	41'	25	56	61	49	81	( $81 > 49$ 須交換)

Pass 1 : 8 次比較

Pass 1 : 5 次交換

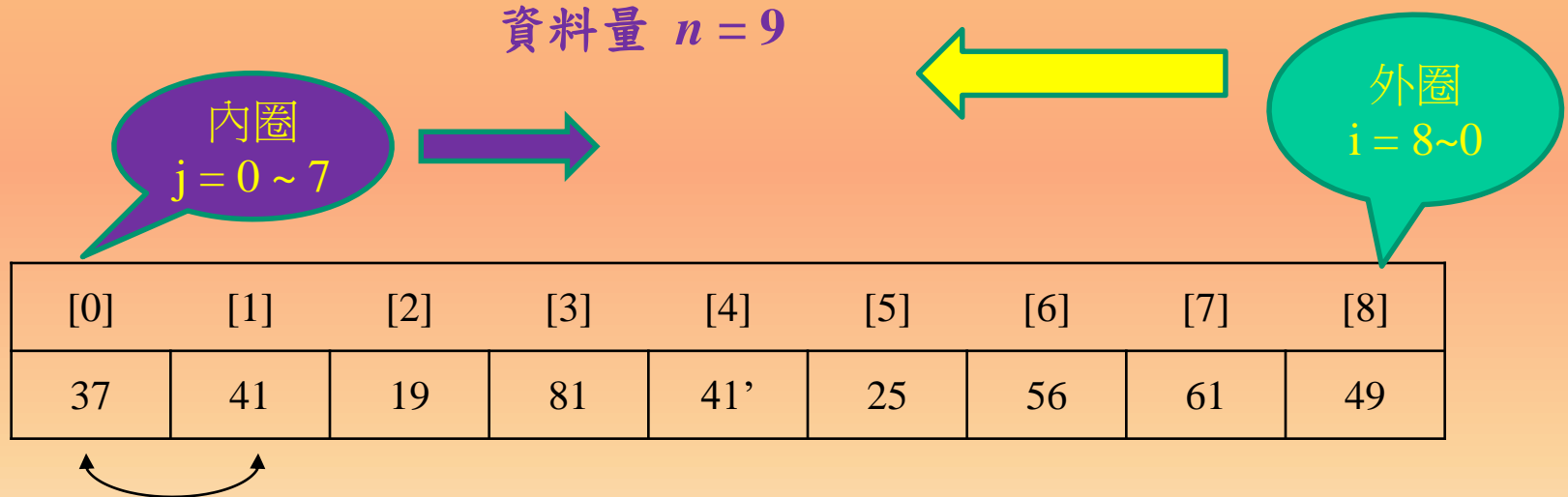


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	說明
原始資料	37	41	19	81	41'	25	56	61	49	Pass 1 : a[0] ~ a[8]
執行Pass 1後	37	19	41	41'	25	56	61	49	81	Pass 2 : a[0] ~ a[7]
執行Pass 2後	19	37	41	25	41'	56	49	61	81	Pass 3 : a[0] ~ a[6]
執行Pass 3後	19	37	25	41	41'	49	56	61	81	Pass 4 : a[0] ~ a[5]
執行Pass 4後	19	25	37	41	41'	49	56	61	81	Pass 5 : a[0] ~ a[4]
執行Pass 5後	19	25	37	41	41'	49	56	61	81	Pass 6 : a[0] ~ a[3]
執行Pass 6後	19	25	37	41	41'	49	56	61	81	Pass 7 : a[0] ~ a[2]
執行Pass 7後	19	25	37	41	41'	49	56	61	81	Pass 8 : a[0] ~ a[1]
執行Pass 8後	19	25	37	41	41'	49	56	61	81	排序完畢

```

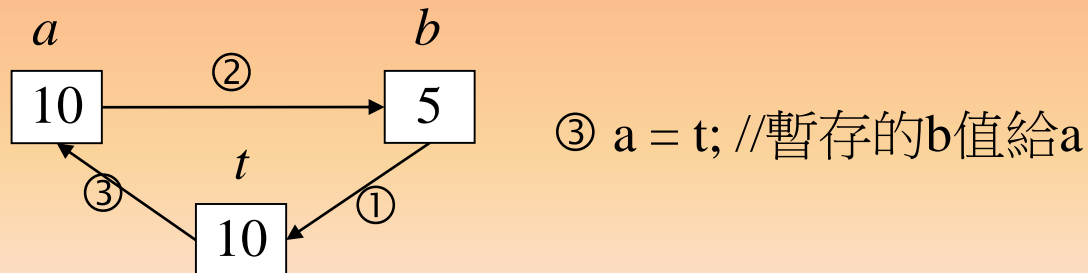
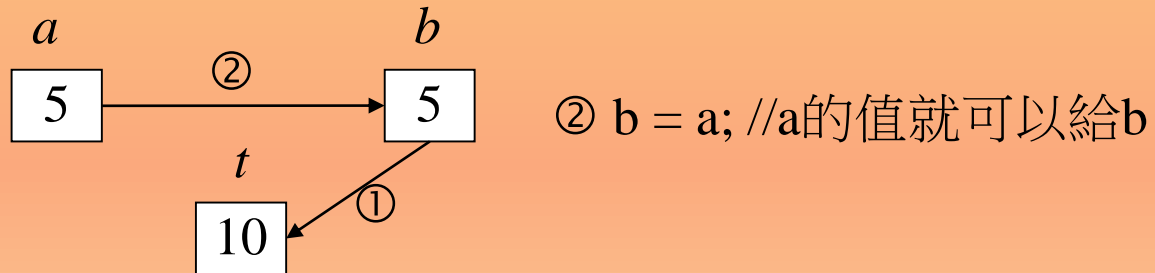
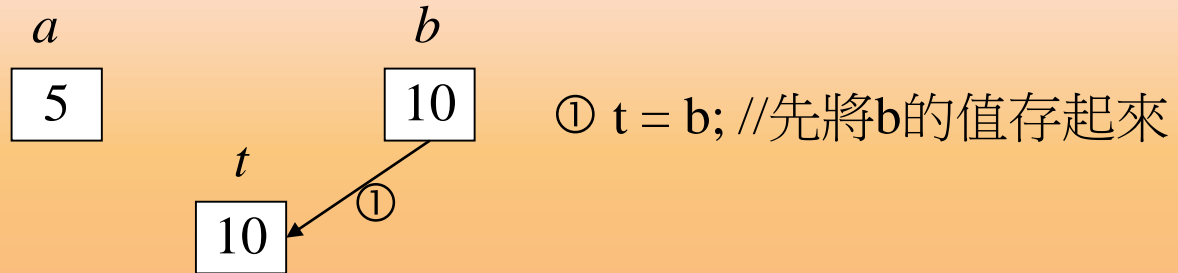
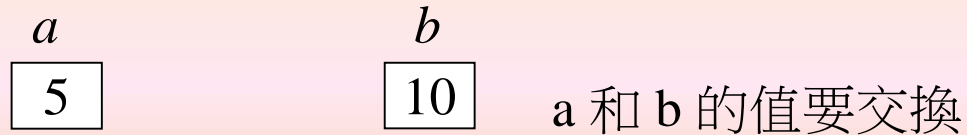
1. void Bubble_Sort(int a[], int n)
2. {   int i, j, temp;
3.     for (i = n-1 ; i > 0 ; i--)           //i是右限
4.         for (j = 0 ; j < i ; j++)
5.             if (a[j] > a[j+1])           //左大右小
6.                 {   temp = a[j];         //將 a[j] 與 a[j+1] 交換
7.                     a[j] = a[j+1];
8.                     a[j+1] = temp;
9.                 }
10. }

```



$a[j]$  與  $a[j+1]$  作比較，若  $a[j] > a[j+1]$  則交換  
當  $j$  由 0 到 7

# 交換三部曲



```

1. void Bubble_Sort(int a[], int n)
2. {   int i, j, temp;
3.     for (i = n-1 ; i > 0 ; i--)           //i是右限
4.         for (j = 0 ; j < i ; j++)
5.             if (a[j] > a[j+1])           //左大右小
6.                 {   temp = a[j];         //將 a[j] 與 a[j+1] 交換
7.                     a[j] = a[j+1];
8.                     a[j+1] = temp;
9.                 }
10. }

```

	比較〈第5行〉		交換〈第6~8行〉	
	最好狀況	最壞狀況	最好狀況	最壞狀況
Pass 1	$n-1$	$n-1$	0	$n-1$
Pass 2	$n-2$	$n-2$	0	$n-2$
Pass 3	$n-3$	$n-3$	0	$n-3$
:	:	:	:	:
Pass $n-1$	1	1	0	1
總 計	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$	0	$\frac{n(n-1)}{2}$

# 選擇排序法

基本動作：**選擇**出範圍內最大的元素，將之與範圍內最右邊的元素**交換**。

逐一掃描，**選擇**出最大的資料 81，最右邊的資料是 49

37      41      19      **81**      41'      25      56      61      **49**

將 81 與最右邊的資料**交換**

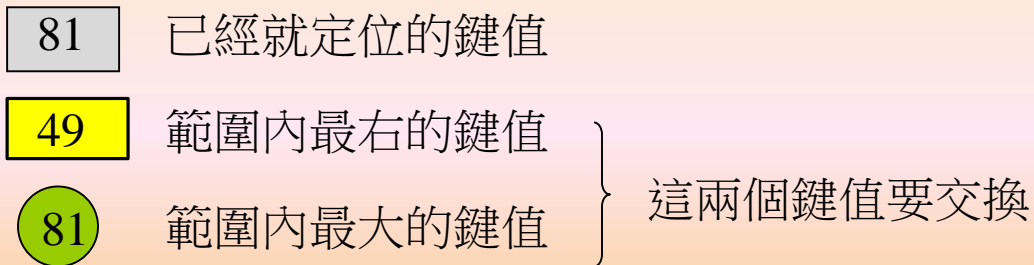
37      41      19      **49**      41'      37      56      61      **81**



**氣泡排序法**是 一步一腳印 - 最大元素慢慢交換到最右邊位置

**選擇排序法**是 一步到位 - 最大元素一次交換到最右邊位置

# 圖示說明



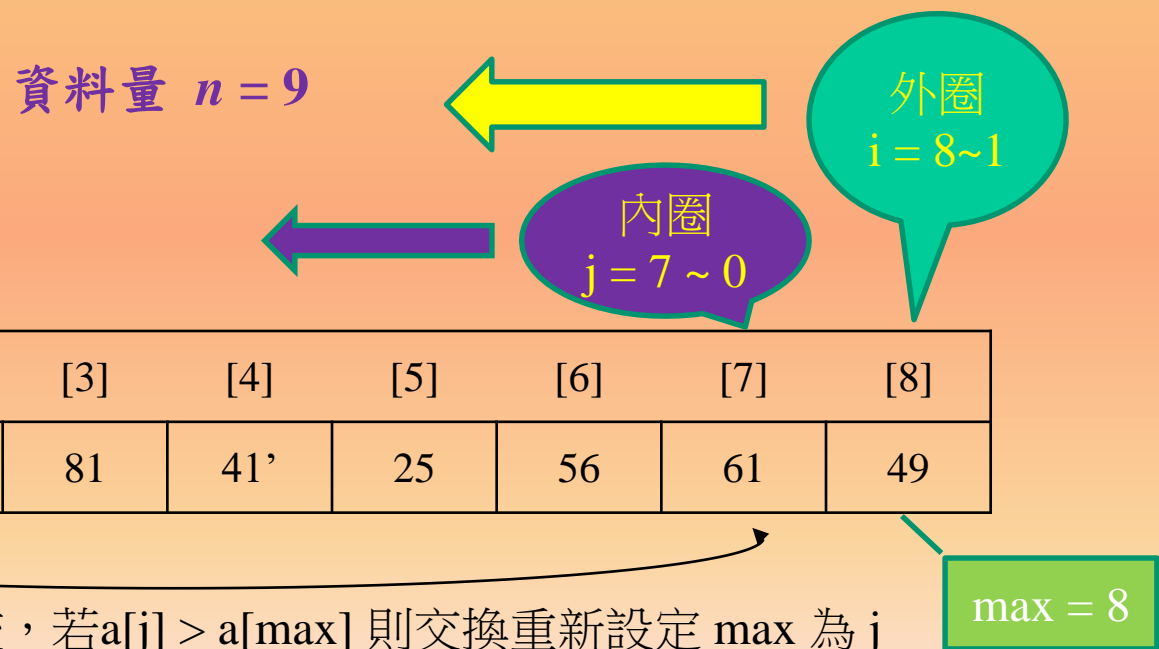
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	說明
原始資料	37	61	19	41	<span style="border: 1px solid black; border-radius: 50%; padding: 2px;">81</span>	25	56	41'	<span style="background-color: yellow; border: 1px solid black; padding: 2px;">49</span>	Pass 1 : a[0]~a[8] , 81↔49
Pass 1後	37	<span style="border: 1px solid black; border-radius: 50%; padding: 2px;">61</span>	19	41	49	25	56	<span style="background-color: yellow; border: 1px solid black; padding: 2px;">41'</span>	<span style="border: 1px solid black; padding: 2px;">81</span>	Pass 2 : a[0]~a[7] , 61↔41'
Pass 2後	37	41'	19	41	49	25	<span style="border: 1px solid black; border-radius: 50%; padding: 2px;">56</span>	61	81	Pass 3 : a[0]~a[6] , 56↔56
Pass 3後	37	41'	19	41	<span style="border: 1px solid black; border-radius: 50%; padding: 2px;">49</span>	<span style="background-color: yellow; border: 1px solid black; padding: 2px;">25</span>	56	61	81	Pass 4 : a[0]~a[5] , 49↔25
Pass 4後	37	41'	19	<span style="border: 1px solid black; border-radius: 50%; padding: 2px;">41</span>	<span style="background-color: yellow; border: 1px solid black; padding: 2px;">25</span>	49	56	61	81	Pass 5 : a[0]~a[4] , 41↔25
Pass 5後	37	<span style="border: 1px solid black; border-radius: 50%; padding: 2px;">41'</span>	19	<span style="background-color: yellow; border: 1px solid black; padding: 2px;">25</span>	41	49	56	61	81	Pass 6 : a[0]~a[3] , 41'↔25
Pass 6後	<span style="border: 1px solid black; border-radius: 50%; padding: 2px;">37</span>	25	<span style="background-color: yellow; border: 1px solid black; padding: 2px;">19</span>	41'	41	49	56	61	81	Pass 7 : a[0]~a[2] , 37↔19
Pass 7後	19	<span style="border: 1px solid black; border-radius: 50%; padding: 2px;">25</span>	37	41'	41	49	56	61	81	Pass 8 : a[0]~a[1] , 25↔25
Pass 8後	19	25	37	41'	41	49	56	61	81	排序完畢 14

```

1. void Select_Sort(int a[], int n)
2. {   int i, j, max, temp;
3.     for (i = n-1 ; i > 0 ; i--)      //i是右限
4.     {   max = i;                      //找出範圍中(a[0] ~ a[i])最大者，先假設a[i]最大
5.         for ( j = i-1 ; j >= 0 ; j--) // j由右而左掃描
6.             if (a[j] > a[max])        //第j個挑戰成功
7.                 max = j;              //max隨時紀錄最大者
8.         temp = a[max];                //將a[i] 與a[max] 交換
9.         a[max] = a[i];
10.        a[i] = temp;
11.    }
12. }

```

資料量  $n = 9$



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
37	41	19	81	41'	25	56	61	49

$a[j]$  與  $a[\max]$  作比較，若  $a[j] > a[\max]$  則交換重新設定  $\max$  為  $j$   
 當  $j$  由 7 到 0

$\max = 8$

# 選擇－掃描一次的自然產物

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	說明
原始資料	37	61	19	41	81	25	56	41'	49	初設最大值 = 49
第1次比較	↑									$41' \leq 49$ , 最大值不變
第2次比較	↑									$56 > 49$ , 最大值 = 56
第3次比較	↑									$25 \leq 56$ , 最大值不變
第4次比較	↑									$81 > 56$ , 最大值 = 81
第5次比較	↑									$41 \leq 81$ , 最大值不變
第6次比較	↑									$19 \leq 81$ , 最大值不變
第7次比較	↑									$61 \leq 81$ , 最大值不變
第8次比較	↑									$37 \leq 81$ , 最大值不變

最大值為 81



```

1. void Select_Sort(int a[], int n)
2. {   int i, j, max, temp;
3.     for (i = n-1 ; i > 0 ; i--)          //i是右限
4.     {   max = i;                          //找出範圍中(a[0] ~ a[i])最大者，先假設a[i]最大
5.         for ( j = i-1 ; j >= 0 ; j--)    // j由右而左掃描
6.             if (a[j] > a[max])          //第j個挑戰成功
7.                 max = j;                //max隨時紀錄最大者
8.         temp = a[max];                  //將a[i] 與a[max] 交換
9.         a[max] = a[i];
10.        a[i] = temp;
11.    }
12. }

```

	比較〈第6行〉		交換及設定〈第4, 7, 8~10行〉	
	最好狀況	最壞狀況	最好狀況	最壞狀況
Pass 1	$n-1$	$n-1$	$1+1$	$n+1$
Pass 2	$n-2$	$n-2$	$1+1$	$n-1+1$
Pass 3	$n-3$	$n-3$	$1+1$	$n-2+1$
:	:	:	:	:
Pass $n-1$	1	1	$1+1$	$2+1$
總 計	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$	$2(n-1)$	$\frac{(n+4)(n-1)}{2}$

# 插入排序法

基本動作：將鍵值插入左邊適當的地方，使範圍內鍵值都排好。

假設要排序第 3 個資料(前2個已經排好)

[0]	[1]	[2]
37	41	19

1. 首先將a[2]的值往上提，放入up變數，則 a[2] 的位置等於空下來一樣。

up 19

[0]	[1]	[2]
37	41	

2. 由於  $41 > 19$ ，因此將41往右移，現在變成a[1] 的位置空下來了。

up 19

[0]	[1]	[2]
37		41

3. 由於  $37 > 19$ ，因此將37往右移，現在變成a[0] 的位置空下來了。

up 19

[0]	[1]	[2]
	37	41

4. 空位已經在最左邊了，因此把 up 中的值 ( 19 ) 放入空位a[0]。

up

[0]	[1]	[2]
19	37	41

因此 a[0] ~ a[2] 已經照順序排好了

以上連續動作相當於將 19 插入到最左邊位置

現在的任務，是要把  $a[8]$  ( 49 ) 插入左邊適當的位置。

1. 首先將  $a[8]$  的值 ( 49 ) 往上提，放入  $up$  變數，則  $a[8]$  產生空位。

up	49
----	----

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
19	25	37	41	41'	56	61	81	

2.  $81 > 49$ ,  $61 > 49$ ,  $56 > 49$ ，因此它們分別往右移一位。41' 不大於 49，因此 41' 不動，空位停在  $a[5]$ 。

up	49
----	----

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
19	25	37	41	41'		56	61	81

3. 將  $up$  的值 ( 49 ) 放入空位  $a[5]$  中。

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
19	25	37	41	41'	49	56	61	81

因此  $a[0] \sim a[8]$  就都照順序排好了。

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	說明
開始	37	<b>41</b>	19	81	41'	25	56	61	49	粗體字為插入資料
Pass 1後	37	41	<b>19</b>	81	41'	25	56	61	49	37不比41大，41不動
Pass 2後	19	37	41	<b>81</b>	41'	25	56	61	49	41,37各往右移，19插入a[0]
Pass 3後	19	37	41	81	<b>41'</b>	25	56	61	49	41不比81大，81不動
Pass 4後	19	37	41	41'	81	<b>25</b>	56	61	49	81>41'往右移，41'插入a[3]
Pass 5後	19	25	37	41	41'	81	<b>56</b>	61	49	比25大的均右移
Pass 6後	19	25	37	41	41'	56	81	<b>61</b>	49	81>56往右移，56插入a[5]
Pass 7後	19	25	37	41	41'	56	61	81	<b>49</b>	81>61往右移，61插入a[6]
Pass 8後	19	25	37	41	41'	49	56	61	81	81,61,56往右移，49插入a[5]

```
1. void Insert_Sort(int a[], int n)
2. {   int i, j, up;
3.     for (i = 1 ; i < n ; i++)    //a[i]是要整理的元素(a[0]~a[i-1]已經整理好)
4.     {   up = a[i];
5.         j = i;                    //j 記錄空格位置
6.         while (j > 0 && a[j-1] > up) //空格左邊的值比up大
7.         {   a[j] = a[j-1];        //空格左邊的值往右移一格
8.             j--;
9.         }
10.        a[j] = up;                //up中的值放入最後的空位
11.    }
12. }
```

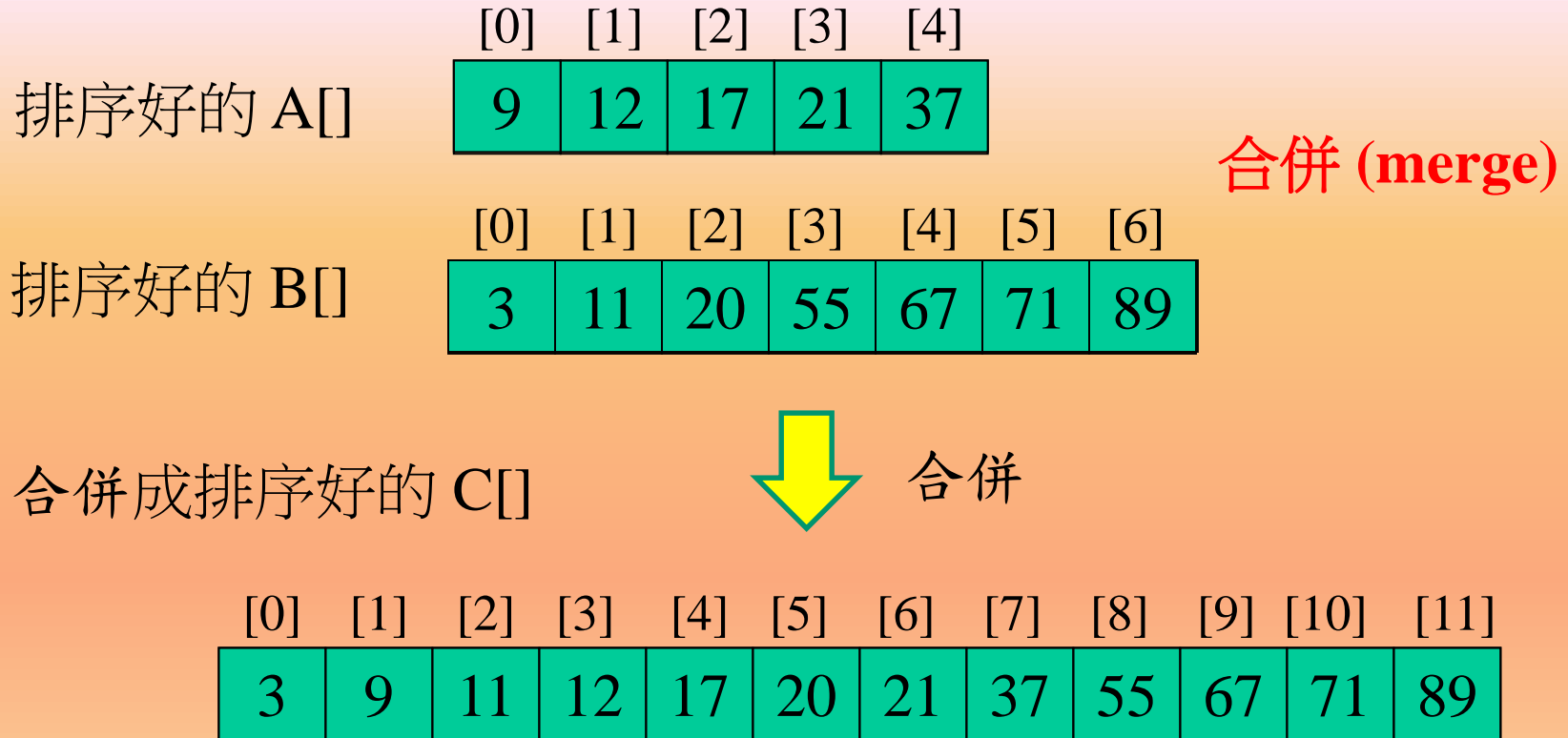
	比較		移動 (包括a[i]指定給 up及up指定給a[j])	
	最好狀況	最壞狀況	最好狀況	最壞狀況
Pass 1	1	1	2	2+1
Pass 2	1	2	2	2+2
Pass 3	1	3	2	2+3
:	:	:	:	:
Pass $n-1$	1	$n-1$	2	$2+(n-1)$
總計	$n-1$	$\frac{n(n-1)}{2}$	$2(n-1)$	$\frac{(n+4)(n-1)}{2}$

## 基本排序法的比較

	最好效率	平均效率	額外記憶體需求	穩定性
氣泡排序法	$O(n^2)$	$O(n^2)$	0	有
選擇排序法	$O(n^2)$	$O(n^2)$	0	無
插入排序法	$O(n)$	$O(n^2)$	0	有

## 7-3 進階排序法

### 合併排序法



合併排序法：兩組各一個資料合併為一組二個、兩組各二個合併為一組四個...直到所有資料合併成一組。

時間點	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	說明
原始資料	37	41	19	81	41'	25	56	61	49	有斜線部分為合併動作
1	37	41	19	81	41'	25	56	61	49	對切成4個, 5個
2	37	41	19	81	41'	25	56	61	49	左半部再對切成2, 2
3	37	41	19	81	41'	25	56	61	49	第一組再對切成1, 1
4	37	41	19	81	41'	25	56	61	49	1, 1 合併成2
5	37	41	19	81	41'	25	56	61	49	第二組再對切成1, 1
6	37	41	19	81	41'	25	56	61	49	1, 1 合併成2
7	19	37	41	81	41'	25	56	61	49	2, 2 合併成4
8	19	37	41	81	41'	25	56	61	49	右半部再對切成2, 3
9	19	37	41	81	41'	25	56	61	49	第一組再對切成1, 1
10	19	37	41	81	25	41'	56	61	49	1, 1 合併成2
11	19	37	41	81	25	41'	56	61	49	第二組再對切成1, 2
12	19	37	41	81	25	41'	56	61	49	最右邊再對切成1, 1
13	19	37	41	81	25	41'	56	49	61	1, 1 合併成2
14	19	37	41	81	25	41'	49	56	61	1, 2 合併成3
15	19	37	41	81	25	41'	49	56	61	2, 3 合併成5
16	19	25	37	41	41'	49	56	61	81	4, 5 合併成9, 排序完成



- 從排序的過程來看，合併排序法是不斷地將鍵值對切，切到不能再切（一個鍵值）再進行合併。合併時則反向兩組兩組進行，直到最後合併成整個檔案。
- 我們從遞迴的角度來看，如果從步驟1,2直接對切後跳至步驟15，接著只是單純地將兩組排好的資料合併而已。但是步驟15的成果，卻是由步驟2~7（左半部）及步驟8~15（右半部）分別進行合併排序累積起來的。
- 先把大問題切成小問題，再分開解決小問題，最後再把解決小問題所獲致的成果累積起來，成為大問題的成果，這種解題方法稱為「個個擊破法」（divide and conquer）。
- 合併排序法就是採用個個擊破法，積小勝為大勝，最後獲得全盤的勝利。

排 $n$ 個資料  
的時間

先遞迴排兩半  
各 $n/2$ 個資料

再合併成 $n$   
個資料

$$T(n) \leq 2 T(n/2) + n, \quad T(1) = 0$$

$$\rightarrow T(n) \leq 2(2T(n/4) + n/2) + n, \quad \because T(n/2) \leq 2 T(n/4) + n/2$$

$$\rightarrow T(n) \leq 2^2 T(n/2^2) + n + n$$

:

$$\rightarrow T(n) \leq 2^k T(n/2^k) + kn$$

當  $n \div 2^k$  ( $\lg n \div k$ ) 時

$$\rightarrow T(n) \leq n T(n/2^k) + kn$$

$$\rightarrow T(n) \leq nT(1) + kn$$

$$= kn$$

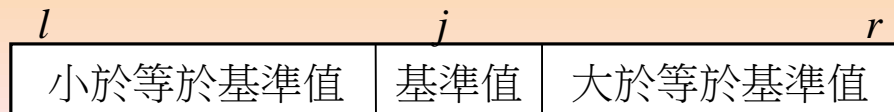
$$= n \lg n$$

因此  $T(n) = O(n \lg n)$

# 快速排序法

## 分割 (partition)


	$l$	$r$	說明
1	37 ....41... 19... .81... .41'.. ..25... .56 ....61 ....49		$i = l + 1$ 往右直到大於等於基準值。 $j = r$ 往左直到小於等於基準值
2	37 ....41... 19... .81... .41'.. ..25... .56 ....61 ....49		暫停未交錯
3	37 ....25... 19... .81... .41'.. ..41... .56 ....61 ....49		交換
4	37 ....25... 19... .81... .41'.. ..41... .56 ....61 ....49		暫停已交錯
5	19 ....25... 37... .81... .41'.. ..41... .56 ....61 ....49		基準值 $a[l]$ 和 $a[j]$ 交換



## 分割的另一個例子

	$l$									$r$	說 明
1	40	41 ↑ $i$	19	81	41'	25	56	21	61	49 ↑ $j$	$i = l + 1$ $j = r$ ; ( $i, j, r, l$ 都是位置)
2	40	41 ↑ $i$	19	81	41'	25	56	21 ↑ $j$	61	49	$i$ 一路往右, 直到 $41 \geq 40$ $j$ 一路往左, 直到 $21 \leq 40$
3	40	21 ↑ $i$	19	81	41'	25	56	41 ↑ $j$	61	49	$i < j$ , 將 $a[i]$ 和 $a[j]$ 交換
4	40	21	19	81 ↑ $i$	41'	25 ↑ $j$	56	41	61	49	$i$ 一路往右, 直到 $81 \geq 40$ $j$ 一路往左, 直到 $25 \leq 40$
5	40	21	19	25 ↑ $i$	41'	81 ↑ $j$	56	41	61	49	$i < j$ , 將 $a[i]$ 和 $a[j]$ 交換
6	40	21	19	25 ↑ $j$	41' ↑ $i$	81	56	41	61	49	$i$ 一路往右, 直到 $41' \geq 40$ $j$ 一路往左, 直到 $25 \leq 40$
7	25	21	19	40 ↑ $j$ (分割點)	41'	81	56	41	61	49	$i > j$ , 已經錯身而過, 將 $a[l]$ (基準值) 和 $a[j]$ 交換

# 圖示說明

 已經就定位的鍵值

	鍵值分割及排序狀況									說明
1	37	41	19	81	41'	25	56	61	49	原始資料
2	[19	25]	37	[81	41'	41	56	61	49]	第一次分割後
3	[]19	[25]	37	[81	41'	41	56	61	49]	空的中括號[]代表 無元素
4	19	25	37	[49	41'	41	56	61]	81[]	
5	19	25	37	[41	41']	49	[56	61]	81	陰影部分為已就定位
6	19	25	37	[]41	[41']	49	[56	61]	81	
7	19	25	37	41	41'	49	[]56	[61]	81	
8	19	25	37	41	41'	49	56	61	81	

# 基數排序法

## LSD ( Least Significant Digit First )

初始順序	139	219	532	655	422	164	098	422'	334
按個位數	53 <b>2</b>	42 <b>2</b>	42 <b>2</b> '	16 <b>4</b>	33 <b>4</b>	65 <b>5</b>	09 <b>8</b>	13 <b>9</b>	21 <b>9</b>
按十位數	2 <b>1</b> 9	4 <b>2</b> 2	4 <b>2</b> 2'	5 <b>3</b> 2	3 <b>3</b> 4	1 <b>3</b> 9	6 <b>5</b> 5	1 <b>6</b> 4	0 <b>9</b> 8
按百位數	<b>0</b> 98	<b>1</b> 39	<b>1</b> 64	<b>2</b> 19	<b>3</b> 34	<b>4</b> 22	<b>4</b> 22'	<b>5</b> 32	<b>6</b> 55

### 每一位數的排序--使用分配法

1. 計算此位數各種值出現的次數
2. 進行累積
3. 將鍵值分配至暫時陣列 b[]
4. 由暫時陣列 b[] 寫回陣列 a[]

如果個位、十位、百位數都使用前面介紹的任一方法，豈不是多此一舉，因為沒有用到將鍵值拆開的好處。因此我們介紹分配法，每位數字只需 $O(n)$ 就可處理

鍵值的每一位數只有 0 ~ 9，共十種可能。例如12個鍵值的個位數：

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
a[]	9	8	2	3	2'	4	4'	7	8'	5	7'	8''

1. 計算此位數各種值出現的次數：使用輔助陣列 `count[10]` ( 10 是因為鍵值只有10 種可能 )。 `count[0]` 代表鍵值 0 出現的次數， `count[1]` 代表鍵值 1 出現的次數， ...。因此陣列 `count[]` 將變成：

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
count[]	0	0	2	1	2	1	0	2	3	1

8出現3次

2. 執行累積： `count[2] = 2` 代表有2個元素小於等於鍵值 2。 `count[4] = 5` 代表有 5 個元素小於等於鍵值 4。因此陣列 `count[]` 將變成：

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
count[]	0	0	2	3	5	6	6	8	11	12

3. 將鍵值分配至暫時陣列 b[] : 先分配 a[11] , 再分配 a[10] , ...

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
a[]	9	8	2	3	2'	4	4'	7	8'	5	7'	8''

➡ 分配 a[11] (8) , count[8] = 11 , 代表有11個元素小於等於鍵值8 , 因此將它分配在陣列b[] 的第11個位置 ( b[10] ) 。並且count[8] 減1成為10

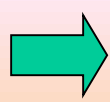
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
b[]											8''	
count[]	0	0	2	3	5	6	6	8	10	12		

➡ 分配 a[10] (7) , count[7] = 8 , 代表有8個元素小於等於鍵值7 , 因此將它分配在陣列b[] 的第8個位置 ( b[7] ) 。並且count[7] 減1成為7

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
b[]								7'			8''	
count[]	0	0	2	3	5	6	6	7	10	12		

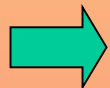


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
a[]	9	8	2	3	2'	4	4'	7	8'	5	7'	8''



分配 a[9] (5)，count[5] = 6，代表有 6 個元素小於等於鍵值 5，因此將它分配在陣列 b[] 的第 6 個位置 (b[5])。並且 count[5] 減 1 成為 5

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
b[]						5		7'			8''	
count[]	0	0	2	3	5	5	6	7	10	12		



以同樣的方法分配 a[8] ~ a[0]

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
b[]	2	2'	3	4	4'	5	7	7'	8	8'	8''	9

4. 由暫時陣列 b[] 寫回陣列 a[]

# 例7.7

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
a[]	9	2	3	5	3'	5'	6	2'	3''	5''	7	0

1. 計數：count陣列先儲存每種鍵值出現的次數。

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
count[]	1	0	2	3	0	3	1	1	0	1

count[5] = 3, 代表鍵值5重複3次，其他以此類推。

2. 再由前往後累積。

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
count[]	1	1	3	6	6	9	10	11	11	12

count[3] = 6, 代表有6個鍵值小於等於3，其他以此類推

3. 再將陣列a的鍵值由後往前分配至陣列b。

A. 分配 a[11] (= 0)

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
count[]	0	1	3	6	6	9	10	11	11	12		
b[]	0											

B. 分配 a[10] (= 7)

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
count[]	0	1	3	6	6	9	10	10	11	12		
b[]	0										7	

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
a[]	9	2	3	5	3'	5'	6	2'	3''	5''	7	0

C. 分配 a[9] (= 5'')

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
count[]	0	1	3	6	6	8	10	10	11	12		
b[]	0								5''		7	

同様方式分配 a[8] ~ a[0] :

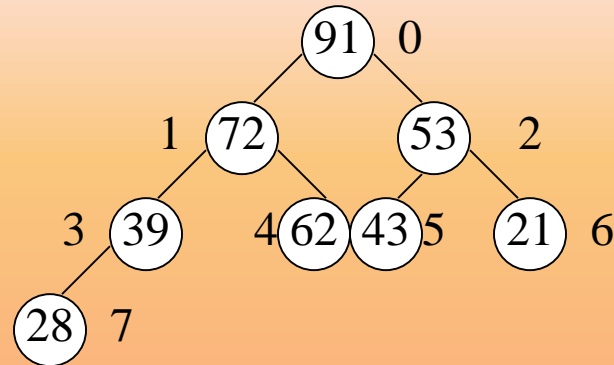
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
count[]	0	1	1	3	6	6	9	10	11	11		
b[]	0	2	2'	3	3'	3''	5	5'	5''	6	7	9

4. 由陣列b[] 回寫至陣列a[] 即完成。

## 7-4 樹狀排序法

### 堆積排序法

堆積 ( heap ) 的**定義**：堆積是一種**完整二元樹**，在堆積中任一個內部節點的鍵值，都**大於等於其子節點**的鍵值



堆積可以按照完整二元樹的方式儲存：

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
a[]	91	72	53	39	62	45	21	28

反過來說，如果有一個任意的陣列a[]，我們也可以將它看成完整二元樹，但是這個完整二元樹不一定是堆積

## 將任意的完整二元樹調整成堆積的方法： 往下調整法

演算法：往下調整法將陣列 $a$ 調整成堆積

從最後一個有子節點的節點  $a[k]$  開始，直到樹根 $a[0]$ ，重複迴圈

當 $a[k]$  有兒子且小於兒子中最大的（稱為大兒子），重複迴圈

$a[k]$  與它的大兒子交換 //  $a[k]$ 相當於降了一代

迴圈結束

迴圈結束

演算法結束

# 往下調整法

演算法：往下調整法將陣列a調整成堆積

從最後一個有子節點的節點  $a[k]$  開始，直到樹根 $a[0]$ ，重複迴圈

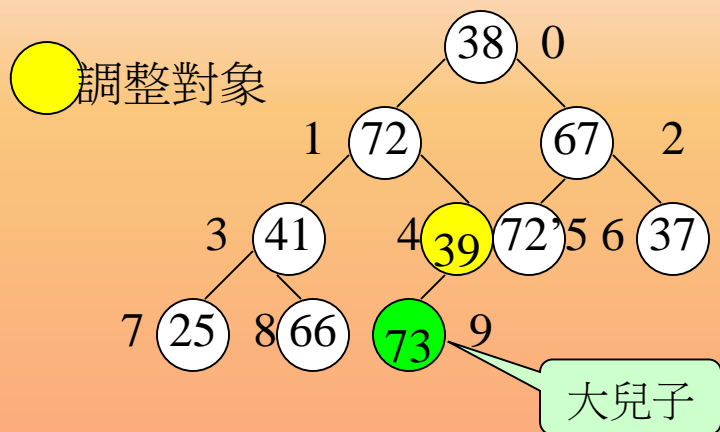
當 $a[k]$  有兒子且小於兒子中最大的（稱為大兒子），重複迴圈

$a[k]$  與它的大兒子交換 //  $a[k]$ 相當於降了一代

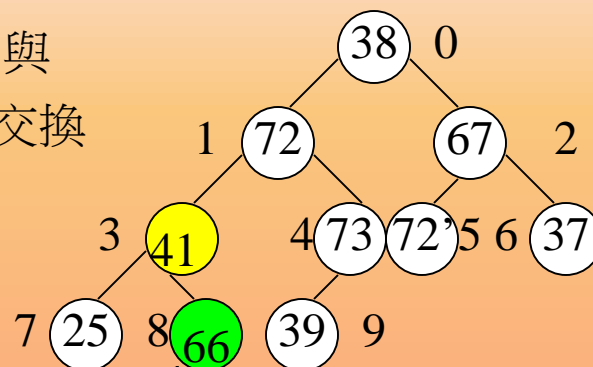
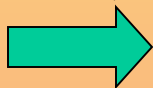
迴圈結束

迴圈結束

演算法結束



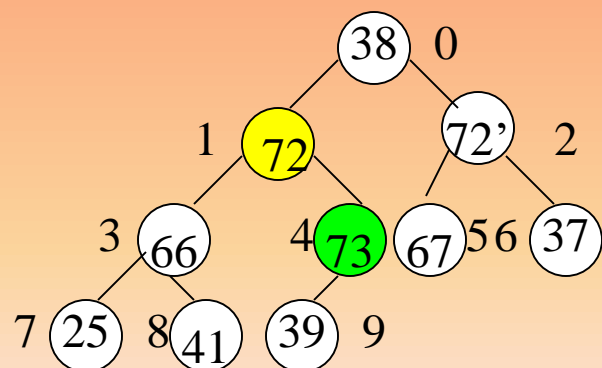
$a[4]$  (39) 與  
 $a[9]$  (73)交換



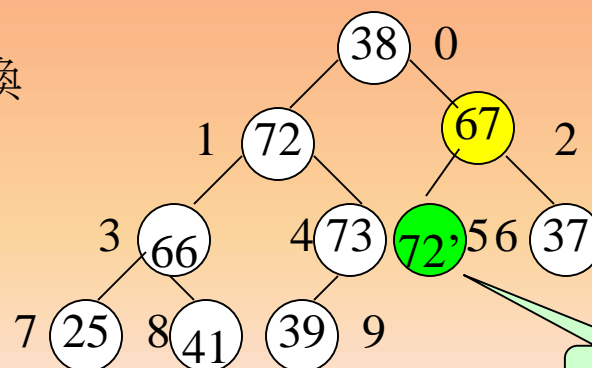
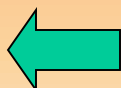
大兒子



$a[3]$  (41) 與  
 $a[8]$  (66)交換

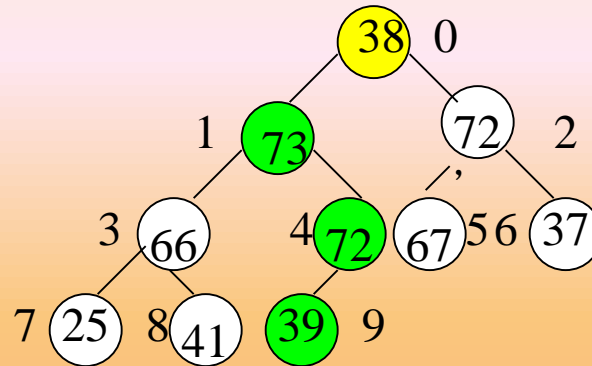


$a[2]$  (67) 與  
 $a[5]$  (72')交換

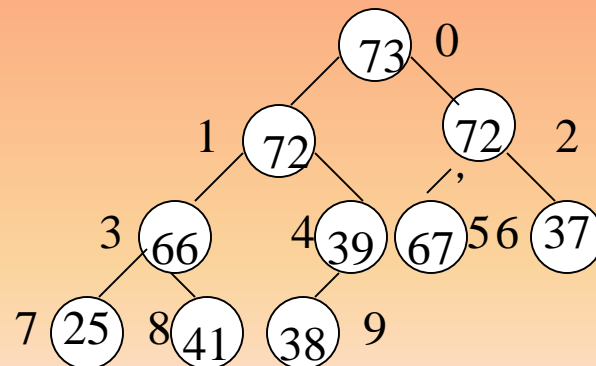


大兒子

a[1] (72) 與  
a[4] (73) 交換



a[0] (38) < a[1] (73) 交換，接著  
a[1] (38) < a[4] (72) 交換，接著  
a[4] (38) < a[9] (39) 交換



# 堆積排序法

演算法：堆積排序法

將陣列調整成堆積

當堆積節點數目大於1時，重複迴圈

將堆積的樹根與堆積的最後一個元素交換

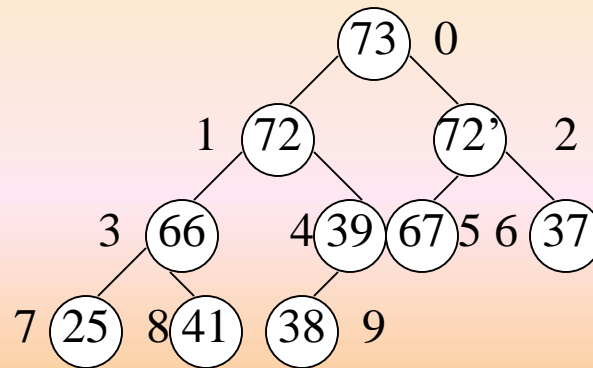
堆積少最後一個節點

重新調整成堆積

迴圈結束

演算法結束



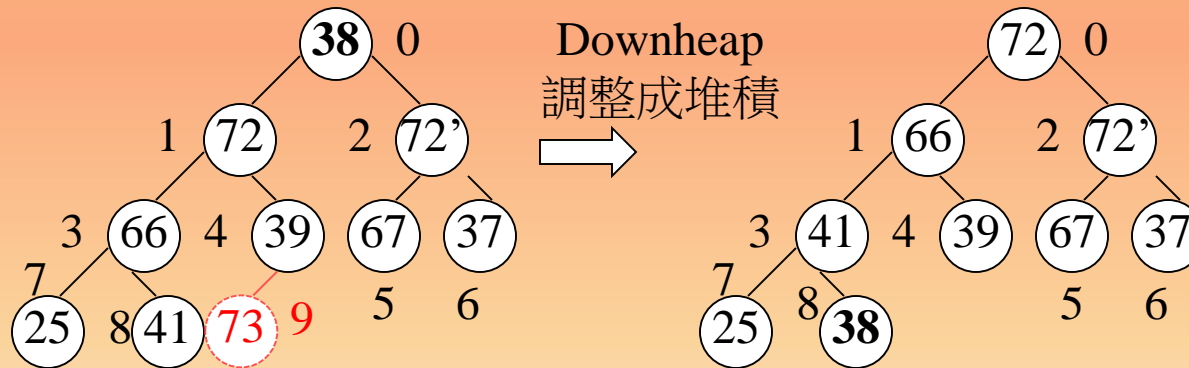


假設這已經是一個堆積 (使用向上或向下調整法其中一種)

實際對應的陣列

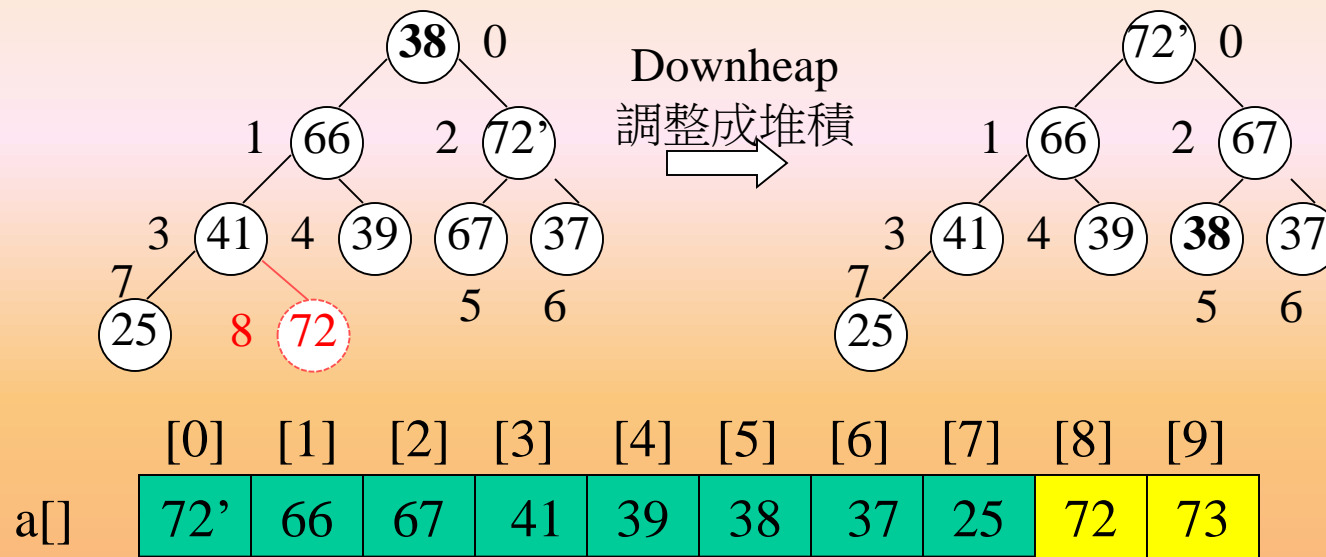
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
a[]	73	72	72'	66	39	67	37	25	41	38

1. 將a[0] ( 73 ) 和 a[9] ( 38 , 堆積的最後一個元素 ) 交換，並且73就不算入堆積的一部份，接著只看 a[0] ~ a[8] ，將a[0] 向下調整成堆積：

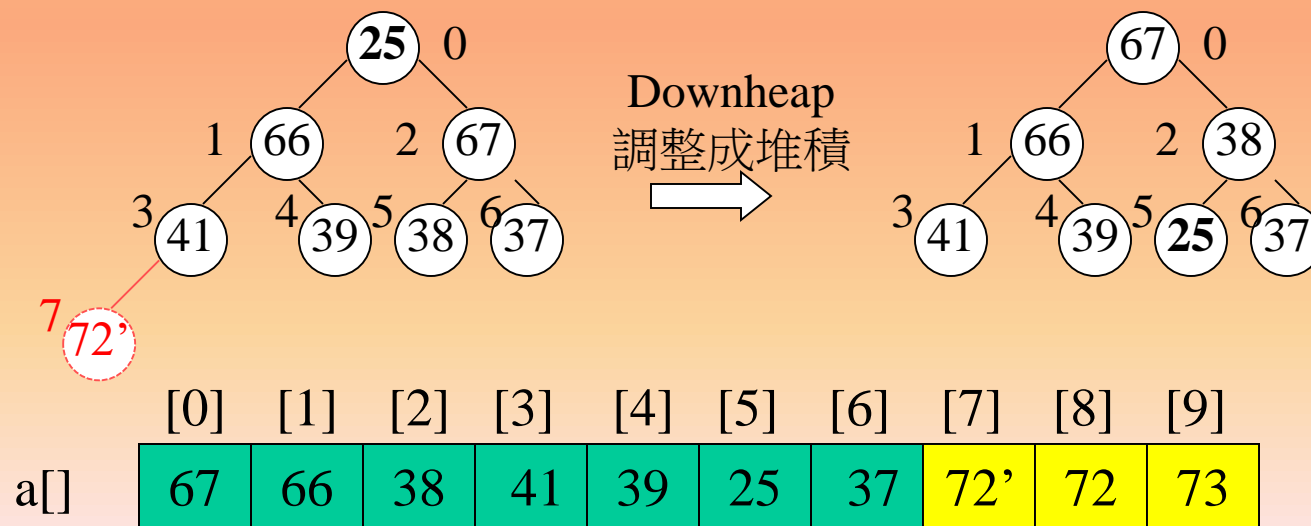


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
a[]	72	66	72'	41	39	67	37	25	38	73

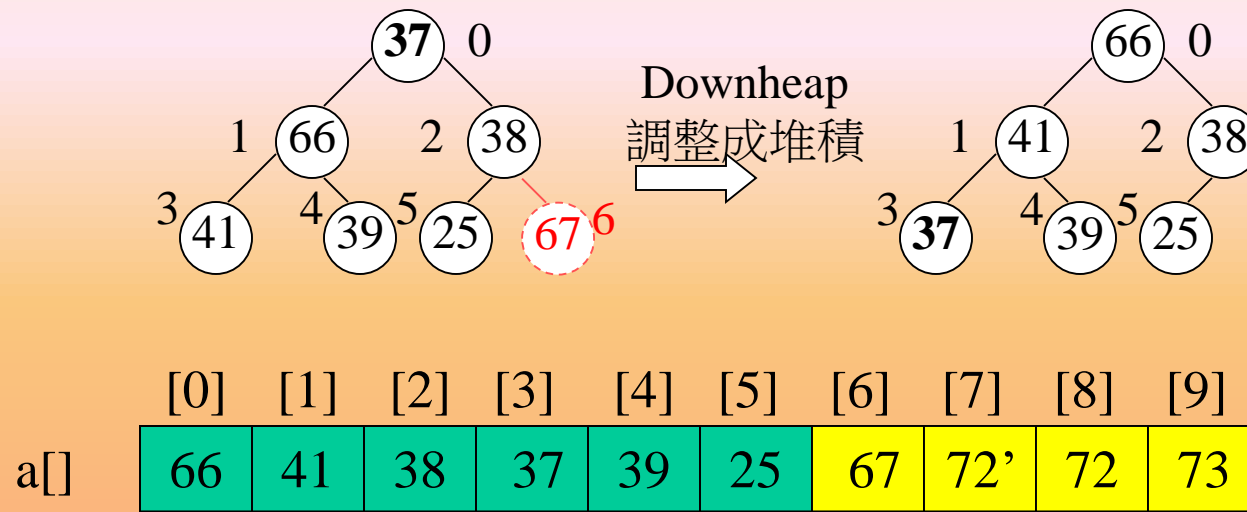
2. 將a[0] ( 72 ) 和 a[8] ( 38 ，堆積的最後一個元素 ) 交換，並重新調整：



3. 將a[0] ( 72' ) 和 a[7] ( 25 ，堆積的最後一個元素 ) 交換，並重新調整：



4. 將a[0] ( 67 ) 和 a[6] ( 37 , 堆積的最後一個元素 ) 交換，並重新調整：



5. 重複 “ 交換 -- 調整 ” 過程，即可將陣列排序好

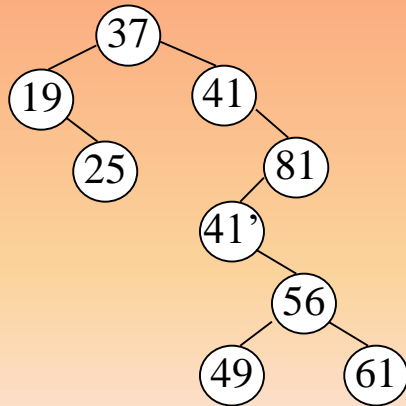
# 二元樹排序法

二元樹排序法是使用「二元搜尋樹」( binary search tree ) 來完成排序，排序的步驟是：

1. 根據鍵值建立二元搜尋樹。
2. 對這棵二元搜尋樹進行中序走訪，並輸出結果。

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
a[]	37	41	19	81	41'	25	56	61	49

1. 建立二元搜尋樹



2. 中序走訪此二元搜尋樹

19, 25, 37, 41, 41', 49, 56, 61, 81

排序法	平均時間複雜度	穩定(Y/N)	額外記憶體空間
<b>Bubble sort</b>	$O(n^2)$	Y	N
<b>Insertion sort</b>	$O(n^2)$	Y	N
<b>Selection sort</b>	$O(n^2)$	N	N
<b>Merge sort</b>	$O(n \log n)$	Y	Y
<b>Quick sort</b>	$O(n \log n)$	N	N
<b>Radix sort</b>	$O(kn)$	Y	Y
<b>Heap sort</b>	$O(n \log n)$	N	N