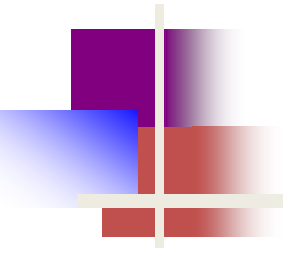


第八章
排序(Sorting)

本章學習目標

1. 讓學生了解**排序的意義與分類**。
2. 讓學生了解**各種排序的方法與運作原理**。

本章內容

8-1 排序(Sorting)

【定義】

指將一組資料依使用者的需要予以重新安排其順序。

而資料在經過排序之後，其優點為容易閱讀、利於統計分析及可以快速搜尋 所要的資料等等。

在「資料結構」課程中，排序法可分為兩大類：

第一類：內部與外部排序法

第二類：穩定與不穩定排序

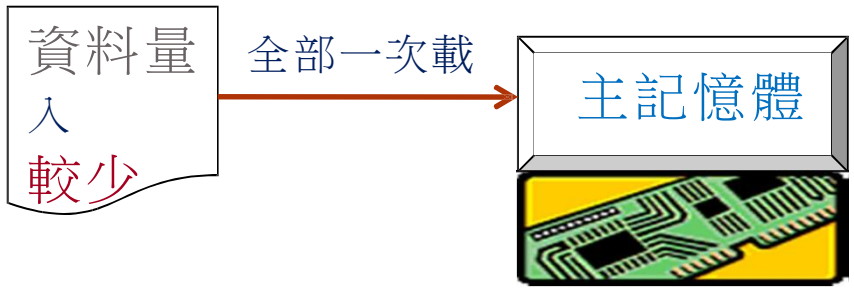
第一類：內部與外部排序法

1.內部排序法(Internal Sort)：又稱為「陣列排序」

【定義】是指要排序的資料全部都是在主記憶體(RAM)內完成。

【適用時機】資料量較少者。

【圖解】



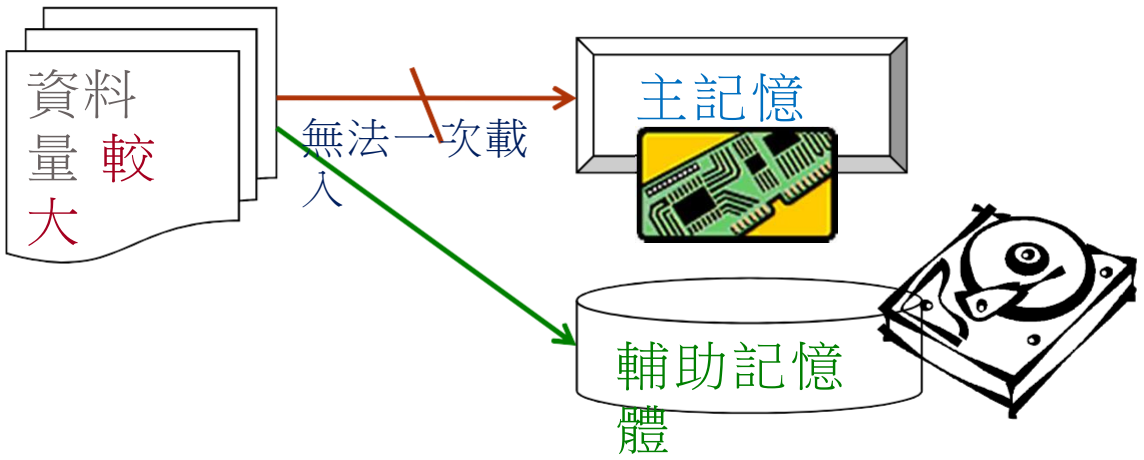
2.外部排序法(External Sort)：又稱為「檔案排序」

【定義】

排序的工作是在輔助記憶體內完成。由於檔案太大，使得要排序的資料無法一次全部載入到主記憶體中，而排序進行時，須藉助 輔助記憶體存取才能完成。

【適用時機】資料量較大者。

【圖解】



第二類：穩定與不穩定排序

1.穩定排序(Stable Sorting)

【定義】如果鍵值相同之資料在排序後的相對位置和排序前相同時， 則稱為穩定排序。

【例如】

(1)排序前：3,5,19,1,3+,10(兩個相同鍵值3，故第二個鍵值3寫成3+)

(2)排序後：1,3,3+,5,10,19(兩個3的相對位置在排序前後是相同的)

第二類：穩定與不穩定排序

2.不穩定排序(UnStable Sorting)

【定義】如果鍵值相同之資料在排序後的相對位置和排序前是不相同時，則稱為不穩定排序。

【例如】

(1)排序前：3,5,19,1,3+,10(兩個相同鍵值3，故第二個鍵值3寫成3+)

(2)排序後：1,3+,3,5,10,19(兩個3的相對位置在排序前後是不相同)

表8-1 各種排序的比較

排序方式	最壞時間	平均時間	穩定度	額外空間	備註說明
氣泡排序 (Bubble Sort)	$O(n^2)$	$O(n^2)$	穩定	$O(1)$	n小時較好
選擇排序 (Selection Sort)	$O(n^2)$	$O(n^2)$	不穩定	$O(1)$	n小時較好
插入排序 (Insertion Sort)	$O(n^2)$	$O(n^2)$	穩定	$O(1)$	大部份已排序者較好
薛爾排序 (Shell Sort)	$O(n^s)$ $1<s<2$	$O(n \log_2 n)$	穩定	$O(1)$	s是分組
快速排序 (Quick Sort)	$O(n^2)$	$O(n \log_2 n)$	不穩定	$O(\log n)$ ~ $O(1)$	n大時較好
堆積排序 (Heap Sort)	$O(n \log_2 n)$	$O(n \log_2 n)$	不穩定	$O(1)$	n大時較好
合併排序 (Merge Sort)	$O(n \log_2 n)$	$O(n \log_2 n)$	穩定	$O(N)$	常用於外部排序
基數排序 (Radix Sort)	$O(n \log_r B)$	$O(n \log_b k) \sim O(n)$	穩定	$O(n * b)$	k:箱子個數 b:基數

8-2 氣泡排序法(Bubble Sort)

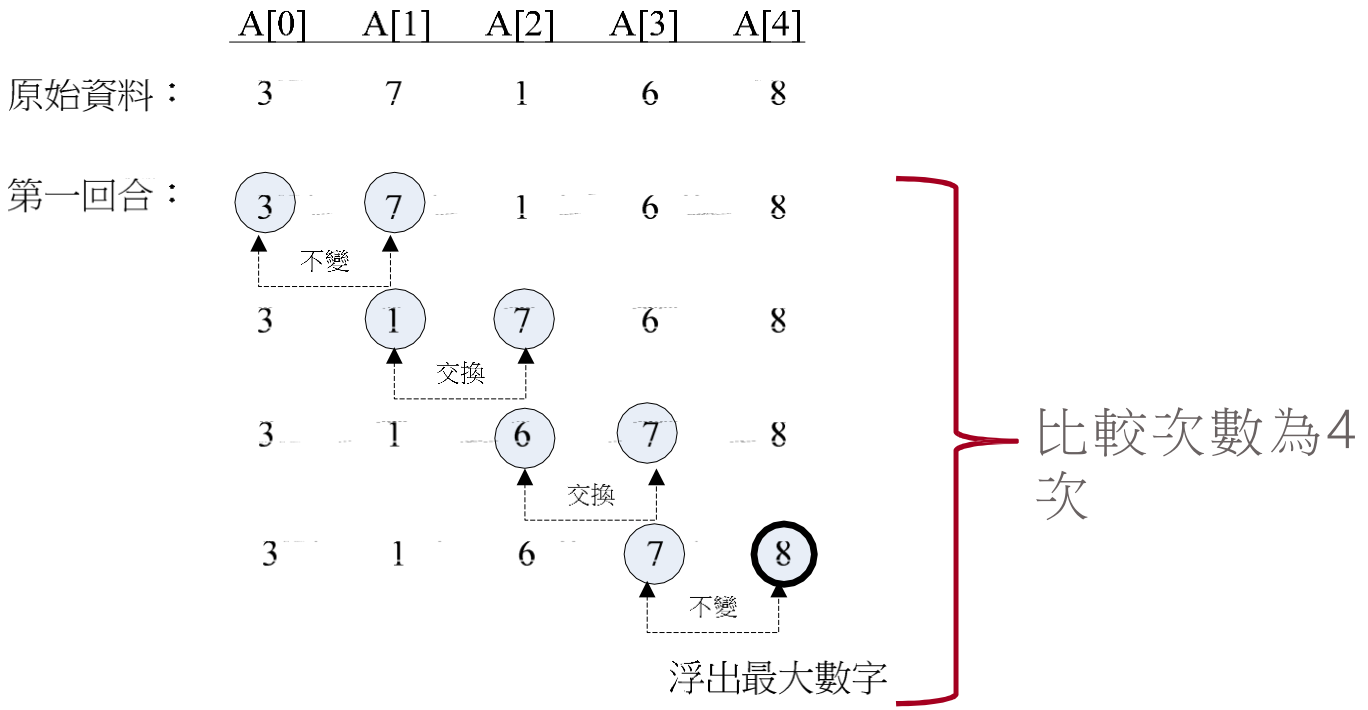
【定義】

是指將兩個相鄰的資料相互做比較，若比較時發現次序不對，則將兩個資料互換，並且資料依序由上往下比，而結果則會依序由下往上浮起，猶如氣泡一般。

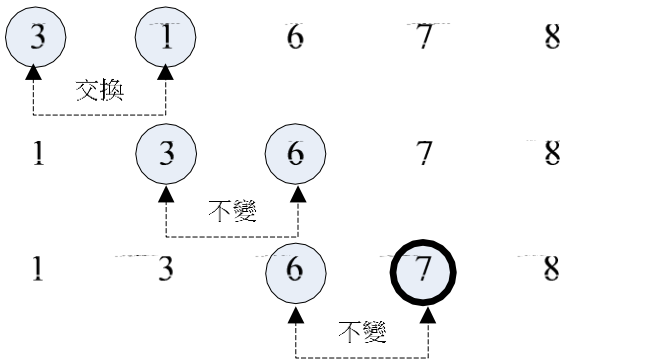
【原理】

由小到大
or
由大到小

逐次比較兩個相鄰的資料，按照排序的條件交換位置，直到全部資料依序排好為止。其排序過程。如下圖所示：



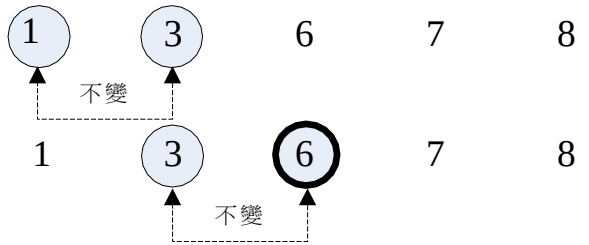
第二回合：



比較次數為3次

浮出第二大數字

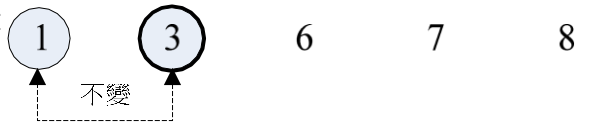
第三回合：



比較次數為2次

浮出第三大數字

第四回合：



比較次數為1次

浮出第四大數字
即完成排序動作

氣泡排序法的演算法如下：

```
01  Procedure BubSort(int A[], intn)
02   begin
03   for (i=n-1; i>=1;i--)                  //排序n-1個回合
04     {
05      for ( j =0; j <=i-1; j++)           //從第0個元素開始掃瞄
06       if (A[ j] > A[ j+1])               //判斷左邊元素是否大於右邊元素
07         {                                // A[ j] 與 A[ j+1]交換
08           Temp = A[ j];
09           A[ j] = A[ j+1];
10           A[ j+1] = Temp;
11         }
12     }
13   end
14  End Procedure
```

【實例】

假設原始資料為：3,7,1,6,3+，在進行排序時，每一回合必定會有 一個元素排到定位，稱為一個回合(Pass)。

	A[0]	A[1]	A[2]	A[3]	A[4]	比較次數	比較範圍
原始資料	3	7	1	6	3+		
Pass 1	3	1	6	3+	**7**	4	(A[0]與A[1]、 A[1]與A[2]、 A[2]與A[3]、 A[3]與A[4])
Pass 2	1	3	3+	**6**	7	3	(A[0]與A[1]、 A[1]與A[2]、 A[2]與A[3])
Pass 3	1	3	**3+**	6	7	2	(A[0]與A[1]、 A[1]與A[2])
Pass 4	1	**3**	**3+**	6	7	1	(A[0]與A[1])
					總比較次數	10	

【分析】

1. 比較之**回合次數**= **資料個數**-1

   (例如：資料**個數**n=5，則**回合次數為**4)

2. 在**每一回合之後**，至少會**有一個資料**可以排列到**正確位置**，**再進行下一個回合**的排列時，便可以**減少此資料的比較**。

   (例如：資料**個數**n=5，則Pass 1**時，比較次數為**4，Pass 2**時，比較次數為**3，以此類推，如上表所示)

3. 需要一個**額外空間**。

   例如：在上面的演算法中的行號08，需要一個Temp變數空間。

4. 為一種**穩定排序**（Stable Sorting）。

   因為氣泡排序法**交換條件**為「**左大右小**」時才必須交換。如下所示：

(1) **排序前**：(兩個相同鍵值3，故第二個鍵值3寫成3^+)

3	7	1	6	3^+

(2) **排序後**：

1	**3**	3^+	6	7

(因為**兩個**3**的相對位置**在**排序前後**是**相同**的)

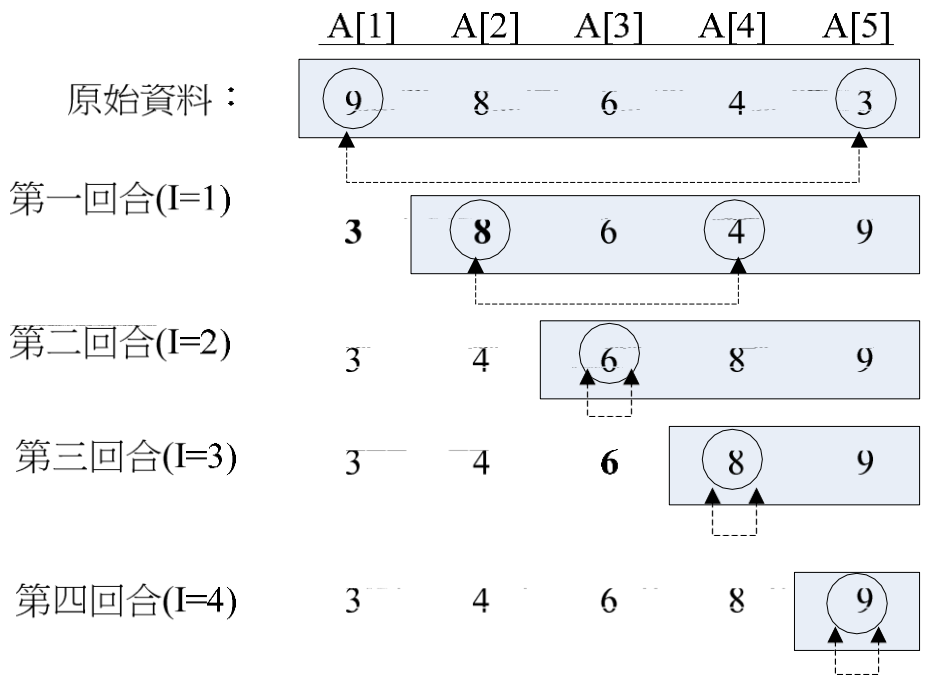
8-3 選擇排序法（Selection Sort）

【定義】

先以某一數值為基準，再由左至右掃瞄比目前大或小的數字，找到時先記錄其位置或索引值，待確定後再進行資料的交換，而這樣的方法我們稱之為選擇排序法（Selection Sort）。

【原理】

第一回合由資料中選取最小的資料和第一個資料對調、第二回合由資料中選取第二小的資料和第二個資料對調(因最小的資料已排到第一個位置)、依此循環直到最後一個資料，即完成資料的排序。如下圖所示：



「選擇排序法」的演算法如下：

```
01  Procedure SelSort(int A [], int n)
02   Begin
03   for (i = 0; i < n - 1; i++)              //控制排序n-1個回合
04     {
05       Min = i;                             //設定最小值索引
06       for ( j = i + 1; j <= n; j++)        //從第1個元素開始掃瞄
07          if (A[Min]>A[ j])                 //如果左邊元素大於右邊元素
08             Min = j;                       //則重新設定最小值索引
09       {                                    //並進行兩個的資料交換位置
10        Temp = A[i];
11        A[i] = A[Min];
12        A[Min] = Temp;
13       }
14     }
15   End
16  End Procedure
```

【實例】

假設原始資料為：7, 3+,1,6,3，在進行排序時，每一回合必定會有一個元素排到定位，稱為一個回合(Pass)。

	A[0]	A[1]	A[2]	A[3]	A[4]	比較次數	比較範圍
原始資料	**7**	**3+**	**1**	**6**	**3**		
Pass 1	**1**	**3+**	**7**	**6**	**3**	**4(1~4)**	**A[0]**與**A[1]~A[4]** 比較
Pass 2	**1**	**3**	**7**	**6**	**3+**	**3(2~4)**	**A[1]**與**A[2]~A[4]** 比較
Pass 3	**1**	**3**	**3+**	**6**	**7**	**2(3~4)**	**A[2]**與**A[3]~A[4]** 比較
Pass 4	**1**	**3**	**3+**	**6**	**7**	**1(4)**	**A[3]**與**A[4]**比較
					總比較次數	**10**	

【分析】

1. 比較之**回合次數**= **資料個數**-1

   (例如：資料**個數**n=5，則**回合次數為**4)

2. 在**每一回合之後**，至少會**有一個資料**可以排列到正確位置，**再進行下一個回合**的排列時，便可以**減少此資料的比較**。(例如：資料個數n=5，則Pass 1時，**比較次數為**4，Pass 2時，比較次數為3，以此類推，如上表所示)

3. 需要暫存最小值的**額外空間**。

   例如：在上面的演算法中的行號05，需要一個Min變數空間。

4. 為一種**不穩定排序** (Unstable Sorting)。

   (1) **排序前**：(兩個相同鍵值3，故第二個鍵值3寫成3^+)

7	3^+	1	6	3

   (2) **排序後**：

1	3	3^+	6	7

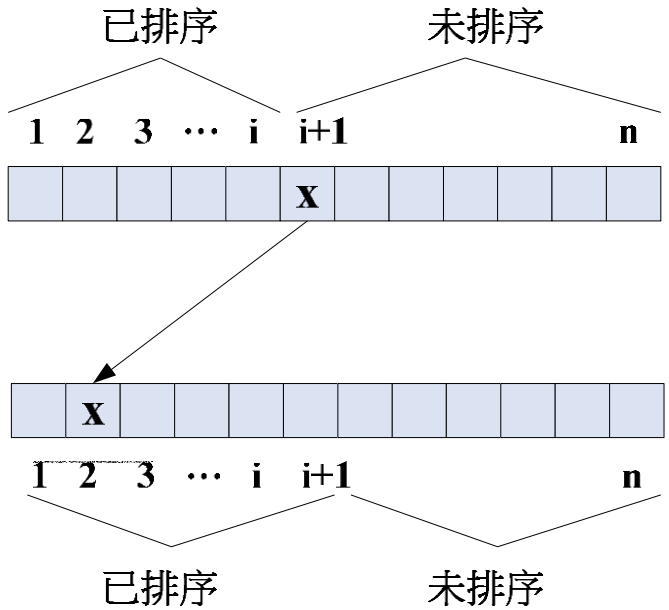
   (因為兩個3的相對位置在排序前後是不相同的)

5. **資料量愈小**，選擇排序法的**效果愈好**。

8-4 插入排序(Insertion Sort)

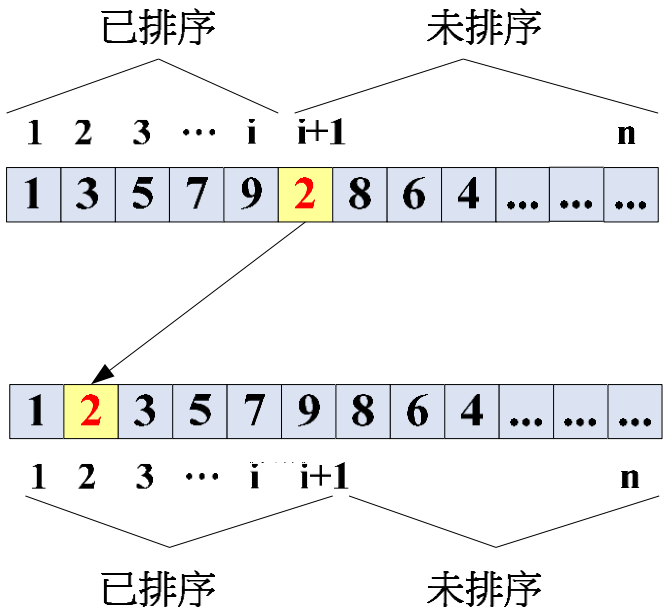
【定義】

是指每一次排序時都必須要往後拿一筆記錄，插入到前面已經排序好的記錄中。像是玩樸克牌一樣，我們將牌分作兩堆(第一堆為已排序，第二堆則尚未排序)，每次從第二堆牌中抽出第一張牌，然後插入到第一堆牌的適當位置。如下圖所示。



【原理】

是指將陣列中的元素(未排序)，逐一與已排序好的資料作比較，再將該陣列元素插入適當的位置。其排序原理如下所示：



「插入排序法」的演算法如下：

```
Procedure Insertion_Sort(list[], int n)
 Begin
        var i, j, instn
        for (i = 1; i < n - 1; i++)
         {
            instn=list[i];
            for ( j = i - 1; j >= 0; j--)
              if (instn<list[j])
                 list[j+1] = list[j];
              else
                 exit for
              end if
            end for
            list[j+1] = instn;
         end for
 End Procedure
```

【實例】

假設原始資料為：7, 3+,1,6,3，**在進行排序時，每一回合必定會有一個元素排到定位，稱為一個回合**(Pass)。

	A[0]	A[1]	A[2]	A[3]	A[4]
原始資料	7	3+	1	6	3
Pass 1	3+	7	1	6	3
Pass 2	1	3+	7	6	3
Pass 3	1	3+	6	7	3
Pass 4	1	3+	3	6	7

【分析】

1. 比較之**回合次數**= **資料個數**-1

   (例如：資料**個數**n=5，則**回合次數為**4)

2. 只需一個**額外的空間**，所以空間複雜度為最佳。

3. 為一種**穩定排序** (Stable Sorting)。

   (1) **排序前**：(兩個相同鍵值3，故第二個鍵值3寫成3^+)

7	3^+	1	6	3

   (2)**排序後**：(因為兩個3的相對位置在排序前後是**相同**的)

1	3^+	3	6	7

4. 此排序法**適用**於大部份資料**已經排序完成的資料**。

8-5 快速排序(Quick Sort)

【定義】

快速排序法又稱**分割交換排序法**，其觀念是先在資料中找到一個**中間值**，把**小於中間值的資料**放在**左邊**，而**大於中間值的資料**放在**右邊**，再以同樣的方式**分別處理左右兩邊**的資料，直到完成為止。

【作法】

1. 取**第一個記錄**的**鍵值** K_0 **當作中間值**。
2. **由左而右**，找到**第一個** K_i，使得$K_i \geqq K_0$。

   由右而左，找到**第一個** K_j，使得$K_j \leqq K_0$。

   < 亦即**從左找比它大**，**從右找比它小**的數字>
3. **若** i < j **則** K_i **與**K_j **對調位置**，並繼續執行**步驟**2.

   否則，K_0與 K_j **對調位置**，此時**以j為基準點**將此記錄資料串列分為**左右兩部份**。並以**遞迴方式**分別為**左右兩半進行排序**，直至完成排序。其排序過程如下所示：

   原始資料：26,5,37,1,61,11,59,15,48,19

其排序過程如下所示：

原始資料：26,5,37,1,61,11,59,15,48,19

K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9
26	**5**	**37** **i=2**	**1**	**61**	**11**	**59**	**15**	**48**	**19** **j=9**

從左找比K_o大，從右找比K_o小的數字，因為i<j所以K_i與K_j交換。因此，繼續比較下去：

K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9
26	5	19	1	**61**	11	59	**15**	48	37
				i=4			j=7		

從左找比K_o大，從右找比K_o小的數字，因為i<j所以K_i與K_j交換。因此，繼續比較下去：

K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9
26	5	19	1	15	11	59	61	48	37
					j=5	i=6			

因為i>j所以K_o與K_j交換。並且以**j=5**為基準點分割成左右兩部份。

K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9
[11	5	19	1	15]	**26**	[59	61	48	37]

同樣的步驟，在各子集合中，將找出第一個鍵值K_o 當作中間值，並且將小於K_o的資料放在左半邊， 而大於K_o的資料放在右半邊，直到全部完成為止。

K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9
[1	5]	**11**	[19	15]	26	[59	61	48	37]

注意：每一回合只能處理一個子集合，其完整的排序過程如下所示：

Pass 1	[11 5 19 1 15] 26 [59 61 48 37]
Pass 2	[1 5] 11 [19 15] 26 [59 61 48 37]
Pass 3	1 [5] 11 [19 15] 26 [59 61 48 37]
Pass 4	1 5 11 [19 15] 26 [59 61 48 37]
Pass 5	1 5 11 [15] 19 26 [59 61 48 37]
Pass 6	1 5 11 15 19 26 [59 61 48 37]
Pass 7	1 5 11 15 19 26 [48 37] 59 [61]
Pass 8	1 5 11 15 19 26 [37] 48 59 [61]
Pass 9	1 5 11 15 19 26 37 48 59 [61]

【分析】

1. 比較之**回合次數**= **資料個數**-1

   (例如：資料**個數**n= 5，則**回合次數為**4)

2.需要**額外的堆疊**（Stack）**空間**。

3. **為一種不穩定排序** (Unstable Sorting)。

(1)**排序前**：(兩個相同鍵值3，故第二個鍵值3寫成3^+)

5	3^+	1	6	3

(2)**排序後**：(因為兩個3的相對位置在排序前後是不相同的)

1	3	3^+	5	6

4. 時間複雜度：**最壞情況為** $O(n^2)$ **與平均情況為** $O(n \log_2 n)$。

5. **快速排序法**是**平均執行時間最快**的內部排序法。

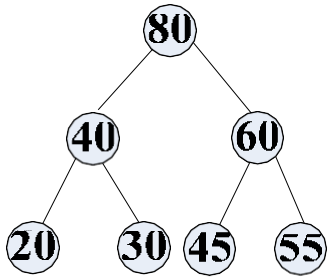
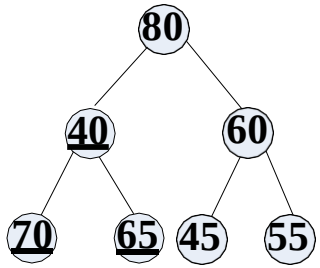
8-6 堆積排序(Heap Sort)

【定義】

堆積排序法就是利用堆積樹的樹根與最後一個節點交換，再重新建立堆積樹，直到只剩下最後一個節點為止，排序也完成了。

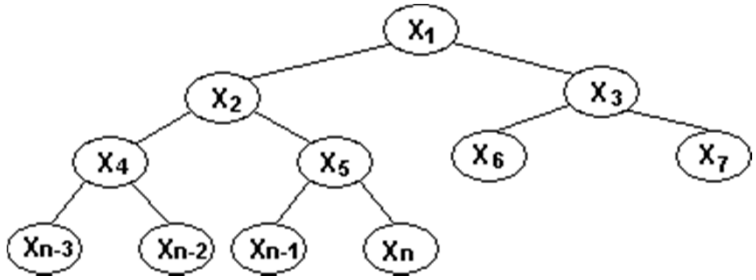
【特性】

1. 堆積樹是一棵完整二元樹(Complete Binary Tree)。
2. 每一個節點之值均大於或等於它的兩個子節點之值。
3. 樹根的值是堆積樹中最大的。如下圖(A)所示。

圖 (A)是堆積樹	圖 (B)非堆積樹
	

【作法】

1. 將原始資料(x_1 , x_2 , x_3 , … , x_n)轉換成完整二元樹。如下圖 所示



2. 將完整二元樹化為堆積樹 (heap tree) 。
3. 將樹根與最後一個節點交換。
4. 二元樹其他鍵值重複依照步驟(2)與步驟(3)的方法交換，直到只剩下最後一個節點為止。

【分析】

1. 比較之回合次數 = 資料個數-1

   (例如：資料個數n=5，則回合次數為4)

2. 需要一個額外的記錄空間。

3. 為一種不穩定排序 (Unstable Sorting)。

   (1)排序前：(兩個相同鍵值3，故第二個鍵值3寫成3+)

7	3+	1	6	3

   (2)排序後：(因為兩個3的相對位置在排序前後是不相同的)

1	3	3+	6	7

4. 時間複雜度：最壞情況與平均情況都是$O(n \log_2 n)$。

【舉例】

假設一個尚未排序的陣列中包含下列整數：

45,83,7,61,12,99,44,77,14,29

請利用「堆積排序法」來完成以上資料的排序。

【解答】其步驟如下：

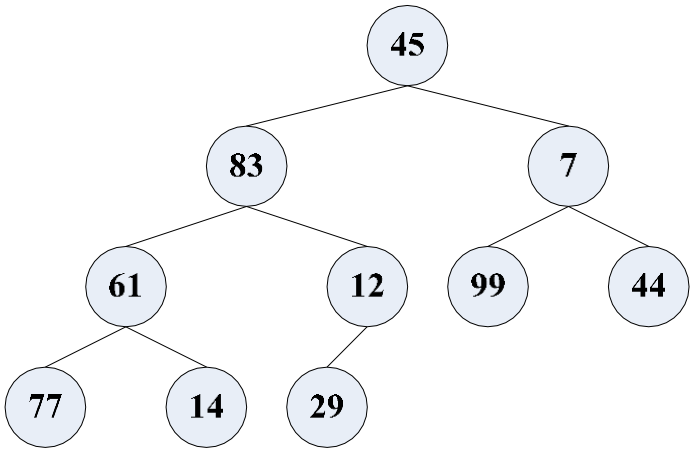
(1)建立完整二元樹(Complete binary tree)

(2)將完整二元樹轉換成堆積樹

(3)堆積排序(Heap sort)

(1)建立完整二元樹(依序加入，不需比較)

原始資料：45,83,7,61,12,99,44,77,14,29

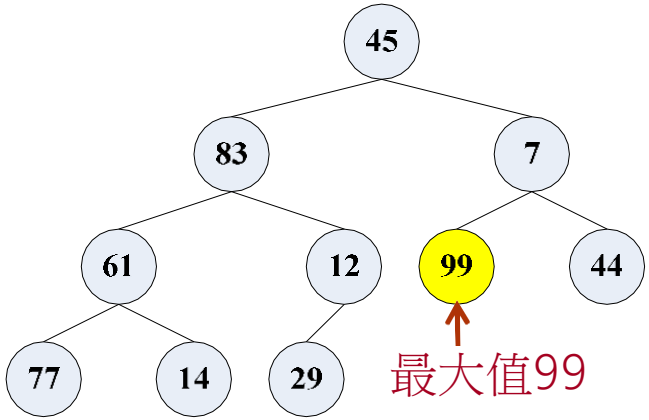


(2)將完整二元樹轉換成堆積樹

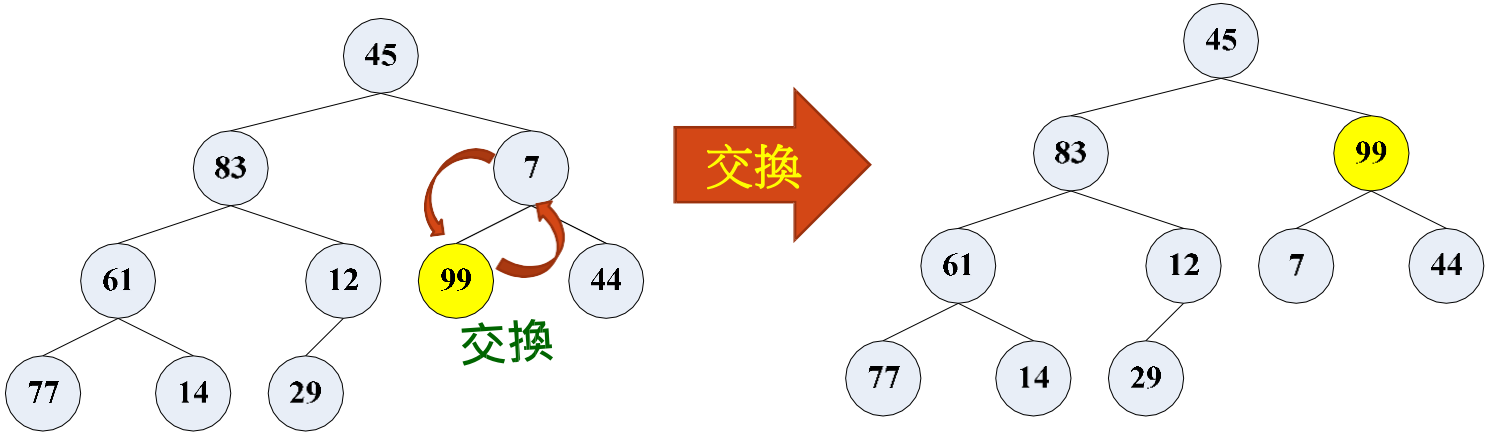
步驟1：在完整二元樹中找出最大值99，依序移到樹根

(99與7交換，再將45與99交換)。

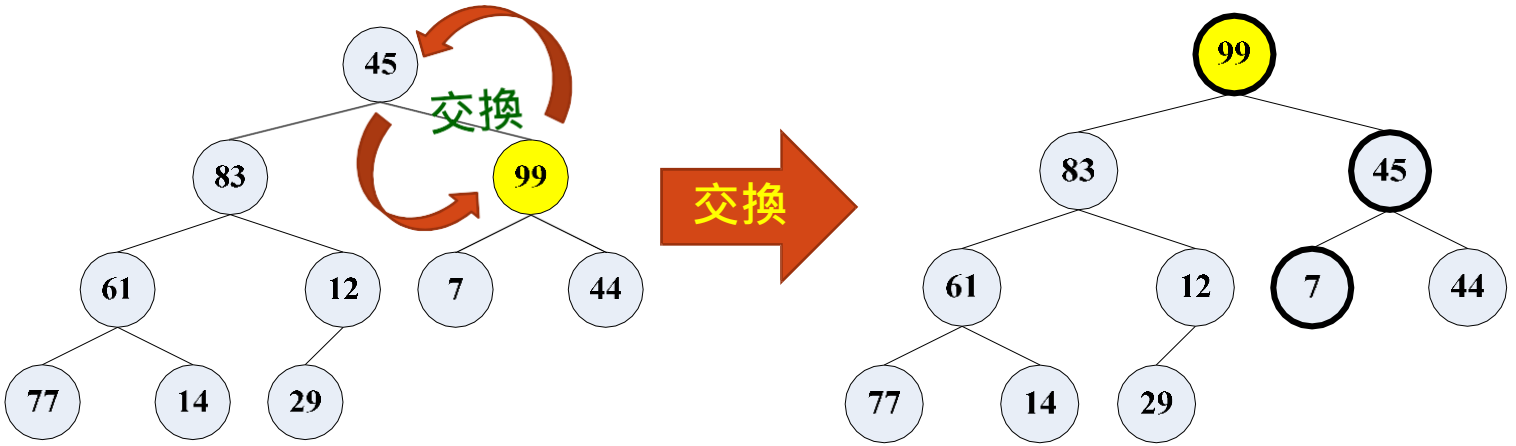
(1)在完整二元樹中找出最大值**99**



(2)依序移到樹根(**99**與7交換)

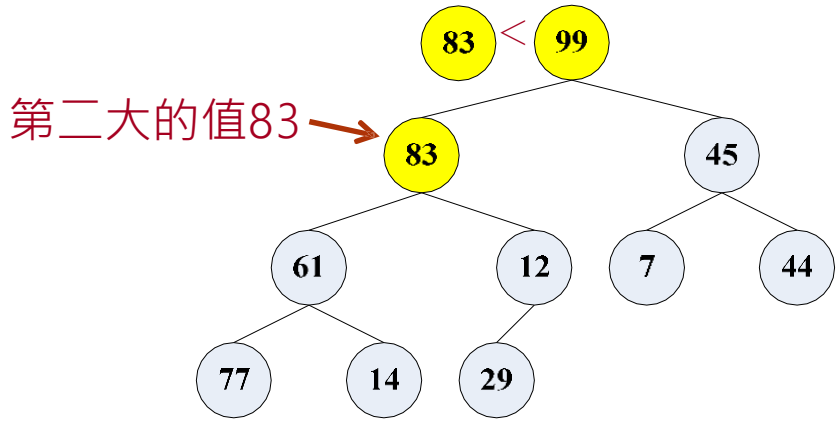


(3) 再將**45**與99交換

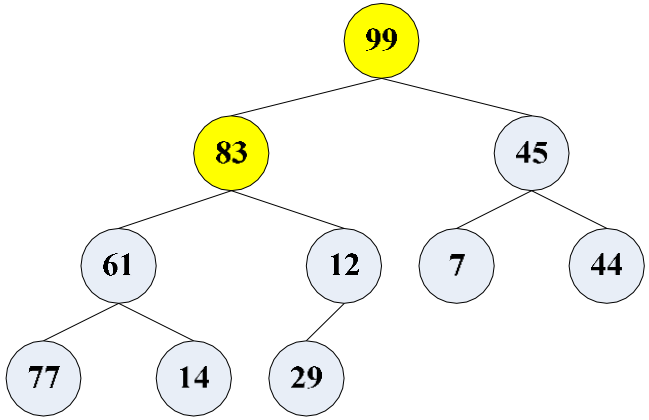


步驟2：在完整二元樹中找出第二大的值83，此時83鍵值小於父節點99。因此，不須做任何交換。

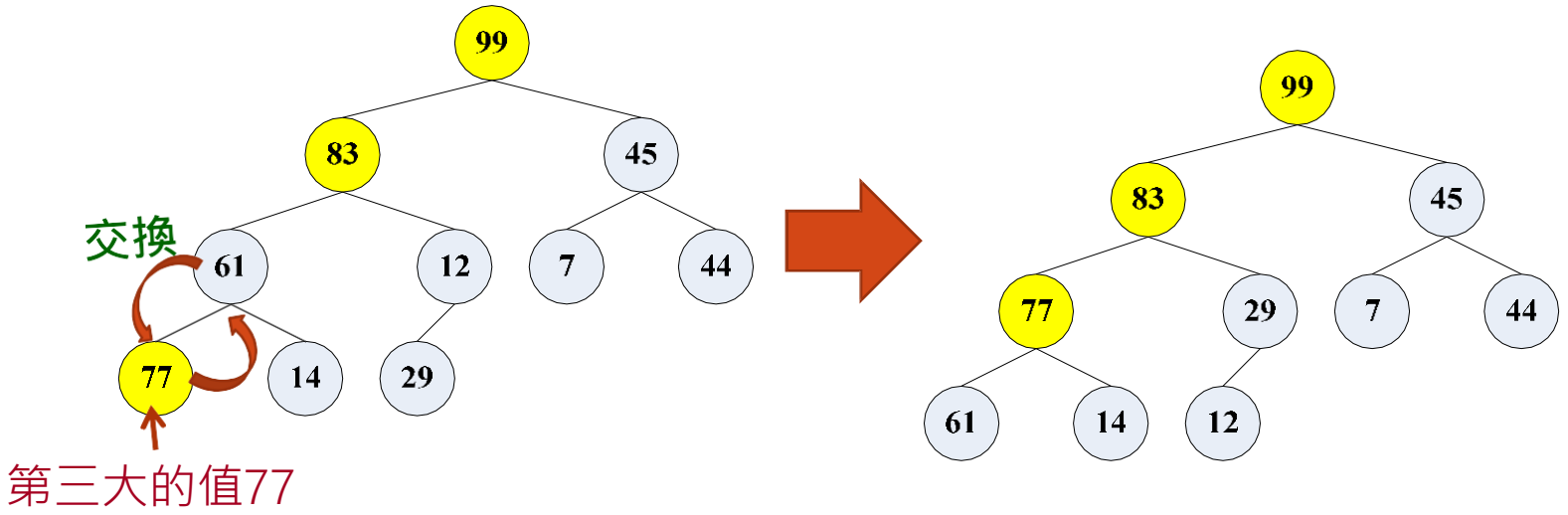
(1)找出第二大的值**83**，此時**83**鍵值小於父節點**99**



(2)不須做任何交換

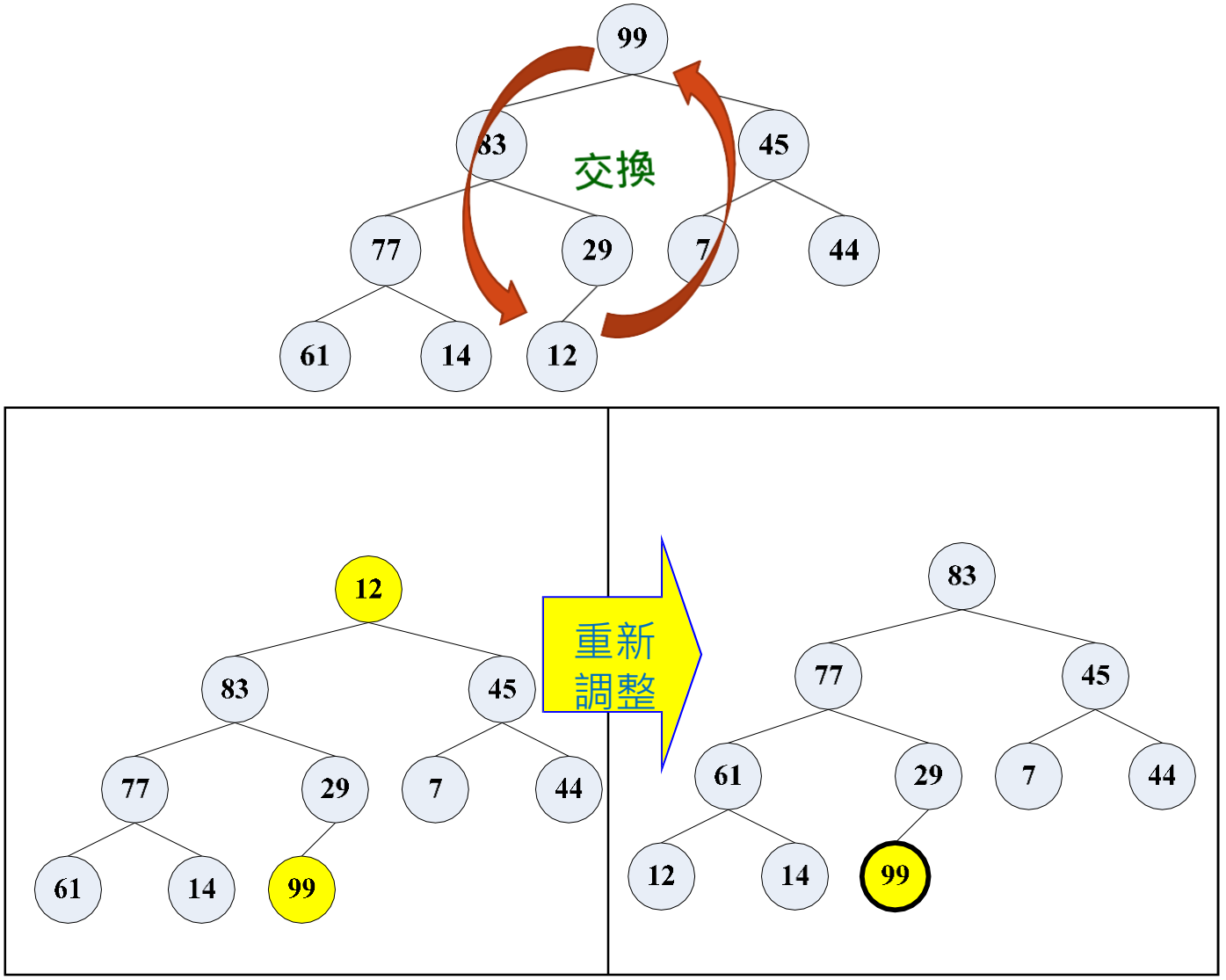


步驟3：最後在完整二元樹中找出第三大的值77，與其上一層的父節點比較，比61大，所以61與77位置交換。直到全部的節點的鍵值大於它的左子樹與右子樹的鍵值，即可成一棵最大堆積樹。



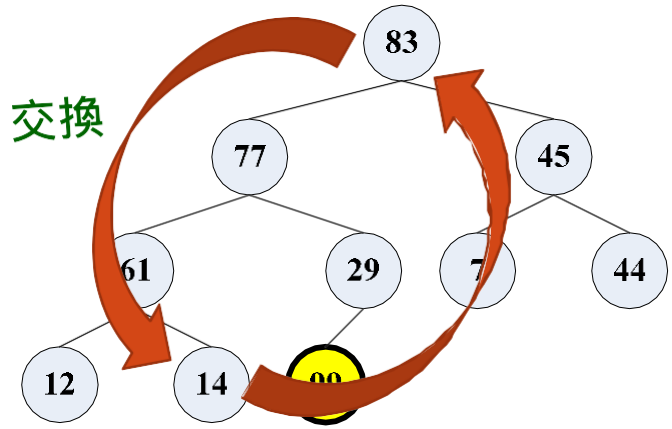
(3)堆積排序(Heap sort)

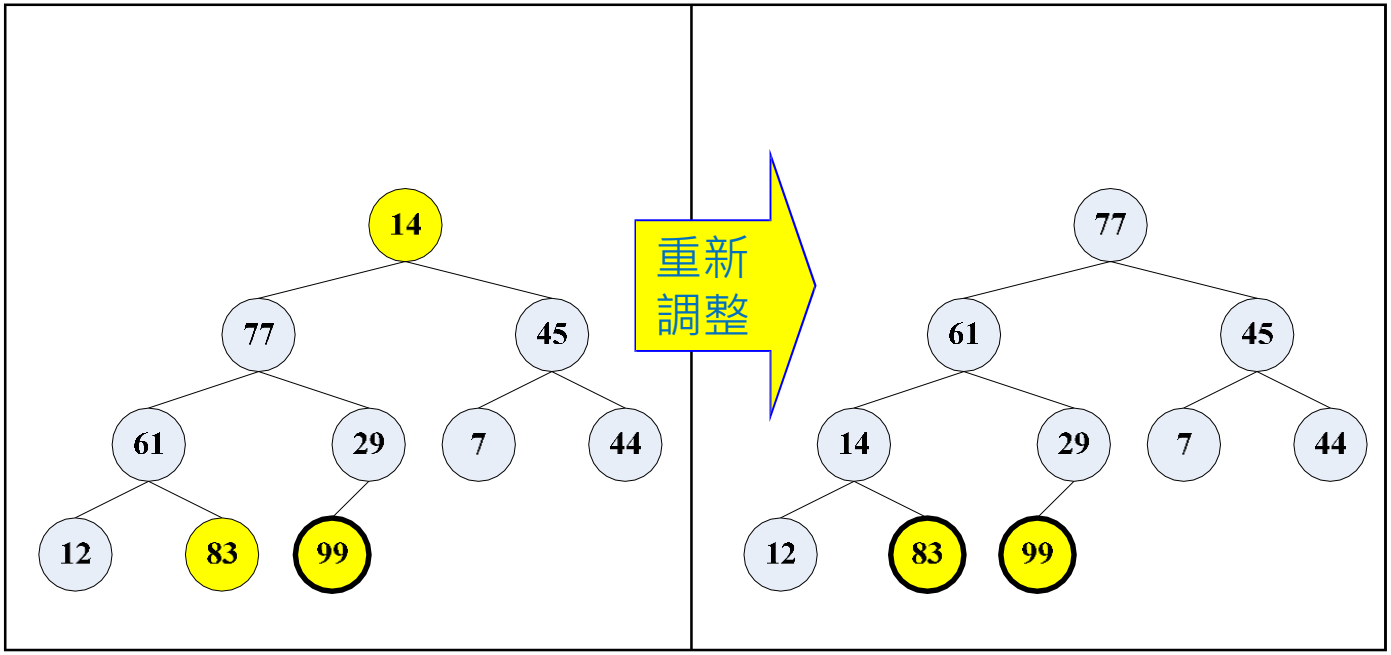
由第一階段：將樹根(99)與最後一個節點(12)交換，再重新調整之後的堆積樹，如下圖所示：



由第二階段：將樹根(83)與**倒**數最後二個節點(14)交換，

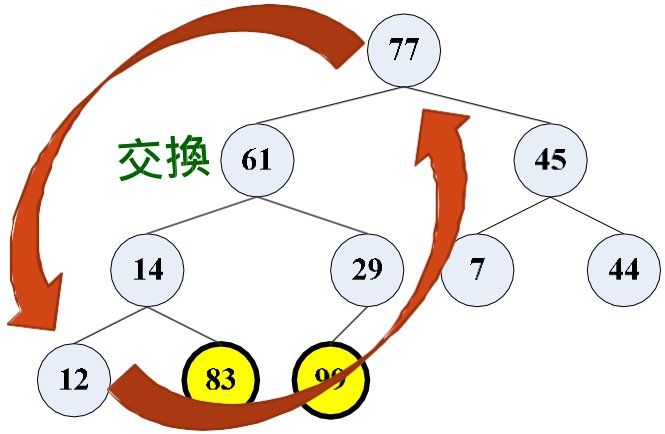
再重新調整之後的堆積樹，如下圖所示：

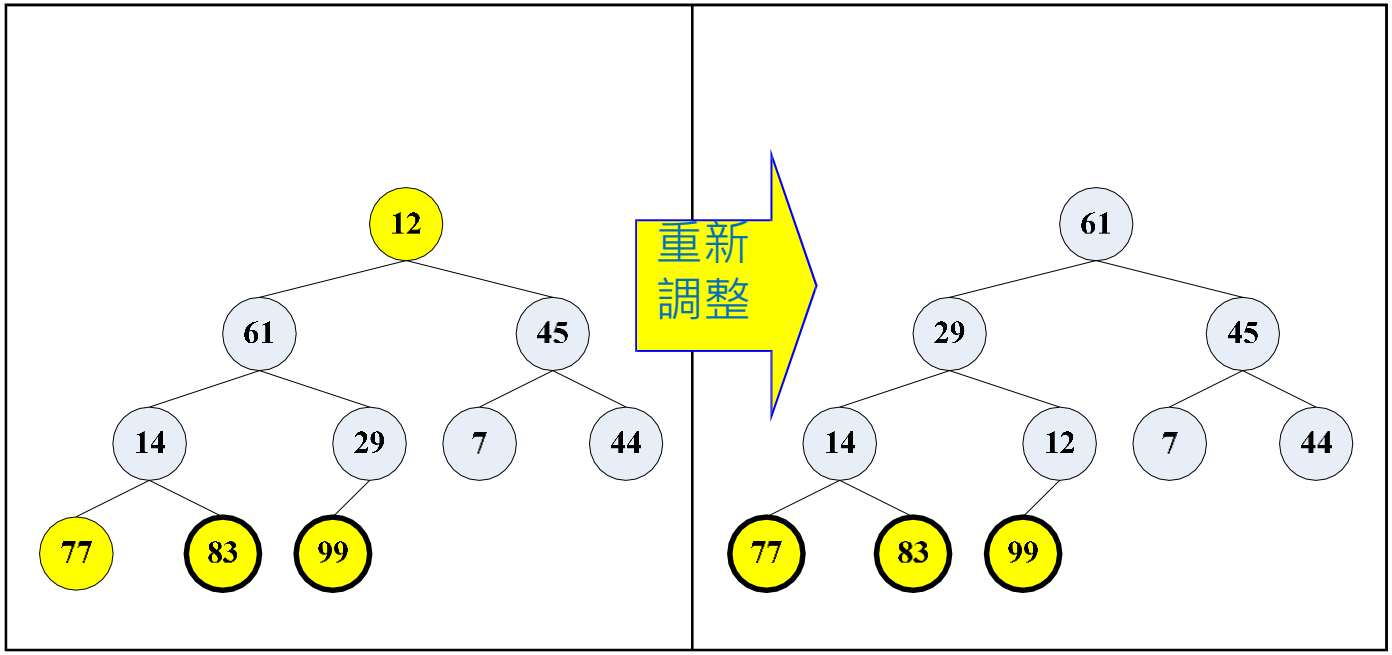




由第三階段：將樹根(77)與**倒**數最後三個節點(12)交換，

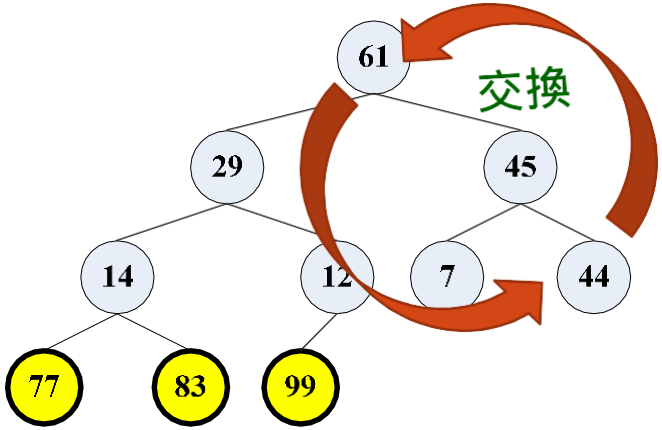
再重新調整之後的堆積樹，如下圖所示：

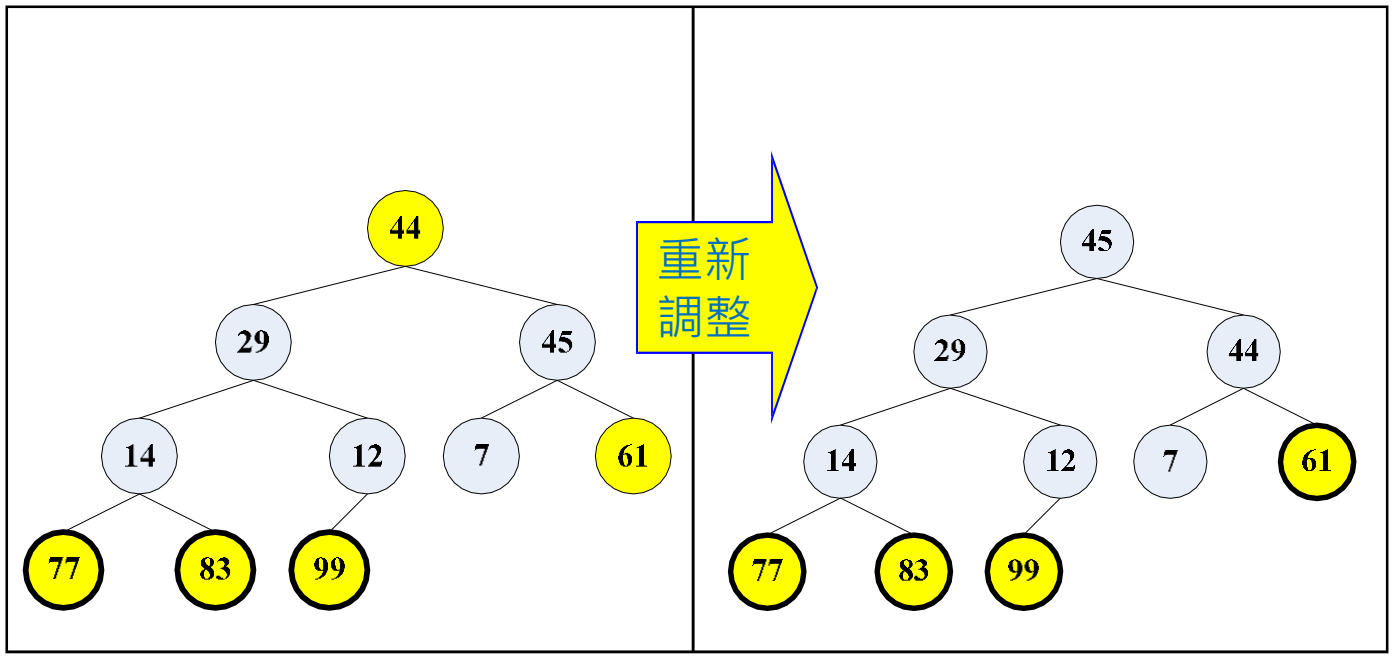




由第四階段：將樹根(61)與**倒**數最後**四**個節點(44)交換，

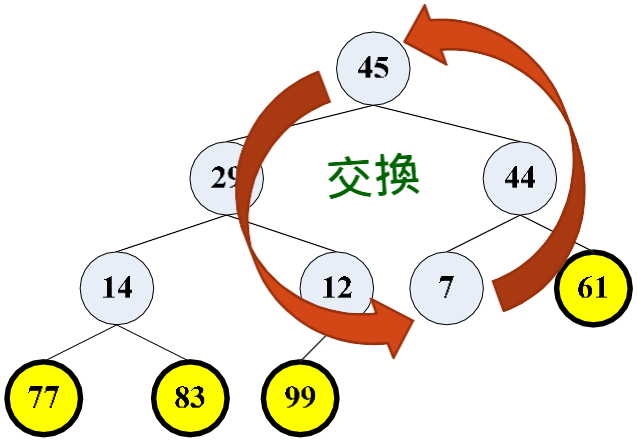
再重新調整之後的堆積樹，如下圖所示：

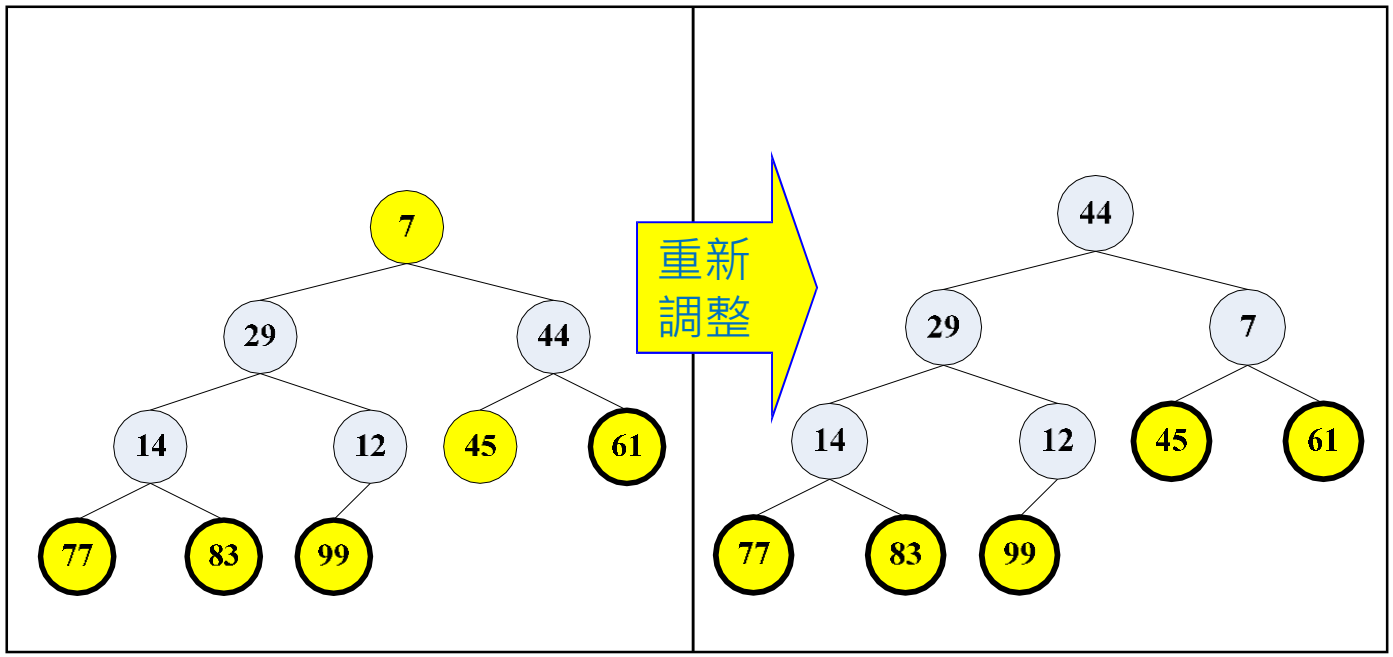




由第五階段：將樹根(45)與**倒**數最後**五**個節點(7)交換，

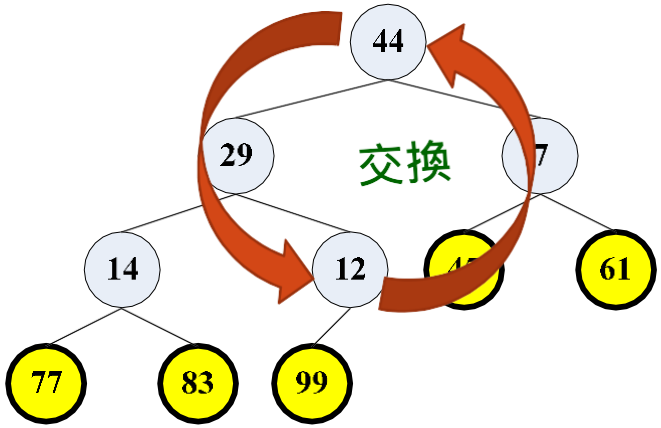
再重新調整之 後的堆積樹，如下圖所示：

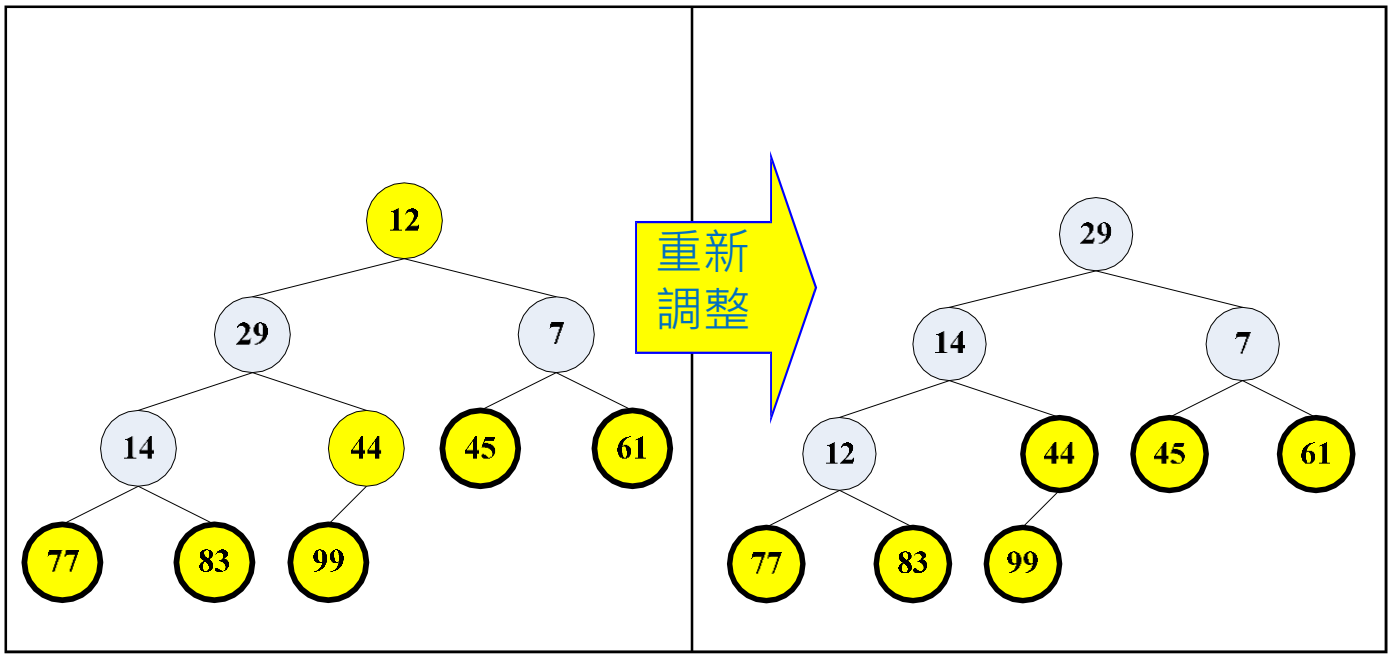




由第六階段：將樹根(44)與**倒**數最後**六**個節點(12)交換，

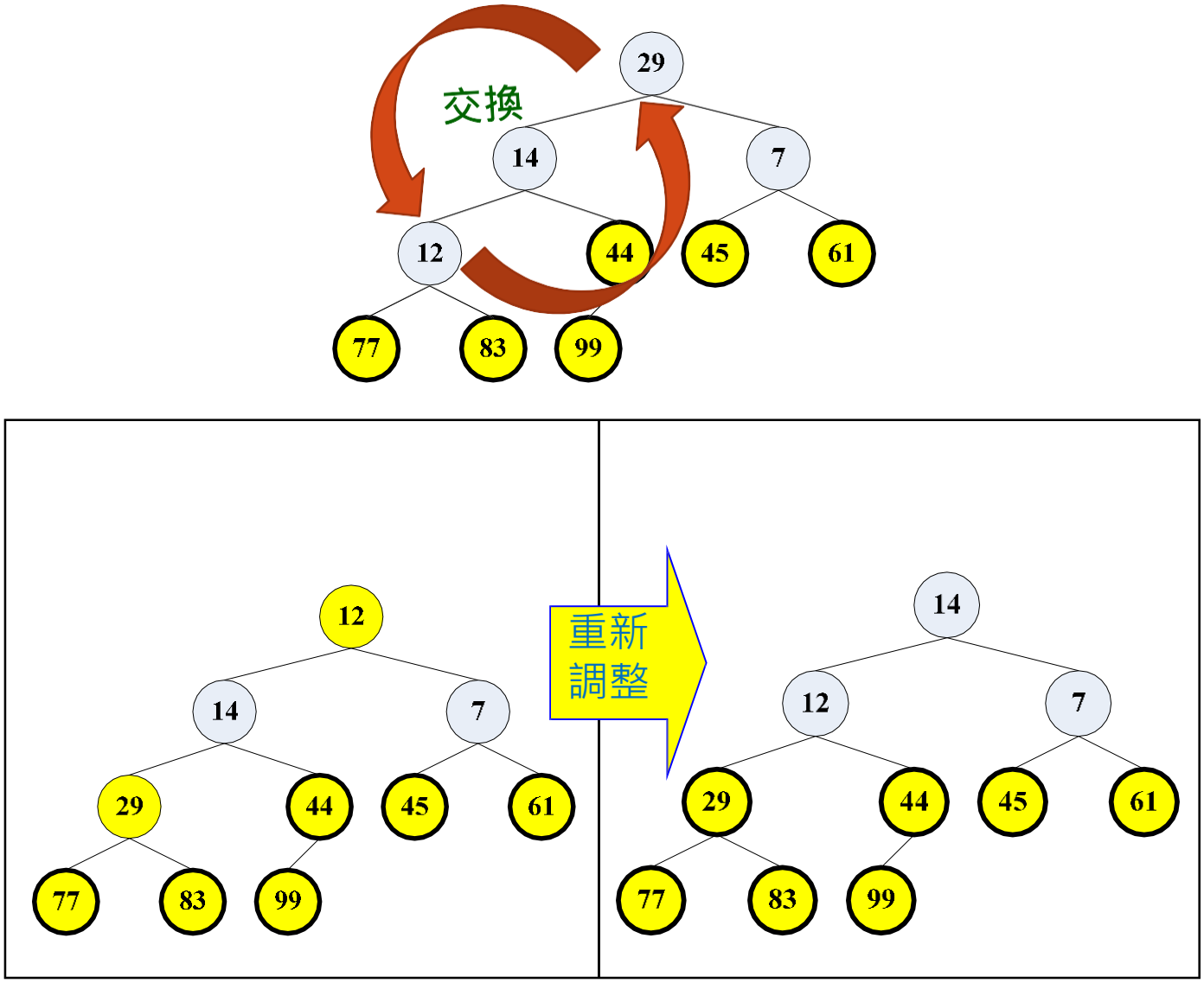
再重新調整之後的堆積樹，如下圖所示：





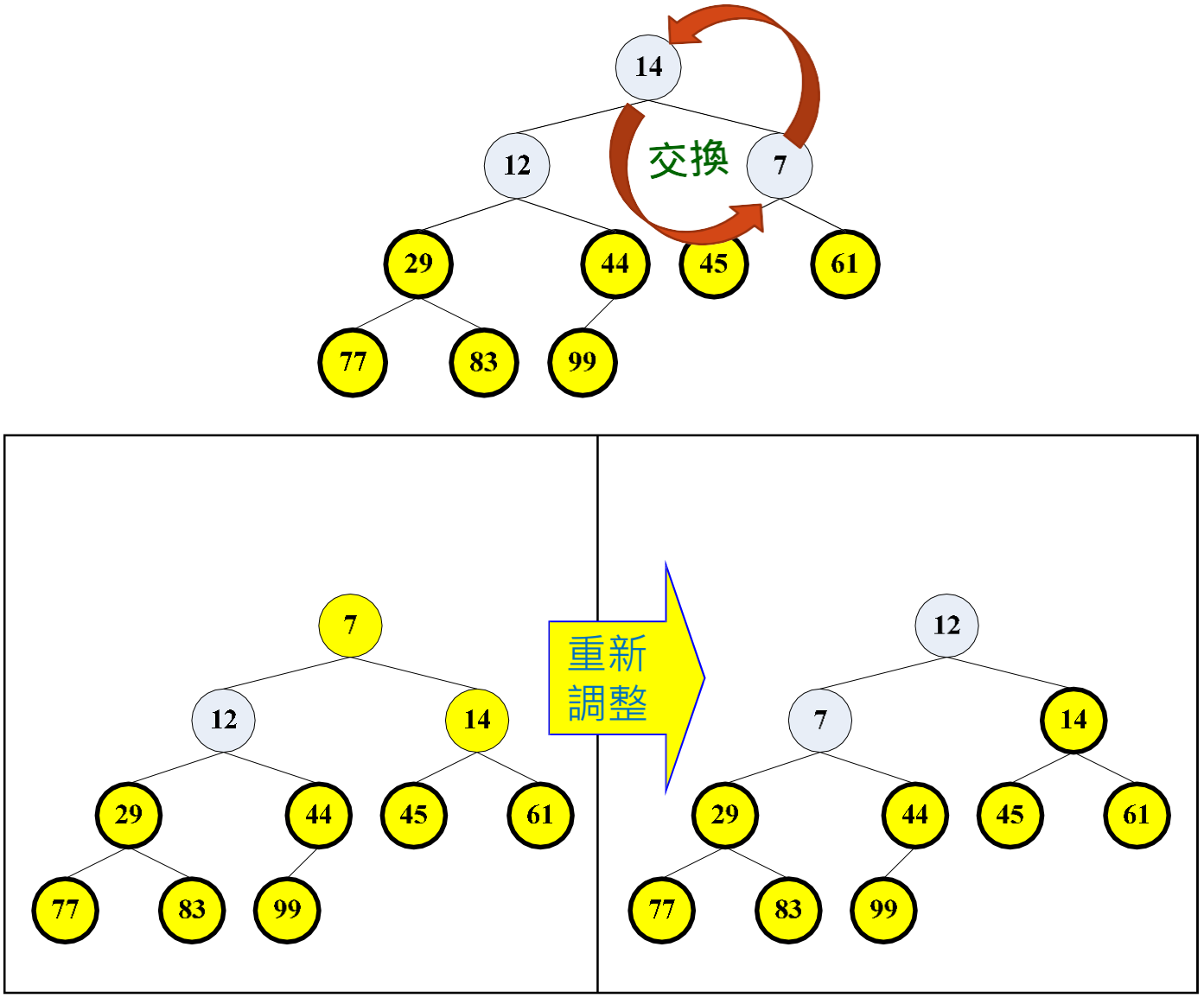
由第七階段：將樹根(29)與**倒**數最後**七**個節點(12)交換，

再重新調整 之後的堆積樹，如下圖所示：

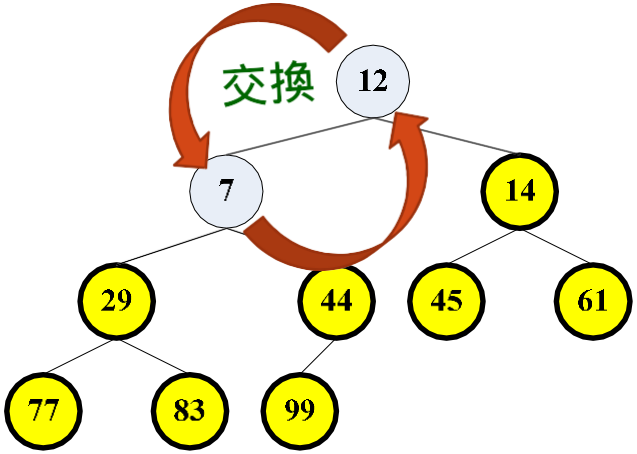


由第八階段：將樹根(14)與**倒**數最後八個節點(7)交換，

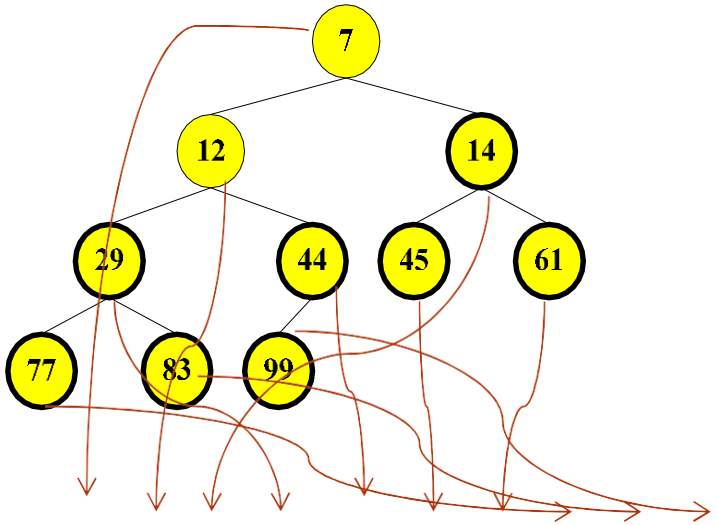
再重新調整之後的堆積樹，如下圖所示：



由第九階段：將樹根(12)與倒數最後九個節點(7)交換



最後已完成堆積排序，如下圖所示：



最後的結果為：7,12,14,29,44,45,61,77,83,99

8-7 謝耳排序(Shell sort)

【定義】由D.L. Shell所提出，方法是插入排序法演進而來，其目的是用來**減少插入排序法**中元素**搬移的次數**，**增快排序的速度**。

【作法】

1. 利用某一間隔值來分割資料，再利用插入排序法進行排序。
2. 若有n筆資料要進行排序時，先求出初始間隔值 Gap =⌊n DIV 2⌋
3. 依照間隔值將資料分割成數個區塊。
4. 再利用插入排序法針對數個區塊內的資料進行排序
5. 最後，再縮小間隔值範圍，重複步驟3與步驟4，直到間隔值Gap=1，排序即可完成。

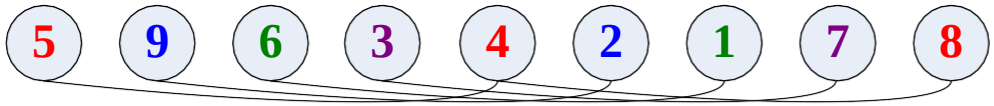
【舉例】

請利用「謝耳排序法」來排序以下的資料：

原始資料：5 9 6 3 4 2 1 7 8

【解答】

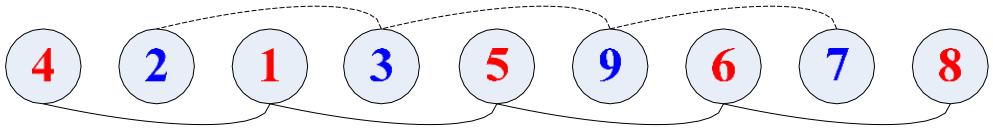
1. 初始間隔值 Gap = $\lfloor 9 \text{ DIV } 2 \rfloor = 4$



(1)依照間隔值將資料分割為四個部份，分別為(5,4,8)(9,2)(6,1)(3,7)

(2)再利用插入排序法來排序，其結果如下：(4,5,8)(2,9)(1,6)(3,7)

2.再**縮小間隔值**Gap= $\lfloor 4 \text{ DIV } 2 \rfloor = 2$



(1)依照間隔值將資料分割為二個部份，分別為(**4,1,5,6,8**)(**2,3,9,7**)

(2)再利用插入排序法來排序，其結果如下：(**1,4,5,6,8**)(**2,3,7,9**)

3.**再縮小間隔值**Gap= $\lfloor 2 \text{ DIV } 2 \rfloor = 1$，最後再利用**插入排序法**來排序

【分析】

假設原始資料為：7, 3+, 1, 6, 3，在進行排序時，每一回合必定會有一個元素排到定位，稱為一個回合(Pass)。

	A[0]	A[1]	A[2]	A[3]	A[4]
原始資料	7	3+	1	6	3
Pass 1	1	3+	3	6	7
Pass 2	1	3+	3	6	7

1. 為一種穩定排序 (Unstable Sorting) 。

(1)排序前：(兩個相同鍵值3，故第二個鍵值3寫成3+)

7	3+	1	6	3

(2)排序後：(因為兩個3的相對位置在排序前後是相同的)

1	3+	3	6	7

2. 時間複雜度：最壞情況為$O(N^S)$，平均情況為$O(n(\log_2 n))$。

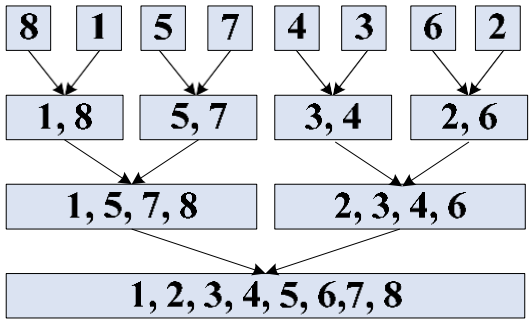
8-8 合併排序(Merge Sort)

【定義】

合併排序適用於內部排序和外部排序，也是一種**典型**的「分而治之」的方法。

【作法】

1. 將 N個長度為 1 的鍵值成對地合併長度為 2 的鍵值組。
2. 將 N/2個長度為 2 的鍵值成對地合併長度為 4 的鍵值組。
3. 將鍵值組成對地合併，直到合併成一組長度為 N 的鍵值組為止。如下圖所示。



【舉例】<偶數個資料>

請利用「合併排序法」由小至大來寫出以下資料之排序的過程。

原始資料：25,57,48,37,12,92,86,33

【解答】

<準備動作> [25 , 57][48 , 37][12 , 92][86 , 33]

<第一回合> [25 , 57][37 , 48][12 , 92][33 , 86]

<第二回合> [25 , 37 , 48 , 57][12 , 33 , 86 , 92]

<第三回合> [12 , 25 , 33 , 37 , 48 , 57 , 86 , 92]

【舉例】<奇數個資料>

請利用「合併排序法」由小至大來寫出以下資料之排序的過程。

原始資料： 37,57,32,23,15

【解答】

<準備動作> [37 , 57][32 , 23][15]

<第一回合> [37 , 57][23 , 32][15]

<第二回合> [23 , 32 , 37 , 57][15]

<第三回合> [15 , 23 , 32 , 37 , 57]

【分析】

假設原始資料為：7, 3+, 1, 6, 3，在進行排序時，每一回合必定會有一個元素排到定位，稱為一個回合(Pass)。

	A[0]	A[1]	A[2]	A[3]	A[4]
原始資料	7	3+	1	6	3
Pass 1	3+	7	1	6	3
Pass 2	1	3+	6	7	3
Pass 3	1	3+	3	6	7

1. 為一種穩定排序 (Unstable Sorting) 。

(1)排序前：(兩個相同鍵值3，故第二個鍵值3寫成3+)

7	3+	1	6	3

(2)排序後：(因為兩個3的相對位置在排序前後是相同的)

1	3+	3	6	7

2. 時間複雜度：最壞情況與平均情況均為$O(n \log_2 n)$。

8-9 基數排序(Radix Sort)

【定義】

1. 先將n筆數字資料依個位數來加以「分配」，並分別放入由數字0,1,2,...9的暫存陣列Temp[10][n]中，再透過「合併」數字的順序放回原陣列。則此時的資料已依個位數大小由小到大排序。
2. 將n筆數字資料依十位數來加以「分配」，並分別放入由數字0,1,2,...9的暫存陣列Temp[10][n]中，再透過「合併」數字的順序放回原陣列。則此時的資料已依十位數和個位數大小由小到大排序。
3. 同理再作百位數、千位數、...即可得由小到大排序好的數字。

【作法】

1. 假設R 為基底 (Base, 或稱進制)，則必須要準備 R個桶子(Bucket)，編號為 0 ~ n-1
2. 假設 D 為n筆資料中的最大鍵值之位數個數，則須執行 D 回合才能完成排序(Sort) 工作
3. **從**最低位數到最高位數，其每一回合必須要完成以下的程序：
   (1)分配：依位數值，將資料分配到對應的桶子(Bucket)中
   (2)合併：指合併R個桶子(Bucket)中(從 0 ~ n-1)

【實例】將下列數字利用「基數排序法」由小至大來進行排序(Base=10) 。

原始資料：79, 8, 6, 93, 59, 84, 55, 9, 71, 33

【解答】

步驟一： Base = 10, ∴準備 10個桶子，編號為 0 ~ 9

步驟二：最大的數值是93，有二個位數，

∴D= 2，同時可知道需執行2個回合才會完成 Sort 工作

步驟三：從最低位數 (個位數) 開始執行各回合

1. 第一回合：把每個整數依其「個位數」為主排序

個位數	0	1	2	3	4	5	6	7	8	9
分配資料		71		93 33	84	55	6		8	79 59 9

合併：71,93,33,84,55,6,8,79,59,9

2.第二回合：把每個整數依其「十位數」為主排序

(將第一回合的結果當作第二回合的輸入資料來源)

十位數	0	1	2	3	4	5	6	7	8	9
分配資料	6 8 9			33		55 59		71 79	84	93

合併：6,8,9,33,55,59,71,79,84,93