Chapter 7 Sorting(1)

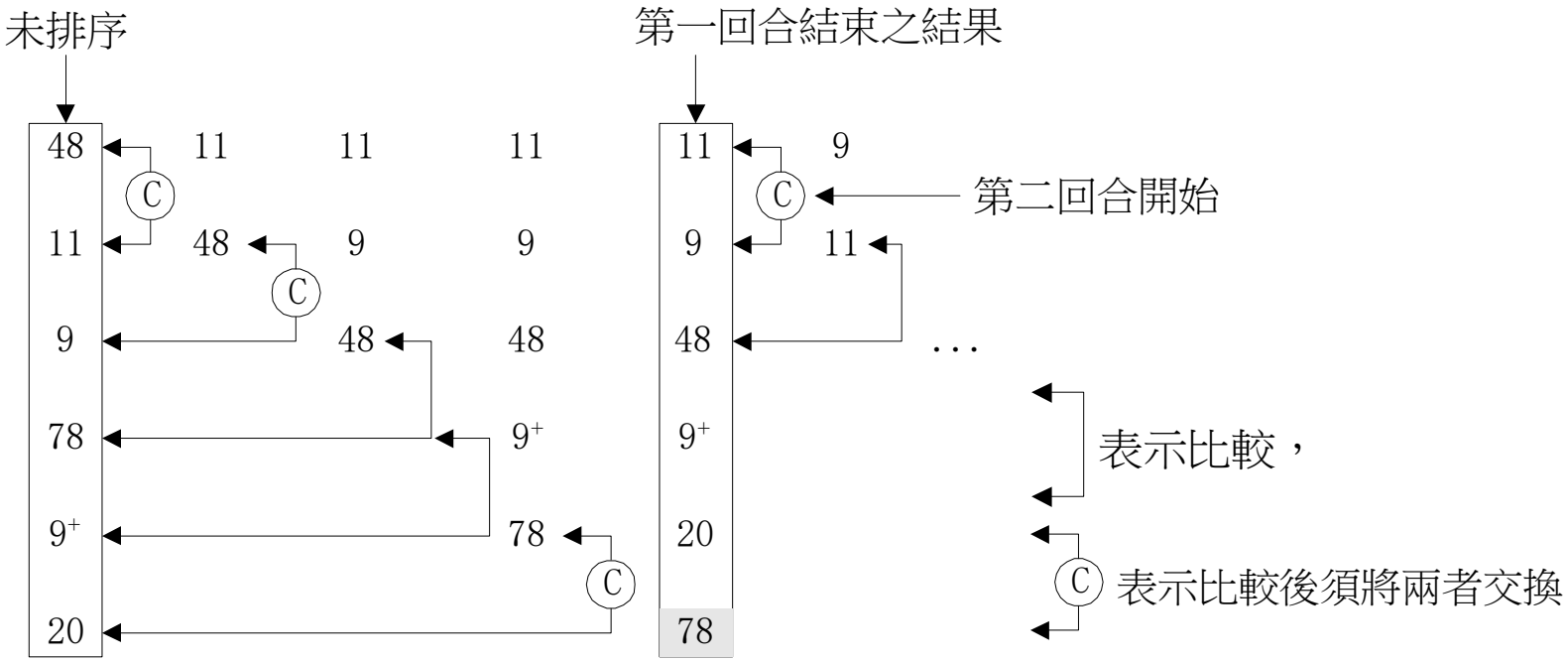
排序的基本觀念

- 檔案(File)
- 記錄(Row或Record)
- 欄位(Column或Field)
- 鍵值(Key Value)

姓名	性別	年齡	電話	身份證字號
劉德華	男	20	0935-1234567	B123456789
鄭秀文	女	21	0919-1231231	A200010101
張學友	男	21	0922-7654321	L112313515
王菲	女	22	0938-8765987	K256787897
成龍	男	22	0958-0123321	Z158963791

氣泡排序(bubble sort) (1)

- 將N筆記錄(編號0至N-1)按鍵值不遞減次序排序
  1. 重複步驟2　N-1回合，直到其中有一回合沒有「交換」情形發生為止。
  2. 比較陣列中相鄰兩元素之鍵值，若前面元素大於後面元素，則立刻將兩元素值交換。



氣泡排序(bubble sort) (2)

```
public static void bubbleSort(int[] data){
    int size = data.length;
    for (int i=size-1;i>0;i--)//掃瞄次數
    {
        for (int j=0;j<i;j++)    //比較、交換次數
        {
            // 比較相鄰兩數，如第一數較大則交換
            if (data[j]>data[j+1])
            {
                int tmp=data[j];
                data[j]=data[j+1];
                data[j+1]=tmp;
            }
        }
    }
}
```

作業07-01

- 請修改bubbleSort方法，若有一回合沒有「交換」情形發生則中止程式。並將其時間複雜度big-O寫出。

選擇排序(selection sort)(1)

- 首先在所有的資料中挑選一個最小的放置在第一個位置（因為由小到大排序），再從第二個以後的資料開始挑選一個最小的放置於第二個位置，.....，一直下去。
  - 例如有5個記錄，其鍵值為18, 2, 20, 34, 12。利用選擇排序，其做法如下：

		[1]	[2]	[3]	[4]	[5]
		18	2	20	34	12
Step 1:	最小為 2	2	18	20	34	12
Step 2:	從第2位置開始挑最小為 12	2	12	20	34	18
Step 3:	從第3位置開始挑最小為 18	2	12	18	34	20
Step 4:	從第4位置開始挑最小為 20	2	12	18	20	34

選擇排序(selection sort)(2)

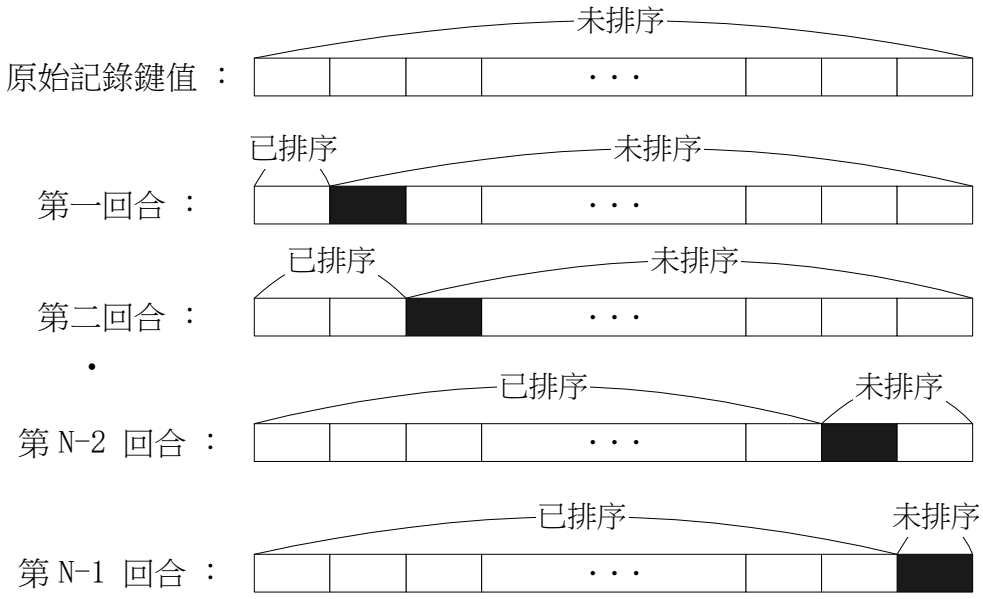
```
public static void selectionSort(int[] data){
    int size = data.length;
    int i,j,tmp;
    for(i=0;i<size-1;i++)          //掃描num-1次
        {
        for(j=i+1;j<size;j++)   //由i+1比較起 {
            if(data[i]>data[j])  //比較第i及第j個元素
            {
                tmp=data[i];
                data[i]=data[j];
                data[j]=tmp;
            }
        }
    }
}
```

作業07-02

- 請實作selectionSort方法。並將其時間複雜度big-O寫出。

插入排序(insertion sort)(1)

- 如同玩撲克牌賽；一拿到牌則插入手上已排序好的牌中。不先選擇資料但一拿到資料則選擇位置插入(insertion)。
- 將N筆記錄(編號1至N)依鍵值不遞減之次序排序的插入排序法為：
  1. 從第2個鍵值至第N個鍵值，分別執行步驟2。
  2. 將該鍵值插入到其前面所有鍵值當中第一個大於本身的鍵值之前，若沒有大於本身者，則保持原狀並繼續下一回合。



原始鍵值	每回合結果				
	一	二	三	四	五
25	25	18	18	5	5
37	37	25	21	18	18
18	18	37	25	21	21
21	21	21	37	25	21^+
5	5	5	5	37	25
21^+	21^+	21^+	21^+	21^+	37

(陰影部分已完成排序)

插入排序(insertion sort)(2)

```
public static void inertionSort(int[] data){
    int size = data.length;
    int i;      //i為掃描次數
    int j;      //以j來定位比較的元素
    int tmp;    //tmp用來暫存資料
    for (i=1;i<size;i++)   //掃描迴圈次數為SIZE-1
    {
        tmp=data[i];
        j=i-1;
        while (j>=0 && tmp<data[j])   //如果第二元素小於第一元素
        {
          data[j+1]=data[j]; //就把所有元素往後推一個位置
          j--;
        }
        data[j+1]=tmp;          //最小的元素放到第一個元素
    }
}
```

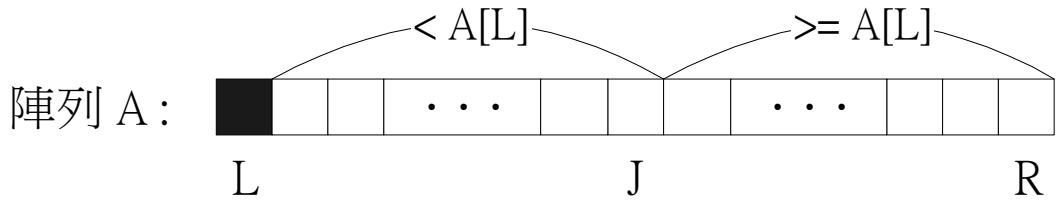
作業07-03

- 請實作insertionSort方法。並將其時間複雜度big-O寫出。

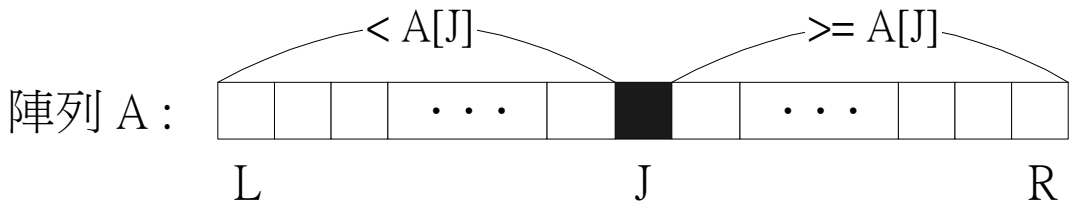
快速排序法(Quick Sort) (1)

- 分而治之(Divide and Conquer)
- 觀念：將待排序的N個鍵值(編號0至N-1)分成左右兩半，左半邊之鍵值小於第一個鍵值(即a[0])，而右半邊則大於或等於第一個

鍵值，如下圖所示



- 將A[L]與A[J]之鍵值交換



快速排序法(Quick Sort) (2)

- 將N筆記錄(編號0至N-1)按鍵值不遞減之順序排列之快速排序法為：
  1. 令K=待排序範圍之第一個(最左邊)鍵值。(第一次為A[0])。
  2. 由左向右找出一個鍵值K_i，滿足$K_i > K$。 。
  3. 由右向左找出一個鍵值K_j，滿足$K_j < K$。
  4. 若$i < j$則將K_i與K_j交換，然後跳到步驟2。
  5. 若$i \geq j$則將K與K_j交換，並以j為基準分割成左右兩半，然後分別針對左右兩半進行步驟1至5，直到左半邊鍵值 = 右半邊鍵值為止。

快速排序法(Quick Sort) (3)

步驟	陣列元素										L	R	K	K_i	K_j	i	j	交換之鍵值對	分割情形
	0	1	2	3	4	5	6	7	8	9									
1	38	69	21	15	43	58	77	21^+	83	5	0	9	38	69	5	1	9	(69,5)	
2	38	5	21	15	43	58	77	21^+	83	69	0	9	38	43	21^+	4	7	($43,21^+$)	
3	38	5	21	15	21^+	58	77	43	83	69	0	9	38	58	21^+	5	4	($38,21^+$)	[0,3], [5,9]
4	[21^+	5	21	15]	38	[58	77	43	83	69]	0	3	21^+	21	15	2	3	(21,15)	
5	[21^+	5	15	21]							0	3	21^+	21	15	2	3	(21^+,15)	[0,1], [3,3]
6	[15	5]	21^+	[21]							0	1	15	x	5	2	1	(15,5)	[0,0]
7	[5]	[15]									0	0							
8	[21]										3	3							
9	[58	77	43	83	69]						5	9	58	77	43	6	7	(77,43)	
10	[58	43	77	83	69]						5	9	58	77	43	7	6	(58,43)	[5,5], [7,9]
11	[43]	58	[77	83	69]						5	5							
12	[77	83	69]								7	9	77	83	69	8	9	(83,69)	
13	[77	69	83]								7	9	77	83	69	9	8	(77,69)	[7,7], [9,9]
14	[77]	69	[83]								7	7							
15	[83]										9	9							

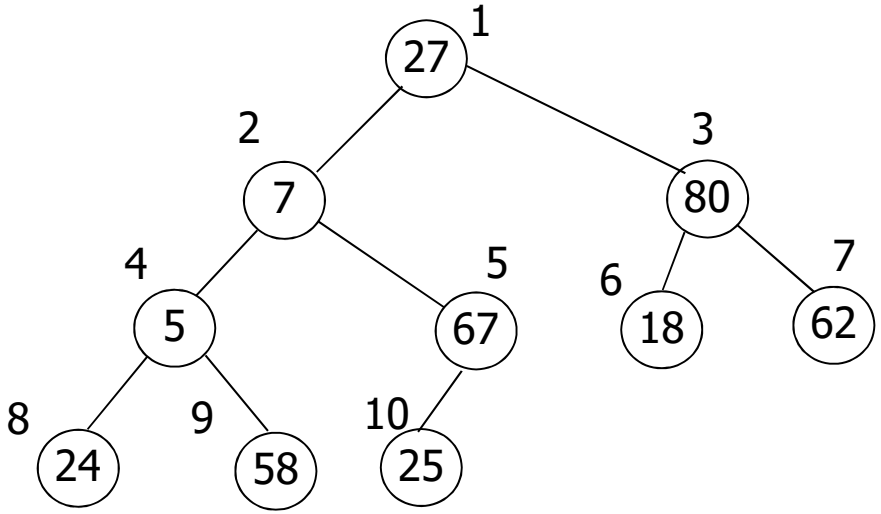
```
public static void quickSort(int d[],int size,int lf,int rg){
    //lf需要排序陣列的開頭index, rg需要排序陣列的最後一個index
    int i,j,tmp;
    int lf_idx; //比開頭index的值還大的值的最小index
    int rg_idx; //比開頭index的值還小的值的最大index
    if(lf<rg) //1:第一筆鍵值為d[lf]
    {
        lf_idx=lf+1;
        rg_idx=rg;
        while(true) //排序
        {
            for(i=lf+1;i<=rg;i++)   //2:由左向右找出一個鍵值大於d[lf]者
            {
                if(d[i]>=d[lf])
                {
                    lf_idx=i;
                    break;
                }
                lf_idx++;
            }
            for(j=rg;j>=lf+1;j--)    //3:由右向左找出一個鍵值小於d[lf]者
            {
                if(d[j]<=d[lf])
                {
                    rg_idx=j;
                    break;
                }
                rg_idx--;
            }
```

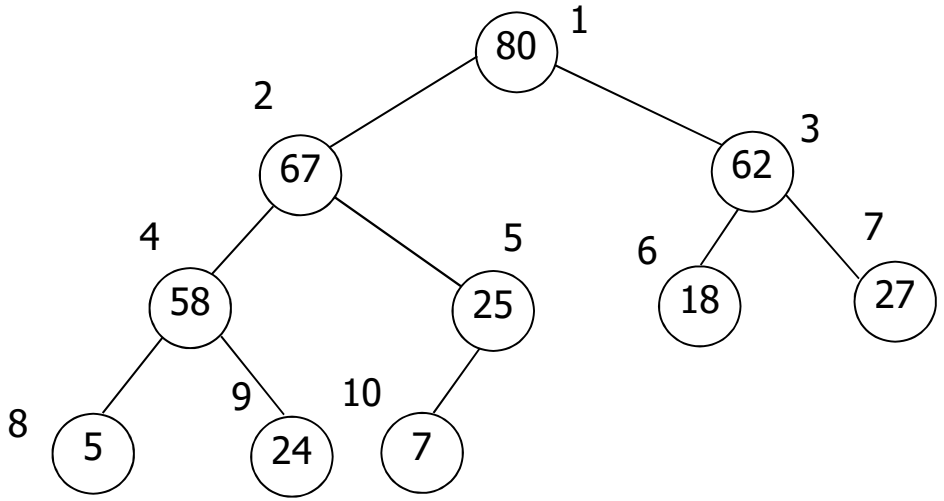
```
            if(lf_idx<rg_idx)          //4-1:若lf_idx<rg_idx
            {
            tmp = d[lf_idx];
            d[lf_idx] = d[rg_idx]; //則d[lf_idx]和d[rg_idx]互換
            d[rg_idx] = tmp;        //然後繼續排序
            }else{
            break;        //否則跳出排序過程
            }
            }

            //整理
            if(lf_idx>=rg_idx)                  //5-1:若lf_idx大於等於rg_idx
            {                                   //則將d[lf]和d[rg_idx]互換
            tmp = d[lf];
            d[lf] = d[rg_idx];
            d[rg_idx] = tmp;
            //5-2:並以rg_idx為基準點分成左右兩半
            Ch08Sorting.quickSort(d,size,lf,rg_idx-1);
            //以遞迴方式分別為左右兩半進行排序
            Ch08Sorting.quickSort(d,size,rg_idx+1,rg); //直至完成排序
            }
        }
}
```

作業07-04

- 請實作quickSort方法。並將其時間複雜度big-O寫出。

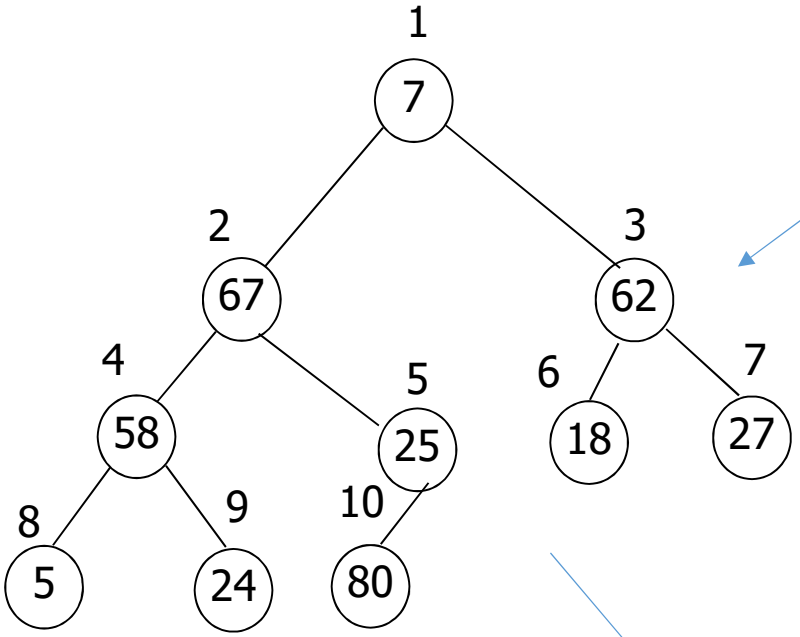
堆積排序 (Heap Sort)
將二元樹轉換成max heap

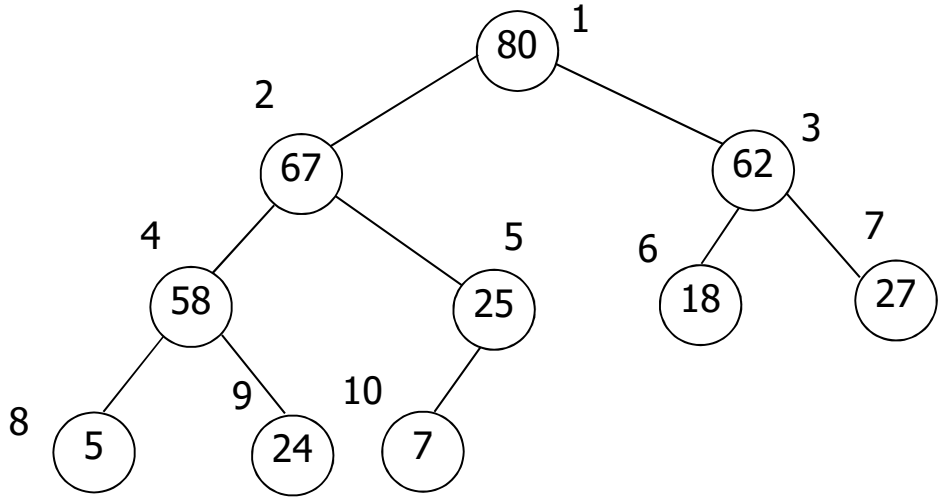




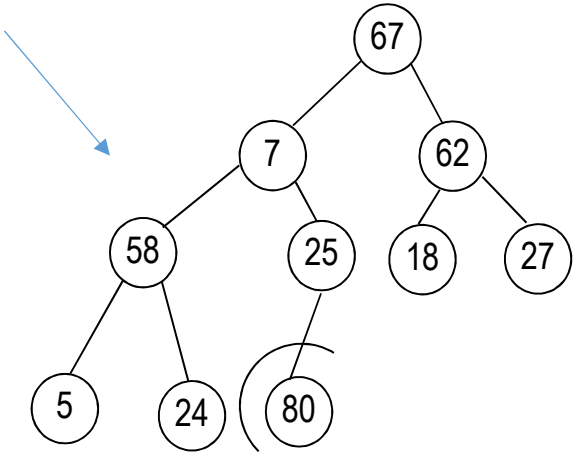
堆積排序
Pass

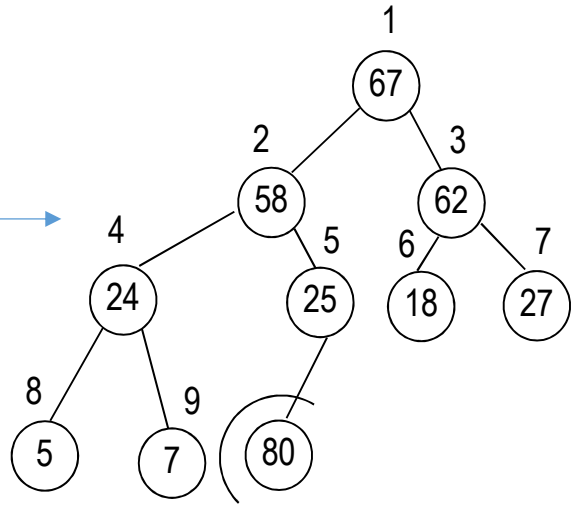
因此i=1時，原先堆積變成：





67與7對調...→





堆積排序
程式碼實作(1)

```
public static void heapSort(int data[] ,int size)
{
    int i,tmp;
    for(i=(size/2);i>0;i--)     //建立堆積樹節點
        adjust(data,i,size-1);
    for(i=size-2;i>0;i--)     //堆積排序
    {
        tmp=data[i+1];     //頭尾節點交換
        data[i+1]=data[1];
        data[1]=tmp;
        adjust(data,1,i);                //處理剩餘節點
    }
}
```

堆積排序
程式碼實作(2)

```
public static void adjust(int data[],int i,int size)
{
     int j,tmp,post;
     j=2*i;
     tmp=data[i];
     post=0;
     while(j<=size && post==0)
     {
          if(j<size)
          {
               if(data[j]<data[j+1])//找出最大節點
               j++;
          }
          if(tmp>=data[j])//若樹根較大，結束比較過程
               post=1;
          else
          {
               data[j/2]=data[j];//若樹根較小，則繼續比較
               j=2*j;
          }
     }
     data[j/2]=tmp;//指定樹根為父節點
}
```

作業07-05

- 請實作Heap Sort方法。並將其時間複雜度big-O寫出。