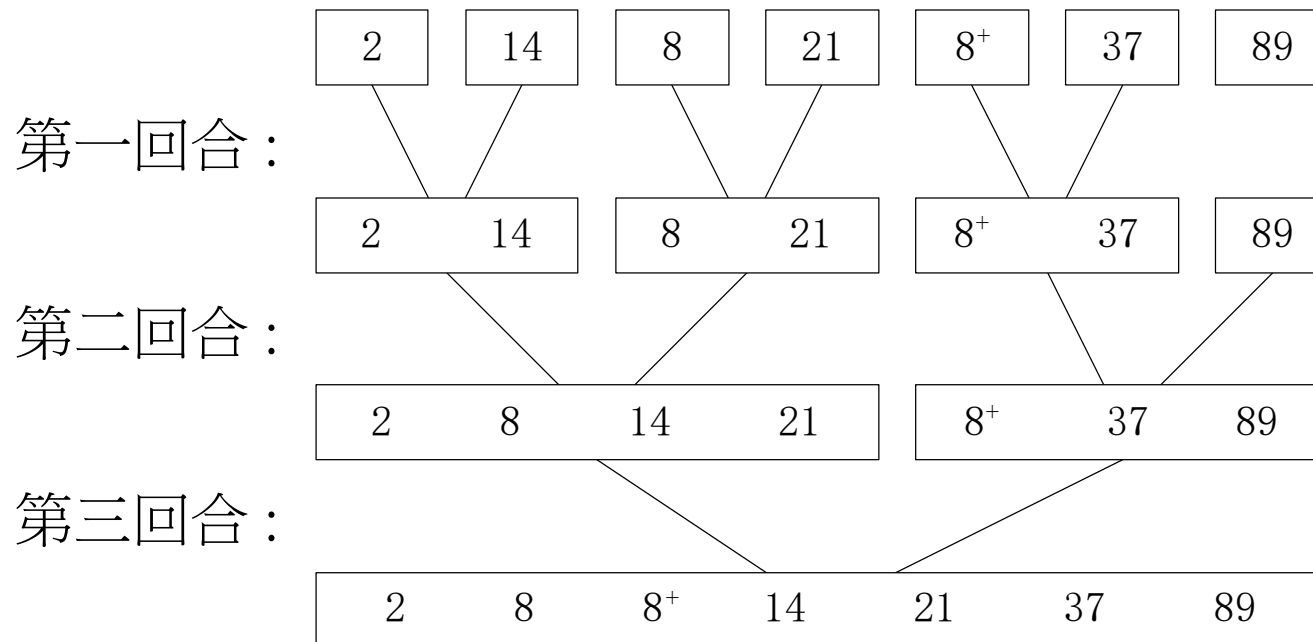


Chapter 7 Sorting(2)

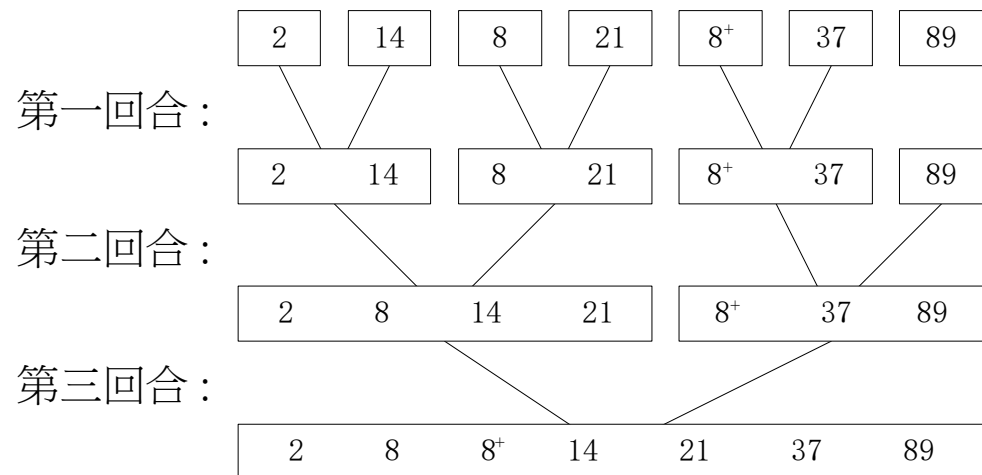
合併排序法(Merge Sort)

- 是一個典型的「分而治之」方法。
- 將N筆記錄依鍵值不遞減順序排序之方法為：
 1. 將N個長度為1的鍵值成對地合併成N/2個長度為2的鍵值組。
 2. 將N/2個長度為2的鍵值組成對地合併成N/4個長度為4的鍵值組。
 3. 將鍵值組成對地合併，直到合併成1組長度為N的鍵值組為止。
- 將鍵值 {2, 14, 8, 21, 8⁺, 37, 89} 按鍵值不遞減順序排序之合併排序法的三個回合如下：



MergeSort(int[] a, int n)

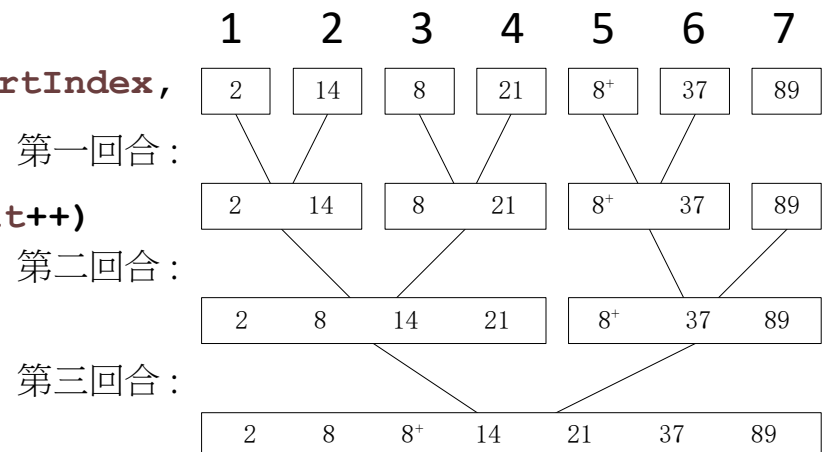
```
public void mergeSort(int[] a, int n)
{
    // Sort a [1:n] into nondecreasing order.
    int[] tempList = new int[n+1];
    // l is the length of the sublist currently being merged
    for(int l = 1; l < n; l = l*2) {
        // 上面的MergePass將初始陣列a，依大小為l的兩兩合併排序後變成tempList
        MergePass(a, tempList, n, l);
        l = l*2; // 將合併大小加大一倍
        // tempList為上一個pass的結果，所以在此次作為初始陣列
        MergePass(tempList, a, n, l);
    }
}
```



Merge pass

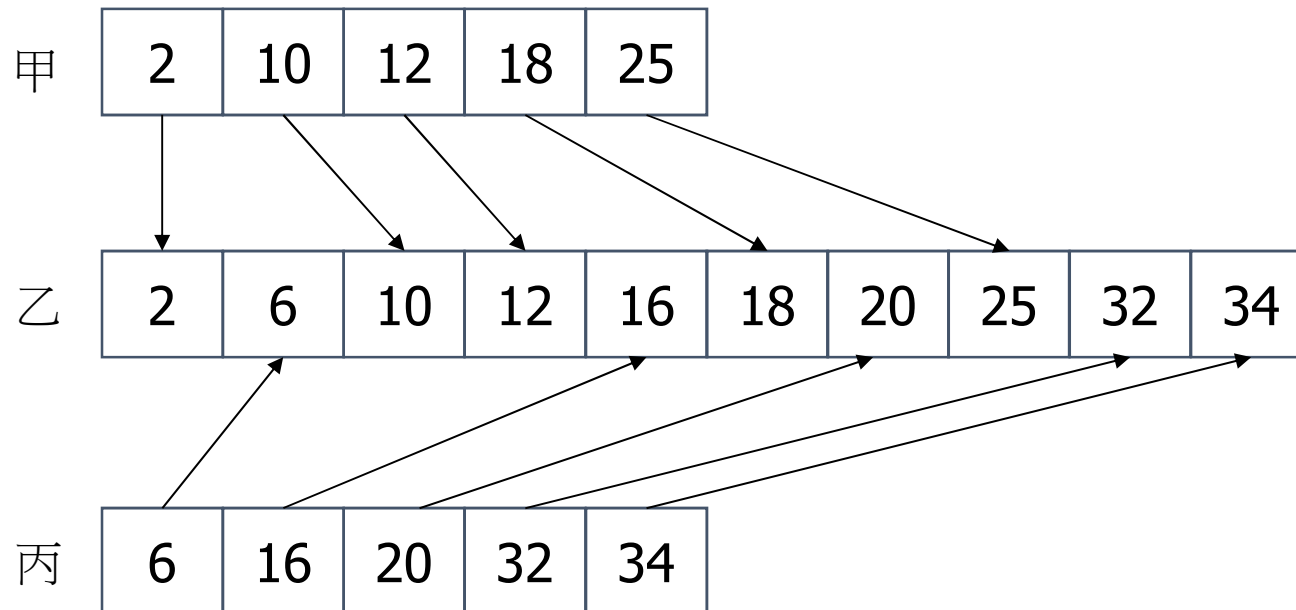
```
public void MergePass(int[] initList, int[] resultList, int n, int s)
{
    // Adjacent pairs of sublists of size s are merged from
    // initList to resultList. n is the number of records in initList.
    int i;
    // i is first position in first of the sublists being merged
    // enough elements for two sublists of length s
    for(i = 1; i <= n-2*s + 1; i += 2*s)
        Merge(initList, resultList, i, i+s-1, i+2*s-1);
    // merge remaining list of size < 2*s
    if((i + s - 1) < n)
        Merge(initList, resultList, i, i+s-1, n);
    else
        copy(initList, i, n, resultList, i);
}
```

```
public static void copy(int[] initList, int startIndex,
int endIndex, int[] mergedList, int iResult)
{
    for(int i=startIndex;i<=endIndex;i++,iResult++)
        mergedList[iResult]=initList[i];
}
```



Merging two sorted list

- 乃是將兩個或兩個以上已排序好的檔案，合併成一個大的已排序好的檔案。



Merging two sorted list

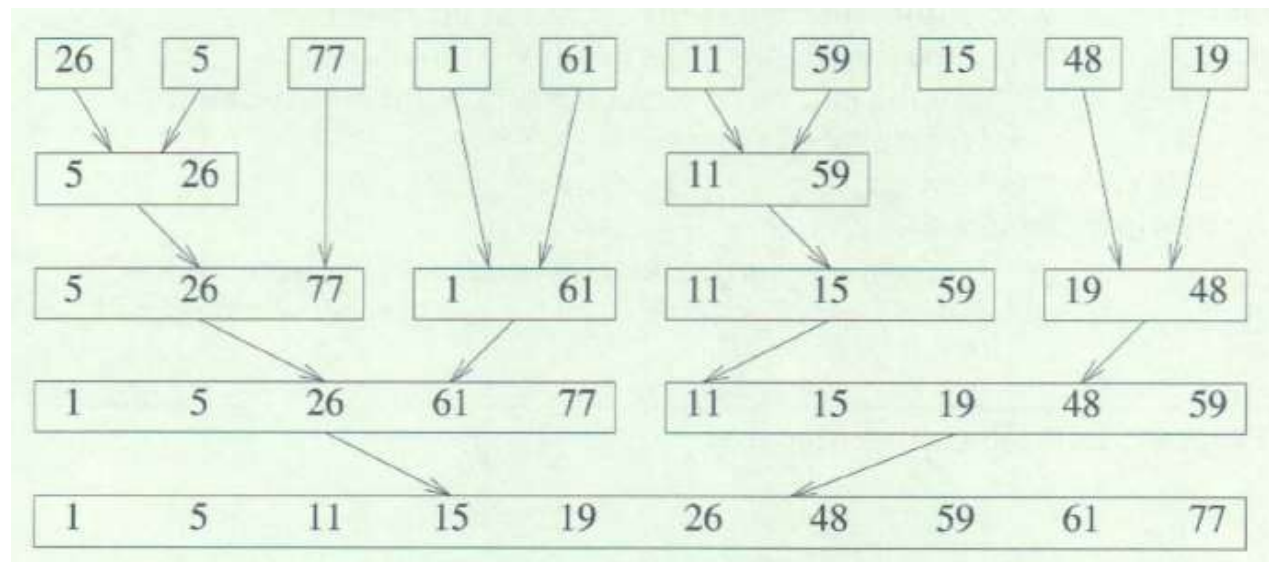
```
public void Merge(int[] initList, int[] mergedList, int l, int m, int
n)
{
    //initList [l:m] and initList [m+1:n] are sorted lists.
    //They are merged to obtain the sorted list mergedList[l :n].
    int i1,iResult,i2;
    for (i1 = l, iResult = l, i2 = m + 1; i1<= m && i2 <= n; iResult++)
        // i1, i2, and iResult are list positions
        if (initList[i1] <= initList[i2])
        {
            mergedList[iResult] = initList[i1];
            i1++;
        }
        else
        {
            mergedList[iResult] = initList[i2];
            i2++ ;
        }
    // copy remaining records of first list
    if(i1<=m)
        copy(initList, i1, m, mergedList, iResult);
    // copy remaining records of second list
    if(i2<=n)
        copy(initList, i2, n, mergedList, iResult);
}
```

進階作業07-01

- 請實作MergeSort的方法

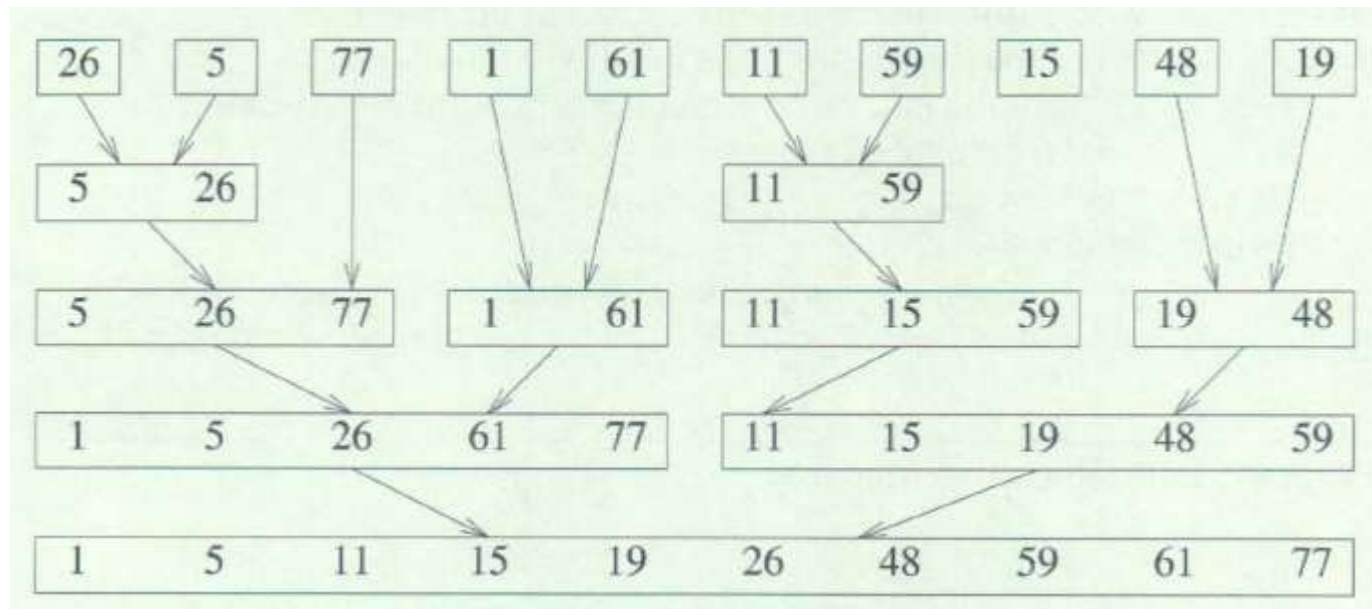
Recursive Merge Sort (1)

```
public static int rmergeSort(int[] a, int[] temp, int left, int right)
{
    // a[left:right] is to be sorted. Link[i] is initially 0 for all i.
    // rMergeSort returns the index of the first element in the sorted chain.
    if(left < right){
        int mid = (left + right)/2;
        rmergeSort(a, temp, left, mid)
        rmergeSort(a, temp, mid + 1, right)
        Merge(a, temp, int l, int m, int n)
        copy(temp, l, n, a, l)
    }
}
```



Recursive Merge Sort (2)

```
public static int rmergeSort(int[] a, int[] link, int left, int right)
{
    // a[left:right] is to be sorted. link[i] is initially 0 for all i.
    // rMergeSort returns the index of the first element in the sorted chain.
    if(left >= right)
        return left;
    int mid = (left + right)/2;
    return listMerge(a, link,
        rmergeSort(a, link, left, mid), //sort left half
        rmergeSort(a, link, mid + 1, right)); //sort right half
}
```



rmergeSort(int[] a, int[] link, 1, 10)

listMerge(a,link, rmergeSort(a, link, 1, 5), rmergeSort(a, link, 6, 10));

listMerge(a,link,
rmergeSort(a, link, 1, 3), rmergeSort(a, link, 4, 5));

listMerge(a,link,
rmergeSort(a, link, 6, 8), rmergeSort(a, link, 9, 10));

listMerge(a,link,
rmergeSort(a, link, 1, 2),
rmergeSort(a, link, 3, 3));

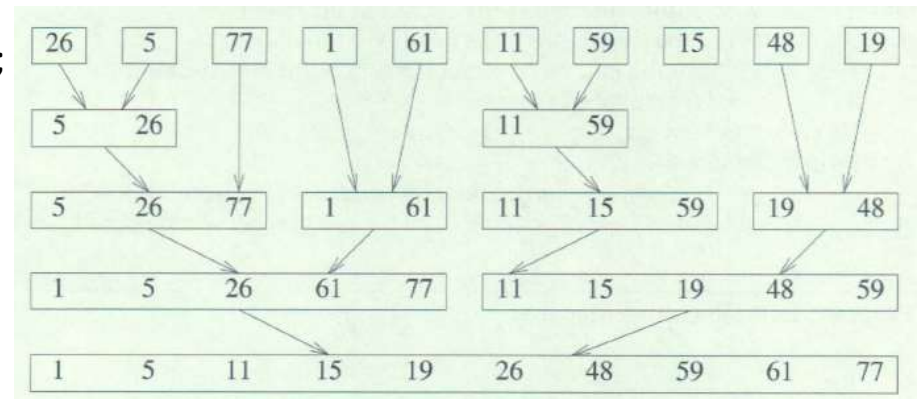
listMerge(a,link,
rmergeSort(a, link, 4, 4),
rmergeSort(a, link, 5, 5));

listMerge(a,link,
rmergeSort(a, link, 6, 7),
rmergeSort(a, link, 8, 8));

listMerge(a,link,
rmergeSort(a, link, 9, 9),
rmergeSort(a, link, 10, 10));

listMerge(a,link,
rmergeSort(a, link, 1, 1),
rmergeSort(a, link, 2, 2));

listMerge(a,link,
rmergeSort(a, link, 6, 6),
rmergeSort(a, link, 7, 7));

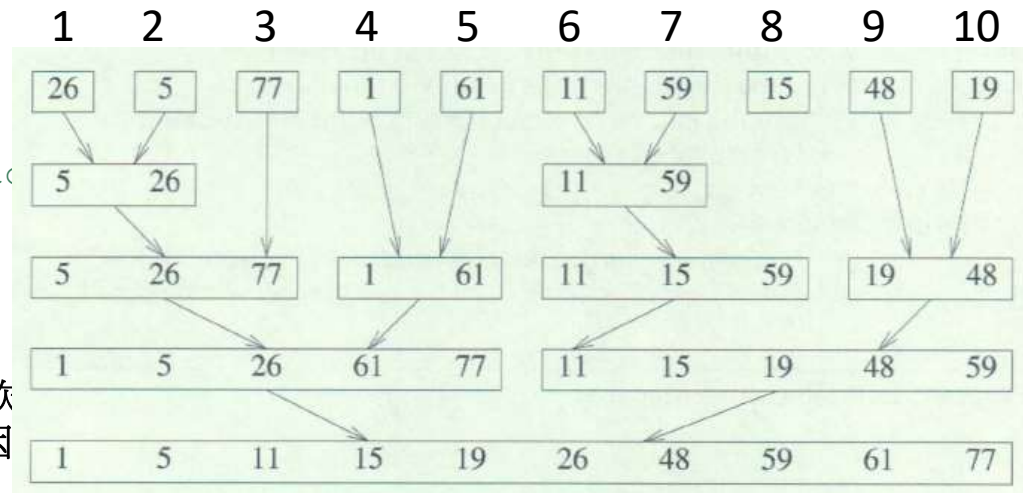


	0	1	2	3	4	5	6	7	8	9	10
link	0	0	0	0	0	0	0	0	0	0	0

```

public static int listMerge(int[] a, int[] link, int start1, int start2)
{
    // The sorted chains beginning at start1 and start2, respectively, are merged.
    // link[0] is used as a temporary header. Return start of merged chain.
    int iResult=0; //last record of result chain 下一個存在link陣列的index，每次
    listMerge中，最小值一定存在link[0]，所以iResult一開始為0
    int i1,i2;
    for (i1 = start1, i2 = start2; i1>0 && i2>0;)
        if (a[i1] <= a[i2])
        {
            link[iResult] = i1 ; //i1為下一個最小值的index，將此存在link[iResult]
            iResult = i1; //所以下一個最小值的index要存在link陣列的index為i1
            i1 = link[i1]; //下一個要比較的值index是i1修改為link[i1]，也就是
        }
        else
        {
            link[iResult] = i2;
            iResult = i2;
            i2 = link[i2];
        }
    // attach remaining records to the result chain
    if(i1 == 0)
        link[iResult] = i2;
    else
        link[iResult] = i1;
    return link[0]; //此變數儲存此次
    每次listMerge後最小值的index，因
}

```



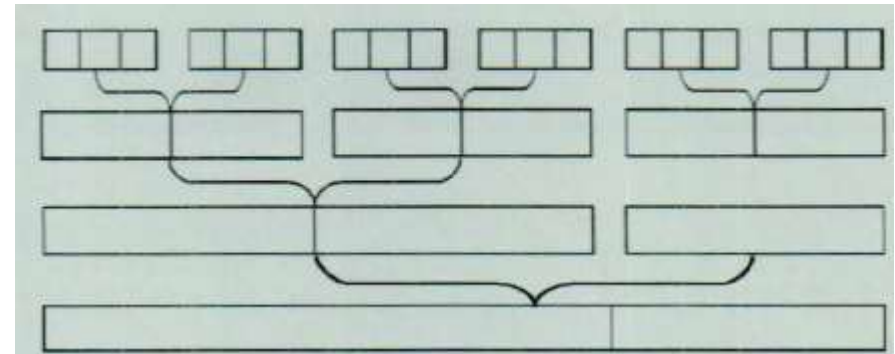
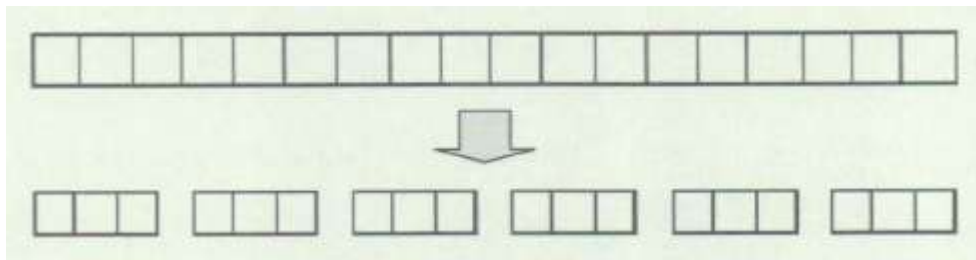
```
public static void printRMergeSort(int[] a, int[] link){  
    int curIndex = 0;  
    while(link[curIndex] != 0) {  
        System.out.print(a[curIndex]+" ");  
        curIndex = link[curIndex];  
    }  
    System.out.print("\n");  
}
```

進階作業07-02

- 請實作MergeSort 遞迴的方法

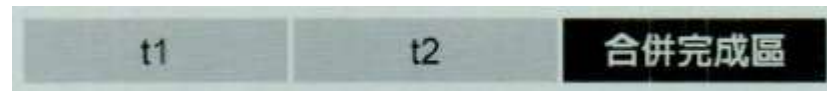
外部排序法- (External sorting)(1)

- 當我們所要排序的資料量太多或檔案太大，無法直接在記憶體內排序，而需依賴外部儲存裝置時，我們就會使用到外部排序法。
- 直接合併排序法(**Direct Merge Sort**) 是外部儲存裝置最常用的排序方法。它可以分為兩個步驟：
 - 步驟1: 將欲排序的檔案分為幾個可以載入記憶體空間大小的小檔案，再使用內部排序法將各檔案內的資料排序。
 - 步驟2: 將第一步驟所建立的小檔案每二個合併成一個檔案。兩兩合併後，把所有檔案合併成一個檔案後就可以完成排序了。
 - 例如: 我們把一個檔案分成6 個小檔案，小檔案都完成排序後，兩兩合併成一個較大的檔案，最後再合併成一個檔案即可完成。



外部排序法- (External sorting)

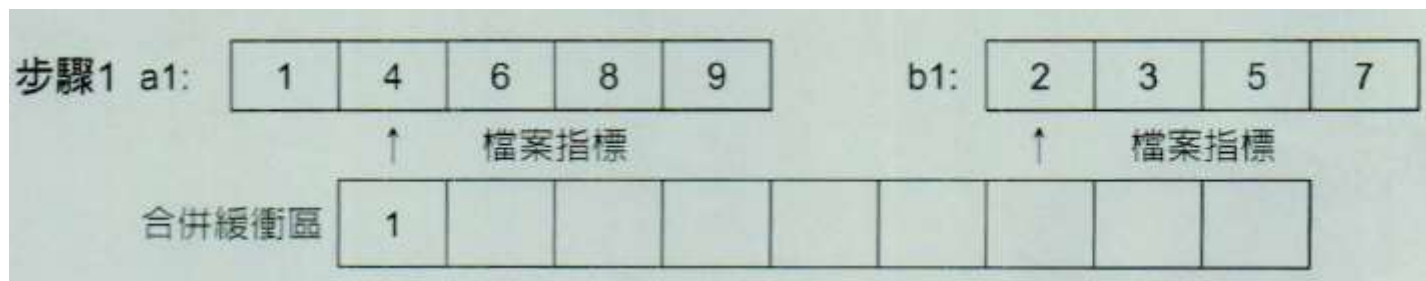
- 對檔案test.txt 進行排序，而test. txt 裡包含的1500筆資料，但記憶體最多一次可處理300 筆資料。
 - 步驟1：將test.txt分成5個檔案t1- t5每個檔案300 筆。
 - 步驟2：以任一內部排序法對t1-t5 進行排序。
 - 步驟3：進行檔案t1、t2合併，將記憶體分成三部份，每部份可存放100 筆資料，
 - 先把t1及t2的前100筆資料放到記憶體，排序後放到合併完成區，等緩衝區滿了之後寫入磁碟。



- 步驟4: 重複步驟3直到完成排序為止。

直接合併排序法合併的方法

- 假設我們有兩個完成排序的檔案要合併，排序由小到大：
a1 : 1, 4, 6, 8, 9
b1 : 2, 3, 5, 7
- 首先在兩個檔案中分別讀出一個元素進行比較，比較後將較小的檔案放入合併緩衝區內。1 跟 2 比較後將較小的 1 放入緩衝區，a1 的檔案指標往後一個元素。



- 依此類推，等到緩衝區的資料滿了就進行寫入檔案的動作; a1或b1 的檔案指標到了最後一筆就讀取下面的資料進來進行比較及排序。

SORTING ON SEVERAL KEYS

基數排序法(Radix Sort)

- 又稱多鍵排序(Multi-Key Sort)、箱子排序法(Bucket Sort)
- 最有效鍵優先(Most Significant Digit First : MSD)
 1. 最有效鍵越先排序。
 2. 採用「分配」、「排序」、「收集」等三個步驟進行。
- 最無效鍵優先(Least Significant Digit First : LSD)
 1. 最無效鍵越先排序。
 2. 採用「分配」和「收集」兩個步驟。





K^1 [Suits]: ♣ < ♦ < ♥ < ♠
 K^2 [Face values]: 2 < 3 < 4 ... < 10 < J < Q < K < A
2♣, ..., A♣, ..., 2♦, ..., A♦

最有效鍵為花色，數字為次有效鍵





基數排序法(Radix Sort) MSD法排序(1)

- 將撲克牌一花色優先，數字次之排序
- 利用MSD法排序{A1，C3，A10，D6，B12，C7，A3，B8，D9，C2，B13，A5，C11等13張牌的過程為：

步驟 1. 準備A、B、C、D四個箱子。





A()	B()	C()	D()

步驟 2. 依次讀入撲克牌，並依花色置入箱中，讀牌依序為A1，C3，…，C11。

A()	B()	C()	D()
A1	B12	C3	D6
A10	B8	C7	D9
A3	B13	C2	
A5		C11	

基數排序法(Radix Sort) MSD法排序(2)

步驟 3. A, B, C, D四個箱子內之牌，個別獨立用插入法排序。

A()	B()	C()	D()
A1	B8	C2	D6
A3	B12	C3	D9
A5	B13	C7	
A10		C11	

步驟 4. 依A, B, C, D箱子的順序收集。

A1, A3, A5, A10, B8, B12, B13, C2, C3, C7, C11, D6, D9

基數排序法(Radix Sort) LSD法排序(1)

- 利用LSD法排序{A1，C3，A10，D6，B12，C7，A3，B8，D9，C2，B13，A5，C11}等13張牌的過程為：

步驟 1. 準備好13個箱子，編號為 $1, 2, \dots, 13$ 。

[illegible]

步驟 2. 讀入撲克牌A1，C3， \cdots ，C11，並置入相同點數的箱中。

1	2	3	4	5	6	7	8	9	10	11	12	13
A1	C2	C3 A3		A5	D6	C7	B8	D9	A10	C11	B12	B13





基數排序法(Radix Sort) LSD法排序(2)

步驟 3. 按1、2、 \dots 、13箱子的順序收集

A1，C2，C3，A3，A5，D6，C7，B8，D9，A10，C11，B12，B13





請注意！目前已按點數由小到大排序好了。

步驟 4. 準備A，B，C，D四個箱子。

A()	B()	C()	D()

基數排序法(Radix Sort) LSD法排序(3)

步驟 5. 依次讀入步驟3之結果A1，C2，C3，...，B13，並置入相同花色的箱中

A()	B()	C()	D()
A1	B8	C2	D6
A3	B12	C3	D9
A5	B13	C7	
A10		C11	

步驟 6. 依次從A、B、C、D箱子收集後即完成排序。

A1，A3，A5，A10，B8，B12，B13，C2，C3，C7，C11，D6，D9

練習

- 利用MSD及LSD方式排列正整數鍵值
{31, 58, 63, 87, 58, 16, 66, 34, 21, 89, 84, 32, 11}, 請將過程寫出

作業07-06

- 請寫一個function，傳入一個參數a為整數陣列，a存20個2位數整數，該function使用MSD將a排序

作業07-07

- 請寫一個function，傳入一個參數a為整數陣列，a存20個2位數整數，該function使用LSD將a排序