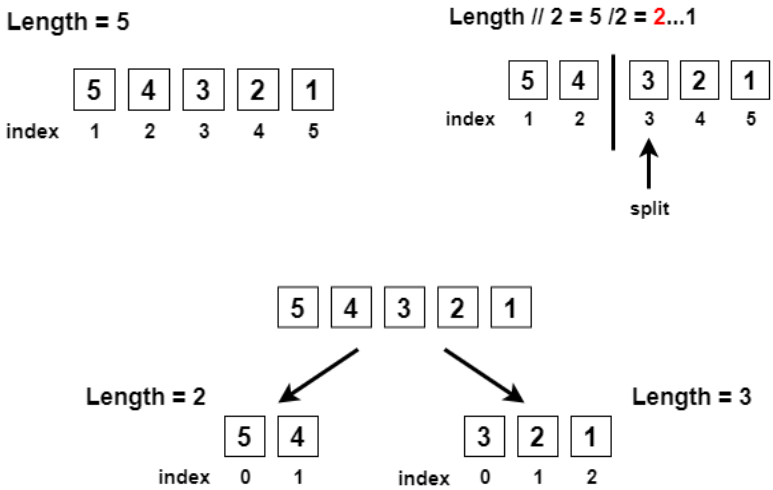
MergeSort 流程圖、學習歷程與文字說明

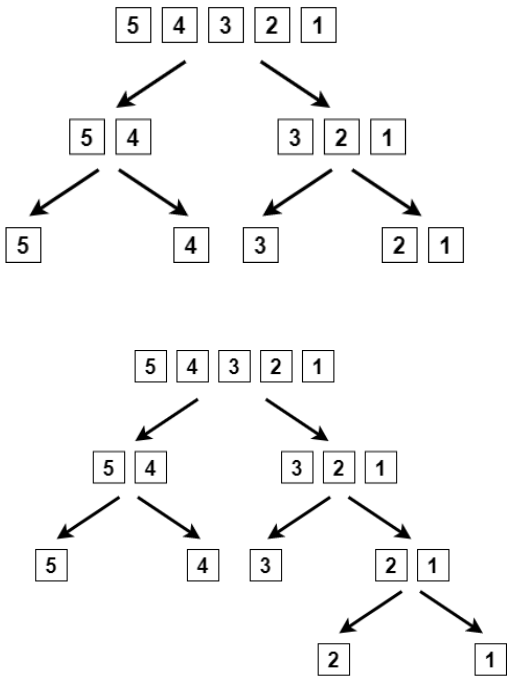
一、流程圖

1. 分割

A. 將一組含有 N 個項目的數列，以「N/2」方式分割其長度，所以數列會分割成兩組，以「N/2」的商當作分割點的索引，左邊的數列範圍為原數列的起始索引到分割點的前一項索引；右邊的數列範圍為分割點的索引到原數列的尾端索引。



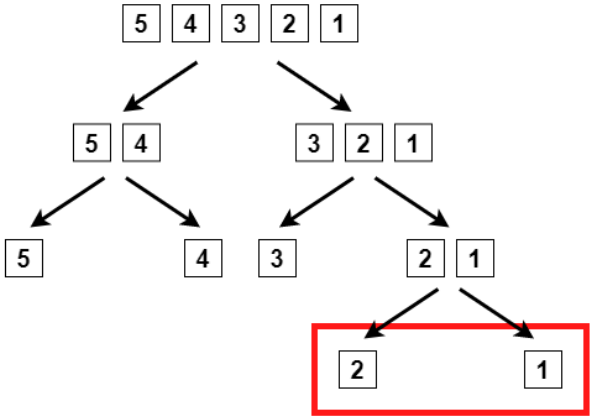
B. 接著，左右兩組分別繼續進行分割，直到數量的長度為 1 為止。



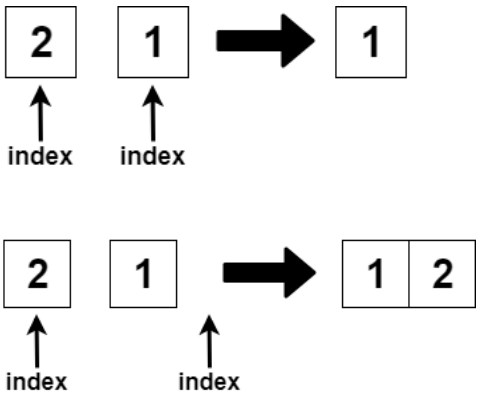
2. 合併與排序

將分割後長度「1」的數列成對地合併並進行排序，直到合併成一組長度與最初數列一樣的數列，而該數列的數值則按照大小順序排序。

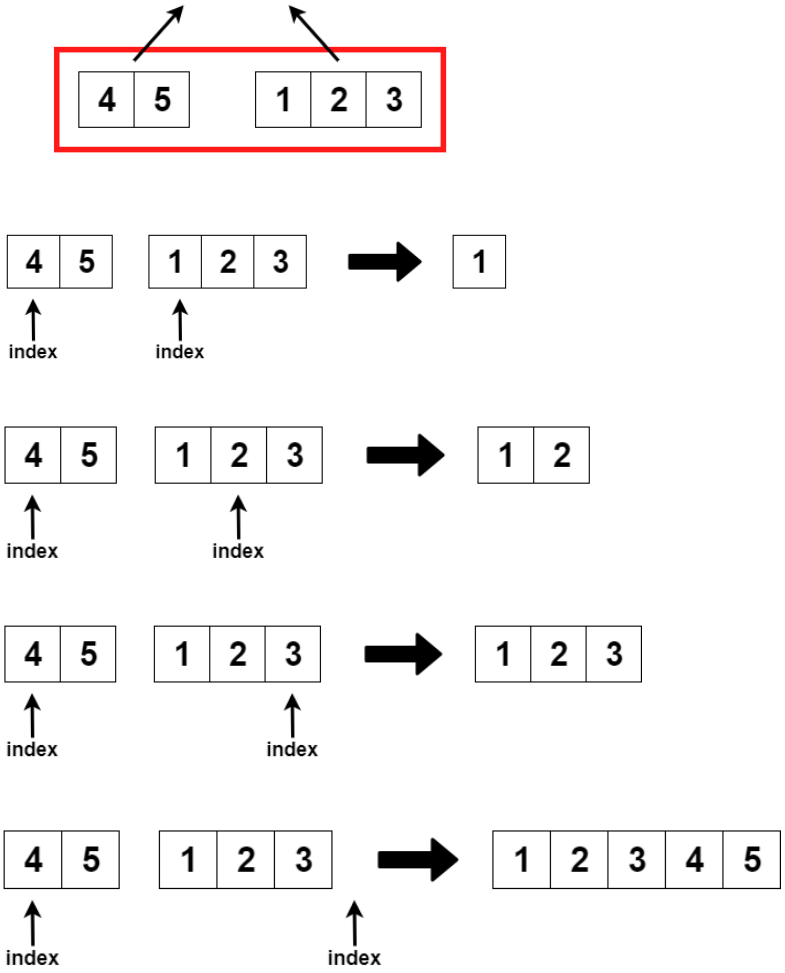
A. 見樹狀圖，最初在數列合併時會先從樹狀圖最底層長度「1」的數列開始合併。



B. 而每次在合併時會同時進行排序，首先，會有兩個指標指向欲合併的兩個數列起始數值；其次，兩個指標指向的數值進行比較，較小的數值將會作為合併後數列的起始值；接著，前一次有較小數值的數列的指標會指向該數列的下一個數值與另一個數列的指標所指向的數值繼續比較，較小的數值將會作為合併後數列的下一個數值。以此類推，直到某一個數列的指標超過數列範圍為止；最後，指標未指完的數列的數值再依序作為合併後數列的數值。



C. 上述過程為一次的合併與排序。完全分割的數列會不斷進行上述的合併與排序，直到合併成一組長度與最初數列一樣的數列。



二、學習歷程

繳交的作業為原創的程式碼，無任何抄襲。參考的程式碼僅作為延伸學習。對照學習歷程中參考程式碼和原創程式碼兩部分可以看出兩者的思考邏輯並不相同。

1. 失敗程式碼

嘗試自行撰寫 MergeSort 程式，但該程式僅能在輸入陣列長度為 2 的 n 次方的情況下成功排序。

原理是先將輸入的陣列裡的每個數值變成一個個子陣列，完成完全分割的步驟。接著再將兩兩子陣列依序合併並排序，直到合併排序成一個與原先輸入陣列長度相同的陣列。但此方法的缺陷在於選取兩兩子陣列合併時，所有子陣列的個數都須為 2 的 n 次方個，否則無法完全配對。

```
def Convert(data):
    L=[]
    for i in data:
        e=[i]
        L.append(e)
    return L

def SortMerge(L):
    begin = 0
    end = len(L)
    D=[]
    for i in range(begin,end,2):
        d1,d2=L[i],L[i+1]
        D.append(Merge(d1,d2))

    if len(D)==1:
        return D[0]
    else:
        return SortMerge(D)

def Merge(d1,d2):
    D=[]
    for ii in d1:
        pivot=ii
        for jj in d2:
            if jj < pivot:
                D.append(jj)
        for ww in D:
            if ww in d2:
                d2.remove(ww)
        D.append(pivot)
```

```
data=[16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1]
     ...: SortMerge(Convert(data))
Out[156]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 13, 14, 15, 16]

In [157]: data=[4,3,2,1]
     ...: SortMerge(Convert(data))
Out[157]: [1, 2, 3, 4]

In [158]: data=[8,7,6,5,4,3,2,1]
     ...: SortMerge(Convert(data))
Out[158]: [1, 2, 3, 4, 5, 6, 7, 8]
```

2. 原創程式碼文字說明

```python
class Solution:
    def Check(self,data):
        import numpy as np
        t=np.array(data)
        if len(t.shape)==1:
            return True
        elif len(t.shape)==2:
            return False

    def Sup(self,data):
        n=len(data)
        k=0
        base=1
        while base < n:
            base=base*2
            k+=1
        num_dummy=2**k-n
        if num_dummy!=0:
            self.dummy=max(data)+1
            #dummy=9999
            data_dummy=data+[self.dummy]*num_dummy
            return data_dummy
        else:
            self.dummy=None
            return data

    def Convert(self,data):
        L=[]
        for i in data:
            e=[i]
            L.append(e)
        return L

    def Merge(self,d1,d2):
        D=[]
        i=0
        j=0
        while i<len(d1) and j<len(d2):
            if d1[i] < d2[j]:
                D.append(d1[i])
                i+=1
            else:
                D.append(d2[j])
                j+=1
        while i<len(d1):
            D.append(d1[i])
            i+=1
        while j<len(d2):
            D.append(d2[j])
            j+=1
        return D

    def sortMerge(self,data):
        if self.Check(data)==True:
            if len(data)==0:
                return 'no element in the list'
            if len(data)==1:
                return data
            data_dummy = self.Sup(data)
            data = self.Convert(data_dummy)
        if self.Check(data)==False:
            if len(data)==1:
                data_final=data[0]
                while self.dummy in data_final:
                    data_final.remove(self.dummy)
                return data_final
        begin = 0
        end = len(data)
        D=[]
        for i in range(begin,end,2):
            d1,d2=data[i],data[i+1]
            D.append(self.Merge(d1,d2))
        return self.sortMerge(D)
```

```
In [297]: data=[6,5,4,3,2,1]
     ...: Solution().sortMerge(data)
Out[297]: [1, 2, 3, 4, 5, 6]

In [298]: data=[1]
     ...: Solution().sortMerge(data)
Out[298]: [1]

In [299]: data=[]
     ...: Solution().sortMerge(data)
Out[299]: 'no element in the list'

In [300]: data=[-1,-2,-3]
     ...: Solution().sortMerge(data)
Out[300]: [-3, -2, -1]
```

邏輯與學習歷程中失敗的程式碼相同。原理是先將輸入的陣列裡的每個數值變成一個個子陣列，完成完全分割的步驟。接著再將兩兩子陣列依序合併並排序，直到合併排序成一個與原先輸入陣列長度相同的陣列。但改進了所有子陣列的個數必須要是 2 的 n 次方個的問題。解決的方法是直接將輸入的數列裡的數值個數補成 2 的 n 次方個，等最後完成排序後再將填補的數值刪除。其中填補的數值為所輸入的數列中最大的數值加上一。

以下為程式完成 Merge Sort 所需的步驟:

1. 第 54~58 行。如果輸入的數列是非遞迴的程式，則會檢查所輸入的數列裡的個數。其中會用 Check 函式判斷輸入的數列是否為遞迴的程式，依據為該數列裡是否存在子數列。如果是遞迴的程式，數列存在子數列，該數列的維度 (shape) 會為 2；如果是非遞迴的程式，則不存在子數列，數列的維度 (shape) 則為 1。檢查所輸入的數列裡的個數方面，個數為 0 時輸出”no element in the list”；個數為 1 時則直接輸出陣列。
2. 第 59 行。利用 Sup 函式將輸入的數列裡的數值個數補成 2 的 n 次方個。其中填補的數值為所輸入的數列中最大的數值加上一。
3. 第 60 行。將所輸入的數列運用 Convert 函數將數列中的每個數值都變成一個子數列，每個子數列長度為 1，例如將 [3,2,1,4] 轉換成 [[3],[2],[1],[4]]。
4. 第 67 到 72 行。將數列裡的子數列兩兩一組透過 Merge 函式做合併排序，合併排序的原理與流程圖相同，再將所有合併排序過的子數列 append 到一個空的母數列中。例如將[[3],[2],[1],[4]]的兩兩子數列做合併排序並 append 到一個空的母數列後會變成 [[2,3],[1,4]]。其中因為陣列的子數列個數永遠一直都保持在 2 的 n 次方個，所以每個子數列都能兩兩湊對。
5. 第 73 行。將子數列做過一次合併排序過的母數列用遞迴的方式繼續做合併排序，其中遞迴的程式都不會經過第 54~60 行。
6. 第 61~66 行。經不斷遞迴進行合併排序後，子數列會被合併成一個子數列，而該子數列中已包含最初輸入的數列裡所有的數值並完成排序。此時會將該子數列取出，並把最初填補的值從該數列中刪除，然後輸出該數列，完成 Merge Sort 所有步驟。

3. 延伸學習的程式碼

參考 <資料結構使用 Python> 這本書中 MergeSort 的程式碼。該方法是先將輸入的數列，不斷對半分割成兩個數列，直到將每個數列分割成長度為一的數列為止。接著再將兩兩數列依序合併並排序，直到合併排序成一個與原先輸入陣列長度相同的數列。

```
1 def MergeSort(data):
2     if len(data) > 1:
3         split = len(data)//2
4         left = data[0:split]
5         right = data[split:]
6         MergeSort(left)
7         MergeSort(right)
8         l=r=index=0
9         while l<len(left) and r<len(right):
10            if left[l] < right[r]:
11                data[index] = left[l]
12                index+=1
13                l+=1
14            else:
15                data[index] = right[r]
16                index+=1
17                r+=1
18        if l < len(left):
19            while l < len(left):
20                data[index]=left[l]
21                index+=1
22                l+=1
23        elif r <len(right):
24            while r < len(right):
25                data[index]=right[r]
26                index+=1
27                r+=1
```

該方法啟發我的是，在第 6、7 行中，將一開始切成的左右兩個數列做遞迴，但在第 8 行後直接拿這兩個數列做合並排序。但依照 MergeSort 的原理，應該是要把數列切成長度為一的數列為止才能進行合併排序，但從程式乍看下會感覺輸入的數列切成兩個數列後就直接把這兩個數列進行合併排序，而這兩列內都還沒有排序過。但事實上每個欲合併排序的數列裡的數值都已經排序過了。因為依照程式的邏輯，愈前面的程式碼會愈先被執行，因此在第 8 行開始合併排序前，已經在第 6、7 行跑過遞迴，而在遞迴的程式中在第 8 行開始合併排序前，也已經在第 6、7 行跑過遞迴，因此在一層層遞迴執行下，其實最先執行的程式其實是最後一層遞迴程式，兩兩數列長度為一，接著在最後一層遞迴程式的第 8 行開時進行合併排序。接著再執行倒數第二層遞迴程式，而欲合併的兩兩數列裡的數值已經在最後一層遞迴程式中排序過了。因此該程式其實是先將數列分割到長度唯一的數列後再依序做合併排序。

```
In [163]: data=[5,6,3,7,9]
     ...: MergeSort(data)
Out[163]: [3, 5, 6, 7, 9]

In [164]: data=[1,2,3,4,5]
     ...: MergeSort(data)
Out[164]: [1, 2, 3, 4, 5]

In [165]: data=[5,4,3,2,1]
     ...: MergeSort(data)
Out[165]: [1, 2, 3, 4, 5]
```

4. 參考資料

<使用資料結構 Python，李淑馨著，深石數位科技> 第九章 9.3.2 節 第 9-24 頁