合并排序 Merge-sort

分治法

很多算法在结构上是递归的：为了解决一个给定的问题，算法要一次或多次地递归调用其自身来解决相关的子问题。分治法在每一次递归上都有三个步骤：

1.分解（Divide）：将原问题分解成一系列子问题.

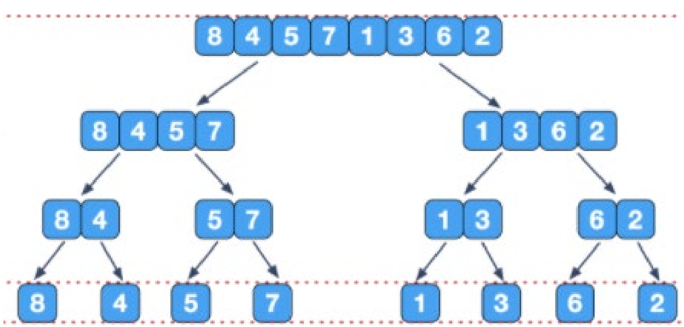
2. 解决（Conquer）：递归地解各个子问题。若子问题足够小，则直接求解.

3. 合并 （Combine）：将子问题的结果合并成原问题的解。

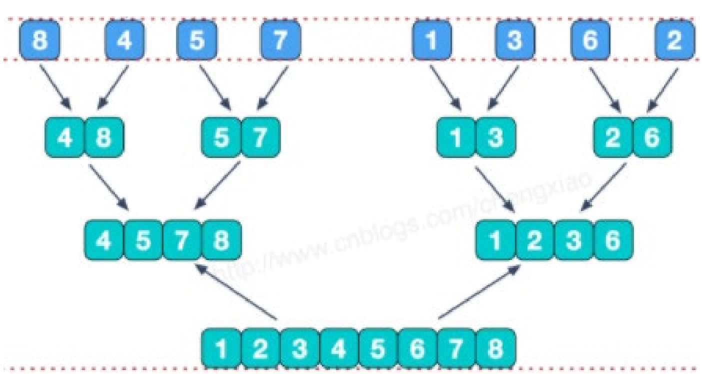
合并排序采用的就是一个非常典型的分治法思路，若表示成分治法思路：

1. 分解：将 n 个元素分成各含 $\frac{n}{2}$ 个元素的子序列。

2. 解决：用合并排序法对两个子序列递归地排序。也就是说，子序列不断产生长度为本身一半的又一个子序列，直到产生的子序列长度为 1。长度为 1 的序列被认为是已排好序的。这个过程可以用下图表示：



3. 合并：合并两个已排序的子序列以得到排序结果。在第二步时，我们生成了很多的子序列，我们依次对它们进行合并：



至此，一个完整的合并排序就完成了。

具体的代码实现可以访问 https://www.geeksforgeeks.org/merge-sort/， 里面已经足够清晰的解释了实现过程。但有一部分的代码我想在这里重新花笔墨解释一遍来加深印象。

```
line 1    void mergeSort(int arr[],int l,int r){
line 2        if(l>=r){
line 3            return;//returns recursively
line 4        }
line 5        int m = (l+r-1)/2;
line 6        mergeSort(arr,l,m);
line 7        mergeSort(arr,m+1,r);
line 8        merge(arr,l,m,r);
line 9    }
```

这是一个递归分解，合并的过程。首先，l 和 r 分别表示 array 的左右侧 index。mergeSort 函数仅有当 l >=r 时才结束递归，开始依次向上处理刚才创造出来的一堆递归。当 l >=r 时,程序回到了上一次递归开始的地方，也就是第 6 行。于是程序往下走，来到了第 7 行。第 7 行的 mergeSort 执行一样的操作，程序会把 arr 分割成长度为 1 的 array。然后就是最后的 merge 函数，当它处理完毕后，就能让程序回到上一个递归中，同样是从第 6 行跳出，开始往下执行。

语言描述有时是苍白的，复习时还是建议在脑海里重新想象一遍这个过程。

分治法分析

当一个算法中含有对其自身的递归调用时，其运行时间可以用一个递归方程来表示。

$$T(n)=\begin{cases}\Theta(1) & \text{如果 } n\leqslant c\\ aT(n/b)+D(n)+C(n) & \text{否则}\end{cases}$$

如果问题的规模足够小，则我们可以在 O(1)内得到结果。

合并排序算法的分析

首先，我把合并排序的主要步骤再次罗列一遍：

1. 分解：将 n 个元素分成各含 $\frac{n}{2}$ 个元素的子序列。

2. 解决：用合并排序法对两个子序列递归地排序。

3. 合并：合并两个已排序的子序列以得到排序结果。

接着我们分析最坏情况下，合并排序运算需要花费的时间：

1. 分解：常量时间即可完成，因为表示为 O(1).

2. 解决：递归地解两个规模为 n/2 的子问题，时间为 2 T(n/2)。注意，T(n/2) = O(n lg n)。

3. 合并：很容易理解，合并过程中会将两个 array 中的元素依次拿出比较后合并，所以花费的总时间是 O(n)

综上所述，我们可以得到合并排序算法的运行时间（用递归方程来表示）：

$$T(n) = \begin{cases} \Theta(1) & \text{如果 } n = 1 \\ 2T(n/2) + \Theta(n) & \text{如果 } n > 1 \end{cases}$$

其中 2T(n/2) + O(n)可以表示为 2O(n lg n) + O(n)，即 O(n lg n).