```
                if (arr[mid] >= num) {
                        res = mid;
                        high = mid - 1;
                } else {
                        low = mid + 1;
                }
        }
        return res;
}
```

计算数组的小和

【题目】

数组小和的定义如下：

例如，数组 s=[1,3,5,2,4,6]，在 s[0]的左边小于或等于 s[0]的数的和为 0，在 s[1]的左边小于或等于 s[1]的数的和为 1，在 s[2]的左边小于或等于 s[2]的数的和为 1+3=4，在 s[3]的左边小于或等于 s[3]的数的和为 1，在 s[4]的左边小于或等于 s[4]的数的和为 1+3+2=6，在 s[5]的左边小于或等于 s[5]的数的和为 1+3+5+2+4=15，所以 s 的小和为 0+1+4+1+6+15=27。

给定一个数组 s，实现函数返回 s 的小和。

【难度】

校 ★★★☆

【解答】

用时间复杂度为 $O(N^2)$的方法比较简单，按照题目例子描述的求小和的方法求解即可，本书不再详述。下面介绍一种时间复杂度为 $O(N\log N)$、额外空间复杂度为 $O(N)$的方法，这是一种在归并排序的过程中，利用组间在进行合并时产生小和的过程。

1．假设左组为 l[]，右组为 r[]，左右两个组的组内都已经有序，现在要利用外排序合并成一个大组，并假设当前外排序是 l[i]与 r[j]在进行比较。

2．如果 l[i]<=r[j]，那么产生小和。假设从 r[j]往右一直到 r[]结束，元素的个数为 m，那么产生的小和为 l[i]*m。

3．如果 l[i]>r[j]，不产生任何小和。

4．整个归并排序的过程该怎么进行就怎么进行，排序过程没有任何变化，只是利用步

骤 1~步骤 3，也就是在组间合并的过程中累加所有产生的小和，总共的累加和就是结果。

还是以题目的例子来说明计算过程。

1．归并排序的过程中会进行拆组再合并的过程。[1,3,5,2,4,6]拆分成左组[1,3,5]和右组[2,4,6]，[1,3,5]再拆分成[1,3]和[5]，[2,4,6]再拆分成[2,4]和[6]，[1,3]再拆分成[1]和[3]，[2,4]再拆分成[2]和[4]，如图 8-1 所示。

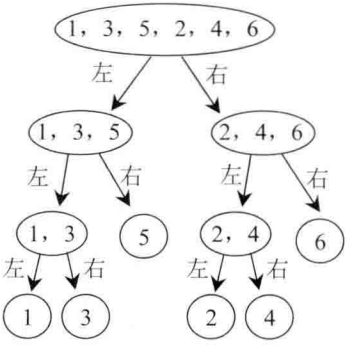

图 8-1

2．[1]与[3]合并。1 和 3 比较，左组的数小，右组从 3 开始到最后一共只有 1 个数，所以产生小和为 1×1=1，合并为[1,3]。

3．[1,3]与[5]合并。1 和 5 比较，左组的数小，右组从 5 开始到最后一共只有 1 个数，所以产生小和为 1×1=1。同理，3 和 5 比较，产生小和为 3×1=3，合并为[1,3,5]。

4．[2]与[4]合并。2 和 4 比较，左组的数小，右组从 4 开始到最后一共只有 1 个数，所以产生小和为 2×1=2，合并为[2,4]。

5．[2,4]与[6]合并。与步骤 3 同理，产生小和为 6，合并为[2,4,6]。

6．[1,3,5]与[2,4,6]合并。1 和 2 比较，左组的数小，右组从 2 开始到最后一共有 3 个数，所以产生小和为 1×3=3。3 和 2 比较，右组的数小，不产生小和。3 和 4 比较，左组的数小，右组从 4 开始到最后一共有 2 个数，所以产生小和为 3×2=6。5 和 4 比较，右组的数小，不产生小和。5 和 6 比较，左组的数小，右组从 6 开始到最后一共有 1 个数，所以产生小和为 5，合并为[1,2,3,4,5,6]。

7．归并过程结束，总的小和为 1+1+3+2+6+3+6+5=27。合并的全部过程如图 8-2 所示。

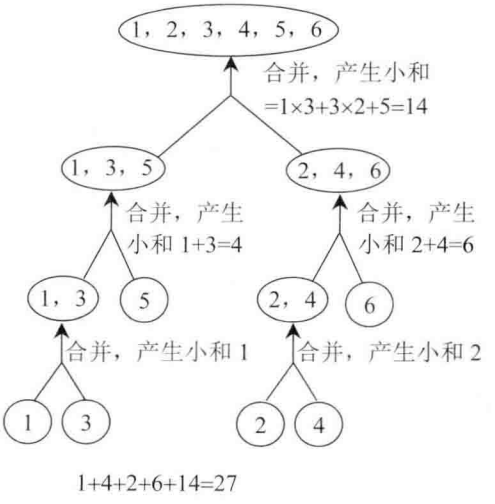

图 8-2

在归并排序中，尤其是在组与组之间进行外排序合并的过程中，按照如上方式把小和一点一点地“榨”出来，最后收集到所有的小和。具体过程请参看如下代码中的 getSmallSum 方法。

```
public int getSmallSum(int[] arr) {
        if (arr == null || arr.length == 0) {
                return 0;
        }
        return func(arr, 0, arr.length - 1);
}

public int func(int[] s, int l, int r) {
        if (l == r) {
                return 0;
        }
        int mid = (l + r) / 2;
        return func(s, l, mid) + func(s, mid + 1, r) + merge(s, l, mid, r);
}

public int merge(int[] s, int left, int mid, int right) {
        int[] h = new int[right - left + 1];
        int hi = 0;
        int i = left;
        int j = mid + 1;
        int smallSum = 0;
        while (i <= mid && j <= right) {
                if (s[i] <= s[j]) {
```

```
                    smallSum += s[i] * (right - j + 1);
                    h[hi++] = s[i++];
                } else {
                    h[hi++] = s[j++];
                }
            }
            for (; (j < right + 1) || (i < mid + 1); j++, i++) {
                h[hi++] = i > mid ? s[j] : s[i];
            }
            for (int k = 0; k != h.length; k++) {
                s[left++] = h[k];
            }
            return smallSum;
        }
```

自然数数组的排序

【题目】

给定一个长度为 N 的整型数组 arr，其中有 N 个互不相等的自然数 1～N，请实现 arr 的排序，但是不要把下标 0～N-1 位置上的数通过直接赋值的方式替换成 1～N。

【要求】

时间复杂度为 $O(N)$，额外空间复杂度为 $O(1)$。

【难度】

士 ★☆☆☆

【解答】

arr 在调整之后应该是下标从 0 到 N-1 的位置上依次放着 1~N，即 arr[index]=index+1。

本书提供两种实现方法，先介绍方法一：

1．从左到右遍历 arr，假设当前遍历到 i 位置。

2．如果 arr[i]==i+1，说明当前的位置不需要调整，继续遍历下一个位置。

3．如果 arr[i]!=i+1，说明此时 i 位置的数 arr[i]不应该放在 i 位置上，接下来将进行跳的过程。

举例来说明，比如[1,2,5,3,4]，假设遍历到位置 2，也就是 5 这个数。5 应该放在位置 4 上，所以把 5 放过去，数组变成[1,2,5,3,5]。同时，4 这个数是被 5 替下来的数，应该放在位置 3，所以把 4 放过去，数组变成[1,2,5,4,5]。同时 3 这个数是被 4 替下来的数，应该放