归并排序

归并排序（MERGE-SORT）是是建立在归并操作上的一种有效，稳定的排序算法。该算法采用经典的分治策略。即先将整个数组进行拆分，然后使每个子序列有序，再使子序列段间有序。

示例

我们现在将通过一个示例，来深刻的理解在归并排序中的一些核心问题。

设有数列{6，202，100，301，38，8，1}

初始状态：6,202,100,301,38,8,1

首先，我们需要对待排序的数组不断的进行拆分，直到每个小数组只有1个数字，因为只有1个数组的数组是不需要排序的！

过程如下：

首先，从中间分割，得到：{6，202，100，301},{38，8，1}

然后，各自从中间分割，得到：{6，202},{100，301},{38，8},{1}

最后，将数组拆分成只有一个数字的集合，即：{6},{202},{100},{301},{38},{8},{1}

接下来，我们开始合并，合并的过程就是沿原有的分割路线原路退回，逐一比较两个数组中数据的大小。

第一次归并共有3次比较，这三次比较分别是：6和202、100和301、38和8，看起来是废话，但是让我们继续往后看。

另外一个内容就是逆序对，因为38大于8，所以此刻我们先记录一组逆序对。

归并结果：{6,202},{100,301},{8,38},{1}

第二次归并的比较次数是4，你看的出来吗？它们分别是：6和100,202和100,202和301，最后一次则是8和1。你看明白了吗？

然后观察逆序对，因为202大于100，所以，这里又出现了第2个逆序对。对于最后的一次比较，因为8大于1，又因为8所在的数组是升序数组，因此，数组内的全部数据都必定是大于1的，数组的长度为2，因此我们需要再加上2个逆序对。到此为止，已经出现了4个逆序对。

归并结果：{6,100,202,301}，{1,8,38}

第三次归并的比较次数也是4;

这四次分别为：6和1,6和8,100和8,100和38。其余的数字则无须再比了，因为此刻另一个数组已经空了。

接下来还是来看逆序对。首先因为6大于1，那么整个数组的全部数据都和1组成逆序对，此处共有4个。接下来因为100大于8，所以100之后的全部数据都和8构成逆序对，此处共有3个。最后100还大于38，所以100以后的全部数据也和38构成逆序对，此处依然是3个。在这一次的比较中，共计出现了10个逆序对，加上前面的4个，共计14个逆序对。

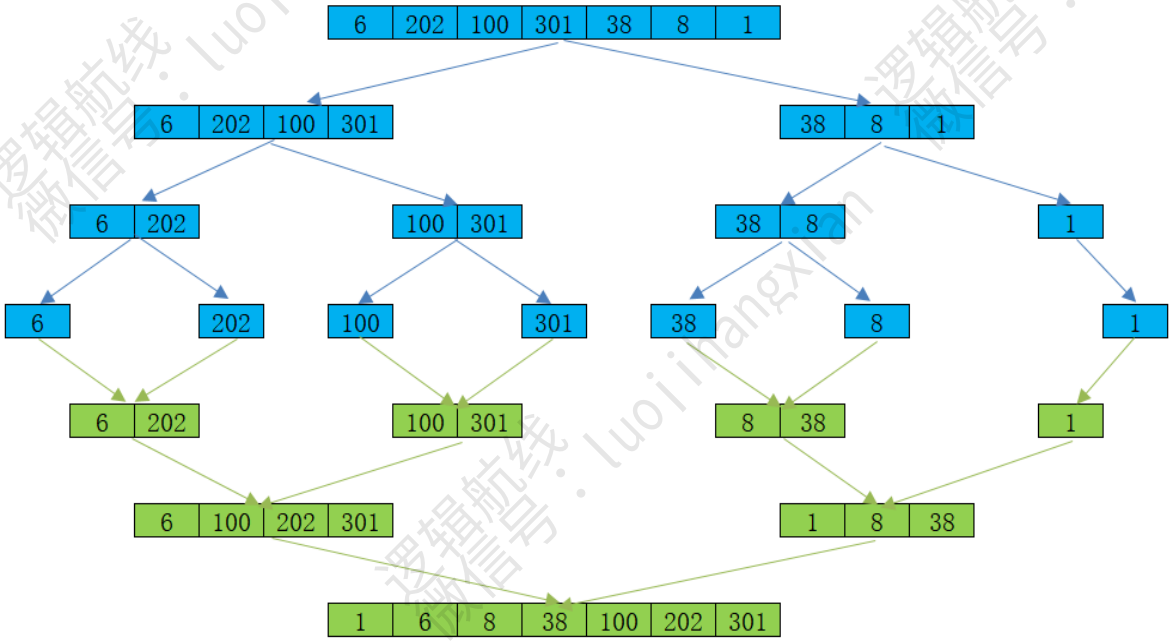
归并结果：{1,6,8,38,100,202,301}

因此，我们得出如下结论：

总的比较次数为：3+4+4=11；

逆序数为14；

整个过程可以用下图表示：



编码

```
#include <bits/stdc++.h>

using namespace std;

long long sum = 0;

void merge(int arr[], int low, int mid, int high) {
    //low为第1有序区的第1个元素，i指向第1个元素，mid为第1有序区的最后1个元素
    int i = low, j = mid + 1, k = 0;  //mid+1为第2有序区第1个元素，j指向第1个元素
    int *temp = new int[high - low + 1]; //temp数组暂存合并的有序序列
    while (i <= mid && j <= high) {
        if (arr[i] <= arr[j]) //较小的先存入temp中
            temp[k++] = arr[i++];
        else {
            temp[k++] = arr[j++];
        }
```

```
    }
    while (i <= mid)//若比较完之后，第一个有序区仍有剩余，则直接复制到t数组中
       temp[k++] = arr[i++];
    while (j <= high)//同上
       temp[k++] = arr[j++];
    for (i = low, k = 0; i <= high; i++, k++) //将排好序的存回arr中low到high这区间
       arr[i] = temp[k];
    delete[]temp;//释放内存，由于指向的是数组，必须用delete []
}

void MergeSort(int arr[], int low, int high) {
    if (low >= high) {
        return;     // 终止递归的条件，子序列长度为1
    }
    int mid = low + (high - low) / 2;  // 取得序列中间的元素
   MergeSort(arr, low, mid);  // 对左半边递归66
    MergeSort(arr, mid + 1, high);  // 对右半边递归
   merge(arr, low, mid, high);  // 合并
}

int a[1000001];

int main(int argc, char **argv) {

    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    MergeSort(a, 0, n - 1);
    for (int i = 0; i < n; i++) {
        cout << a[i];
    }
    return 0;
}
```