**归并的含义是将两个或两个以上的有序表合并成一个新的有序表。**其中两个有序表合并成一个有序表的归并称为二路归并排序，否则称为多路归并排序。

**归并排序既可用于内部排序，也可以用于外部排序。**

**二路归并的基本思想**

将待排序列nums看成一个含有n个长度为1的有序子表，把这些子表依次进行两两归并，得到n/2个有序的子表；然后把这n/2个有序的子表进行两两归并，如此重复，直到最后一个长度为n的有序表为止。**核心操作是将两个相邻的有序序列归并为一个有序序列。**

**实现**

1.两个相邻有序序列的归并

假设前后有两个有序序列分别存在一维数组nums的nums[h…m]和nums[m+1…t]中，首先在两个有序序列中，分别从第1个记录开始对应关键字的比较，将关键字值比较小的记录放入一个有序组order中；然后，因此对两个有序序列中剩余记录进行相同处理，直到两个有序序列中的所有记录都加入到有序数组order中为止；最后这个有序数组order中存放的记录序列就是归并排序的结果。

public static void merge(int[] order,int[] nums,int start,int mid,int end){

//第一个子序列为[start-mid]

//第二个子序列为[mid+1-end]

int i = start;

int j = mid+1;

int k=0;

while(i<=mid&&j<=end){

if(nums[i]>nums[j]) //当前子序列关键字小就加入到order中然后向后移动一位

order[k++] = nums[j++];

else

order[k++] = nums[i++];

}

while(i<=mid) //把前一个子序列的剩余元素复制到order中

order[k++]=nums[i++];

while(j<=end) //把后一个子序列的剩余元素复制到order中

order[k++]=nums[j++];

}

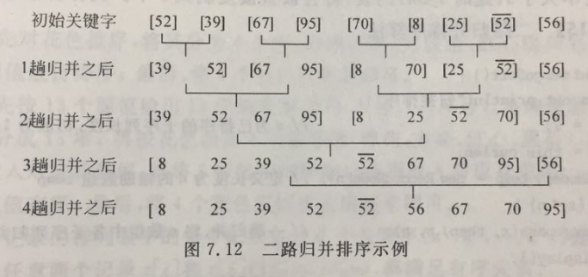
2.一趟归并排序

人们把完成一次将待排序序列中所有两两有序序列合并的过程称为一趟归并排序过程。

实现

假设nums为待排序列的数组，n为带排序列的长度，s为待归并的有序子序列的长度，一趟归并排序的结果存放在数组order中。

当待排序序列中含有偶数个有序子序列时，则只需要调用n/2s(向上取整)次两两归并算法即可完成一趟归并排序；当待排序序列中还有奇数个有序子序列时，则只需调用n/2s(向下取整)次两两递归算法后再将最后一个有序子序列复制到排序结果数组order中，方可完成一趟归并排序。



例如，如图，第二趟归并排序，其中n=9，奇数，s=2，9/(2\*2)(向下取整)=2，所以进行了2次归并排序好，再将最有一个子序列[56]复制到排序结果order数组中。

public void mergepass(int[] nums,int[] order,int n,int s){

//nums为原始数组

//order为存储结果的数组

//n为数组长度

//s为子表长度

int p = 0; //p为每一对待合并表的第1个元素下标，初值为0

//每两个待比较的子表为[p,p+s],[p+s+1,p+2s]

//因为时0开头，[p,p+s-1],[p+s,p+2s-1]

while(p+2\*s-1<=n-1){ //先合并前面长度相等的子序列

merge(order,nums,p,p+s-1,p+2\*s-1);

p=p+2\*s;//跳过两个合并完的子表，下两个子表进行合并

}

if(p+s-1<n-1){ //合并完长度相等的子序列，或者没有长度相等的子序列，合并长度不相等的子序列

merge(order,nums,p,p+s-1,n-1);//归并两个长度不相等的序列，如图第3趟排序

}

else { //将剩余的有序表复制到order中

for(int i=p;i<n;i++){

order[i]=nums[i];

}

}

}

我们在实现的时候，要注意有的时候并不能完全的平分，所以需要写出，处理不相等的序列，出现两个不相等的序列，是在剩余数组不够分为成两个相等的子序列，当能够分成一个子序列的情况，例如途中的第三趟排序。

3.二路归并排序

假设待排序的n个记录保存在数组nums[n]中，归并过程中需要引入辅助数组temp[n]，第1趟有nums归并到temp，第二趟有temp归并到nums；如此反复，直到n个记录成为一个有序表为止。

在归并过程中，为了将最后的排序结果扔置于数组nums中，需要进行的归并趟数为偶数，如果实际上只需要奇数趟就可以完成，那么最后一趟还要进行一趟，正好此时temp中的n个有序记录为一个长度不大于s（此时s>=n）的表，将会被直接复制到nums中。

public void mergeSort(int[] nums){

int s = 1; //s为已排列的子序列长度，初值为1

int n = nums.length;

int[] temp = new int[n]; //辅助数组

while(s<n){

mergepass(nums,temp,n,s);//一趟归并将nums数组子序列归并到temp中

s\*=2; //子序列长度加倍

mergepass(temp,nums,n,s);//二趟归并将temp数组子序列归并到nums中

s\*=2;//子序列长度再加倍

}

}

**算法性能分析**

**1.时间复杂度**

二路归并时间复杂度等于归并趟数于每一趟时间复杂度的乘积。归并趟数为(向上取整)。由于每一趟归并就是将两两有序序列归并，而每一对有序序列归并时，记录的比较次数均不大于记录的移动次数，而记录的移动次数等于这一对有序序列的长度之和，所以每一趟归并的移动次数等于数组中记录的个数n，即每一趟归并的时间复杂度为O(n).因此二路归并排序的时间复杂度为**O(n)**

**2.空间复杂度**

二路归并排序需要一个于待排序记录序列等长的辅助数组来存放排序过程中的中间结果，所以空间复杂度为**O(n)**

**3.算法稳定性**

**稳定**