**[关于数组的几道面试题](http://www.cnblogs.com/graphics/archive/2010/08/24/1761620.html)**

2011年2月15日更新，加入[找出绝对值最小的元素](http://www.cnblogs.com/graphics/archive/2010/08/24/1761620.html" \l "link18)一题

数组是最基本的数据结构，关于数组的面试题也屡见不鲜，本文罗列了一些常见的面试题，仅供参考，如果您有更好的题目或者想法，欢迎留言讨论。目前有以下18道题目，如果有好的题目，随时更新。

* [数组求和](http://www.cnblogs.com/graphics/archive/2010/08/24/1761620.html#link01)
* [求数组的最大值和最小值](http://www.cnblogs.com/graphics/archive/2010/08/24/1761620.html#link02)
* [求数组的最大值和次大值](http://www.cnblogs.com/graphics/archive/2010/08/24/1761620.html#link03)
* [求数组中出现次数超过一半的元素](http://www.cnblogs.com/graphics/archive/2010/08/24/1761620.html#link04)
* [求数组中元素的最短距离](http://www.cnblogs.com/graphics/archive/2010/08/24/1761620.html#link05)
* [求两个有序数组的共同元素](http://www.cnblogs.com/graphics/archive/2010/08/24/1761620.html#link06)
* [求三个数组的共同元素](http://www.cnblogs.com/graphics/archive/2010/08/24/1761620.html#link07)
* [找出数组中唯一的重复元素](http://www.cnblogs.com/graphics/archive/2010/08/24/1761620.html#link08)
* [找出出现奇数次的元素](http://www.cnblogs.com/graphics/archive/2010/08/24/1761620.html#link09)
* [求数组中满足给定和的数对](http://www.cnblogs.com/graphics/archive/2010/08/24/1761620.html#link10)
* [最大子段和](http://www.cnblogs.com/graphics/archive/2010/08/24/1761620.html#link11)
* [最大子段积](http://www.cnblogs.com/graphics/archive/2010/08/24/1761620.html#link12)
* [数组循环移位](http://www.cnblogs.com/graphics/archive/2010/08/24/1761620.html#link13)
* [字符串逆序](http://www.cnblogs.com/graphics/archive/2010/08/24/1761620.html#link14)
* [组合问题](http://www.cnblogs.com/graphics/archive/2010/08/24/1761620.html#link15)
* [合并两个数组](http://www.cnblogs.com/graphics/archive/2010/08/24/1761620.html#link16)
* [重排问题](http://www.cnblogs.com/graphics/archive/2010/08/24/1761620.html#link17)
* [找出绝对值最小的元素](http://www.cnblogs.com/graphics/archive/2010/08/24/1761620.html#link18)

**数组求和**

给定一个含有n个元素的整型数组a，求a中所有元素的和。可能您会觉得很简单，是的，的确简单，但是为什么还要说呢，原因有二，第一，这道题要求用递归法，只用一行代码。第二，这是我人生中第一次面试时候遇到的题，意义特殊。

**分析**

简单说一下，两种情况

1. 如果数组元素个数为0，那么和为0。

2. 如果数组元素个数为n，那么先求出前n - 1个元素之和，再加上a[n - 1]即可

**代码**

[复制代码](javascript:void(0);)

// 数组求和  
int sum(int\*a, int n)  
{  
     return n ==0?0 : sum(a, n -1) + a[n -1];  
}

[复制代码](javascript:void(0);)

**求数组的最大值和最小值**

给定一个含有n个元素的整型数组a，找出其中的最大值和最小值

**分析**

常规的做法是遍历一次，分别求出最大值和最小值，但我这里要说的是分治法(Divide and couquer)，将数组分成左右两部分，先求出左半部份的最大值和最小值，再求出右半部份的最大值和最小值，然后综合起来求总体的最大值及最小值。这是个递归过程，对于划分后的左右两部分，同样重复这个过程，直到划分区间内只剩一个元素或者两个元素。

**代码**

[复制代码](javascript:void(0);)

// 求数组的最大值和最小值，返回值在maxValue和minValue  
void MaxandMin(int \*a, int l, int r, int& maxValue, int& minValue)  
{  
 if(l == r) // l与r之间只有一个元素  
 {  
 maxValue = a[l] ;  
 minValue = a[l] ;  
 return ;  
 }  
  
 if(l + 1 == r) // l与r之间只有两个元素  
 {  
 if(a[l] >= a[r])  
 {  
 maxValue = a[l] ;  
 minValue = a[r] ;  
 }  
 else  
 {  
 maxValue = a[r] ;  
 minValue = a[l] ;  
 }  
 return ;  
 }  
  
 int m = (l + r) / 2 ; // 求中点  
  
 int lmax ; // 左半部份最大值  
 int lmin ; // 左半部份最小值  
 MaxandMin(a, l, m, lmax, lmin) ; // 递归计算左半部份  
  
 int rmax ; // 右半部份最大值  
 int rmin ; // 右半部份最小值  
 MaxandMin(a, m + 1, r, rmax, rmin) ; // 递归计算右半部份  
  
 maxValue = max(lmax, rmax) ; // 总的最大值  
 minValue = min(lmin, rmin) ; // 总的最小值  
}

[复制代码](javascript:void(0);)

**求数组的最大值和次大值**

给定一个含有n个元素的整型数组，求其最大值和次大值

**分析**

思想和上一题类似，同样是用分治法，先求出左边的最大值leftmax和次大值leftsecond，再求出右边的最大值rightmax和次大值rightsecond，然后合并，如何合并呢？分情况考虑

1 如果leftmax > rightmax，那么可以肯定leftmax是最大值，但次大值不一定是rightmax，但肯定不是rightsecond，只需将leftsecond与rightmax做一次比较即可。

2 如果rightmax > leftmax，那么可以肯定rightmax是最大值，但次大值不一定是leftmax，但肯定不是leftsecond，所以只需将leftmax与rightsecond做一次比较即可。

**注意**

这种方法无法处理最大元素有多个的情况，比如3,5,7,7将返回7，7而不是7,5。感谢网友 从无到有靠谁人 指出。

**代码**

[复制代码](javascript:void(0);)

// 找出数组的最大值和次大值，a是待查找的数组，left和right是查找区间，max和second存放结果  
void MaxandMin(int a[], int left, int right, int&max, int&second)  
{  
 if(left == right)  
 {  
 max = a[left] ;  
 second =  INT\_MIN;  
 }  
 elseif(left +1== right)  
 {  
 max = a[left] > a[right] ? a[left] : a[right] ;  
 second = a[left] < a[right] ? a[left] : a[right] ;  
 }  
 else  
 {  
 int mid = left + (right - left) /2 ;  
  
 int leftmax ;  
 int leftsecond ;  
 MaxandMin(a, left, mid, leftmax, leftsecond) ;  
  
 int rightmax ;  
 int rightsecond ;  
 MaxandMin(a, mid +1, right, rightmax, rightsecond) ;  
  
 if (leftmax > rightmax)  
 {  
 max = leftmax ;  
 second = leftsecond > rightmax ? leftsecond : rightmax ;  
 }  
 else  
 {  
 max = rightmax ;  
 second = leftmax < rightsecond ? rightsecond : leftmax ;  
 }  
 }  
}

[复制代码](javascript:void(0);)

**求数组中出现次数超过一半的元素**

给定一个n个整型元素的数组a，其中有一个元素出现次数超过n / 2，求这个元素。据说是百度的一道题

**分析**

设置一个当前值和当前值的计数器，初始化当前值为数组首元素，计数器值为1，然后从第二个元素开始遍历整个数组，对于每个被遍历到的值a[i]

1 如果a[i]==currentValue，则计数器值加1

2 如果a[i] != currentValue， 则计数器值减1，如果计数器值小于0，则更新当前值为a[i]，并将计数器值重置为1

**代码**

[复制代码](javascript:void(0);)

// 找出数组中出现次数超过一半的元素  
int Find(int\* a, int n)  
{  
 int curValue = a[0] ;  
 int count = 1 ;  
  
 for (int i = 1; i < n; ++i)  
 {  
 if (a[i] == curValue)  
 count++ ;  
 else  
 {  
 count-- ;  
 if (count < 0)  
 {  
 curValue = a[i] ;  
 count = 1 ;  
 }  
 }  
 }  
  
 return curValue ;  
}

[复制代码](javascript:void(0);)

另一个方法是先对数组排序，然后取中间元素即可，因为如果某个元素的个数超过一半，那么数组排序后该元素必定占据数组的中间位置。

**求数组中元素的最短距离**

给定一个含有n个元素的整型数组，找出数组中的两个元素x和y使得abs(x - y)值最小

**分析**

先对数组排序，然后遍历一次即可

**代码**

[复制代码](javascript:void(0);)

int compare(const void\* a, const void\* b)  
{  
 return \*(int\*)a - \*(int\*)b ;  
}  
  
// 求数组中元素的最短距离  
void MinimumDistance(int\* a, int n)  
{  
 // Sort  
 qsort(a, n, sizeof(int), compare) ;  
  
 int i ; // Index of number 1  
 int j ; // Index of number 2  
  
 int minDistance = numeric\_limits<int>::max() ;  
 for (int k = 0; k < n - 1; ++k)  
 {  
 if (a[k + 1] - a[k] < minDistance)  
 {  
 minDistance = a[k + 1] - a[k] ;  
 i = a[k] ;  
 j = a[k + 1] ;  
 }  
 }  
  
 cout << "Minimum distance is: " << minDistance << endl ;  
 cout << "i = " << i << " j = " << j << endl ;  
}

[复制代码](javascript:void(0);)

**求两个有序数组的共同元素**

给定两个含有n个元素的有序（非降序）整型数组a和b，求出其共同元素，比如

a = 0, 1, 2, 3, 4

b = 1, 3, 5, 7, 9

输出 1, 3

**分析**

充分利用数组有序的性质，用两个指针i和j分别指向a和b，比较a[i]和b[j]，根据比较结果移动指针，则有如下三种情况

1. a[i] < b[j]，则i增加1，继续比较

2. a[i] == b[j]，则i和j皆加1，继续比较

3. a[i] < b[j]，则j加1，继续比较

重复以上过程直到i或j到达数组末尾。

**代码**

[复制代码](javascript:void(0);)

// 找出两个数组的共同元素  
void FindCommon(int\* a, int\* b, int n)  
{  
 int i = 0;  
 int j = 0 ;  
  
 while (i < n && j < n)  
 {  
 if (a[i] < b[j])  
 ++i ;  
 else if(a[i] == b[j])  
 {  
 cout << a[i] << endl ;  
 ++i ;  
 ++j ;  
 }  
 else// a[i] > b[j]  
 ++j ;  
 }  
}

[复制代码](javascript:void(0);)

这到题还有其他的解法，比如对于a中任意一个元素，在b中对其进行Binary Search，因为a中有n个元素，而在b中进行Binary Search需要logn。所以找出全部相同元素的时间复杂度是O(nlogn)。

另外，上面的方法，只要b有序即可，a是否有序无所谓，因为我们只是在b中做Binary Search。如果a也有序的话，那么再用上面的方法就有点慢了，因为如果a中某个元素在b中的位置是k的话，那么a中下一个元素在b中的位置一定位于k的右侧，所以本次的搜索空间可以根据上次的搜索结果缩小，而不是仍然在整个b中搜索。也即如果a和b都有序的话，代码可以做如下修改，记录上次搜索时b中元素的位置，作为下一次搜索的起始点。

**求三个数组的共同元素**

给定三个含有n个元素的整型数组a,b和c，求他们最小的共同元素

**分析**

如果三个数组都有序，那么可以设置三个指针指向三个数组的头部，然后根据这三个指针所指的值进行比较来移动指针，直道找到共同元素。

**代码**

[复制代码](javascript:void(0);)

// 三个数组的共同元素-只找最小的  
void FindCommonElements(int a[], int b[], int c[], int x, int y, int z)  
{  
 for(int i = 0, j = 0, k = 0; i < x && j < y && k < z;)  
 {  
 if(a[i] < b[j])  
 {  
 i++ ;  
 }  
 else // a[i] >= b[j]  
 {  
 if(b[j] < c[k])  
 {  
 j++ ;  
 }  
 else // b[j] >= c[k]  
 {  
 if(c[k] < a[i])  
 {  
 k++ ;  
 }  
 else // c[k] >= a[i]  
 {  
 cout << c[k] << endl ;  
 return ;  
 }  
 }  
 }  
 }  
  
 cout << "Not found!" << endl ;  
}

[复制代码](javascript:void(0);)

如果三个数组都无序，可以先对a, b进行排序，然后对c中任意一个元素都在b和c中做二分搜索。

**代码**

[复制代码](javascript:void(0);)

// 找出三个数组的共同元素  
// O(NlogN)  
int UniqueCommonItem(int \*a, int \*b, int \*c, int n)  
{  
 // sort array a  
 qsort(a, n, sizeof(int), compare) ; // NlogN  
  
 // sort array b  
 qsort(b, n, sizeof(int), compare) ; // NlogN  
  
 // for each element in array c, do a binary search in a and b  
 // This is up to a complexity of N\*2\*logN  
 for (int i = 0; i < n; i++)  
 {  
 if(BinarySearch(a, n, c[i]) && BinarySearch(b, n, c[i]))  
 return c[i] ;  
 }  
  
 return - 1 ; // not found  
}

[复制代码](javascript:void(0);)

也可以对a进行排序，然后对于b和c中任意一个元素都在a中进行二分搜索

**代码**

[复制代码](javascript:void(0);)

// 找出三个数组唯一的共同元素  
// O(NlogN)  
int UniqueCommonItem1(int \*a, int \*b, int \*c, int n)  
{  
 // sort array a  
 qsort(a, n, sizeof(int), compare) ; // NlogN  
  
 // Space for time  
 bool \*bb = new bool[n] ;  
 memset(bb, 0, n) ;  
  
 bool \*bc = new bool[n] ;  
 memset(bb, 0, n) ;  
  
 // for each element in b, do a BS in a and mark all the common element  
 for (int i = 0; i < n; i++) // NlogN  
 {  
 if(BinarySearch(a, n, b[i]))  
 bb[i] = true ;  
 }  
  
 // for each element in c, do a BS only if b[i] is true  
 for (int i = 0; i < n; i++) // NlogN  
 {  
 if(b[i] && BinarySearch(a, n, c[i]))  
 return c[i] ;  
 }  
  
 return - 1 ; // not found  
}

[复制代码](javascript:void(0);)

排序和二分搜索代码如下

[复制代码](javascript:void(0);)

// Determine whether a contains value k  
bool BinarySearch(int \*a, int n, int k)  
{  
 int left = 0 ;  
 int right = n - 1 ;  
 while (left <= right)  
 {  
 int mid = (left + right) ;  
  
 if(a[mid] < k)  
 left = mid + 1 ;  
 if(a[mid] == k)  
 return true ;  
 else  
 right = mid - 1 ;  
 }  
  
 return false ;  
}  
  
// Compare function for qsort  
int compare(const void\* a, const void\* b)  
{  
 return \*(int\*)a - \*(int\*)b ;  
}

[复制代码](javascript:void(0);)

小小总结一下，对于在数组中进行查找的问题，可以分如下两种情况处理

1. 如果给定的数组有序，那么首先应该想到Binary Search，所需O(logn)

2. 如果给定的数组无序，那么首先应该想到对数组进行排序，很多排序算法都能在O(nlogn)时间内对数组进行排序，然后再使用二分搜索，总的时间复杂度仍是O(nlogn)。

如果能做到以上两点，大多数关于数组的查找问题，都能迎刃而解。

**找出数组中唯一的重复元素**

给定含有1001个元素的数组，其中存放了1-1000之内的整数，只有一个整数是重复的，请找出这个数

**分析**

求出整个数组的和，再减去1-1000的和

**代码**

略

**找出出现奇数次的元素**

给定一个含有n个元素的整型数组a，其中只有一个元素出现奇数次，找出这个元素。

**分析**

因为对于任意一个数k，有k ^ k = 0，k ^ 0 = k，所以将a中所有元素进行异或，那么个数为偶数的元素异或后都变成了0，只留下了个数为奇数的那个元素。

**代码**

[复制代码](javascript:void(0);)

int FindElementWithOddCount(int\*a, int n)  
{  
     int r = a[0] ;  
  
     for (int i =1; i < n; ++i)  
     {  
          r ^= a[i] ;  
     }  
  
     return r ;  
}

[复制代码](javascript:void(0);)

**求数组中满足给定和的数对**

给定两个有序整型数组a和b，各有n个元素，求两个数组中满足给定和的数对，即对a中元素i和b中元素j，满足i + j = d(d已知)

**分析**

两个指针i和j分别指向数组的首尾，然后从两端同时向中间遍历，直到两个指针交叉。

**代码**

[复制代码](javascript:void(0);)

// 找出满足给定和的数对  
void FixedSum(int\* a, int\* b, int n, int d)  
{  
 for (int i = 0, j = n - 1; i < n && j >= 0)  
 {  
 if (a[i] + b[j] < d)  
 ++i ;  
 else if (a[i] + b[j] == d)  
 {  
 cout << a[i] << ", " << b[j] << endl ;  
 ++i ;  
 --j ;  
 }  
 else // a[i] + b[j] > d  
 --j ;  
 }  
}

[复制代码](javascript:void(0);)

**最大子段和**

给定一个整型数组a，求出最大连续子段之和，如果和为负数，则按0计算，比如1， 2， -5， 6， 8则输出6 + 8 = 14

**分析**

编程珠玑上的经典题目，不多说了。

**代码**

[复制代码](javascript:void(0);)

// 子数组的最大和  
int Sum(int\* a, int n)  
{  
 int curSum = 0;  
 int maxSum = 0;  
 for (int i = 0; i < n; i++)  
 {  
 if (curSum + a[i] < 0)  
 curSum = 0;  
 else  
 {  
 curSum += a[i] ;  
 maxSum = max(maxSum, curSum);  
 }  
 }  
 return maxSum;  
}

[复制代码](javascript:void(0);)

**最大子段积**

给定一个整型数组a，求出最大连续子段的乘积，比如 1， 2， -8， 12， 7则输出12 \* 7 = 84

**分析**

与最大子段和类似，注意处理负数的情况

**代码**

[复制代码](javascript:void(0);)

// 子数组的最大乘积  
int MaxProduct(int \*a, int n)  
{  
 int maxProduct = 1; // max positive product at current position  
 int minProduct = 1; // min negative product at current position  
 int r = 1; // result, max multiplication totally  
  
 for (int i = 0; i < n; i++)  
 {  
 if (a[i] > 0)  
 {  
 maxProduct \*= a[i];  
 minProduct = min(minProduct \* a[i], 1);  
 }  
 else if (a[i] == 0)  
 {  
 maxProduct = 1;  
 minProduct = 1;  
 }  
 else // a[i] < 0  
 {  
 int temp = maxProduct;  
 maxProduct = max(minProduct \* a[i], 1);  
 minProduct = temp \* a[i];  
 }  
  
 r = max(r, maxProduct);  
 }  
  
 return r;  
}

[复制代码](javascript:void(0);)

**数组循环移位**

将一个含有n个元素的数组向右循环移动k位，要求时间复杂度是O(n)，且只能使用两个额外的变量，这是在微软的编程之美上看到的一道题

**分析**

比如数组 1 2 3 4循环右移1位 将变成 4 1 2 3， 观察可知1 2 3 的顺序在移位前后没有改变，只是和4的位置交换了一下，所以等同于1 2 3 4 先划分为两部分

1 2 3 | 4，然后将1 2 3逆序，再将4 逆序 得到 3 2 1 4，最后整体逆序 得到 4 1 2 3

**代码**

[复制代码](javascript:void(0);)

// 将buffer中start和end之间的元素逆序  
void Reverse( int buffer[], int start, int end )  
{  
 while ( start < end )  
 {  
 int temp = buffer[ start ] ;  
 buffer[ start++ ] = buffer[ end ] ;  
 buffer[ end-- ] = temp ;  
 }  
}  
  
// 将含有n个元素的数组buffer右移k位  
void Shift( int buffer[], int n, int k )  
{  
 k %= n ;  
  
 Reverse( buffer, 0, n - k - 1) ;  
 Reverse( buffer, n - k, n - 1 ) ;  
 Reverse( buffer, 0, n - 1 ) ;  
}

[复制代码](javascript:void(0);)

稍微扩展一下,如果允许分配额外的数组,那么定义一个新的数组,然后将移位后的元素直接存入即可,也可以使用队列,将移动后得元素出对,再插入队尾即可.

**字符串逆序**

给定一个含有n个元素的字符数组a，将其原地逆序。

**分析**

可能您觉得这不是关于数组的，而是关于字符串的。是的。但是别忘了题目要求的是原地逆序，也就是不允许额外分配空间，那么参数肯定是字符数组形式，因为字符串是不能被修改的（这里只C/C++中的字符串常量），所以，和数组有关了吧，只不过不是整型数组，而是字符数组。用两个指针分别指向字符数组的首位，交换其对应的字符，然后两个指针分别向数组中央移动，直到交叉。

**代码**

[复制代码](javascript:void(0);)

// 字符串逆序  
void Reverse(char\*a, int n)  
{  
     int left =0;   
     int right = n -1;  
  
     while (left < right)  
     {  
         char temp = a[left] ;  
         a[left++] = a[right] ;  
         a[right--] = temp ;  
     }  
}

[复制代码](javascript:void(0);)

**组合问题**

给定一个含有n个元素的整型数组a，从中任取m个元素，求所有组合。比如下面的例子

a = 1, 2, 3, 4, 5

m = 3

输出

1 2 3, 1 2 4, 1 2 5, 1 3 4, 1 3 5, 1 4 5

2 3 4, 2 3 5, 2 4 5  
3 4 5

**分析**

典型的排列组合问题，首选回溯法，为了简化问题，我们将a中n个元素值分别设置为1-n

**代码**

[复制代码](javascript:void(0);)

// n选m的所有组合  
int buffer[100] ;  
  
void PrintArray(int \*a, int n)  
{  
 for (int i = 0; i < n; ++i)  
 cout << a[i] << "";  
 cout << endl ;  
}  
  
bool IsValid(int lastIndex, int value)  
{  
 for (int i = 0; i < lastIndex; i++)  
 {  
 if (buffer[i] >= value)  
 return false;  
 }  
 return true;  
}  
  
void Select(int t, int n, int m)  
{  
 if (t == m)  
 PrintArray(buffer, m);  
 else  
 {  
 for (int i = 1; i <= n; i++)  
 {  
 buffer[t] = i;  
 if (IsValid(t, i))  
 Select(t + 1, n, m);  
 }  
 }  
}

[复制代码](javascript:void(0);)

**合并两个数组**

给定含有n个元素的两个有序（非降序）整型数组a和b。合并两个数组中的元素到整型数组c，要求去除重复元素并保持c有序（非降序）。例子如下

a = 1, 2, 4, 8

b = 1, 3, 5, 8

c = 1, 2, 3, 4, 5, 8

**分析**

利用合并排序的思想，两个指针i,j和k分别指向数组a和b，然后比较两个指针对应元素的大小，有以下三种情况

1. a[i] < b[j]，则c[k] = a[i]。

2. a[i] == b[j]，则c[k]等于a[i]或b[j]皆可。

3. a[i] > b[j]，则c[k] = b[j]。

重复以上过程，直到i或者j到达数组末尾，然后将剩下的元素直接copy到数组c中即可

**代码**

[复制代码](javascript:void(0);)

// 合并两个有序数组  
void Merge(int \*a, int \*b, int \*c, int n)  
{  
 int i = 0 ;  
 int j = 0 ;  
 int k = 0 ;  
  
 while (i < n && j < n)  
 {  
 if (a[i] < b[j])// 如果a的元素小，则插入a中元素到c  
 {  
 c[k++] = a[i] ;  
 ++i ;  
 }  
 else if (a[i] == b[j])// 如果a和b元素相等，则插入二者皆可，这里插入a  
 {  
 c[k++] = a[i] ;  
 ++i ;  
 ++j ;  
 }  
 else // a[i] > b[j] // 如果b中元素小，则插入b中元素到c  
 {  
 c[k++] = b[j] ;  
 ++j ;  
 }  
 }  
  
 if (i == n) // 若a遍历完毕，处理b中剩下的元素  
 {  
 for (int m = j; m < n; ++m)  
 c[k++] = b[m] ;  
 }  
 else//j == n, 若b遍历完毕，处理a中剩下的元素  
 {  
 for (int m = i; m < n; ++m)  
 c[k++] = a[m] ;  
 }  
}

[复制代码](javascript:void(0);)

**重排问题**

给定含有n个元素的整型数组a，其中包括0元素和非0元素，对数组进行排序，要求：

1. 排序后所有0元素在前，所有非零元素在后，且非零元素排序前后相对位置不变

2. 不能使用额外存储空间

例子如下

输入 0, 3, 0, 2, 1, 0, 0

输出 0, 0, 0, 0, 3, 2, 1

**分析**

此排序非传统意义上的排序，因为它要求排序前后非0元素的相对位置不变，或许叫做整理会更恰当一些。我们可以从后向前遍历整个数组，遇到某个位置i上的元素是非0元素时，如果a[k]为0，则将a[i]赋值给a[k]，a[k]赋值为0。实际上i是非0元素的下标，而k是0元素的下标

**代码**

[复制代码](javascript:void(0);)

void Arrange(int\* a, int n)  
{  
 int k = n -1 ;  
 for (int i = n -1; i >=0; --i)  
 {  
 if (a[i] !=0)  
 {  
 if (a[k] ==0)  
 {  
 a[k] = a[i] ;  
 a[i] =0 ;  
 }  
 --k ;  
 }  
 }  
}

[复制代码](javascript:void(0);)

**找出绝对值最小的元素**

给定一个有序整数序列（非递减序），可能包含负数，找出其中绝对值最小的元素，比如给定序列 -5, -3, -1, 2, 8 则返回1。

**分析**

由于给定序列是有序的，而这又是搜索问题，所以首先想到二分搜索法，只不过这个二分法比普通的二分法稍微麻烦点，可以分为下面几种情况

* 如果给定的序列中所有的数都是正数，那么数组的第一个元素即是结果。
* 如果给定的序列中所有的数都是负数，那么数组的最后一个元素即是结果。
* 如果给定的序列中既有正数又有负数，那么绝对值得最小值一定出现在正数和负数的连接处。

为什么？因为对于负数序列来说，右侧的数字比左侧的数字绝对值小，如上面的-5, -3, -1, 而对于整整数来说，左边的数字绝对值小，比如上面的2, 8，将这个思想用于二分搜索，可先判断中间元素和两侧元素的符号，然后根据符号决定搜索区间，逐步缩小搜索区间，直到只剩下两个元素。

**代码**

单独设置一个函数用来判断两个整数的符号是否相同

[复制代码](javascript:void(0);)

// 找出一个非递减序整数序列中绝对值最小的数  
int MinimumAbsoluteValue(int\* a, int n)  
{  
 // Only one number in array  
if (n ==1)  
 {  
 return a[0] ;  
 }  
  
 // All numbers in array have the same sign  
if (SameSign(a[0], a[n -1]))  
 {  
 return a[0] >=0? a[0] : a[n -1] ;  
 }  
  
 // Binary search  
int l =0 ;  
 int r = n -1 ;  
  
 while(l < r)  
 {  
 if (l +1== r)  
 {  
 return abs(a[l]) < abs(a[r]) ? a[l] : a[r] ;  
 }  
  
 int m = (l + r) /2 ;  
  
 if (SameSign(a[m], a[r]))  
 {  
 r = m -1;  
 continue;  
 }  
 if (SameSign(a[l], a[m]))  
 {  
 l = m +1 ;  
 continue;  
 }  
 }  
}

[复制代码](javascript:void(0);)

=THE END=