**1.** **查找数组中第k大的数字？**

因为快排每次将数组划分为两组加一个枢纽元素，每一趟划分你只需要将k与枢纽元素的下标进行比较，如果比枢纽元素下标大就从右边的子数组中找，如果比枢纽元素下标小从左边的子数组中找，如果一样则就是枢纽元素，找到，如果需要从左边或者右边的子数组中再查找的话，只需要递归一边查找即可，无需像快排一样两边都需要递归，所以复杂度必然降低。

最差情况如下：假设快排每次都平均划分，但是都不在枢纽元素上找到第k大第一趟快排没找到，时间复杂度为O(n)，第二趟也没找到，时间复杂度为O(n/2)，第k趟找到，时间复杂度为O(n/2k)，所以总的时间复杂度为O(n(1+1/2+....+1/2k))=O(n)，明显比冒泡快，虽然递归深度是一样的，但是每一趟时间复杂度降低。

**2、连续子数组的最大和**

**问题描述：**

输入一个整型数组，数组中连续的一个或多个整数组成一个子数组，每个子数组都有一个和，求所有子数组和的最大值。  
例如输入的数组为1,-2,3,10,-4,7,2,-5，和最大的子数组为3,10,-4,7,2，因此输出为该子数组的和18。

[复制代码](javascript:void(0);)

int max\_sub(int a[], int size) //动态规划

{

int i;

int max=0;

int temp\_sum=0;

for (i=0; i<size; i++) {

temp\_sum += a[i];

if (temp\_sum>max) {

max = temp\_sum;

} else if (temp\_sum<0) {

temp\_sum = 0;

}

}

// if all data are negative

if (max == 0) {

max = a[0];

for (i=1; i<size; i++) {

if (a[i] > max) max = a[i];

}

}

return max;

}

[复制代码](javascript:void(0);)

**3、数对之差的最大值**

**问题描述：**  
在数组中，数字减去它右边的数字得到一个数对之差，求所有数对之差的最大值。  
例如在数组{2,4,1,16,7,5,11,9}中，数对之差的最大值是11，是16-5的结果。

这个问题用暴力法破解的话，时间复杂度是O(n^2)，我们可以换下思路。

构建一个辅助数组diff，并且diff[i]=number[i]-number[i+1] (0<=i<N-1)，那么 diff[i]+diff[i+1]+...+diff[j]=numbers[i]-number[j+1]，

可见最大的数对之差numbers[i]-number[j+1]，同时也是 diff[i]+diff[i+1]+...+diff[j]的最大值，也就是连续子数组的最大和，这就回到了上面那个问题了。

C代码实现如下：

[复制代码](javascript:void(0);)

#define N 8

int main()

{

int array[N] = {2,4,1,16,7,5,11,9};

int diff[N-1];

int j = 0;

for (; j<N-1; j++) diff[j] = array[j] - array[j+1];

j = max\_sub(diff, N-1);

printf("the max sub is: %d\n", j); // 11

return 0;

}

[复制代码](javascript:void(0);)

上面这种方法比较巧妙，一般不容易想到，我们再用一种比较常规的解法。

构建数组 diff[i] = max(number[h] - number[i])，（0<=h<i），现在问题就变成了求数组diff[i] (0<=i<N-1)的最大值。

假设我们已经知道了diff[i]，那么该怎么求diff[i+1]呢？对于diff[i]，肯定存在一个h（h<i），满足number[h]-number[i]最大，也就是number[h]应该是number[i]之前的所有数字的最大值。当我们求diff[i+1]的时候，需要找到第i+1个数字之前的最大值，这个值只有两种可能，一个是number[h]，另一个是number[i]。

C代码实现如下：

[复制代码](javascript:void(0);)

#define N 8

#define MAX(a,b) (a>b?a:b)

int main()

{

int i;

int array[N] = {2,4,1,16,7,5,11,9};

int diff;

int last\_max = array[0];

int max\_diff = array[0] - array[1];

for (i=2; i<N; i++) {

last\_max = MAX(last\_max, array[i-1]);

diff = last\_max - array[i];

if (diff > max\_diff)

max\_diff = diff;

}

printf("the max sub is: %d\n", max\_diff); // 11

return 0;

}

[复制代码](javascript:void(0);)

**4、把数组排成最小的数**

**问题描述：**

输入一个正整数数组，将它们连接起来排成一个数，输出能排出的所有数字中最小的一个。  
例如输入数组{3，32，321}，则输出这两个能排成的最小数字321323.

class Solution {

public:

string PrintMinNumber(vector<int> numbers) {

vector<string> sortedNums;

for(auto &i : numbers)

sortedNums.push\_back(to\_string(i));

sort(sortedNums.begin(), sortedNums.end(), [](string &s1, string &s2){return s1+s2 <s2+s1;});

string res = "";

for(auto &s : sortedNums)

res += s;

return res;

}

};

**5、有序数组中和为给定值的两个数字**

**问题描述：**

输入一个已经按升序排序过的数组和一个数字，在数组中查找两个数，使得它们的和正好是输入的那个数字。要求时间复杂度是O(n)。如果有多对数字的和等于输入的数字，输出任意一对即可。例如输入数组1、2、4、7、11、15和数字15。由于4+11=15，因此输出4和11。

**6、扑克牌的顺子**

**问题描述：**

从扑克牌中随机抽5张牌，判断是不是一个顺子，即这5张牌是不是连续的。2-10为数字本身，A为1，J为11，Q为12，K为13，而大小王可以看成任意数字。

需要做三件事情：把数组排序，统计数组中0的个数，统计排序之后的数组相邻数字之间的空缺总数。如果空缺的总数小于或者等于0的个数，那么这个数组就是连续的；反之则不连续。最后，我们还需要注意的是，如果数组中的非0数字重复出现，则该数组不是连续的。换成扑克牌的描述方式，就是如果一副牌里含有对子，则不可能是顺子。

**7、子数组换位问题**

**问题描述：**

设a[0:n-1]是一个有n个元素的数组，k(0<=k<=n-1)是一个非负整数。 试设计一个算法将子数组a[0:k]与a[k+1,n-1]换位。

PS：要求算法在最坏情况下耗时O(n)，且只用到O(1)的辅助空间。

例如，数组 {a0, a1, a2, a3, a4, a5, a6, a7, a8, a9}，

1、若k=4（两个子数组等长），即需要将数组变成：{a5, a6, a7, a8, a9, a0, a1, a2, a3, a4}，两个子数组的长度一样，直接两两交换a[i]与a[i+k]即可；

2、若k=1（后面的子数组更长），即需要将数组变成：{a2, a3, a4, a5, a6, a7, a8, a9, a0, a1}，可以先把第一个子数组交换到整个数组的最后，得到：

{a8, a9, a2, a3, a4, a5, a6, a7,  a0, a1}，然后对前面的子数组再交换一次，得到：

{a2, a3, a4, a5, a6, a7, a8, a9,  a0, a1}

3、若k=6（前面的子数组更长），即需要将数组变成：{a8, a9, a0, a1, a2, a3, a4, a5, a6, a7}，可以先把第二个子数组交换到整个数组的前面，得到：

{a8, a9, a2, a3, a4, a5, a6, a7, a0, a1}，然后问题就变成了怎么把{a2, a3, a4, a5, a6, a7}与{a0, a1}交换了，递归处理即可。

**8、求数组的主元素**

**问题描述：**  
大小为Ｎ的数组Ａ，其主元素是一个出现超过Ｎ／２次的元素（从而这样的元素最多有一个），怎么用线性时间算法得到一个数组的主元素（如果有的话）。

**算法思想：**

1、先将数组排序，如果数组的主元素存在，那么主元素一定位于数组中间位置，也就是说主元素一定是数组的中位数。那么问题就变成了怎么求中位数？

2、**另外一种思路：**

若数组中主元素存在，且其个数为m，则有 m > N/2，两边同时减1，既有 m-1 > (N-2)/2。

也就是说在数组中随机找出两个元素e1、e2，如果e1 != e2，并且其中一个是主元素，则删除这两个元素，主元素在剩下的数组中仍是主元素。注意：如果e1、e2没有主元素，那么可能导致某个不是主元素的元素在剩下的数组中变成主元素。

**9、连续数打乱判断少了哪个数**

**问题描述：**

N个连续的数（比如0～999）打乱 之后，随机取出1个数 ，问如何最快速的判断出少了哪一个？

**算法描述：**

由于数组顺序被打乱，最可行的办法就是建立一个bitmap，然后扫描一遍数组，并在bitmap中相应位置1，比如数组元素7就在bitmap第7位置1。最后再扫描一遍bitmap就知道缺失哪个数了，这样做的时间复杂度是O(2n)。

也可以用异或运算来分析下从0到N的异或结果，并将该结果与数组进行异或

这个问题还可以进行一些拓展，比如：

1、一个连续数组，所有元素位于0~(N-1)之间，但有一个元素出现了2次，怎么快速找到多出来的这个数字？同样使用异或就可以搞定它！

2、一个连续数组，一个元素缺失，另一个元素重复了，怎么快速找到这两个数字？（）

3、在一个整型数组中，除了1个数字之外，其他的数字都出现了两次，怎么找到这1个只出现一次的数字？（异或）

4、在一个整型数组中，除了2个数字之外，其他的数字都出现了两次，怎么找到这2个只出现一次的数字？（异或，分类）

5、在一个整型数组中，除了3个数字之外，其他的数字都出现了两次，怎么找到这3个只出现一次的数字？（异或，分类，再分类）