# 第3章 排 序

排序(sorting)是计算机内经常进行的一种操作,其目的是将一组“无序”的数据元素(或记录)序列调整为“有序”的序列。

排序的概念:将杂乱无章的数据元素,通过一定的方法按关键字顺序排列的过程叫作排序。

常见的排序算法如下。

插入排序:直接插入排序、希尔排序。

选择排序:选择排序、堆排序。

交换排序:冒泡排序、快速排序。

归并排序:归并排序。

假设待排序的文件中存在两条或两条以上的记录具有相同的关键字,在用某种排序法排序后,若这些相同关键字的元素的相对次序仍然不变,则这种排序方法是稳定的。其中冒泡排序、插入排序、基数排序、归并排序属于稳定排序,选择排序、快速排序、希尔排序、堆排序属于不稳定排序。多种排序算法的对比如表 3-1 所示。

表 3-1 多种排序算法的对比

| 排序方法 | | 时间复杂度 | | | 空间复杂度 | 稳定性 |
|---|---|---|---|---|---|---|
| | | 平均情况 | 最好情况 | 最坏情况 | | |
| 插入排序 | 直接插入 | $O(n^2)$ | $O(n)$ | $O(n^2)$ | $Q(1)$ | 稳定 |
| | 希尔排序 | $O(n^{1\sim2})$ | $O(n\log^2 n)$ | $O(n^2)$ | $Q(1)$ | 不稳定 |
| 选择排序 | 直接选择 | $O(n^2)$ | $O(n^2)$ | $O(n^2)$ | $O(1)$ | 不稳定 |
| | 堆排序 | $O(n\log_2 n)$ | $O(n\log_2 n)$ | $O(n\log_2 n)$ | $O(1)$ | 不稳定 |
| 交换排序 | 冒泡排序 | $O(n^2)$ | $O(n)$ | $O(n^2)$ | $O(1)$ | 稳定 |
| | 快速排序 | $Q(n\log_2 n)$ | $O(n\log_2 n)$ | $O(n^2)$ | $O(\log_2 n)$ | 不稳定 |
| 归并排序 | | $O(n\log_2 n)$ | $O(n\log_2 n)$ | $O(n\log_2 n)$ | $O(n)$ | 稳定 |
| 基数排序 | | $O(d(n+r))$ | $O(d(n+r))$ | $O(d(n+r))$ | $O(n+rd)$ | 稳定 |

注:1. 希尔排序的时间复杂度和增量的选择有关。

2. 基数排序的复杂度中,$r$ 代表关键字的基数,$d$ 代表长度,$n$ 代表关键字的个数。

关于时间复杂度：

1. 平方阶（$O(n^2)$）排序

各类简单排序：直接插入、直接选择和冒泡排序。

2. 线性对数阶（$O(n\log_2 n)$）排序

如快速排序、堆排序和归并排序。

3. $O(n+\S)$排序

$\S$ 是介于 0 和 1 的整数，如希尔排序。

4. 线性阶（$O(n)$）排序

如基数排序，此外还有桶、箱排序。

5. 说明

当原表有序或基本有序时，直接插入排序和冒泡排序将大大减少比较次数和移动记录的次数，时间复杂度可降至 $O(n)$。

而快速排序则相反，当原表基本有序时，它将蜕化为冒泡排序，时间复杂度提高为 $O(n^2)$。

表是否有序，对简单选择排序、堆排序、归并排序和基数排序的时间复杂度影响不大。

关于稳定性：

（1）排序算法的稳定性：若待排序的序列中存在多个具有相同关键字的记录，经过排序，这些记录的相对次序保持不变，则称该算法是稳定的；若经排序后记录的相对次序发生了改变，则称该算法是不稳定的。

（2）稳定性的好处：排序算法如果是稳定的，那么从一个键上排序，然后再从另一个键上排序，第一个键排序的结果可以为第二个键排序所用。基数排序就是这样，先按低位排序，逐次按高位排序，低位相同的元素其顺序在高位也相同时是不会改变的。另外，如果排序算法稳定，可以避免多余的比较。

（3）稳定的排序算法：冒泡排序、直接插入排序、归并排序和基数排序。

（4）不稳定的排序算法：选择排序、快速排序、希尔排序、堆排序。

选择排序算法的准则和依据：

（1）准则：每种排序算法各有优缺点。因此，使用时需根据不同情况适当选用，甚至可以将多种方法结合起来使用。

（2）依据：影响排序算法的因素有很多，平均时间复杂度低的算法并不一定就是最优的。相反，有时平均时间复杂度高的算法可能更适合某些特殊情况。同时，选择算法时还得考虑它的可读性，以利于软件的维护。一般而言，需考虑以下 4 个因素。

① 待排序的记录数目 $n$ 的大小；

② 记录本身数据量的大小，也就是记录中除关键字外的其他信息量的大小；

③ 关键字的结构及其分布情况；

④ 对排序稳定性的要求。

设待排序元素的个数为 $n$：

（1）当 $n$ 较大时，可采用时间复杂度为 $O(n\log_2 n)$的排序方法，如快速排序、堆排序或归并排序。

快速排序：是目前基于比较的内部排序中被认为是最好的方法，当待排序的关键字是随机分布时，快速排序的平均时间最短。

堆排序：如果内存空间允许且要求稳定性的；

归并排序：它有一定数量的数据移动，所以可以与插入排序结合，先获得一定长度的序列，然后再合并，这样在效率上会有所提高。

（2）当 $n$ 较小时，可采用直接插入排序或直接选择排序。

直接插入排序：当元素分布有序，直接插入排序将大大减少比较次数和移动记录的次数。

直接选择排序：元素分布有序，如果不要求稳定性，可选直接选择排序。

（3）一般不使用或不直接使用传统的冒泡排序。

（4）基数排序：它是一种稳定的排序算法，但有一定的局限性：

① 关键字可分解；

② 记录的关键字位数较少，如果密集更好；

③ 如果是数字，最好是无符号的，否则将增加相关的映射复杂度，可先将其按正负分开排序。

# 3.1 冒泡排序

## 3.1.1 冒泡排序的基本原理

冒泡排序（Bubble Sort）是一种简单直观的排序算法。它重复地走访要排序的数列，依次比较相邻的数据，将小数据放在前，大数据放在后，即第一趟先比较第 1 个和第 2 个数，大数在后，小数在前，再比较第 2 个数与第 3 个数，大数在后，小数在前，以此类推，则将最大的数“冒泡”到最后一个位置；第二趟则将次大的数滚动到倒数第二个位置……第 $n-1$（$n$ 为无序数据的个数）趟即能完成排序。走访数列的工作是重复地进行，直到不再需要交换，也就是说，该数列已经排序完成。这个算法的名字由来是较小的元素经交换会慢慢“浮”到数列的顶端。

## 3.1.2 冒泡排序的算法步骤

（1）比较相邻的元素。如果第一个数比第二个数大，就交换它们的位置。

（2）对每一对相邻元素做同样的工作，从开始的第一对数据到结尾的最后一对数据。这步做完后，最后的元素会是最大的数。

（3）针对所有元素重复以上步骤，除最后一个。

（4）持续对越来越少的元素重复上面的步骤，直到没有任何一对数需要比较。

什么时候最快：当输入的数据已经是正序时。

什么时候最慢：当输入的数据是反序时。

## 3.1.3 冒泡排序的基本算法实现

```
int main ()
{
    int a[7]={5,2,9,10,3,7,1};
```

```
    int len = sizeof(a)/sizeof(a[0]); //测数组的长度
    int i = 0;
    int j = 0;
    int tmp=0;                        //空瓶子,用于交换数组
    for(j = 0;j < len ;j ++)          //比较多少趟
    {
        for(i =0;i < len-j-1 ;i++)    //第一次少比较 0 个;第二次少比较 1 个;第三次少
                                      //比较 2 个;……;第 i 次就少比较 i-1 个
        {                             //每趟的比较
            if(a[i] > a[i+1])
            {
                tmp = a[i];
                a[i]=a[i+1];
                a[i+1]=tmp;
            }
        }
    }
    for(i =0;i < len ; i ++)
    {
        printf("%d ",a[i]);
    }
    return 0 ;
}
```

### 3.1.4 冒泡排序的优化

当一堆数相对比较整齐时,我们就简化代码,方法就是加一个标志,如果上一轮进去没有做过交换,就不用再继续下一趟的比较。

```
int main ()
{
    int a[7]={5,2,9,10,3,7,1};
    int len = sizeof(a)/sizeof(a[0]);     //测数组的长度
    int i = 0;
    int j = 0;
    int tmp=0;                            //空瓶子,用于交换数组
    bool flag = true;                     //一个标志
    int count = 0;
    for(j = 0;j < len ;j ++)              //比较多少趟
    {
        flag = true;                      //判断完了,就恢复真
        for(i =0;i < len-j-1 ;i++)        //第一次少比较 0 个;第二次少比较 1 个;第三
                                          //次少比较 2 个;……;第 i 次就少比较 i-1 个
        {                                 //每趟的比较
            if(a[i] > a[i+1])
            {
                count ++;
                tmp = a[i];
                a[i]=a[i+1];
                a[i+1]=tmp;
```

```
                flag = false;                    //如果交换过了,就返回 false,标志交换过
            }

        }
        if(flag)
        {
            break;
        }

    }
    for(i =0;i < len ; i ++)
    {
        printf("%d ",a[i]);
    }

    return 0 ;
}
```

当数相对比较整齐时,明显可以看出效率提高了。

## 3.2 快速排序

### 3.2.1 快速排序的基本原理

同冒泡排序一样,快速排序也属于交换排序,它通过元素之间的比较和交换位置达到排序的目的。

不同的是,冒泡排序在每一轮只把一个元素冒泡到数列的一端,而快速排序在每一轮挑选一个基准元素,并让其他比它大的元素移动到数列一边,比它小的元素移动到数列的另一边,从而把数列拆解成两部分。

### 3.2.2 快速排序算法的步骤

快速排序算法通过多次比较和交换实现排序,其排序流程如下。

(1) 首先设定一个分界值,通过该分界值将数组分成左、右两部分。

(2) 将大于或等于分界值的数据集中到数组右边,将小于分界值的数据集中到数组的左边。此时,左边部分中各元素都小于或等于分界值,而右边部分中各元素都大于或等于分界值。

(3) 然后,左边和右边的数据可以独立排序。左侧的数组数据,又可以取一个分界值,将该部分数据分成左、右两部分,同样在左边放置较小值,在右边放置较大值。右侧的数组数据也可以做类似处理。

(4) 重复上述过程,可以看出,这是一个递归定义。通过递归将左侧部分排好序后,再递归排好右侧部分的顺序。当左、右两部分各数据排序完成后,整个数组的排序也就完成了。

### 3.2.3 快速排序的基本算法实现

```
//快速排序
#include<iostream>
```

```
using namespace std;
const int N = 1e5 + 10;
int n;
int q[N];
void quick_sort(int q[], int l, int r) {
    if (l >= r) {              //判边界，如果区间中只有一个数字,或没有数字，就直接返回
        return;
    }
    int x = q[(l + r) >> 1];                                    //分界点
    int i = l - 1, j = r + 1;
    while (i < j) {
        do i++; while (q[i] < x);
        do j--; while (q[j] > x);
        if (i < j) {
            swap(q[i], q[j]);
        }
    }
    quick_sort(q, l, j), quick_sort(q, j + 1, r);          //递归处理左右两段
}

int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> q[i];
    }
    quick_sort(q, 0, n - 1);
    for (int i = 0; i < n; i++) {
        if (i > 0) {
            cout << " ";
        }
        cout << q[i];
    }
    cout << endl;
    return 0;
}
```

## 3.3 其他排序

排序的算法还有很多，如堆排序、归并排序和桶排序等。有兴趣的读者可以查找相关资料和文献进行了解和学习。

## 3.4 实例演示

### 3.4.1 出现次数超过一半的数

问题描述：

某校的惯例是在每学期的期末考试之后发放奖学金。发放的奖学金共有 5 种，获取的条件各自不同：

(1) 院士奖学金，每人 8000 元，期末平均成绩高于 80 分(>80)，并且在本学期内发表 1 篇或 1 篇以上论文的学生均可获得；

(2) 五四奖学金，每人 4000 元，期末平均成绩高于 85 分(>85)，并且班级评议成绩高于 80 分(>80)的学生均可获得；

(3) 成绩优秀奖，每人 2000 元，期末平均成绩高于 90 分(>90)的学生均可获得；

(4) 西部奖学金，每人 1000 元，期末平均成绩高于 85 分(>85)的西部省份学生均可获得；

(5) 班级贡献奖，每人 850 元，班级评议成绩高于 80 分(>80)的学生干部均可获得。

只要符合条件就可以得奖，每项奖学金的获奖人数没有限制，每名学生也可以同时获得多项奖学金。例如，姚林的期末平均成绩是 87 分，班级评议成绩 82 分，同时他还是一位学生干部，那么他可以同时获得五四奖学金和班级贡献奖，奖金总额是 4850 元。

现在给出若干学生的相关数据，请计算哪些同学获得的奖金总额最高(假设总有同学能满足获得奖学金的条件)。

输入格式：

在第一行输入一个整数 $N$($1 \leqslant N \leqslant 100$)，表示学生总数。接下来的 $N$ 行每行是一位学生的数据，从左向右依次是姓名、期末平均成绩、班级评议成绩、是否是学生干部、是否是西部省份学生，以及发表的论文数。姓名是由大小写英文字母组成的长度不超过 20 的字符串(不含空格)；期末平均成绩和班级评议成绩都是 0～100 的整数(包括 0 和 100)；是否是学生干部和是否是西部省份学生分别用一个字符表示，Y 表示是，N 表示不是；发表的论文数是 0 到 10 的整数(包括 0 和 10)。每两个相邻数据项之间用一个空格分隔。

输出格式：

输出包括三行，第一行是获得最多奖金的学生的姓名，第二行是这名学生获得的奖金总额。如果有两名或两名以上的学生获得的奖金最多，请输出他们中在输入文件中出现最早的学生的姓名。第三行是这 $N$ 个学生获得的奖学金总额。

输入样例：

```
4
YaoLin 87 82 Y N 0
ChenRuiyi 88 78 N Y 1
LiXin 92 88 N N 0
ZhangQin 83 87 Y N 1
```

输出样例＃1：

```
ChenRuiyi
9000
28700
```

参考程序：

```
#include<iostream>
#include<cstring>
using namespace std;
```

```
int main()
{
    int n;
    char cadre,west;
    int avr_grade,class_grade,paper;
    int prize=0,max=0,sum=0;
    string name,max_name=" ";
    int i;

    cin>>n;
    for(i=1;i<=n;i++)
    {
        cin>>name>>avr_grade>>class_grade>>cadre>>west>>paper;
                                                    //输入每个人的信息

        /*按要求进行奖学金汇总*/
        if( avr_grade>80 && paper>0)  prize+=8000;
        if( avr_grade>85 && class_grade>80)   prize+=4000;
        if( avr_grade>90 )   prize+=2000;
        if( avr_grade>85 && west=='Y')   prize+=1000;
        if( class_grade>80 && cadre=='Y')    prize+=850;

        sum+=prize;                                 //计算奖学金总额
        if( prize>max || (sum==0&&sum==max) )       //记录最大奖学金
        {
            max=prize;
            max_name=name;
        }
        prize=0;
    }

    /*数据输出*/
    cout<<max_name<<endl;
    cout<<max<<endl;
    cout<<sum<<endl;

    return 0;
}
```

## 3.4.2 奖学金发放

问题描述：

给出一个含有 $n(0<n\leqslant 1000)$ 个整数的数组，请找出其中出现次数超过一半的数。数组中的数大于 $-50$ 且小于 50。

输入：

第一行包含一个整数 $n$，表示数组大小；

第二行包含 $n$ 个整数，分别是数组中的每个元素，相邻两个元素之间用单个空格隔开。

输出：

若存在这样的数，则输出这个数；否则输出 no。

输入样例：

```
3
1 2 2
```

输出样例：

```
2
```

参考程序：

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
int main()
{
    int a[101]={0};
    int n,b;
    int i;
    bool flag=false;

    cin>>n;
    for(i=0;i<n;i++)
    {
        cin>>b;
        a[b+50]++;
    }
    for(i=0;i<100;i++)
    {
        if(a[i]>=n/2)
        {
            flag=true;
            cout<<i-50<<endl;
        }
    }
    if(flag==0)
        cout<<"no";
    cout<<endl;
    return 0;
}
```

### 3.4.3 魔法照片

问题描述：

一共有 $n$（$n \leqslant 20000$）个人（以 1～$n$ 编号）向佳佳要照片，而佳佳只能把照片给其中的 $k$ 个人。佳佳按照自己与他们关系好坏的程度给每个人赋予了一个初始权值 $W[i]$。然后将初始权值从大到小进行排序，每人就有了一个序号 $D[i]$（取值同样是 1～$n$）。按照这个序号对 10 取模的值将这些人分为 10 类。也就是说，定义每个人的类别序号 $C[i]$的值为 $(D[i-1] \bmod 10+1$，显然类别序号的取值为 1～10。第 $i$ 类的人将会额外得到 $E[i]$的权

值。你需要做的就是求出加上额外权值以后，最终权值最大的 $k$ 个人，并输出他们的编号。在排序中，如果两人的 $W[i]$相同，编号小的优先。

输入格式：

第一行输入用空格隔开的两个整数，分别是 $n$ 和 $k$。

第二行给出 10 个正整数，分别是 $E[1]$到 $E[10]$。

第三行给出 $n$ 个正整数，第 $i$ 个数表示编号为 $i$ 的人的权值 $W[i]$。

输出格式：

输出一行用空格隔开的 $k$ 个整数，分别表示最终的 $W[i]$从高到低的人的编号。

输入样例：

```
10 10
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
```

输出样例＃1：

```
10 9 8 7 6 5 4 3 2 1
```

思路：输入→排号→排序→权值处理→排序→输出。

参考程序：

```
#include<iostream>
#include<algorithm>
using namespace std;
int extra[11],initial[20001],order[20001];
bool cmp(int a,int b)
{
    if(initial[a]==initial[b])  return a<b;              //从大到小排序
    else    return initial[a]>initial[b];                //序号小的优先
}
int main()
{
    int n,k;
    int i;
    cin>>n>>k;
    for(i=1;i<=10;i++)   cin>>extra[i];
    for(i=1;i<=n;i++)
    {
        cin>>initial[i];
        order[i]=i;
    }
    sort(order+1,order+n+1,cmp);                         //第一次排序
    for(i=1;i<=n;i++)                                    //分类处理
        initial[order[i]]+=extra[(i-1)%10+1];
    sort(order+1,order+n+1,cmp);                         //第二次排序
    for(i=1;i<=k;i++)
        cout<<order[i]<<" ";
    cout<<endl;
    return 0;
}
```

### 3.4.4 输出前 k 大的数

问题描述：

给定一个数组，统计前 $k$ 大的数并且把这 $k$ 个数从大到小输出。

输入：

第一行包含一个整数 $n$，表示数组的大小，$n<100000$。

第二行包含 $n$ 个整数，表示数组的元素，整数之间以一个空格分开。每个整数的绝对值不超过 100000000。

第三行包含一个整数 $k$，$k<n$。

输出：

从大到小输出前 $k$ 大的数，每个数一行。

输入样例：

```
10
4 5 6 9 8 7 1 2 3 0
5
```

输出样例：

```
9
8
7
6
5
```

参考程序：

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<cstring>
#include<algorithm>
#include<string>
#define INF 999999999
#define N 1000001
#define MOD 1000000007
using namespace std;

int a[N];

int cmp(const void *a,const void *b){
    return (*(int *)a) - (*(int *)b);
}

void Find(int st,int ed,int k){
    if(st-ed+1==k)
        return;

    int i=st,j=ed,key=a[st];
```

```
    while(i<j){
        while(i<j&&a[j]>=key)
            j--;
        a[i]=a[j];
        while(i<j&&a[i]<=key)
            i++;
        a[j]=a[i];
    }
    a[i]=key;
    if(ed-i+1==k)
        return;
    else if(ed-i+1>k)
        Find(i+1,ed,k);
    else
        Find(st,i-1,k-(ed-i+1));
}
int main(){
    int n,k;
    scanf("%d",&n);
    for(int i=0; i<n; i++)
        scanf("%d",&a[i]);
    scanf("%d",&k);

    Find(0,n-1,k);

    qsort(a+n-k,k,sizeof(a[0]),cmp);
    for(int i=n-1; i>=n-k; i--)
        printf("%d\n",a[i]);
    return 0;
}
```

### 3.4.5 不重复地输出数

问题描述：

输入 $n$ 个数，从小到大将它们输出，重复的数只输出一次，保证不同的数不超过 500 个。

输入：

第一行是一个整数 $n$，$1\leqslant n\leqslant 100000$。

之后 $n$ 行，每行一个整数，整数大小在 int 范围内。

输出：

一行，从小到大不重复地输出这些数，相邻两个数之间用单个空格隔开。

输入样例：

```
5
2 4 4 5 1
```

输出样例：

```
1 2 4 5
```

参考程序：

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<cstring>
#include<cmath>
#include<algorithm>
#include<string>
#define INF 999999999
#define N 1000001
#define MOD 1000000007
#define E 1e-3
using namespace std;
int a[N];
int main()
{
    int n;
    cin>>n;
    for(int i=1;i<=n;i++)
        cin>>a[i];
    sort(a+1,a+1+n);
    cout<<a[1];
    for(int i=2;i<=n;i++)
        if(a[i]!=a[i-1])
            cout<<" "<<a[i];
    cout<<endl;
    return 0;
}
```

### 3.4.6 单词排序

问题描述：

输入一行单词序列，相邻单词之间由 1 个或多个空格间隔，请按照字典序输出这些单词，要求重复的单词只输出一次（区分大小写）。

输入：

一行单词序列，最少 1 个单词，最多 100 个单词，每个单词长度不超过 50，单词之间用至少 1 个空格间隔。数据不含除字母、空格外的其他字符。

输出：

按字典序输出这些单词，重复的单词只输出一次。

输入样例：

```
She wants to go to Peking University to study Chinese
```

输出样例：

```
Chinese
Peking
She
University
go
```

```
study
to
wants
```

参考程序：

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<string>
#include<algorithm>
using namespace std;
int main()
{
    string a[100];
    int k=0;
    bool flag;
    int i;

    while(cin>>a[k])
    {
        flag=false;
        for(i=0;i<k;i++)
        {
            if(a[i].compare(a[k])==0)
            {
                flag=true;
                break;
            }
        }
        if(!flag)
            k++;
    }
    sort(a,a+k);
    for(i=0;i<k;i++)
        cout<<a[i]<<endl;
    return 0;
}
```

### 3.4.7 快速排序

问题描述：

利用快速排序算法将读入的 $N$ 个数从小到大排序后输出。

快速排序是信息学竞赛的必备算法之一。对快速排序不是很了解的同学可以自行上网查询相关资料，掌握后独立完成。C++ 选手请不要试图使用 STL(标准模板库)，虽然你可以使用 sort()函数一遍过，但是你并没有掌握快速排序算法的精髓。

输入格式：

输入的第 1 行为一个正整数 $N$，第 2 行包含 $N-1$ 个空格隔开的正整数 $a[i]$，为需要进行排序的数，保证 $a[i]$不超过 1000000000。

输出格式：

将给定的 $N$ 个数从小到大输出，数之间用空格隔开，行末换行且无空格。

输入样例：

```
5
4 2 4 5 1
```

输出样例：

```
1 2 4 4 5
```

参考程序：

```
#include<iostream>
using namespace std;
void quick_sort(int left,int right);
int a[100001];
int main()
{
    int n,i;
    cin>>n;
    for(i=0;i<n;i++) cin>>a[i];
    quick_sort(0,n);
    for(i=1;i<=n;i++)
        cout<<a[i]<<" ";
    cout<<endl;
    return 0;
}

void quick_sort(int left,int right)
{
    int i=left,j=right;
    int mid,temp;

    mid=a[(i+j)/2];

    while(i<j)
    {
        while(a[i]<mid)  i++;
        while(a[j]>mid)  j--;

        if(i<=j)
        {
            temp=a[i];a[i]=a[j];a[j]=temp;
            i++;j--;
        }
    }
    if(left<j)   quick_sort(left,j);
    if(right>i)  quick_sort(i,right);
}
```

### 3.4.8 第 k 个数

给定一个长度为 $n$ 的整数数列，以及一个整数 $k$，请用快速选择算法求出数列从小到大排序后的第 $k$ 个数。

输入格式：

第一行包含两个整数 $n$ 和 $k$。

第二行包含 $n$ 个整数(所有整数均在 $1\sim10^9$ 范围内)，表示整数数列。

输出格式：

输出一个整数，表示数列的第 $k$ 个数。

数据范围：

$1\leqslant n\leqslant 100000$，

$1\leqslant k\leqslant n$。

输入样例：

```
5 3
2 4 1 5 3
```

输出样例：

```
3
```

参考程序：

```
//第 k 个数
#include<bits/stdc++.h>
using namespace std;
int a[1000000];
int n,k;
int quicksort(int a[],int begin,int end,int k)
{
    if(begin>=end)
        return a[begin];
    int mid=a[(begin+end)/2];
    int i=begin-1,j=end+1;          //这里如果不是使用 do while 循环实现(因为 do
                                    //while 循环是运行后判断),用 while 时需要改
                                    //成 i=begin,j=end;
    while(i<j)
    {
        do{
            i++;
        }while(a[i]<mid);
        do{
            j--;
        }while(a[j]>mid);           //以上是快速排序,排序的过程是先右后左
        if(i<j)
        {
            swap(a[i],a[j]);        //swap 函数用于交换 a[i]与 a[j]的 value
        }
    }
```

```
    if(j-begin+1>=k) return quicksort(a,begin,j,k);
                              //如果第 k 个数在左边,就直接递归左边的数据,继续快速排序
    else return quicksort(a,j+1,end,k-(j-begin+1));
                              //如果第 k 个数在右边,就直接递归右边的数据,继续快速排序
}
int main()
{

    scanf("%d %d",&n,&k);
    for(int i=0;i<n;i++)
    {
        scanf("%d",a[i]);
    }
    printf("%d",quicksort(a,0,n-1,k));

}
```