第9讲 排序和查找

教材8.4节

MOOC第8周

哈尔滨工业大学

苏小红

为什么这两类算法很重要？

- 排序（Sorting）和查找（Searching）算法
  - 计算科学中很重要的算法，很多复杂算法的基础
  - 与我们的生活息息相关
    - n个大小不同的螺帽和n个大小不同的螺栓原本是相互匹配的，但现在不小心把它们混在一起了，如何实现螺帽和螺栓的快速匹配？

网页全文搜索引擎

100,000,000条检索记录

9,530,000条检索记录

Google和百度要完成的任务是什么？

- 在“世界上最大的一堆稻草”中寻找你想要的“绣花针”
  - 使每个人与问题答案之间只有点击一下鼠标那么远
  - 这么大的信息量，这么快的速度——习以为常，其实“不简单”！
    - 不仅仅是它拥有数量庞大的服务器

谢尔盖布林

拉里-佩奇

1996

Google的背后是什么？

- 搜索引擎要完成的两件最重要的事情
  - matching和ranking
- 搜索的第一步： matching
  - 搜索的关键词在哪？哪些网页与查询匹配？
  - 怎样快速匹配？
    - 20世纪90年代中期的“搜索之王”AltaVista
    - 索引技术 → 高效的匹配和web搜索
    - forward index：文档到关键词的映射，文档 → 关键词 （检索困难）
    - Inverted index——倒排索引（反向索引）
      - 关键词到文档的映射， 关键词 → 文档

谢尔盖布林

拉里-佩奇

Google成功的法宝？

- 你希望这些检索记录以怎样的方式呈现给你?
- 搜索的第二步： ranking
- Google靠什么打败了AltaVista?
  - PageRank网页排序算法
    - 公司成立仅仅3个月，就被PC Magazine评为当年（1998）Top 100网站
    - 理由:“uncanny knack for returning extremely relevant results”
  - 招数1：ranking by hyperlink number（数量）
  - 招数2：ranking by authorty of hyperlink（质量，主题无关）
    - 将最相关、最重要的结果放在搜索结果顶端——排序
    - Learning to Rank——用机器学习来解决排序问题


谢尔盖布林


拉里-佩奇

线性查找（Linear Search）

- 不要求数据表有序
  - 依次将记录的关键字与查找关键字（key）进行比较

学号	姓名	成绩
1173710128	何一夫	60
1163450201	江建东	59
1173710105	曾钰城	55
1173710135	刘强	55
1173710212	韩紫嫣	55
1173710132	牟虹霖	55
1173710231	李松源	54
1173710119	关威	52
1173710111	张淑慧	50
1173710228	许文滨	50

```
int LinSearch(long num[], long x, int n)
{
    int i;
    for (i=0; i<n; i++)
    {
        if (num[i] == x)
        {
            return i;
        }
    }
    return -1;
}
```

最好情况：1次

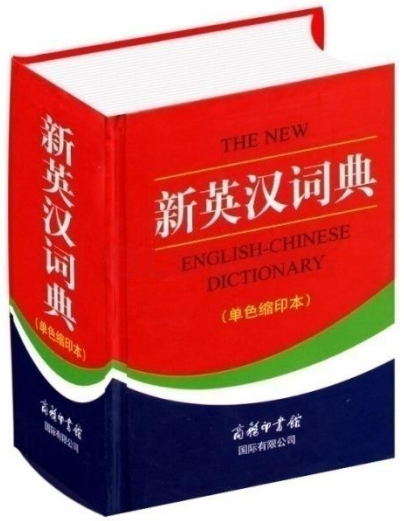
最坏情况：n次

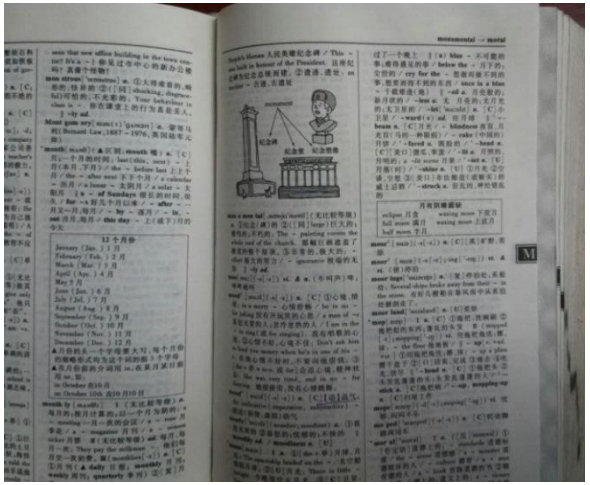
平均情况：n/2次

```
……
printf("Input the searching ID:");
scanf("%ld", &x);
pos = LinSearch(num, x, n);
if (pos != -1)
  printf("score=%d\n", score[pos]);
else
  printf("Not found!\n");
```

如何找得更快？

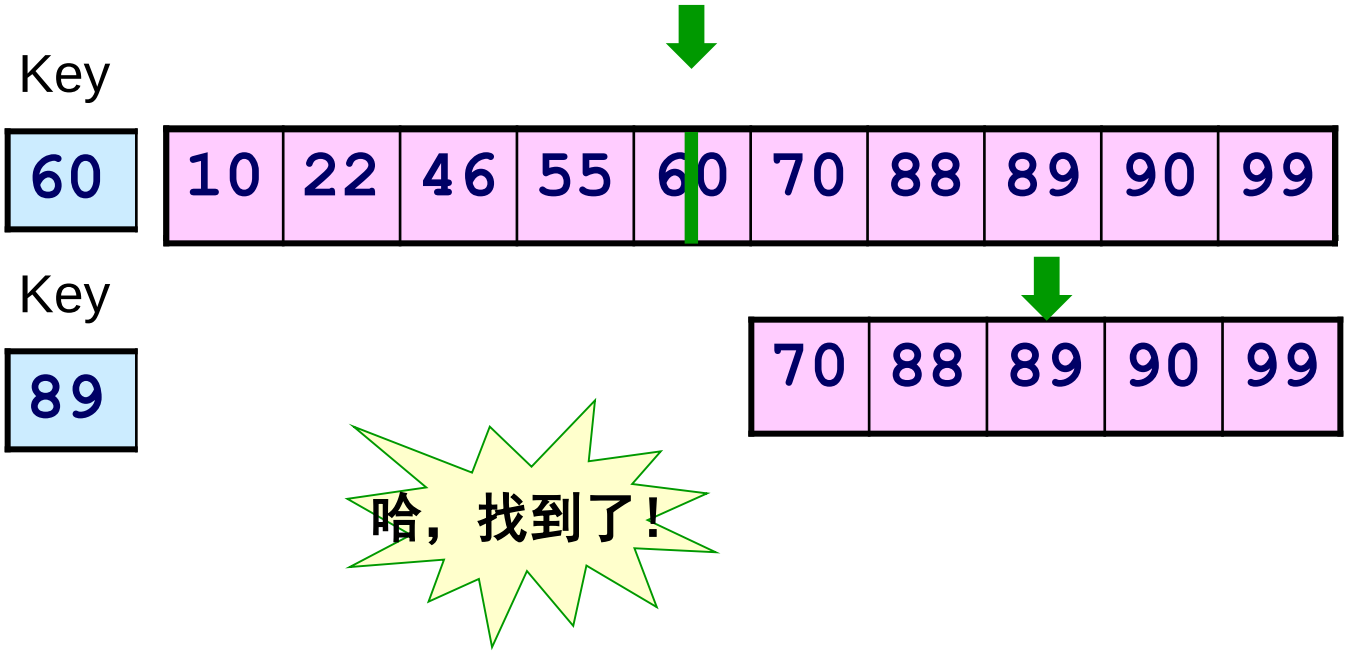
- 当你登录QQ时，QQ服务器要核对你的身份，在庞大的用户群中快速找到你的信息，会用线性查找的方法吗？
- 你是怎样查字典的？





二分查找（Binary Search）

- 要求数据表有序
  - 先将表的中间位置记录的关键字与查找关键字比较
    - 如果两者相等，则查找成功
    - 否则将表分成前、后两个子表，根据比较结果，决定查找哪个子表



二分查找（Binary Search）

何时确定没找到?

mid=(low+high)/2
=4

low=0

high=9

Key

65

0	1	2	3	4	5	6	7	8	9
10	22	46	55	60	70	88	89	90	99

low=mid+1

70	88	89	90	99

high=mid-1

high=mid-1

70	88

low<=high为假

即low>high为真

二分查找的递归实现

```
int BinSearch(long num[], long x, int low, int high)
{
    int mid = (high + low) / 2;
    if (low > high)
    {
        return -1; //没找到
    }
    if (x > num[mid])
    {
        return BinSearch(num, x, mid+1, high); //在后一子表查找
    }
    else  if (x < num[mid])
    {
        return BinSearch(num, x, low, mid-1);  //在前一子表查找
    }
    return mid;    //返回找到的位置下标
}
```

二分查找的迭代实现

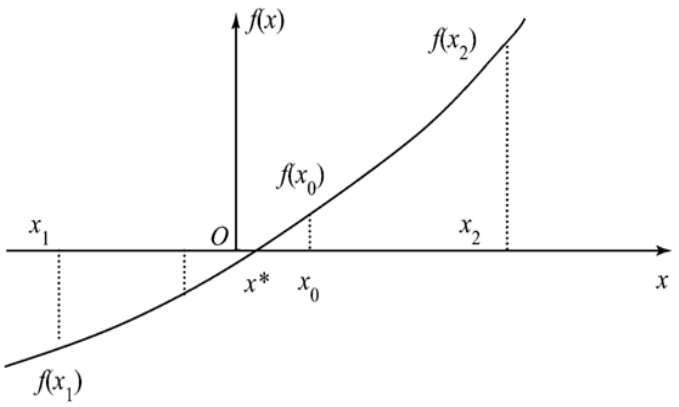
```
int BinSearch(long num[],long x,int n)
{
   int  low = 0, high = n - 1, mid;
   while (low <= high)
   {
        mid = (high + low) / 2;
        if (x > num[mid])
        {
           low = mid + 1; //在后一子表查找
        }
        else  if (x < num[mid])
        {
           high = mid - 1; //在前一子表查找
        }
        else
        {
           return mid; //返回找到的位置下标
        }
   }
   return -1; //没找到
}
```

- 这个程序存在什么漏洞？
- 防止溢出的解决方案
  - 修改计算中间值的方法，用减法代替加法

```
mid = low + (high - low) / 2;
```

二分查找（Binary Search）

- 优点
  - 比较次数少，查找速度快，平均性能好
  - 每执行一次，都将查找空间减少一半
  - 计算机科学中分治思想的完美体现
- 缺点
  - 要求待查表按关键字有序排列
  - 必须顺序存储，增删数据移动量大
  - 适用于不常变动而频繁查找的有序表
- 应用
  - 猜数游戏
  - 计算方程的根



“萝卜”查询

- 某单位有10个职位对外招聘（编号0~9），输入n个应聘者选择的职位编号，统计输出没有应聘者报名的职位编号
  - 例如，输入2 5 2 1 8，输出0 3 4 6 7 9

一个萝卜一个坑

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0	0	0
0	0	2	0	0	1	0	0	0	0
0	1	2	0	0	1	0	0	0	0
0	1	2	0	0	1	0	0	1	0

“萝卜”排序

- 从键盘输入0~9之间的5个数，升序排序
  - 例如，输入2 5 2 1 8，输出1 2 2 5 8

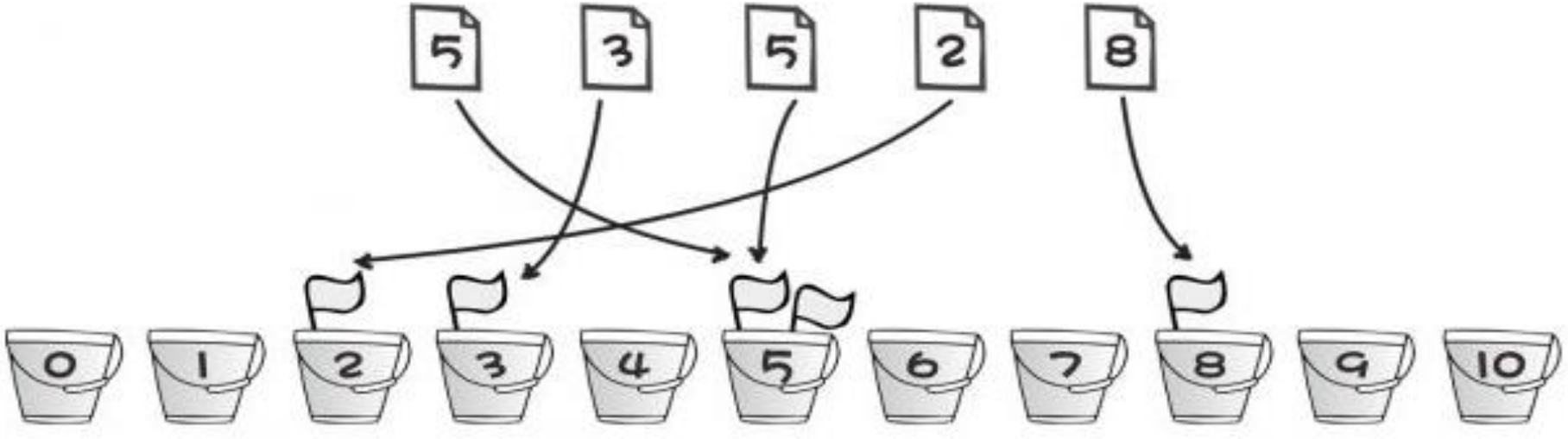
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0	0	0
0	0	2	0	0	1	0	0	0	0
0	1	2	0	0	1	0	0	0	0
0	1	2	0	0	1	0	0	1	0

若降序排序呢？

简化版的桶排序

一个萝卜一个坑

- 从键盘输入0~9之间的5个数，升序排序

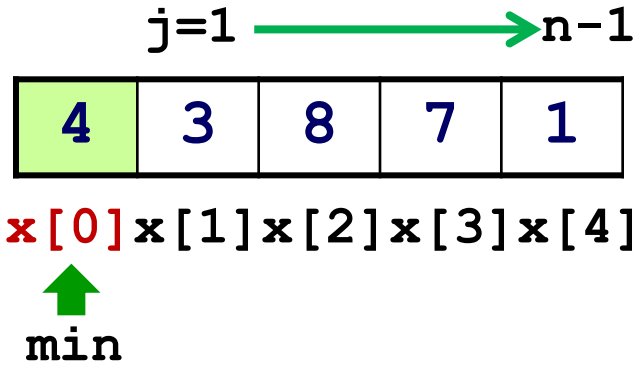


存在缺陷，且本质上还不能算
是一个真正意义上的排序算法

你知道求最值和排序有什么关系吗？

```
int FindMin(int x[], int n)
{
  int min, j;

  ......

    返回最小值min



  return min;
}
```

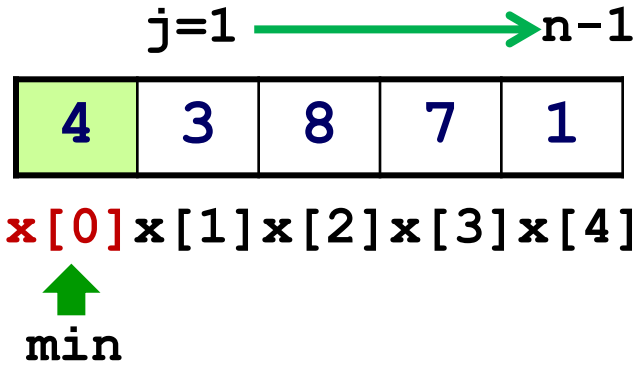
```
#include <stdio.h>
int FindMin(int x[], int n);
int main()
{
   int  a[5]={4,3,8,7,1}, min, n=5;

   min = FindMin(_______);

   printf("min=%d\n", min);
   return 0;
}
```



你知道求最值和排序有什么关系吗？

```
int FindMin(int x[], int n)
{
  int min, j;
  min = x[0]; //假设x[0]最小
  for (j=1; j<n; j++)
  {
    if (x[j] < min)
    {
        min = x[j];
    }
  }
  return min;
}
```

```
void FindMin(int x[], int n)
{
    int temp, j;
    for (j=1; j<n; j++)
    {
        if (x[j] < x[0])
        {
            temp = x[j];
            x[j] = x[0];
            x[0] = temp;
        }
    }
}
```

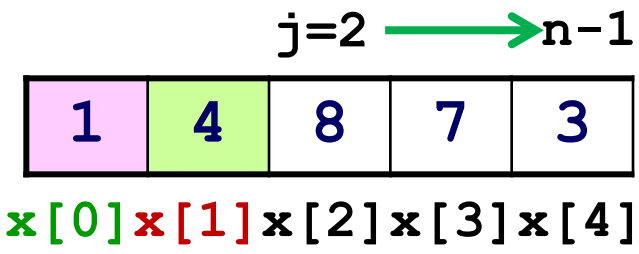


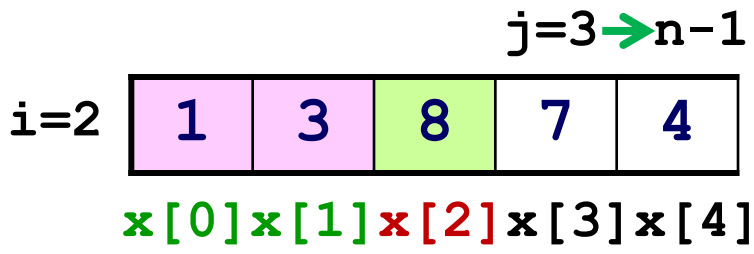
如果想在比较的过程中把最小值保存到x[0]里，怎么办？

你知道求最值和排序有什么关系吗？

```
void FindMin(int x[], int n)
{
    int temp, j;
    for (j=2; j<n; j++)
    {
        if (x[j] < x[1])
        {
            temp = x[j];
            x[j] = x[1];
            x[1] = temp;
        }
    }
}
```

```
void FindMin(int x[], int n)
{
    int temp, j;
    for (j=3; j<n; j++)
    {
        if (x[j] < x[2])
        {
            temp = x[j];
            x[j] = x[2];
            x[2] = temp;
        }
    }
}
```





你知道求最值和排序有什么关系吗？

```
void ExchangeSort(int x[],int n)
{
   int i, j, temp;
   for (i=0; i<n-1; i++)
   {
        for (j=i+1; j<n; j++)
        {
            if (x[j] < x[i])//升序
            {
                temp = x[j];
                x[j] = x[i];
                x[i] = temp;
            }
        }
   }
}
```

交换排序

```
void FindMin(int x[], int n)
{
    int temp, j;
    for (j=3; j<n; j++)
    {
        if (x[j] < x[2])
        {
            temp = x[j];
            x[j] = x[2];
            x[2] = temp;
        }
    }
}
```

n个数需要做多少次这样的求最小值？

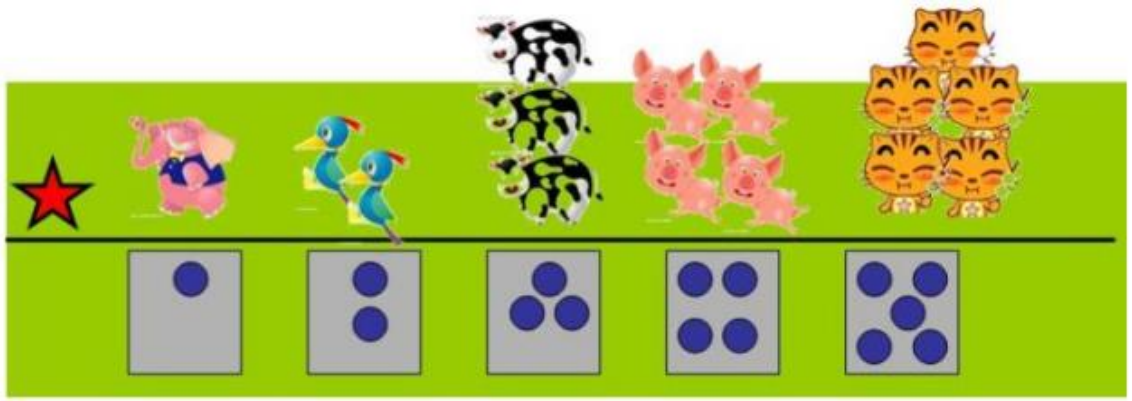
j=3→n-1

1	3	8	7	4

x[0]x[1]x[2]x[3]x[4]

排序算法

1. 交换法 → 2. 选择法

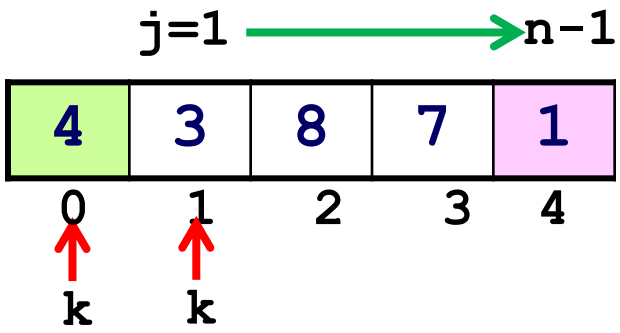


你知道求最值和选择法排序有什么关系吗？

```
int FindMin(int x[], int n)
{
  int j, k;

  ......

  return k;
}
```

返回最小值所在下标位置k

```
#include <stdio.h>
int FindMin(int x[], int n);
int main()
{
   int  a[5]={4,3,8,7,1}, pos, n=5;

   pos = FindMin(_______);

   printf("min=%d\n", a[pos]);
   return 0;
}
```

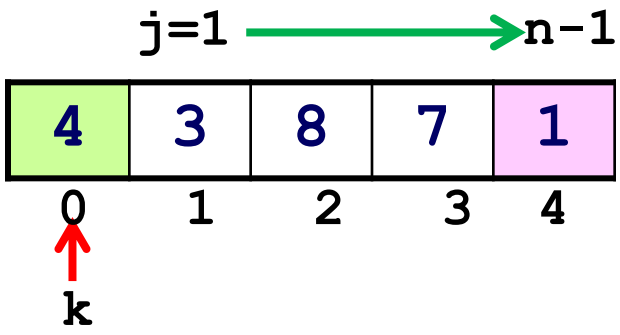


求最小值及其所在的下标位置

你知道求最值和选择法排序有什么关系吗？

```
int FindMin(int x[], int n)
{
  int j, k;
  k = 0;   //假设最小值的下标为0
  for (j=1; j<n; j++)
  {
    if (x[j] < x[k])
    {
        k = j;
    }
  }
  return k;
}
```

返回下标位置

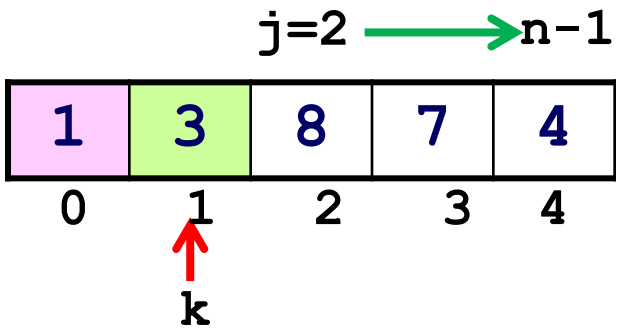



```
void FindMin(int x[], int n)
{
    int temp, j, k;
    k = 0;
    for (j=1; j<n; j++)
    {
        if (x[j] < x[k])
        {
            k = j;
        }
    }
    if (k != 0)
    {
        temp = x[k];
        x[k] = x[0];
        x[0] = temp;
    }
}
```

如何把最小值保存到x[0]里？

你知道求最值和选择法排序有什么关系吗？

```
void FindMin(int x[], int n)
{
    int temp, j, k;
    k = 0;
    for (j=1; j<n; j++)
    {
        if (x[j] < x[k])
        {
            k = j;
        }
    }
    if (k != 0)
    {
        temp = x[k];
        x[k] = x[0];
        x[0] = temp;
    }
}
```




```
void FindMin(int x[], int n)
{
    int temp, j, k;
    k = 1;
    for (j=2; j<n; j++)
    {
        if (x[j] < x[k])
        {
            k = j;
        }
    }
    if (k != 1)
    {
        temp = x[k];
        x[k] = x[1];
        x[1] = temp;
    }
}
```

如何把剩余的最小值保存到x[1]里？

你知道求最值和选择法排序有什么关系吗？

```
void SelectionSort(int x[],int n)
{
    int temp, i, j, k;
    for (i=0; i<n-1; i++)
    {
       k = i;//假设剩余序列第一个数(下标为i)最小
       for (j=i+1; j<n; j++)
       {
         if (x[j] < x[k])
         {
            k = j;
         }
       }
       if (k != i)
       {
          temp = x[k];
          x[k] = x[i];
          x[i] = temp;
       }
    }
}
```

```
void FindMin(int x[], int n)
{
    int temp, j, k;
    k = 1;
    for (j=2; j<n; j++)
    {
        if (x[j] < x[k])
        {
            k = j;
        }
    }
    if (k != 1)
    {
        temp = x[k];
        x[k] = x[1];
        x[1] = temp;
    }
}
```

n个数需要做n-1次这样的求最小值，每次放到x[i]中

```
void SortScore(int score[], long num[],int n)
{
    int i, j, k, temp1;
    long temp2;
    for (i=0; i<n-1; i++)
    {
        k = i;
        for (j=i+1; j<n; j++)
        {
            if (score[j] > score[k])
            {
                k = j;
            }
        }
        if (k != i)
        {
            temp1 = score[k];
            score[k] = score[i];
            score[i] = temp1;
            temp2 = num[k];
            num[k] = num[i];
            num[i] = temp2;
        }
    }
}
```

成绩降序

学号随成绩一起变化

学号	姓名	成绩
1173710128	何一夫	60
1163450201	江建东	59
1173710105	曾钰城	55
1173710135	刘强	55
1173710212	韩紫嫣	55
1173710132	牟虹霖	55
1173710231	李松源	54
1173710119	关威	52
1173710111	张淑慧	50
1173710228	许文滨	50

```
void SortNum(int score[], long num[],int n)
{
    int i, j, k, temp1;
    long temp2;
    for (i=0; i<n-1; i++)
    {
        k = i;
        for (j=i+1; j<n; j++)
        {
            if (num[j] < num[k])
            {
                k = j;
            }
        }
        if (k != i)
        {
            temp2 = num[k];
            num[k] = num[i];
            num[i] = temp2;
            temp1 = score[k];
            score[k] = score[i];
            score[i] = temp1;
        }
    }
}
```

学号升序

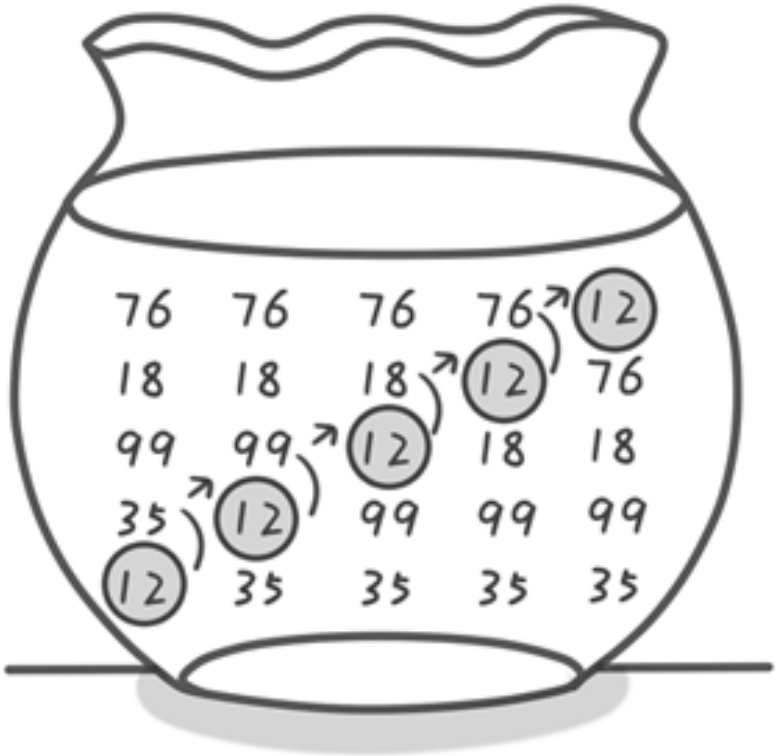
成绩随学号一起变化

学号	姓名	成绩
1173710128	何一夫	60
1163450201	江建东	59
1173710105	曾钰城	55
1173710135	刘强	55
1173710212	韩紫嫣	55
1173710132	牟虹霖	55
1173710231	李松源	54
1173710119	关威	52
1173710111	张淑慧	50
1173710228	许文滨	50

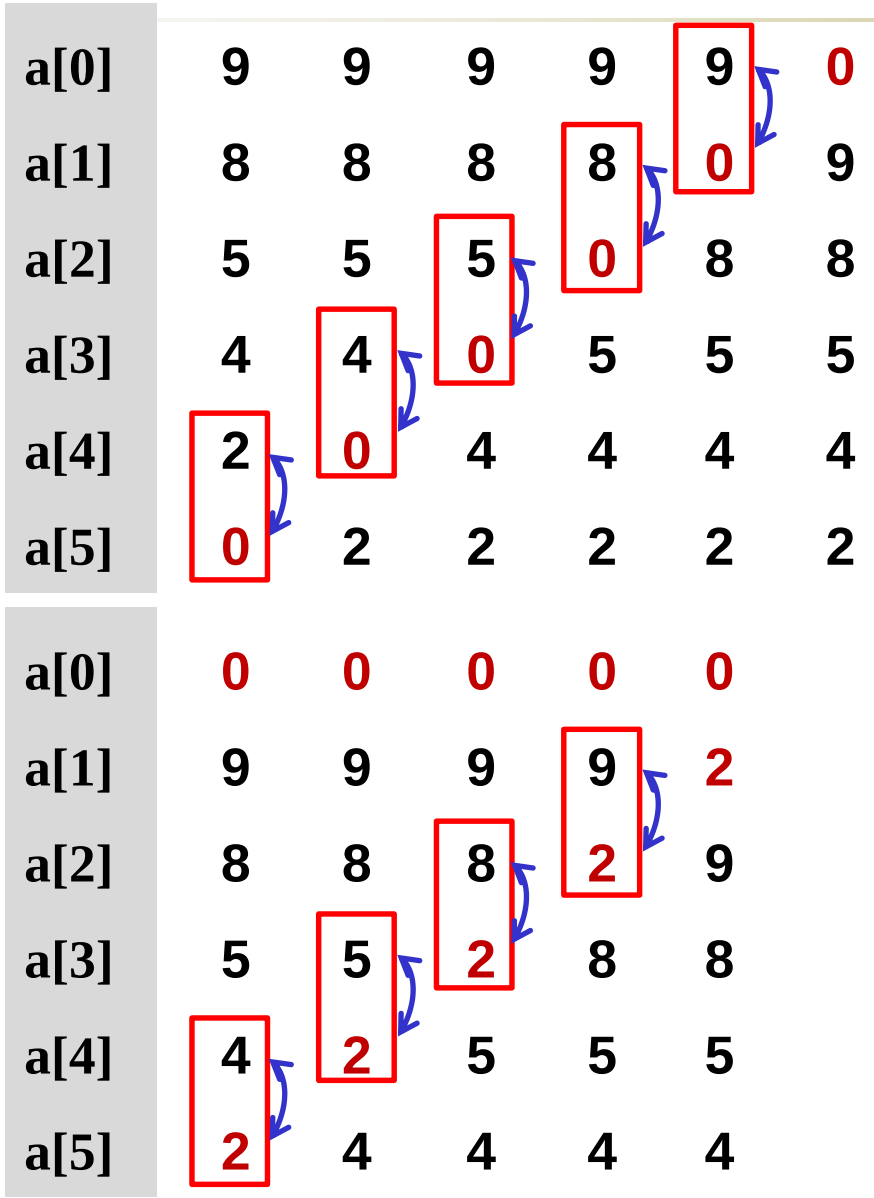
学号	姓名	成绩
1163450201	江建东	59
1173710105	曾钰城	55
1173710111	张淑慧	50
1173710119	关威	52
1173710128	何一夫	60
1173710132	牟虹霖	55
1173710135	刘强	55
1173710212	韩紫嫣	55
1173710228	许文滨	50
1173710231	李松源	54

排序算法

3. 冒泡法


76 76 76 76 12
18 18 18 12 76
99 99 12 18 18
35 12 99 99 99
12 35 35 35 35

冒泡法排序（以升序为例）

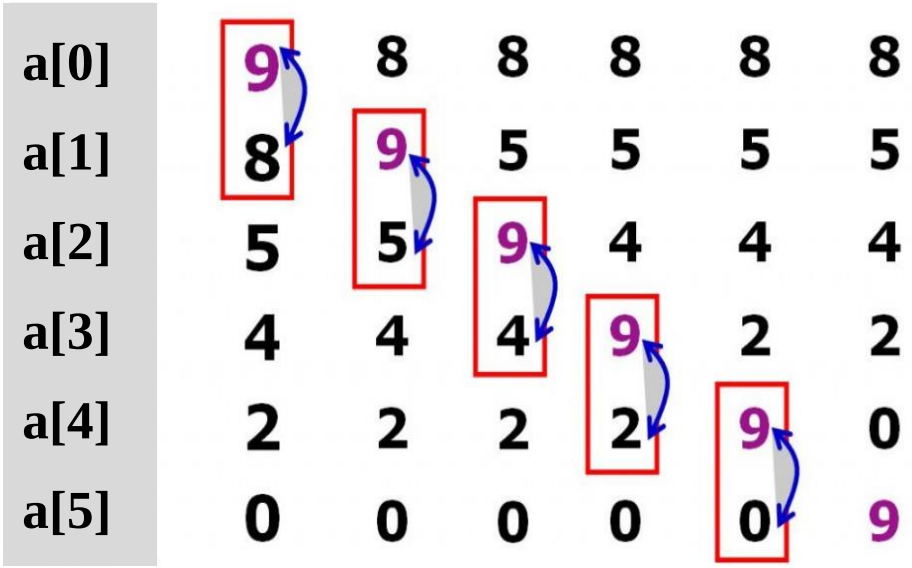



```
void BubbleSort(int a[], int n)
{
    int  i, j, temp;
    for (i=0; i<n-1; i++)
    {
        for (j=n-1; j>i; j--)
        {
            if (a[j] < a[j-1])
            {
                temp = a[j];
                a[j] = a[j-1];
                a[j-1] = temp;
            }
        }
    }
}
```

从后往前比较，小数上（前）移

比较n-1次

冒泡法排序（以升序为例）



从前往后比较，大数下（后）移

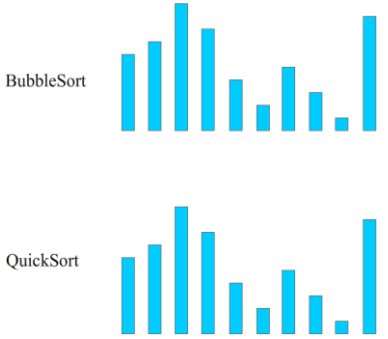
比较n−1次

```
void BubbleSort(int a[], int n)
{
    int  i, j, temp;
    for (i=0; i<n-1; i++)
    {
        for (j=0; j<n-i-1; j++)
        {
            if (a[j] > a[j+1])
            {
                temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    }
}
```

冒泡排序的优缺点

- 优点：
  - 规则简单，易于理解，实现容易，比较次数已知，算法稳定
- 缺点
  - 效率低，每次只能移动相邻两个数据，每次交换仅向最终目标移动一个位置

快速排序（Quick Sort）



英国计算机科学家霍尔（C.A. R.Hoare）—分治法的经典应用实例

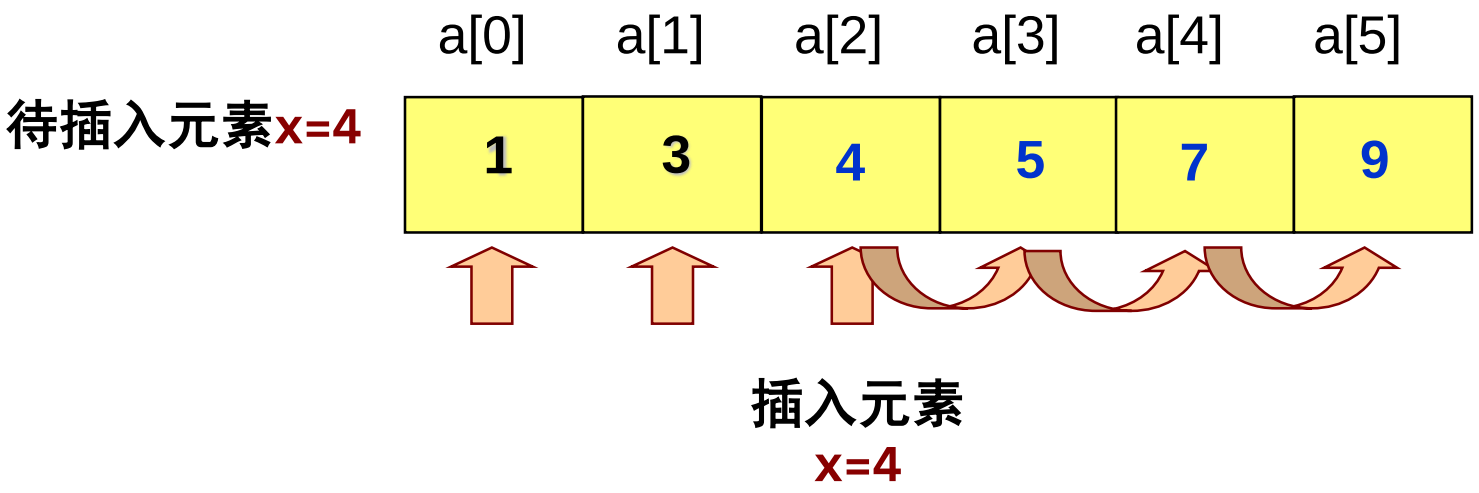
从后往前找地位高于吕方的站其前　　基准：小温侯吕方（第54号）　从前往后找地位低于吕方的站其后

基准：立地太岁阮小二（第27号）

基准：操刀鬼曹正（第81号）

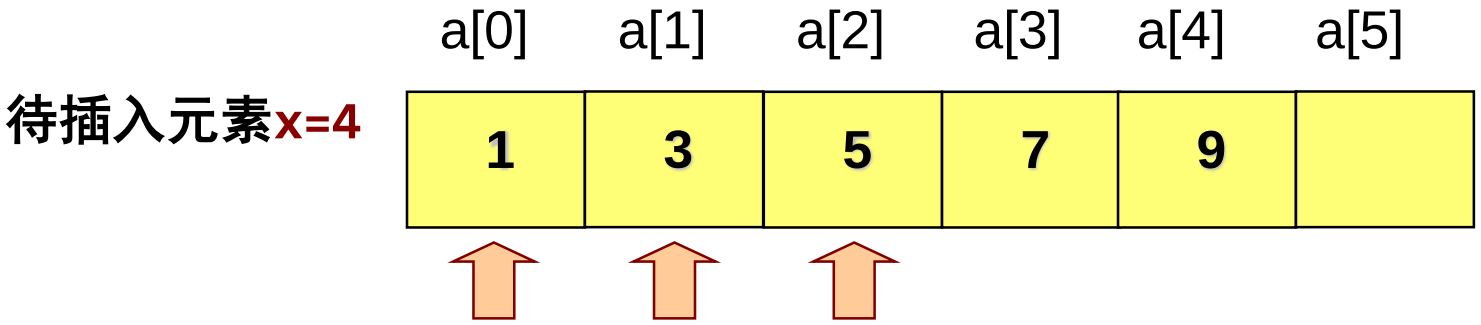
插入一个数据到有序数列中

- **算法的关键：**
  - 再一个有序数组中先找到待插入的位置
  - 依次移动插入位置及其后的所有元素
  - 腾出这一位置放入待插入的元素



插入一个数据到有序数列中

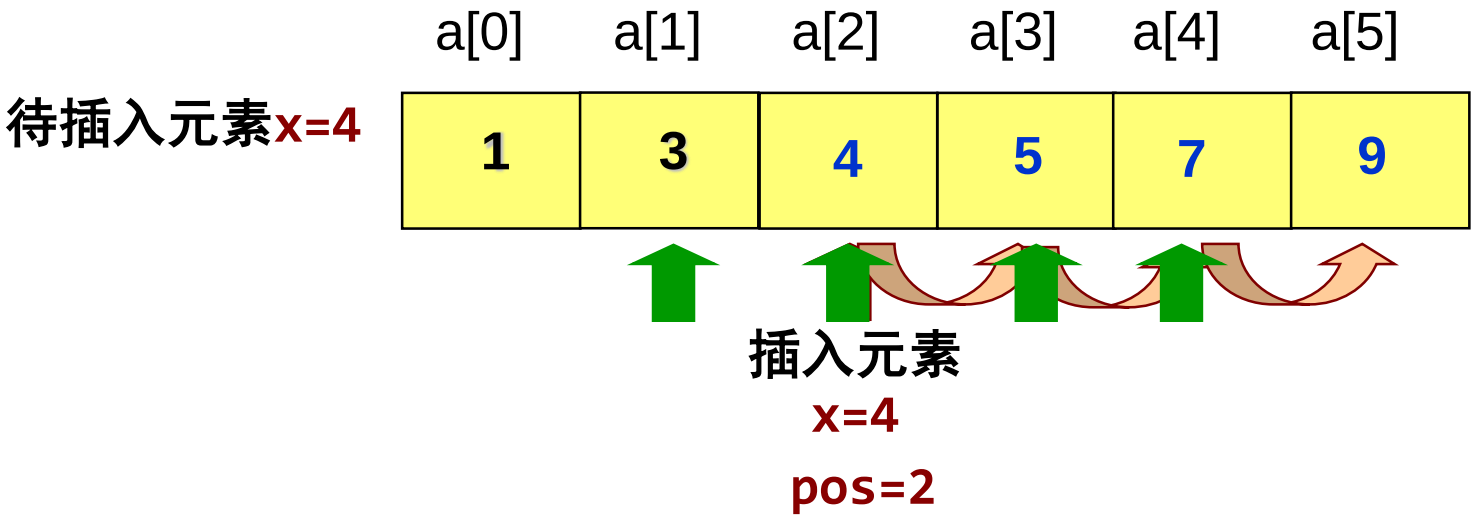
```
//找到待插入的位置
for (i=0; i<n && x>a[i]; i++ )
{
}
```



找到了待插入的位置

插入一个数据到有序数列中

```
pos = i;
for (i=n-1; i>=pos; i--)
{
    a[i+1] = a[i];      //元素a[i]后移
}
a[pos] = x;     //插入元素x到位置pos
```



插入一个数据到有序数列中→插入排序

```
void  Insert(int a[], int n, int x)
{
   int  i, pos;
   for (i=0; i<n && x>a[i]; i++)
   {
   }
   pos = i;   //找到插入位置
   for (i=n-1; i>=pos; i--)
   {
       a[i+1] = a[i]; //元素后移
   }
   a[pos] = x;   //插入元素
}
```

```
void  InsertSort(int a[], int n)
{
    int  i, pos, j, x;
    for (j=1; j<n; j++)
    {
        x = a[j];
        for (i=0; i<j && x>a[i]; i++)
        {
        }
        pos = i; //找到插入位置
        for (i=j-1; i>=pos; i--)
        {
            a[i+1] = a[i]; //元素a[i]后移
        }
        a[pos] = x; //插入元素
    }
}
```

插入排序

```
void  InsertSort(int a[], int n)
{
    int  i, pos, j, x;
    for (j=1; j<n; j++)
    {
        x = a[j];
        for (i=0; i<j && x>a[i]; i++)
        {
        }
        pos = i; //找到插入位置
        for (i=j-1; i>=pos; i--)
        {
            a[i+1] = a[i];
        }
        a[pos] = x; //插入元素
    }
}
```

先找后移

边找边移

```
void  InsertSort(int a[], int n)
{
    int i, j, x;
    for (j=1; j<n; j++)
    {
        x = a[j];
        for (i=j-1; i>=0 && x<a[i]; i--)
        {
            a[i+1] = a[i]; //元素a[i]后移
        }
        a[i+1] = x; //插入元素
    }
}
```

向函数传递二维数组

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

1	6	11	16	21
2	7	12	17	22
3	8	13	18	23
4	9	14	19	24
5	10	15	20	25

```
void SetArray(int a[][N], int m, int n)
{
   int i, j, len = 1;
   for (i=0; i<m; i++)
   {
        for (j=0; j<n; j++)
        {
           a[i][j] = len;
           len++;
        }
   }
}
```

```
void SetArray(int a[][N], int m, int n)
{
   int i, j, len = 1;
   for (j=0; j<n; j++)
   {
        for (i=0; i<m; i++)
        {
           a[i][j] = len;
           len++;
        }
   }
}
```

- 声明函数的二维数组形参时可省略第一维的长度，但不能省略第二维的长度

矩阵转置

```
/* 函数功能：计算m*n矩阵a的转置矩阵at */
void Transpose(int a[][N], int at[][M], int m, int n)
{
    int i, j;
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < n; j++)
        {
            at[j][i] = a[i][j];
        }
    }
}
```

- 样例输入

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

- 样例输出

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

- 样例输入

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

- 样例输出

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

```
#include <stdio.h>
#define M 1000
#define N 1000
void Transpose(int a[][N], int at[][M], int m, int n);
void InputMatrix(int a[][N], int m, int n);
void PrintMatrix(int at[][M], int n, int m);
int main()
{
    int s[M][N], st[N][M], m, n;
    printf("Input m, n:");
    scanf("%d,%d", &m, &n);
    InputMatrix(s, m, n);
    Transpose(s, st, m, n);
    printf("The transposed matrix is:\n");
    PrintMatrix(st, n,  m);
    return 0;
}
```

课后思考题：挑战类型的极限

- 从键盘任意输入一个数n（0<n<=1000000），编程计算并输出S=1!+2!+...+n!的**末6位**（不含前导0）。
  - 若S不足6位，则直接输出S。
  - 不含前导0的意思是，如果末6位为001234，则只输出1234即可。
  - 如果输入的n不在1到1000000之间，则输出“Input error！”。

```
Enter a number to be calculated:
40
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
21! = 51090942171709440000
22! = 1124000727777607680000
23! = 25852016738884976640000
24! = 620448401733239439360000
25! = 15511210043330985984000000
26! = 403291461126605635584000000
27! = 10888869450418352160768000000
28! = 304888344611713860501504000000
29! = 8841761993739701954543616000000
30! = 265252859812191058636308480000000
31! = 8222838654177922817725562880000000
32! = 263130836933693530167218012160000000
33! = 8683317618811886495518194401280000000
34! = 295232799039604140847618609643520000000
35! = 10333147966386144929666651337523200000000
36! = 371993326789901217467999448150835200000000
37! = 13763753091226345046315979581580902400000000
38! = 523022617466601111760007224100074291200000000
39! = 20397882081197443358640281739902897356800000000
40! = 815915283247897734345611269596115894272000000000
```

```c
#include <stdio.h>
#define MOD 1000000
long Func(int n);
int main()
{
   int  n;
   long s;
   printf("Input n:");
   scanf("%d", &n);
   if (n>0 && n<=1000000)
   {
       s = Func(n);
       printf("%ld\n", s);
   }
   else
   {
       printf("Input error!\n");
   }
   return 0;
}
```

```c
//函数功能：计算1!+2!+...+n!的末6位数？？
long Func(int n)
{
    int i;
    long s = 0, f = 1;
    for (i=1; i<=n; i++)
    {
        f = f * i;
        s = s + f;
    }
    return s % MOD;
}
```

- 从键盘任意输入一个数n（0<n<=1000000），编程计算并输出S=1!+2!+...+n!的末6位（不含前导0）。
- 这个程序能否得到你期望的结果？

```c
#include <stdio.h>
#define MOD 1000000
long Func(int n);
int main()
{
   int  n;
   long s;
   printf("Input n:");
   scanf("%d", &n);
   if (n>0 && n<=1000000)
   {
       s = Func(n);
       printf("%ld\n", s);
   }
   else
   {
       printf("Input error!\n");
   }
   return 0;
}
```

```c
//函数功能：计算1!+2!+...+n!的末6位数
long Func(int n)
{
    int i;
    long s = 0, f = 1;
    if (n > 24) n = 24;
    for (i=1; i<=n; i++)
    {
        f = f * i % MOD;  //保留阶乘值末6位
        s = (s + f) % MOD;//保留阶乘和末6位
    }
    return s;
}
```

- 从键盘任意输入一个数n（0<n<=1000000），编程计算并输出S=1!+2!+...+n!的末6位（不含前导0）。
- 这个程序能否得到你期望的结果？

飞机游戏V5.0：敌机下移

```
void UpdateWithoutInput()  //与用户输入无关的更新
{
    if (bulletX > -1)
    {
        bulletX--;//子弹移动，超出屏幕上边界不显示
    }
    if (bulletX == enemyX && bulletY == enemyY) //击中敌机
    {
        enemyX = -1;  //敌机消失
    }
    //用来控制敌机向下移动的速度，每隔10次循环才移动一次敌机
    static int speed = 0; //静态局部变量，仅初始化1次
    if (speed < 10)
    {
        speed++;   //记录本函数被调用的次数
    }
    else if (speed == 10) //每执行10次函数调用后
    {
        enemyX++;  //敌机下移1次
        speed = 0; //重新开始计数
    }
}
```

飞机游戏V6.0：随机产生移动的敌机

```
void UpdateWithoutInput()  //与用户输入无关的更新
{
    srand(time(NULL));
    if (bulletX > -1)
    {
        bulletX--;//子弹移动，超出屏幕上边界不显示
    }
    if (bulletX == enemyX && bulletY == enemyY) //击中敌机
    {
        enemyX = -1;  //敌机消失，隔10帧后重新出现在屏幕顶端
        enemyY = rand() % width;  //水平位置随机生成
    }
    if (enemyX > high)    //敌机跑出屏幕下边界时产生新的敌机
    {
        enemyX = 0;
        enemyY = rand() % width;
    }
    if (planeX == enemyX && planeY == enemyY) //撞上敌机则游戏结束
    {
        printf("Game over!\n");
        system("pause");
        exit(0);
    }
    //用来控制敌机向下移动的速度，每隔几次循环才移动一次敌机
    ……
}
```

飞机游戏V7.0：记录游戏得分

```
void UpdateWithoutInput()  //与用户输入无关的更新
{
    srand(time(NULL));
    if (bulletX > -1)
    {
        bulletX--;//子弹移动，超出屏幕上边界不显示
    }
    if (bulletX == enemyX && bulletY == enemyY) //击中敌机
    {
        score++;
        enemyX = -1;  //敌机消失
        enemyY = rand() % width;
    }
    if (enemyX > high)    //敌机
    {
        enemyX = 0;
        enemyY = rand() % width;
    }
    if (planeX == enemyX && plan
    {
        printf("Game over!\n");
        system("pause");
        exit(0);
    }
    ……
}
```

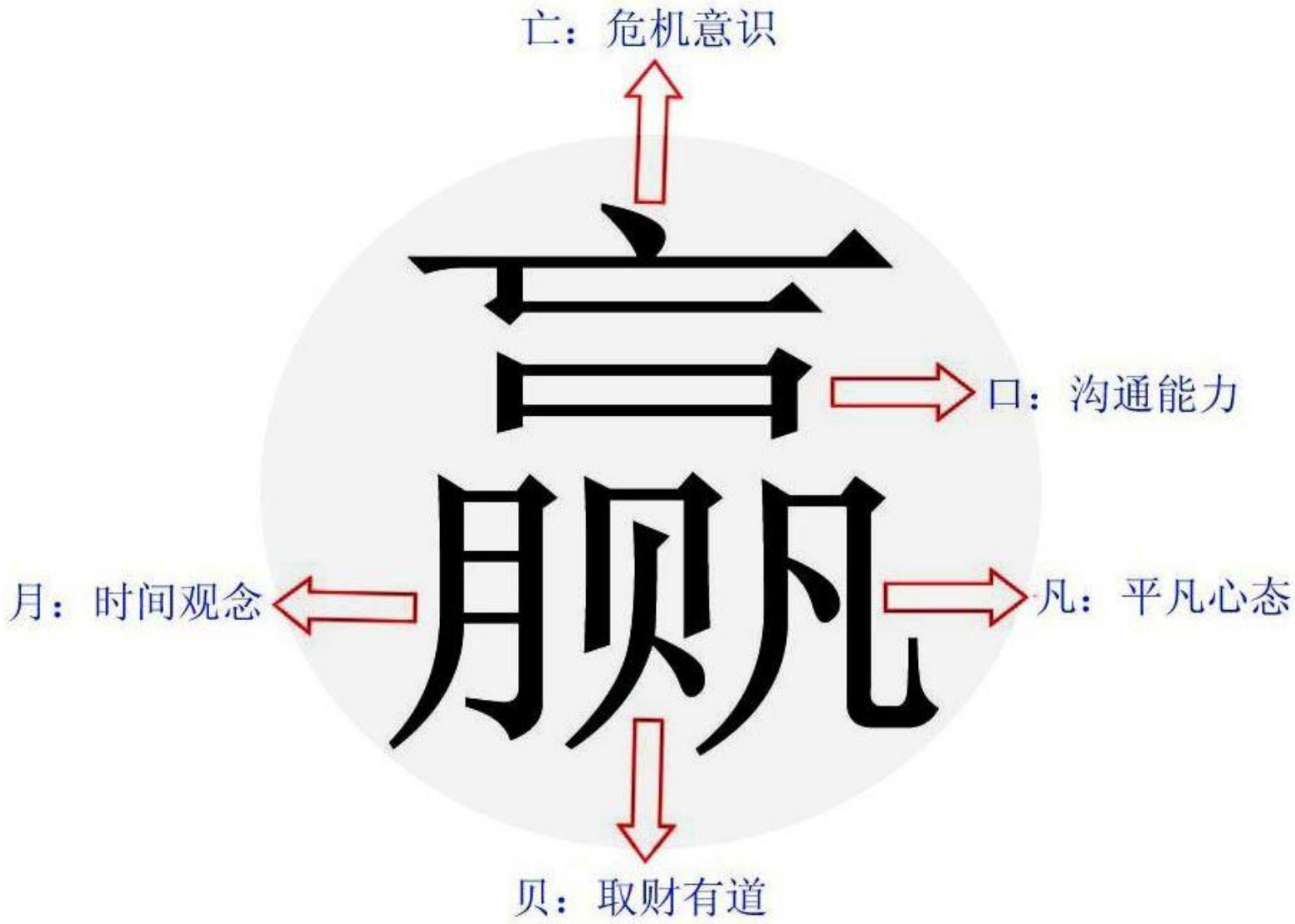
```
//全局变量
……
int score;
void Initialize()    //数据的初始化
{
    ……
    score = 0;
}
```

```
void Show()              //显示画面
{
    ……
    printf("得分：%d\n", score);
}
```

打飞机加强版

- 敌机数量变多
- 有加分和减分
- 积分到达一定程度敌机下落速度变快，子弹变厉害

人生目标 一赢



找对路子，学渣也有春天！

- 日本明治大学文学部教授教育学博士斋藤孝
  - 把学习阶段概括为“找到适合自己的学习方式->坚持->突破”，并分三部分教你如何学。
  - 他的速效学习法有“一气呵成学习法”、“连休集训学习法”、“集中学习法”、“白纸活用法”、“身临其境学习法”、“聊天术”等。