E6 - Solution

注：部分题目的题解助教头子增加了不同的思路“示例代码 - 2”，大家可以参考学习。

A “智能的”加减法

难度	考点
2	字符串

题目分析

先设定一个求和的变量 `sum=0` ，然后处理字符串。当字符串读取到符号的时候，使用一个变量进行记录，然后读取到数字的时候，根据数字前的符号决定加或减这个数，最后读取到字符串末尾或 `=` 号时，输出结果即可。

本题中可能需要做字符串与数字的转换。示例代码中提供了一种转换方式，即一位一位地读取数字，添加进一个求和变量 `num` 中，然后每读取一位，`num` 先自乘 `10` ，然后加入一个数码，同时要注意 `char` 型的数字字符与 `int` 类型的转换问题（诸如将字符 `'3'` 转换为 `int` 型的 `3` 的问题）。 同时，采用 `sscanf` 等也可转换，相关代码参考扩展阅读部分。

示例代码

```
#include<stdio.h>
#include<string.h>
int main() {
    char a[1002];
    gets(a);
    int len = strlen(a);
    int sum = 0, sgn = 1, num = 0;
    for (int i = 0; i < len; i++) {
        if (a[i] >= '0' && a[i] <= '9') {
            num = num * 10 + (a[i] - '0');//字符串与int型数字的转换
        } else {
            sum += sgn * num;
            num = 0;
            if (a[i] == '+')sgn = 1;
            else if (a[i] == '-')sgn = -1;
        }
    }
    printf("%d\n", sum);
    return 0;
}
```

示例代码 - 2

利用 `scanf` 逐个读入也可做出本题。

```
#include <stdio.h>
int main()
{
    char c;
    int ans, t;
    scanf("%d", &ans);
    while(~scanf(" %c%d", &c, &t))
    {
        if(c == '=') break;
        ans += ((c == '+') ? t : -t);
    }
    printf("%d", ans);
    return 0;
}
```

扩展阅读

本题中的计算式为典型的“中缀表达式”，学有余力的同学可自行探索。

采用 `sscanf` 转换字符与数字的方式：

```
#include<stdio.h>
#include<string.h>
int main() {
    char a[1002];
    gets(a);
    int len = strlen(a);
    int sum = 0, sgn = 1, num = 0;
    for (int i = 0; i < len; i++) {
        if (a[i] >= '0' && a[i] <= '9' &&
        (i==0 || (i>0 && a[i-1]==' '))) {
            sscanf(a+i,"%d",&num);
            //与scanf相比，sscanf在前部多了一个参数
            //一般可用“字符串变量名+字符串对应位置的下标”的方式表示
            //本处以字符串“1 + 2 =”为例，当我们需要读取'2'时，2对应的i为4，即前面的参数本质
上是“a+4”
        } else {
            sum += sgn * num;
            num = 0;
            if (a[i] == '+')sgn = 1;
            else if (a[i] == '-')sgn = -1;
        }
    }
    printf("%d\n", sum);
    return 0;
}
```

B 数组的大小

难度	考点
2	对于 `gets` 和 `strcmp` 函数的使用

题目分析

这题主要的难点在于换行符的处理。由于输入的字符串可能含空格，而 `scanf("%s")` 是以空白字符作为结束标志的，因此这题读取字符串需要使用 `gets` 函数。`gets` 函数以换行符作为结束读取的标志，**并且会接收换行符**，并在字符数组大小足够的前提下将读入字符串末尾的 `\n` 替换成 `\0`。注意每组数据我们还需要读入一个整型，而这里的换行符不会被 `scanf` 函数处理，需要我们额外处理一下。

示例代码

```c
#include <stdio.h>
#include <string.h>
int main(void)
{
    char str[32];
    int num, size;
    while (gets(str) != NULL)
    {
        scanf("%d", &num);
        getchar();
        if (!strcmp(str, "int"))
            size = sizeof(int);
        else if (!strcmp(str, "char"))
            size = sizeof(char);
        else if (!strcmp(str, "short"))
            size = sizeof(short);
        else if (!strcmp(str, "long"))
            size = sizeof(long);
        else if (!strcmp(str, "long long"))
            size = sizeof(long long);
        else
        {
            printf("ErrOr!\n");
            continue;
        }
        printf("%d\n", size * num);
    }
    return 0;
}
```

拓展

在大部分人的 IDE 上，各整数类型对应的大小可能都是这样的：

类型	`char`	`short`	`int`	`long`	`long long`
大小 / 字节	1	2	4	4	8

而在我们的 OJ 平台上，各整数类型对应的大小是这样的：

类型	`char`	`short`	`int`	`long`	`long long`
大小 / 字节	1	2	4	8	8

从中我们可以认识到在不同的编译环境下，同一个整数类型可能会有不同的性质。于是在这一题中我们想要获得每个整数类型所占用的内存空间，就不能想当然的使用我们自己 IDE 上这些类型的大小，而是应该使用 `sizeof` 关键字，确保能够在不同的编译环境下都能正确输出各整数类型所占的内存空间。

C CNN

难度	考点
3	二维数组 循环

题目分析

卷积层的计算需要三个循环。前两层循环用来定位当前计算的元素位置，第三层循环实现当前元素的卷积计算。

计算过程需要注意前两层循环的循环次数（即输出矩阵的形状）为 $m-n+1$。

也需要注意卷积计算时要保证输入图像和卷积核的对齐，输出矩阵的第 i 行第 j 列的计算公式为 $\sum_{k=1}^{n}\sum_{l=1}^{n}x[k][l]\times y[i+k][j+l]$

示例代码

```
#include <stdio.h>
int x[100][100];
int y[100][100];
int output[100][100];
int cnn(int m,int n)
{
    for (int i=0;i<m-n+1;i++)
        for (int j=0;j<m-n+1;j++)
        {
            output[i][j]=0;//注意初始化
            for (int k = 0; k < n; ++k) {//第i行第j列的输出矩阵卷积计算
                for (int l = 0; l < n; ++l) {
                    output[i][j]+=x[k][l]*y[i+k][j+l];
                }
            }
        }
    return m-n+1;//返回输出矩阵大小
}
int main()
{
    int m,n;
    scanf("%d%d",&n,&m);
    for (int i=0;i<n;i++)//卷积核
        for (int j=0;j<n;j++)
```

```
            scanf("%d",&x[i][j]);
    for (int i=0;i<m;i++)//输入矩阵
        for (int j=0;j<m;j++)
            scanf("%d",&y[i][j]);
    int r=cnn(m,n);
    for (int i=0;i<r;i++)
    {
        for (int j=0;j<r;j++)
            printf("%d ",output[i][j]);
        printf("\n");
    }
}
```

D czx 学化学

难度	考点
3	字符串

题目分析

先用 `scanf("%s", s)` 读入字符串，然后逐字符遍历一遍这个字符串。

在遍历的时候，如果它是字母，那么在答案中加上 `上一个字符的相对原子质量 * 数量`，如果它是数字，那么更新当前的数量 num。

在处理的时候需要注意的细节主要有：

- 如果 num 的值为 0，说明相应的元素字母后面的数字被省略，应认为 num 为 1。
- 在遍历结束的时候，最后一个元素并没有被算入答案当中，需要在最后加上去。

示例代码

```
#include <stdio.h>
#include <string.h>
#define C 12
#define H 1
#define O 16
#define N 14

char s[205], c;
int res, num;

int main() {
    scanf("%s", s);
    c = s[0], num = 0;
    for (int i = 1; s[i]; i++) {
        if (s[i] >= '0' && s[i] <= '9') {
            num = num * 10 + s[i] - '0';
        } else {
            if (!num)
                num = 1;
```

```
            if (c == 'C')
                res += C * num;
            else if (c == 'H')
                res += H * num;
            else if (c == 'O')
                res += O * num;
            else if (c == 'N')
                res += N * num;
            c = s[i], num = 0;
        }
    }
    if (!num)
        num = 1;
    if (c == 'C')
        res += C * num;
    else if (c == 'H')
        res += H * num;
    else if (c == 'O')
        res += O * num;
    else if (c == 'N')
        res += N * num;
    printf("%d", res);
    return 0;
}
```

示例代码 - 2

利用 `scanf` 的返回值判断。

```
#include <stdio.h>
int main()
{
    char c;
    int n, m = 0;
    while (~(c = getchar()))
    {
        if (scanf("%d", &n) != 1) n = 1;
        switch (c)
        {
            case 'C':
                m += n * 12;
                break;
            case 'H':
                m += n * 1;
                break;
            case 'O':
                m += n * 16;
                break;
            case 'N':
                m += n * 14;
                break;
        }
    }
    printf("%d", m);
    return 0;
```

```
}
```

E 水獭密码

难度	考点
3	二维数组

题目分析

本题考察同学们对于二维数组的基本使用，我们只需要开辟一个大小足够的二维数组，然后将读入的字符串按行放入这个二维数组中，再按列依次进行输出即可。如果二维数组成功初始化的话，实际上我们在输出字符时并不需要特判是不是空字符，因为空字符本来就是没有输出的（你可以运行 `printf("%c",(char)0);` 观察结果，注意字符 '0' 和空字符的区别）。

要注意的是有同学的做法是先求出满足 $n \times n \geq k$ 的最小正整数 n，然后再找到 $n \times n$ 矩阵中每个位置对应原字符位置的值，但是这样可能带来的一个问题就是访问的一位数组下标可能超出 100000（为大于 100000 的最小完全平方数），所以一维数组不开大一点采用这种方法就会越界。

示例代码

```c
#include <stdio.h>
#include <math.h>
#include <string.h>

char c[100010];
char c2[320][320];

int main()
{
    scanf("%s",c);
    int len = strlen(c);
    int n = ceil(sqrt(len));    // ceil 是取上整函数

    int pos = 0;
    while(pos < len) {
        int i = pos / n ;
        int j = pos % n ;   // 按行放置
        c2[i][j] = c[pos];
        pos++;
    }

    for(int j = 0;j < n;j++) {
        for(int i = 0;i < n;i++) {
            printf("%c",c2[i][j]);
        }
    }

    return 0;
}
```

也有同学使用一维数组计算应该输出的下标的方法做的，这样当然也可以，但是可以比较一下是不是使用二维数组之后逻辑更简单了。

示例代码 - 2

```
#include <stdio.h>
#include <math.h>
#include <string.h>
char s[102400];
char a[320][320];
int main()
{
    scanf("%s", s);
    int n = ceil(sqrt(strlen(s)));
    int k = 0;
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            a[i][j] = s[k++];
    for(int j = 0; j < n; j++)
        for(int i = 0; i < n; i++)
            if(a[i][j]) putchar(a[i][j]);
    return 0;
}
```

示例代码 - 3

```
#include <stdio.h>
#include <math.h>
#include <string.h>
char s[102400];
char a[320][320];
int main()
{
    scanf("%s", s);
    int n = ceil(sqrt(strlen(s)));
    for(int j = 0; j < n; j++)
        for(int i = 0; i < n; i++)
            if(s[i * n + j]) putchar(s[i * n + j]);
    return 0;
}
```

F czx 学字符串

难度	考点
4	字符串，函数

题目分析

对于一个字符串 s，给定一个起点 st，怎么样才能提取出它所对应的表示方式呢？显然，这种表示方式的第 i 个字符为 s 的第 $st+i$ 个字符。然而，$st+i$ 可能会超过 s 的长度，当超过了字符串长度时，需要回到字符串的第 0 位。结合取模操作不难想到，s 的第 $st+i$ 个字符即是 $s_{(st+i) \bmod len}$（len 为字符串的长度）。

定义一个函数 `int less(int p, int q)`，判断起点为 p 的表示是否比起点为 q 的小，我们只需要同时遍历两个字符串，在第一个有差异的地方进行判断并返回相应的结果，便可以实现。

有了 less 函数之后，只需要按照求解最大最小值的思路，记录当前最小的位置，并在遍历的过程中比较并更新即可。

示例代码

```
#include <stdio.h>
#include <string.h>

char s[105];
int len, res;

int less(int p, int q) { //判断起点p是否比起点q小
    for (int i = 0; i < len; i++) {
        if (s[(p + i) % len] != s[(q + i) % len]) {
            return s[(p + i) % len] < s[(q + i) % len];
        }
    }
    return 0;
}

int main() {
    scanf("%s", s);
    len = strlen(s);
    for (int i = 1; i < len; i++) { //枚举字符串起点
        if (less(i, res))
            res = i;
    }
    for (int i = 0; i < len; i++) {
        printf("%c", s[(res + i) % len]);
    }
    return 0;
}
```

G HDB3码 题解

难度	考点
4	数组操作

题目解析

本题的思路非常清晰，按照所给的步骤一步一步模拟即可。实现时，有一些细节需要注意，例如用`EOF`判断输入结束、操作时防止出现访问超过范围的数组下标等。

示例代码

```
#include <stdio.h>
#define MAXN 5+1000

int main() {
    char c;
    int a[MAXN];
    int n = 1;
    while ((c = getchar()) != EOF) {
        if (c == '0') {
            a[n] = 0;
        } else if (c == '1') {
            a[n] = 1;
        } else {
            --n; //删除无效位
            break;
        }
        ++n;
    }

    //加V，令2代表V
    for (int i = 4; i <= n; ++i) {
        if (a[i] == 0) {
            if (a[i - 3] + a[i - 2] + a[i - 1] + a[i] == 0) {
                a[i] = 2;
            }
        }
    }
    //加B，令3代表B
    for (int i = 1; i <= n; ++i) {
        if (a[i] == 2) {
            int cnt = 0; //cnt:两个V之间非零码个数
            for (int j = i - 1; j >= 1; --j) {
                if (a[j] == 2) {
                    break;
                }
                if (a[j] != 0) {
                    ++cnt;
                }
            }
            if (cnt % 2 == 0 && i - 3 >= 1) {
                a[i - 3] = 3;
            }
        }
    }
    //对信码加符号
    int flag = 1;
    for (int i = 1; i <= n; ++i) {
        if (a[i] == 1 || a[i] == 3) {
```

```
            a[i] = flag;
            flag *= -1;
        }
    }
    //对v加符号
    for (int i = 1; i <= n; ++i) {
        if (a[i] == 2) {
            for (int j = i - 1; j >= 1; --j) {
                if (a[j] != 0) {
                    a[i] = a[j] > 0 ? 1 : -1;
                    break;
                }
            }
        }
    }
    for (int i = 1; i <= n; ++i) {
        if (a[i] != 0){
            printf("%+d ",a[i]);
        } else {
            printf("0 ");
        }
    }
    return 0;
}
```

H Cirno 的初等函数教室

难度	考点
4	二分法

问题分析

记一个整数 n 的位数为 k，易知有 $n < 10^k$。因此对于本题，我们只需要求解满足不等式 $x \log_{10} x < n$ 的整数 x 的最大值即可。考虑到 n 的范围在 `int` 内，直接遍历会导致 `TLE`，使用二分法求解。

参考代码 #1

```
#include <stdio.h>
#include <math.h>

double f(int x)
{
    return x * log10(x);
}

int main()
{
    int n, x, l = 1, m, r = 0x7fffffff;
    scanf("%d", &n);
    while (l <= r)
```

```
    {
        m = l + (r - l) / 2;
        if (f(m) < n)
        {
            l = m + 1;
            x = m;
        }
        else
            r = m - 1;
    }
    printf("%d", x);
    return 0;
}
```

参考代码 #2

对于 $x \log_{10} x < n$，注意到一定有 $x > \dfrac{n}{\log_{10} n}$，因此从 $\dfrac{n}{\log_{10} n}$ 开始遍历亦可通过本题。

```
#include <stdio.h>
#include <math.h>

int main()
{
    int n;
    scanf("%d", &n);
    int i;
    for (i = n / log10(n); i * log10(i) < n; i++)
        ;
    printf("%d", i - 1);
    return 0;
}
```

I Permutation?

难度	考到
5~6	递归，数组

题目分析

本题用到了递归实现深度优先搜索 (Depth First Search, DFS) 的思想。

设答案数组 ans，$ans[n]$ 表示输入为 n 时的答案。

首先建立一个标记数组 a，$a[i]$ 表示整数 i 当前是否用过。设函数 $dfs(i,k)$ 表示当前遍历到排列的第 k 个数，为整数 i。在函数中，首先将 $a[i]$ 置 1 表示 i 被用过了，然后此时递归调用 $dfs(i-3,k+1), dfs(i-2,k+1), dfs(i+3,k+1), dfs(i+3,k+1)$ 遍历下一个数。

递归调用之后，要注意将 $a[i]$ 重新置零，表示 $a[i]$ 未被用过，防止影响回溯后影响其他递归调用的计算。

递归的基本情况有 3 种：

1. 如果当前遍历的 i 不在数组范围 $[0, n-1]$ 内，则直接返回；
2. 否则，如果当前遍历的 i 用过了，即 $a[i]=1$，则直接返回；
3. 否则，如果当前遍历到第 n 个数，则说明成功找到一个满足条件的排列，将 $ans[n]$ 自增 1。

在主函数中，循环调用 $dfs(i,1), i=0,1,\cdots,n-1$，即可得到输入为 n 时的答案。

总结一下递归实现深度优先搜索算法的核心要点：

1. 确定由当前位置搜索下一个位置的方案，根据此设计递归函数的一般情况；
2. 合理正确使用标记数组 $a[i]$；
3. 根据题目条件设计递归基本情况。

注意到本题中对于给定的 n，答案 $ans[n]$ 也是确定的，因此当计算过 $ans[n]$ 之后可以不必再次计算。

初始化答案数组 ans 的每个值均为 -1 ，对于每组数据，若 $ans[n]=-1$ 说明未计算过，调用函数计算答案并输出；若 $ans[n]\neq -1$，说明计算过了，直接输出答案即可。

以上是本题的核心思路。

学习以下示例代码，找找共同点和不同点。

示例代码

1. dfs 函数只实现搜索，计算用全局变量实现。

```
#include <stdio.h>
int n;
int ans[30];
int a[30];
void dfs(int i, int k)
{
    if(i < 0 || i >= n || a[i]) return;
    if(k == n)
    {
        ans[n]++;
        return;
    }
    a[i] = 1;
    dfs(i - 3, k + 1);
    dfs(i - 2, k + 1);
    dfs(i + 2, k + 1);
    dfs(i + 3, k + 1);
    a[i] = 0;
    return;
}
int main()
{
    for(int i = 0; i < 30; i++)
        ans[i] = -1;
    while(~scanf("%d", &n))
    {
        if(ans[n] == -1)
        {
            ans[n] = 0;
            for(int i = 0; i < n; ++i)
                dfs(i, 1);
```

```
        }
        printf("%d\n", ans[n]);
    }
    return 0;
}
```

2. 合并不同的 n 的递归过程，$dfs(i,k,m)$ 中参数 m 代表当前遍历过的最大的数。

```
#include <stdio.h>
#define max(a,b) ((a)>(b)?(a):(b))
int n = 24;
int ans[30];
int a[30];
void dfs(int i, int k, int m)
{
    if(i < 0 || i >= n || a[i]) return;
    if(m < k) ans[k]++;
    a[i] = 1;
    dfs(i - 3, k + 1, m);
    dfs(i - 2, k + 1, m);
    dfs(i + 2, k + 1, max(m, i + 2));
    dfs(i + 3, k + 1, max(m, i + 3));
    a[i] = 0;
    return;
}

int main()
{
    for(int i = 0; i < 24; i++)
        dfs(i, 1, i);
    while(~scanf("%d", &n))
        printf("%d\n", ans[n]);
    return 0;
}
```

第 2 种方法是出题人目前想出来的计算所有输入 n 对应答案的最优方法（不算打表），如果你有更好的做法，欢迎和助教团队进行交流。

J 摩卡与水獭乐团派对

难度	考点
5	冒泡排序、逆序对、归并排序、分治

题目分析

假如现在编号为 1 到 n 的 n 只水獭站成一排，我们首先的想法是，既然 n 号水獭（这里的号都是水獭自身的编号）最后肯定要到位置 n，我们不妨一直让 n 号水獭与后面的一只水獭交换位置，直到 n 号水獭到达位置 n。这时我们就可以只考虑剩下的 $n-1$ 只水獭，重复上述步骤，直到所有水獭都到了自己该到的位置。假设进行完前 t 回合时，编号前 t 大的水獭都已经到达了目标位置，这一回合我们要考虑编号 1 到编号 k 的 k 只水獭，让编号为 k 的水獭到达位置 k，我们有两件事是确定的：

- 之前的操作相当于把前 t 大的水獭从水獭列中抽出去而已，而 1 **号到** k **号水獭的相对位置从来没有变过**，即若 i 号水獭一开始就在 j 号水獭之间，且 $1 \leq i, j \leq k$，那么现在 i 号水獭也一定在 j 号水獭之前
- 本回合如果要将 k 号水獭交换到位置 k，那么交换的次数**等于现在** k **号水獭后面的水獭数**（除去已经到达目标位置的前 t 只水獭），根据上一点，也**等于一开始在** k **号水獭之后，且编号小于** k **的水獭的个数**

根据上述两点，我们不难知道，交换的最小总次数，等于在初始状态时，对于每只水獭在它后面且编号比它小的水獭的个数和；假设我们用数组元素 $otter_i$ 表示初始时在位置 i 的水獭的编号，那么就相当于找**满足** $i < j$ **且** $otter_i > otter_j$ **的有序数对** (i, j) **的个数**，这样的数对还有一个名字，叫做**逆序对**。即问题转化为了找逆序对个数。

有了这个想法，我们就可以动手了，基于上面的想法，我们可以直接进行编程模拟。但是我们经过简单思考过后，不难发现它与我们学过的冒泡排序有着微妙的关系：冒泡排序过后，逆序对数量一定是 0（因为排序过后的数组一定是从 1 到 n 的顺序排列）；冒泡排序的每次临项交换，减少且只减少一个逆序对。据此，我们惊奇地发现——**冒泡排序的交换次数也等于逆序对数。**

但是事情远没有那么简单，如果我们采用模拟提取最大项或者模拟冒泡的方法，我们不难发现 n 最大可以达到 100000，而两种方法的复杂度会达到 $O(n^2)$，这样会超时。直到现在为止我们相当于一共讨论了三种方法：给定一个数找它后面比它小的数，最后求和（按照我们现在所学的知识只能枚举它之后的每一个元素进行比较）；冒泡排序次数方法；或者我们最先提出来的方法，记录每一只水獭的位置，每次能直接算出当前要处理的水獭到达目标位置要交换的次数，但是需要将它之后的水獭的编号都减 1，按照我们现在所学，我们只能遍历其之后的水獭，将它们的编号减一，这跟方法一是差不多的。显然，这三种方法都不行。

对于像 Hint1 中提到的数组，如 1 4 5 6 2 3 7 8，我们有没有什么不同于以上三种方法的更好的方法。若将前半段有序的数列称为 a，后半段有序的数列称为 b，显然有：

- 在 a 和 b 内部不可能包含逆序对，即逆序对中的元素一定前者属于 a，后者属于 b
- 若 a 中元素 a_i 大于 b 中元素 b_j，那么 a_i 一定大于 b 中在 b_j 之前的所有元素，这也就是所有以 a_i 开头的逆序对数；$a_i < b_j$ 时有类似对称的结论

结合以上两点，我们处理 1 4 5 6 2 3 7 8，流程如下。

- a = 1 4 5 6，b = 2 3 7 8，用 c 表示已经排序好的元素，初始时 c 为空。
- 由于 $1 < 2$，所以不可能有以 1 开头的逆序对，有 a = 4 5 6，b = 2 3 7 8，c = 1
- 由于 $4 > 2$，$4 > 3$，$4 < 7$，所以只有两个以 4 开头的逆序对，有 a = 5 6，b = 7 8，c = 1 2 3 4
- 在下一次比较开始前，我们很显然不用比较 5 与 b 中前两个元素的大小，就知道 5 一定大于它们了，这是因为 5 的前一个元素 4 都大于它们，所以我们只需要发现 $5 < 7$，就知道只有两个以 5 开头的逆序对，有 a = 6，b = 7 8，c = 1 2 3 4 5
- 同理，由于 $6 < 7$，只有两个以 6 开头的逆序对，有 a 为空，b = 7 8，c = 1 2 3 4 5 6
- 由于 a 中没有元素了，所有逆序对已经被我们统计完了，将 b 中剩余元素放入 c，有 a 为空，b 为空，c = 1 2 3 4 5 6 7 8

综上，我们只经过 6 次比较就找到了全部的 6 个逆序对，而用上面的三种方法比较次数都比这个多。这是因为上面的三种方法都只有比较两个数后才知道它们的大小，即使知道 $3 < 4$ 且 $4 < 6$，也需要比较 3 和 6 才知道 $3 < 6$，而在下面的方法中，我们可以省略后面的比较，不经过比较就知道一些元素的大小关系，从而用较少的操作找到所有的逆序对数。 在合并两个有序的数列时，我们每次只需要比较两个

数列的最小元素；同时我们可以发现只要两个数列是有序的就可以这么操作，跟这两个数列合并之后是不是恰好是 1 到 k 是没有关系的。

但是我们发现题目中给的数列完全是 1 到 n 的随机排列，并不能保证前一半有序后一半有序，这个时候我们就可以参考 Hint2 了。我们不难知道以下事实：**只有一个元素的数列是有序的**；**将两个有序列合并之后得到的新的数列是有序的。**所以，结合我们前面学过的递归，我们知道，数列的逆序对数 = 前一半数列内部的逆序对数 + 后一半数列内部的逆序对数 + 第一个数属于前一个数列第二个数属于第二个数列的逆序对数，而最后一项跟将前后两个数列分别排序后用上述合并方法计算出的逆序对数是完全相同的。而计算前一个数列内部的逆序对数，可以用相同公式进行递归计算，合并前一半数列的前一半数列和后一半数列，直到递归到子列只有一个或者没有元素，这时这个子列一定是有序的。这其实也就是归并排序的思想，它的复杂度是 O(nlogn) 。

最后注意到数据范围，最坏的情况下是 100000 到 1 完全逆序，可能超过 int 的最大值(大概 20 亿，具体来说 2147483647)。

示例代码

```
#include <stdio.h>

int a[100010];
int tem[100010];    // 暂存正在排序的数列

// 计算合并开销
long long merge(int l,int r) {
    int mid = (l + r) >> 1; // 根据优先级，括号不写也行
    int i = l, j = mid + 1; // 两端的开始和结束
    int k = l;
    long long ans = 0;
    while((i <= mid) || (j <= r)) { // 合并
        if(j > r || (i <= mid && a[i] <= a[j])) {
            // 如果右子列空了或者左子列当前元素小于右子列当前元素，新的位是左面第一位，且计算逆序对数
            tem[k++] = a[i++];
            ans += j - mid - 1;
        }
        else {// 否则，新的位是右面第一位
            tem[k++] = a[j++];
        }
    }

    for(int i = l;i <= r;i++) {
        a[i] = tem[i];    // 归还排好序的段
    }

    return ans;
}

long long mergeSort(int l,int r) {
    int mid = (l + r) >> 1; // 根据优先级，括号不写也行

    // 内部开销和合并开销
    if(l < r) return mergeSort(l,mid) + mergeSort(mid+1,r) + merge(l,r);
    else return 0;
```

```
}

int main()
{
    int n;
    scanf("%d",&n);
    for(int i = 1;i <= n;i++) {
        scanf("%d",&a[i]);
    }
    printf("%lld\n",mergeSort(1,n));

    return 0;
}
```

在学习指针后，我们对于实现的函数可能有新的写法，而不需要在函数内操作全局数组。同时，基于记录每个元素后面比它小的数的思想也是可以的，但是同样也不能枚举比较，需要利用一些超出教学范围的数据结构，有兴趣的同学可以参考如下几个链接：

- 归并排序
- 树状数组
- P1908 逆序对

- End -