# 零、复杂度

| | |
|---|---|
| **Time limit** | 1000 ms |
| **Memory limit** | 10000 kB |

每道题目都有时空限制。对代码的时间复杂度和空间复杂度进行要求。

时间复杂度

```
for (int i=1; i <= n; ++i) {
    cnt++;
}
```

操作数为$n$，时间复杂度就是$O(n)$.

```
for (int i=1; i <= n; ++i) {
    for (int j=1; j <= n; ++j) {
        cnt++;
    }
}
```

操作数为$n^2$，时间复杂度是$O(n^2)$.

```
for (int i=1; i <= n; ++i) {
    for (int j=1; j <= i; ++i) {
        cnt++;
    }
}
```

操作数为$1+2+3+\ldots+n=\frac{n^2}{2}+\frac{n}{2}$.

时间复杂度为$O(n^2)$.

# 一、枚举

> 枚举出全部的结果找到答案。

百钱买百鸡问题：有一个人有一百块钱，打算买一百只鸡。到市场一看，公鸡一只3元,母鸡一只5元,小鸡3只1元,试求用100元买100只鸡,各为多少才合适?

```
int x,y,z;
for (x=0; x <= 100; ++x) {
    for (y=0; y <= 100; ++y) {
        for (z=0; z <= 100; z+=3) {
            if (x+y+z == 100 && 3 * x + 5 * y + z / 3 == 100) printf("%d %d
%d\n",x,y,z);
        }
    }
}
```

> 优化枚举的范围，减少不必要的枚举

```
    int x,y,z;
    for (x=0; x <= 100; ++x) {
        for (z=0; z <= 100-x; z+=3) {
            y=100-x-z;
            if (x+y+z == 100 && 3 * x + 5 * y + z / 3 == 100) printf("%d %d
%d\n",x,y,z);
        }
    }
```

# 二、模拟

> 模拟就是用计算机来模拟题目中要求的操作。

例一Climbing Worm

一只虫子在n英寸深的井底，它每分钟向上爬u英寸，然后休息一分钟下滑d英寸，重复这个过程，爬出井口需要多长时间?

（ps:不足一分钟的算作一分钟，到达顶部就算爬出井口）

```
int n = 0, u = 0, d = 0;
while (scanf("%d%d%d", &n, &u, &d)) {
    if (n == 0) break;
    int time = 0, dist = 0;
    while (true) {
        dist += u;//向上爬
        time++;
        if (dist >= n) break;//判断是否爬出
        dist -= d;//下滑
        time++;
    }
    printf("%d\n", time);
}
```

# 三、前缀和

# 1、一维前缀和

> 对于一个给定的数列A，它的前缀和数列S通过累加得出：
>
> $$S[i]=\sum_{j=1}^{i}A[j]=S[i-1]+A[i]$$
>
> 数列A的部分和就可以表示成前缀和相减的形式：
>
> $$sum(l,r)=\sum_{i=l}^{r}A[i]=S[r]-S[l-1]$$

例一：acwing 795 前缀和

输入一个长度为n的整数序列。

再输入m次询问，每个询问输入一对l，r。

对于每对询问，输出原序列中从第l个数到第r个数的和。

# 2、二维/多维前缀和

> 在二维或者多维矩阵中，可以类似的求出二维或者多维的前缀和，然后计算部分和。
>
> $$S[i][j]=\sum_{x=1}^{i}\sum_{y=1}^{j}A[x][y]$$

例二：acwing 99 激光炸弹

一种新型的激光炸弹，可以摧毁一个边长为$R$的正方形内的所有目标。

现在地图上有$N$个目标，用整数$X_i$,$Y_i$表示目标在地图上的位置，每个目标都有一个价值$W_i$。

激光炸弹的投放是卫星定位的，但是有一个缺点，就是其爆炸范围，即那个边长为$R$的正方形的边必须和$x$，$y$轴平行。

若目标位于爆破正方形的边上，该目标不会被摧毁。

求一颗炸弹最多能炸掉地图上总价值为多少的目标。

# 四、差分

> 对于一个给定的数列A，它的差分数列B通过后一项减前一项得出：
>
> $$B[1]=A[1];B[i]=A[i]-A[i-1](2\le i\le n)$$
>
> 不难发现“前缀和”和“差分”是一对互逆运算，差分数组B的前缀和序列就是原序列A，前缀和序列S的差分序列也是原序列A。

例一：acwing 797 差分

输入一个长度为n的整数序列。

接下来输出m个操作，每个操作包含三个整数l，r，c，表示将序列中[l，r]之间的每个数加上c。

请你输出进行完所有操作后的序列。

# 五、二分

## 1、二分查找

> 二分查找，也称为折半搜索，是用来在一个**有序**数组中查找某一元素的算法。
>
> 以在一个升序数组中查找一个数为例。
>
> 设置边界l、r，每次考察边界内的中间元素
>
> 1.如果恰好的查找的元素，则结束搜索。
>
> 2.如果边界内的中间元素小于要查找的元素，那么要查找的元素只会出现在中间元素的右侧，压缩查找范围，l=(l+r)/2+1，重新设置左边界。
>
> 3.如果边界内的中间元素大于要查找的元素，那么要查找的元素只会出现在中间元素的左侧，r=(l+r)/2，重新设置右边界。
>
> 二分搜索过程中，每次都把查询的区间减半，对于一个长度为n的数组，至多进行O（$\log_2(n)$）次查找。

```
int binary_search(int l, int r, int x) { //以升序序列为例
    while(l < r) {
        int mid=(l + r) / 2; //mid=(l + r + 1) / 2
        if (a[mid] <= x)
            l = mid + 1; //l = mid;
        else
            r = mid; //r = mid-1;
    }
    return a[(l+r)/2];
}
```

ps:返回值是递增序列A中$\geq X$的数中最小的一个（即X或者X的后继）

mid、l和r分别改成注释的内容后，返回值变成递增序列A中$\leq X$的数中最大的一个（即X或者X的前继）

必须是有序数组才能使用二分搜索，否则答案错误。

## 2、二分答案

> 解题的时候往往会考虑枚举答案然后检验枚举的值是否正确。若满足单调性，则满足使用二分法的条件。把这里的枚举换成二分，就变成了“二分答案”。

例一：洛谷 1873 砍树

伐木工人米尔科需要砍倒M米长的木材。这是一个对米尔科来说很容易的工作，因为他有一个漂亮的新伐木机，可以像野火一样砍倒森林。不过，米尔科只被允许砍倒单行树木。

米尔科的伐木机工作过程如下：米尔科设置一个高度参数H（米），伐木机升起一个巨大的锯片到高度H，并锯掉所有的树比H高的部分（当然，树木不高于H米的部分保持不变）。米尔科就行到树木被锯下的部分。

例如，如果一行树的高度分别为20，15，10和17，米尔科把锯片升到15米的高度，切割后树木剩下的高度将是15，15，10和15，而米尔科将从第1棵树得到5米，从第4棵树得到2米，共得到7米木材。

米尔科非常关注生态保护，所以他不会砍掉过多的木材。这正是他为什么尽可能高地设定伐木机锯片的原因。帮助米尔科找到伐木机锯片的最大的整数高度H，使得他能得到木材至少为M米。换句话说，如果再升高1米，则他将得不到M米木材。

# 六、贪心

> 贪心算法，是用计算机来模拟一个“贪心”的人做出决策的过程。这个人十分贪婪，每一步行动总是按某种指标选取最优的操作。**而且他目光短浅，总是只看眼前，并不考虑以后可能造成的影响**。
>
> 贪心算法在有最优子结构的问题中尤为有效。最优子结构的意思是问题能够分解成子问题来解决，子问题的最优解能递推到最终问题的最优解
>
> 可想而知，并不是所有的时候贪心法都能获得最优解，所以一般使用贪心法的时候，都要确保自己能证明其正确性。

证明方法

> 贪心算法有两种证明方法：反证法和归纳法。一般情况下，一道题只会用到其中的一种方法来证明。
>
> 1. 反证法：如果交换方案中任意两个元素/相邻的两个元素后，答案不会变得更好，那么可以推定目前的解已经是最优解了。
> 2. 归纳法：先算得出边界情况（例如$n = 1$）的最优解$F_1$，然后再证明：对于每个$n$，$F_{n+1}$都可以由$F_n$推导出结果。

例一：洛谷 P1969 搭积木

春春幼儿园举办了一年一度的“积木大赛”。今年比赛的内容是搭建一座宽度为$n$的大厦，大厦可以看成由$n$块宽度为1的积木组成，第i块积木的最终高度需要是$h_i$。

在搭建开始之前，没有任何积木（可以看成$n$块高度为00的积木）。接下来每次操作，小朋友们可以选择一段连续区间$[l, r]$，然后将第第$L$块到第 $R$块之间（含第$L$ 块和第 $R$块）所有积木的高度分别增加1。

小M*M*是个聪明的小朋友，她很快想出了建造大厦的最佳策略，使得建造所需的操作次数最少。但她不是一个勤于动手的孩子，所以想请你帮忙实现这个策略，并求出最少的操作次数。

例一：国王游戏

恰逢 H 国国庆，国王邀请 n 位大臣来玩一个有奖游戏。首先，他让每个大臣在左、右手上面分别写下一个整数，国王自己也在左、右手上各写一个整数。然后，让这 n 位大臣排成一排，国王站在队伍的最前面。排好队后，所有的大臣都会获得国王奖赏的若干金币，每位大臣获得的金币数分别是：排在该大臣前面的所有人的左手上的数的乘积除以他自己右手上的数，然后向下取整得到的结果。

国王不希望某一个大臣获得特别多的奖赏，所以他想请你帮他重新安排一下队伍的顺序，使得获得奖赏最多的大臣，所获奖赏尽可能的少。注意，国王的位置始终在队伍的最前面。

两个相邻的大臣$i$和$i+1$，他们左右手的数字分别为$a_i,b_i$和$a_{i+1},b_{i+1}$假设他们之前的左手和为$A$.

$$ans1=max(\frac{A}{b_i},\frac{A\times a_i}{b_{i+1}})$$

假设这两个大臣交换之后

$$ans2=max(\frac{A}{b_{i+1}},\frac{A\times a_{i+1}}{b_i})$$

替换成$ans1=max(k1,k2),ans2=max(k3,k4)$.

可以得到$k4>k1,k2>k3$.

如果$ans1<ans2$，可以得到$k4>k2$，即$\frac{A\times a_{i+1}}{b_i}>\frac{A\times a_i}{b_{i+1}}$，即$a_i\times b_i<a_{i+1}\times b_{i+1}$.

所以为了$ans$取得较小的值，我们只需要将$a_i\times b_i$乘积较小的放在前面即可.

ps:统计结果时要使用高精度。

# 七、高精度

> 高精度算法，属于处理大数字的数学计算方法。在一般的科学计算中，会经常算到小数点后几百位或者更多，当然也可能是几千亿几百亿的大数字。一般这类数字我们统称为高精度数，高精度算法是用计算机对于超大数据的一种模拟加，减，乘，除，乘方，阶乘，开方等运算。对于非常庞大的数字无法在计算机中正常存储，于是，将这个数字拆开，拆成一位一位的，或者是四位四位的存储到一个数组中， 用一个数组去表示一个数字，这样这个数字就被称为是高精度数。高精度算法就是能处理高精度数各种运算的算法，但又因其特殊性，故从普通数的算法中分离，自成一家。

例一：高精度加法

第一行输入一个$T$，表示测试样例的个数

接下来每一行输入一个$A$，$B$（小于1000位）

输出$A+B$

例二：洛谷 2142 高精度减法

输入一个$A$，$B$（第二个可能比第一个大）

输出$A-B$（结果负数要输出负号）