二分搜索 1

二分搜索，也称为二分答案，在近年的算法竞赛中常被考核到。

二分搜索需要满足以下条件才可以使用：
1、 搜索的范围是确定
2、 搜索的答案是有序的

对于一些问题，我们难以直接求解，但该答案在一个有序范围内，我们则可以假设一个可行解，以二分搜索的方式逐步逼近最优解。

二分搜索的类型，我们可以分为求最大值最小和求最小值最大两种类型。本节课先讲第一种类型。

最大值最小

例：在单调递增序列 a 中查找>=x 的数中的最小一个
【分析】
本题是求最大值最小的模板题。首先，序列是有序的，我们可以从小到大，逐个枚举，找到第一个>=x 的值，即为答案，这种操作的时间复杂度是 O（n）。我们也可以像猜数一样，先猜一个数，如果这个数比 x 要小，那我们下次再猜一个大一点的数，如果这数比 x 大，那我们猜一个小一点的数，不断逼近，直到剩下最后一个数为止。在这里，我们设定一个区间，左边界是 lb，右边界是 ub，我们设定的区间是(lb，ub]，前开后闭的区间，每次选其中间的数 mid=（lb+ub）/2，如果 a[mid]>=x，那么可以在区间(lb,mid]中选一个小的继续判断，注意，这个区间一定存在可行解，最坏情况下可以选 mid；如果 a[mid]<x，那么可以在区间(mid,ub]中选一个大一点的值继续判断，注意，在这个范围内是不包含 mid 的，mid 已被判断不符合条件的。一直进行，直至剩下一个数为止。当剩下一个数时，ub-lb=1，因为这是一个前开后闭的区间。
【代码】

```
#include<iostream>
using namespace std;
int n,a[100],ub,lb,x;
int main(){
    cin>>n>>x;
    for(int i=1;i<=n;i++)cin>>a[i];
    ub=n+1;
    while(ub-lb>1){
        int mid=(ub+lb)/2;
        if(a[mid]>=x)ub=mid;
        else lb=mid;
    }
    cout<<ub;
```

```
14. }
```

二分搜索的代码经常容易出错，一不小心就会写成死循环，关键是理解区间的定义。有另外一种代码风格，将区间定义成一个闭合区间[lb,ub]，那么对应的程序就改为如下：

```
1.  #include<iostream>
2.  using namespace std;
3.  int n,a[100],ub,lb,x;
4.  int main(){
5.      cin>>n>>x;
6.      for(int i=1;i<=n;i++)cin>>a[i];
7.      ub=n+1;
8.      while(ub>lb){ //ub=lb 时就是最后的数
9.          int mid=(ub+lb)/2;
10.         if(a[mid]>=x)ub=mid;//mid 是可行解，区间为[lb,mid]
11.         else lb=mid+1;//mid 不能用，区间是[mid+1,ub]
12.     }
13.     cout<<ub;//输出 ub 或者 lb 都一样，因为 ub=lb
14. }
```

lower_bound()

其实在 c++中，有一个函数也实现了求有序序列中>=x 的值，这个函数就是 lower_bound()，这个函数的实现原理就是以上的二分搜索。lower_bound()的用法如下：

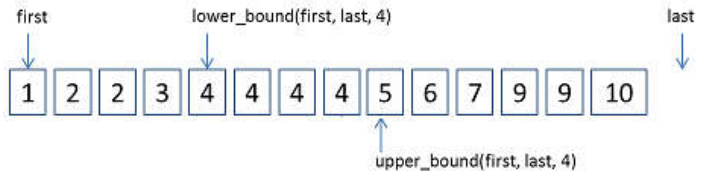
lower_bound(first，last，x)

lower_bound 需要用到三个参数，其中 first 是查找序列的头指针，last 是查找序列的末尾指针，这两个参数的用法有点像 sort 函数的参数，first 和 last 构成一个前闭后开的区间[first,last)，在使用时需特别注意。x 表示要查找的数。lower_bound 返回的是查找到>=x 的第一个数的地址，强调，返回的是一个指针。如果找不到>=x 的数，则返回 last。
以上例子可以用 lower_bound()改写成如下：

```
1.  #include<iostream>
2.  #include<algorithm>  //lower_bound 需要用到的头文件
3.  using namespace std;
4.  int n,a[100],x;
5.  int main(){
6.      cin>>n>>x;
7.      for(int i=1;i<=n;i++)cin>>a[i];
8.      int k=lower_bound(a+1,a+n+1,x)-a; //地址减去 a 得到对应的下标
9.      cout<<k; //k 是下标，输出值应表示为 a[k]
10. }
```

Upper_bound()

lower_bound()是求第一个>=x 的值，那么 upper_bound()是求第一个>x 的值，不包含等于。upper_bound()的参数和 lower_bound()是一样的。下图说明了 lower_bound 和 upper_bound 的区别。



可以结合 lower_bound 和 upper_bound 来求一个数在序列中出现的个数，如：
upper_bound(a+1,a+n+1,x)-lower_bound(a+1,a+n+1,x)
这个式子求出了在序列 a+1 到 a+n 之间数 x 出现的个数。

在使用二分答案之前，我们还经常进行二分查找。

差

【题目描述】
楠楠在网上刷题，感觉第一题：求两数的和(A+B Problem)太无聊了，于是增加了一题：A-B Problem，难倒了一群小朋友，哈哈。
题目是这样的：给出 N 个从小到大排好序的整数，一个差值 C，要求在这 N 个整数中找两个数 A 和 B，使得 A-B=C，问这样的方案有多少种？
例如：N=5，C=2，5 个整数是：2 2 4 810。答案是 3。具体方案：第 3 个数减第 1 个数；第 3 个数减第 2 个数；第 5 个数减第 4 个数。
【输入】
第一行 2 个正整数：N,C。
第二行 N 个整数：已经有序。注意：可能有相同的。
【输出】
一个整数，表示该串数中包含的所有满足 A-B=C 的数对的方案数。
【样例输入】
4 1
1 1 2 2
【样例输出】
4
【数据范围】
5 个数据：N 的范围是[1…1,000]。
5 个数据：N 的范围是[1…100,000]。
所有数据：
C 的范围是[1…1,000,000,000]。
N 个整数中每个数的范围是：[0…1,000,000,000]。
【分析】
本题可以采取多种算法解决。1、枚举，可以枚举任意两个数，判断这两个数的差是不是 c，

如果是则方案加 1，时间复杂度为 O(n^2)，超时。2、枚举每个数，将这个数当作 A，查找是否存在 B=A-c，为了查找到 A-c，可以使用桶排的方式标记每个数出现的次数，时间复杂度为 O(n)，但是数据的范围达到 1,000,000,000，桶排会爆空间。3、还是枚举每个数，将这个数当作 A，查找 A-c 出现的次数，设 B=A-c，因为序列是有序的，则可以使用二分查找的方法查找 B 是否出现。因为 B==A-c，B 一定比 A 小，所以二分查找 B 可以用 lower_bound 找到第一个 B 出现的位置，使用 upper_bound 查找第一个大于 B 的数出现的位置，将 upper_bound 减去 lower_bound 则可以得到 B 出现的次数。时间复杂度为 O(nlogn)。代码如下：

【代码】

```
#include<iostream>
#include<algorithm>
using namespace std;
int n,c,a[100000],ans=0;
int main(){
    cin>>n>>c;
    for(int i=0;i<n;i++){
        cin>>a[i];
        ans+=upper_bound(a,a+i,a[i]-c)-lower_bound(a,a+i,a[i]-c);
    }
    cout<<ans;
}
```

母鸡下蛋

【题目描述】

鸡国中的母鸡最擅长下蛋了，MGMG 是鸡国中一只以下蛋产量高而闻名全鸡国的母鸡。鸡国专供下蛋的 n 个鸡窝呈一字排列在鸡国的“下蛋中心”，从左到右依次编号为 1 到 n。每个鸡窝都有一个最大可下蛋的量，其中第 i 个鸡窝的最大可下蛋量为 ci。有时候由于 MGMG 产量实在太大而无法在一个鸡窝中下完所有的蛋，不得不转移到隔壁的鸡窝继续下蛋，如果隔壁的鸡窝还是不能让它下完所有的蛋，则 MGMG 继续转移，直到下完所有的蛋，或者向“下蛋中心”管理员投诉“鸡窝数量实在太少了，我一只鸡的下蛋量都装不下！”。为了节省转移时所耗费的体力，请你编程帮助 MGMG 找若干个连续的鸡窝(个数尽量少)，让它能下完所有的蛋。

【输入】

输入共 2 行。

第 1 行输入两个整数 n 和 t，表示“下蛋中心”有 n 个可供下蛋的鸡窝，MGMG 一次总共要下 t 个鸡蛋。

第 2 行 n 个正整数 ci(1≤i≤n)，依次表示第 i 个鸡窝最大可下蛋量为 ci 个。

【输出】

输出 1 行一个整数或一个单词。当输出整数时表示让 MGMG 下完所有的蛋至少需要几个连续的鸡窝。当 MGMG 用完所有的鸡窝都无法下完所有的蛋时，MGMG 表示非常愤怒，输出单词“Angry”（不包含双引号，注意大小写）。

【输入样例 1】

5 4
1 2 1 2 3
【输入样例 2】
3 9
3 3 3
【输入样例 3】
3 5
1 2 1
【输出样例 1】
2
【输出样例 2】
3
【输出样例 3】
Angry
【样例 1 解释】
样例 1 中，有 5 个鸡窝，可下蛋量分别为 1，2，1，2，3。MGMG 如果选择第 1，2，3 号鸡窝能下完 4 个蛋，但要用 3 个鸡窝，而选择第 4 号和第 5 号鸡窝也能下完 4 个蛋（还有 1 个多余的容量），但用到的鸡窝只有 2 个。
注意：由于第 2 号和第 4 号鸡窝不连续，不可以作为选择的方案之一。
【样例 2 解释】
样例 2 中，有 3 个鸡窝，可下蛋量分别为 3，3，3，MGMG 可以在这 3 个连续的鸡窝中每个下 3 个蛋，这样正好总共下 9 个蛋。
【样例 3 解释】
样例 3 中，所有鸡窝的可下蛋总量小于 MGMG 的下蛋量，无法满足 MGMG 的下蛋需求，，输出“Angry”。
【数据范围约定】

测试点编号	n	t	$c_i(1\le i\le n)$
1～6	$1\le n\le 100$	$1\le t\le 1000$	$1\le c_i\le 100$
7～14	$1\le n\le 10^5$	$1\le t\le 10^8$	$1\le c_i\le 10000$
15～20	$1\le n\le 10^6$		

【分析】
本题是求一段最少的连续区间，使得其区间和大于等于给定的值 t。首先，关系到区间和的问题，我们要考虑使用前缀和数组，设 s[i]为前 i 个数的和，那么，朴素的算法可以枚举任意一个区间[i,j]，其区间和为 s[j]-s[i-1]，判断这个区间和是否大于等于 t，如果满足条件，则统计其区间个数，与全局数比较。这个算法的时间复杂度为 O(n^2)。如何优化呢？我们构造一个 O(nlogn)的算法。对于区间和 s[i]-s[j]>=t，我们枚举每个 i，对于给定的 i，查找满足一个 s[i]-t 的 j，那么，j 到 i 所构成的区间和便是大于等于 t 的，同时统计 i-j 的最小值。枚举每个 i 的时间为 O(n)，每找一个数，利用二分查找 s[i]-t，时间为 O(logn)，所以总时间复杂度为 O(nlogn)。
【代码】

```
1. #include<iostream>
```

```
#include<algorithm>
using namespace std;
long long sum[1000001];
int n,t,x,ans=99999999;
int main(){
    cin>>n>>t;
    for(int i=1;i<=n;i++){
        cin>>x;
        sum[i]=sum[i-1]+x;  //构造前缀和
        if(sum[i]-t<0)continue;  //整个数列和小于 t，继续循环
        int j=lower_bound(sum,sum+i,sum[i]-t)-sum;  //二分查找 sum[i]-t 的值
        if(j==0)ans=min(ans,i-j);  //特判，刚好满足区间和的情况
        else ans=min(ans,i-j+1);  //统计最小区间个数
    }
    if(ans==99999999){  //如果无解
        cout<<"Angry"<<endl;
    }else{
        cout<<ans<<endl;
    }
}
```

隐形的翅膀

【题目描述】
小杉终于进入了天堂。他看到每个人都带着一双隐形翅膀，他也想要。
（小杉是怎么看到的？……）天使告诉小杉，每只翅膀都有长度，两只翅膀的长度之比越接近黄金分割比例（你可以认为黄金分割比就是 0.6180339887498949），就越完美。
现在天使给了小杉 N 只翅膀，小杉想挑出一对最完美的。
【输入】
每组测试数据的
第一行有一个数 N(2<=N<=30000)
第二行有 N 个不超过 1e5 的正整数，表示 N 只翅膀的长度。
20%的数据 N<=100
【输出】
对每组测试数据输出两个整数，表示小杉挑选出来的一对翅膀。
注意，比较短的在前，如果有多对翅膀的完美程度一样，请输出最小的一对。
【样例输入】
4
2 3 4 6
【样例输出】
2
3
【分析】
本题是给定 n 个数，求 a[i]/a[j]最接近给定值 p 的数，当然，我们可以枚举两个数，将这两

个数的比值与 p 比较，求出最接近的两个数，时间复杂度为 O(n^2)。显然会超时。对于 n 为 30000 的数据，我们需要构造 O(nlogn)的算法。先对数据排序，枚举每个数 a[i]，可以查出，和 a[i]比值最接近 p 的值 a[j]。因为 a[j]/a[i]=p，那么，a[j]=a[i]*p，因为 p 小于 1，所以 a[j]肯定小于 a[i]，所以只需在 a[i]前面的数列中找大于等于 a[i]*p 的值即可。因为求的是比值最接近，所以 a[j-1]/a[i]和 a[j]/a[i]这两个数都有可能，找接近的一个即可。使用二分查找，时间复杂度为 O(nlogn)。

【代码】

```
#include<algorithm>
#include<cmath>
#include<iostream>
using namespace std;
const double p=0.6180339887498949;  //定义常量
int n,x,y,a[30000];
double best;

int main(){
    cin>>n;
    for(int i=0;i<n;i++)cin>>a[i];
    sort(a,a+n);  //排序，保持有序性
    for(int i=1;i<n;i++){  //枚举每个 i
        int j=lower_bound(a,a+i-1,a[i]*p)-a;  //二分查找大于等于 a[i]*p 的位置
        if(fabs((double)a[j]/(double)a[i]-p)<fabs(best-p)){
            best=(double)a[j]/(double)a[i];
            x=i;
            y=j;
        }
        if(j==0)continue;  //没有前一个数的情况跳过
        if(fabs((double)a[j-1]/(double)a[i]-p)<fabs(best-p)){
            best=(double)a[j-1]/(double)a[i];
            x=i;
            y=j-1;
        }
    }
    cout<<a[y]<<endl;
    cout<<a[x]<<endl;
}
```

假设可行解

对于一类答案在有序区间范围但不能直接求解的问题，我们采用的方法是先假设一个可行解，然后验证该解是否可行，如果可行，区间向左移，如果不可行，区间向右移，模板如下：

```
lb=0;
ub=maxn;
```

```
3.  while(ub-lb>1){
4.      int mid=(ub+lb)/2;//mid 是假设的可行解
5.      if(check(mid)){//checke()函数验证可行解
6.          ub=mid;
7.      }else{
8.          lb=mid;
9.      }
10.     cout<<ub;
11. }
```

我们通过以下三道题感受二分搜索的魅力。

和谐分组

【题目描述】

s 班共有 n 名学生，按照学号从 1 到的顺序每名学生的身高分别为 a[1],a[2]…a[n]。由于是新学期，s 班需要进行分组，分组的要求如下：

进行分组的组数不能超过 k。

每组的人的学号必须相邻。

由于身高差过大的人分在同一个组会激起组内内部矛盾（QAQ），所以我们定义一个分 组方案的不和谐度为每个组的身高极差（最高的身高-最矮的身高）的最大值。

我们希望最小化这个不和谐度，输出这个不和谐度。

【输入】

第一行包括两个正整数 n,k。

第二行包括用空格隔开的 n 个正整数，第 i 个正整数描述学号为 i 的学生的身高。

【输出】

一行包括一个整数，表示不和谐度最小的分组方案的不和谐度。

【样例输入】

8 3

5 7 2 3 8 5 9 4

【样例输出】

5

【样例解释】

一种可能的分组是 5 7 2 / 3 8 5 / 9 4

【数据规模】

所有测试数据范围和特点如下：

测试点编号	N,K	$A[i]$
1-3	$1 \le K \le N \le 20$	$1 \le A[i] \le 10^9$
4-5	$1 \le K \le N \le 100$	
6-7	$1 \le K \le N \le 10^3$	
8-10	$1 \le K \le N \le 10^5$	

【分析】

本题是求大值最小。先假设可行解在验证，根据验证结果调整区间范围，见代码实现：

【代码】

```
#include<iostream>
using namespace std;
long long n,k,l=-1,r,a[100001];
bool check(int x){  //x 表示当前分组的不和谐度
    long long sum=1,minn=9999999999LL,maxn=0;
    for(int i=1;i<=n;i++){  //枚举每个人
        minn=min(minn,a[i]);  //找到最小值
        maxn=max(maxn,a[i]);  //找到最大值
        if(maxn-minn>x){  //如果不和谐度超过 x，则另起一组
            sum++;  //组数加 1
            minn=a[i];  //统计新一组的最小值、最大值
            maxn=a[i];
        }
    }
    return sum<=k;  //如果分组数小于等于 k，则方案可行
}
int main(){
    cin>>n>>k;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        r=max(r,a[i]);  //求出右边界 r 值
    }
    while(r-l>1){
        int mid=(r+l)/2;  //mid 为假设的可行解
        if(check(mid))r=mid;  //check(mid)验证可行性，可行，往左边界查找
        else l=mid;  //不可行，往右边界查找
    }
    cout<<r;
}
```

营救

【题目描述】

一座摩天大楼起了大火，n 个人都被困在了顶层狭长的走廊上，大家排着长长的队伍等着逃离险境。但火势很猛，消防员升起的救生舱只有 m 次运人下来的机会，并且每次运的人总重量还不能太重，避免将救生舱压垮。此时如何将这一排人分隔成 m 个连续的小组，（大家遵守逃生守则，没有人会往前插队），并且让这 m 个组中总重量最重的那个组的重量尽量小。这样才能快速安全的将大家都救离险境。

现在告诉你这 n 个人的体重，请你找出一种分组方法，让这 m 个组中总重量最重的那个组的重量尽量小，并输出这个组的总重量。

【输入】

第一行两个正整数 n 和 m，中间用一个空格隔开，表示有 n 个逃生的人和要分隔成 m 个连

续的小组。
第二行 n 个正整数，每个整数之间用一个空格隔开，表示 n 个人的体重(单位:公斤)。
【输出】
一个正整数，表示 m 个组中总重量最重的那个组的重量。
【样例输入】
6 3
20 30 50 80 100 120
【样例输出】
180
【样例解释】
一种合理的分法（20 30 50）（80 100）(120)
【数据范围】
30% 1<=n<=10；5<=m<=10;
70% 1<=n<=100；5<=m<=20;
100% 1<=n<=10000; 100<=m<=1000;
【分析】
本题与上一题类似，假设可行解，然后验证，标准二分搜索求最大值最小
【代码】

```
#include<iostream>
using namespace std;
int a[10001],n,m,lb=1,ub=99999999;  //给 ub 设定一个上限
bool check(int x){  //x 为分组的重量
    int c=1,s=0;  //c 表示分组数量，s 为当前分组的重量
    for(int i=1;i<=n;i++){
        s+=a[i];
        if(s>x){  //如果这个分组重量超过 x，则另起一组
            c++;  //分组数量加 1
            s=a[i];  //新组重量为 a[i]
        }
    }
    return c<=m;  //如果可以分组数量小于 m，则方案可行
}
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>a[i];
    while(ub-lb>1){
        int mid=(ub+lb)/2;  //假定 mid 为可行解
        if(check(mid))ub=mid;  //检验可行解
        else lb=mid;
    }
    cout<<ub;
}
```

Tractor

【题目描述】
小 A 有一片农场区域，大小为 N×N，这 N*N 个格子中，有些为山丘，高度都为正整数，有些是平地，高度为 0。小 A 从当前格子走到相邻格子（东、南、西、北四个方向）的代价为高度差。求小 A 以最小的高度差能走遍所有格子的一半。（如果格子总数为奇数，则一半的值为四舍五入的值）。
【输入】
第一行为一个整数 N；
以下 N 行，每行为 N 个空格分隔开的整数（0..1,000,000）。
【输出】
求遍历一半格子的最小代价。
【样例输入】
5
0 0 0 3 3
0 0 0 0 3
0 9 9 3 3
9 9 9 3 3
9 9 9 9 3
样例输出
3
【样例解释】
高度差为 3。
【数据规模】
对于 20%的数据：1≤N≤10；
对于 40%的数据：1≤N≤50；
对于 60%的数据：1≤N≤100；
对于 100%的数据：1≤N≤500；
【分析】
我们可以假设一个高度差，然后以这个高度差 dfs 棋盘，如果遍历到的数量超过一半，则可以下次再假设小一点的高度差继续判断。同样是二分搜索求最大值最小。
【代码】

```
#include<iostream>
#include<cmath>
#include<cstring>
using namespace std;
int n,a[502][502],l,r;
bool v[502][502];
const int xx[4]={-1,0,1,0};
const int yy[4]={0,1,0,-1};
bool bound(int x,int y){  //判断 x，y 有没有在边界内
    if(x<1||y<1||x>n||y>n)return false;
    return true;
```

```
}
int dfs(int x,int y,int k){  //以 k 的差值从 x，y 访问连通块
    int ans=1;
    v[x][y]=false;
    for(int i=0;i<4;i++){
        int nx=x+xx[i];
        int ny=y+yy[i];
        if(bound(nx,ny)&&v[nx][ny]&&abs(a[x][y]-a[nx][ny])<=k){
            ans+=dfs(nx,ny,k);
        }
    }
    return ans;
}

bool check(int k){  //判断能否以 k 的差值遍历一半棋盘
    memset(v,true,sizeof(v));  //初始化
    for(int i=1;i<=n;i++){  //枚举每一个点
        for(int j=1;j<=n;j++){
            if(v[i][j]){  //如果这个点没访问过
                int ans=dfs(i,j,k);  //统计从 i，j 遍历的连通块数量
                if(ans>=n*n/2)return true;  //如果超过一半，则返回 true
            }
        }
    }
    return false;  //退出表示不合要求
}
int main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            cin>>a[i][j];
        }
    }
    r=1000000;  //设置一个理论上限
    while(r-l>1){
        int mid=(r+l)/2;
        if(check(mid)){  //检验答案是否可行
            r=mid;
        }else{
            l=mid;
        }
    }
    cout<<r;
}
```