本篇针对完全零基础同学。阅读完本篇或者有基础的同学，可以转战推荐的另一篇博客：

https://blog.csdn.net/2302_79577794/article/details/139581587

问题一

给出一组单调递增的**有序**数列：1，3，7，11，19，23，53，77。请问11是数列的第几个数字？23呢？你是通过怎样的思路解决这个问题的？

问题二

扩大数列长度，推广到长度为$n(1 \le n \le 10^6)$，给定**升序**排列的数列$\{a_n\}$，询问数字k在其中出现的位置，保证k一定在数列中出现，保证数列中的数字**互不相同**。沿用第一个问题的思路，如何解决这个问题？

针对这两个问题，我们都可以使用顺序查找数字的思路，从头开始一个个数，数到我们想要的数字k，就可以停下来，记录答案了。

代码：

```
cin>>k;//读入询问数字k
for(int i=1;i<=n;++i)
{
    if(a[i]==k)
    {
        cout<<i<<'\n';
        break;
    }
}
```

这段代码的时间复杂度为$O(n)$，对于这个问题的数据范围要求而言完全够用。

问题三

继续推广问题。给定**升序**排列的数列$\{a_n\}(1 \le n \le 10^6)$，给定询问次数$q(1 \le q \le 10^6)$，每次询问数字$k$在数列中出现的位置，保证$k$一定在数列中出现，保证数列中的数字**互不相同**。

如果我们还是使用顺序查找数字的思路，从头开始数，会怎么样呢？

代码：

```
cin>>q;//读入询问次数
for(int i=1;i<=q;++i)
{
    cin>>k;//读入询问数字k
    for(int j=1;j<=n;++j)
    {
        if(a[j]==k)
        {
            cout<<j<<'\n';
            break;
        }
    }
}
```

这段代码的时间复杂度为$O(qn)$，在给定的数据范围下最多会跑到约10^{12}次，显然会超时。

思考

我们是否能加速这个查找过程呢，比如提高第二重循环的效率?

算法的优化可以从数据的特性出发，思考一下，我们遇到的三个问题的数列a_n有什么特点?

单调性

单调递增一直是上面问题所强调的特性，我们似乎可以更好地利用它，这就需要用到本篇的主题——**二分法。**

整数二分

什么是二分?

假设我们在玩一个游戏，猜数字：主持人在 1~100 中选定一个数字，然后参与者依次说出自己猜的数字，然后主持人会告诉你：猜大了，猜小了，或者猜中了。

为了快速缩小范围，我们猜的第一个数往往都是50，为什么?

因为选择50的话，猜对了不谈。如果猜大了，那么就意味着51~100这些更大的数字一定也会猜大了，这样就排除了一半的错误答案；同理，如果猜小了，那么1 ~ 49也一定会猜小了，同样会排除了一半的错误答案。无论是什么结果，我们最差都能排除一半的错误答案，也就是缩小了一半的范围。

如果我们每次都在剩下的数字的范围中，选中间的数猜，那么每次都至少缩小**一半**的范围，直到最后剩下一个数，那必然就是我们要的答案。

这就是二分法的核心思想——**折半**，每次查找都可以去除一半的数据量，直到最后将所有不符合条件的结果都去除，只剩下一个符合条件的结果。

这种比大小折半的算法的前提，就是查找的整数序列必须**有序**，所以我们才能与序列其中一个数比大小的结果，来推断序列中其他数或大或小或不可知。显然，我们开篇遇到的问题也具备这个前提。

问题一

回到开篇的问题：给出一组单调递增的**有序**数列：1，3，7，11，19，23，53，77。请问11是数列的第几个数字？23呢？

如果使用二分查找11或者23，应该怎么找？需要查找几次？（画图模拟更直观）

问题二

给定**升序**排列的数列$\{a_n\}(1 \le n \le 10^6)$，询问数字$k$在数列中出现的位置，保证$k$一定在数列中出现，保证数列中的数字**互不相同。**

沿用二分查找的思路，这里直接给出代码：

```
int l=1,r=n,Ans;//l,r是查找数列的下标区间的左右端点，Ans就是要找的答案
while(l<=r)
{
    int mid=(l+r)/2;
    if(a[mid]==k)//找到的数字k，记录答案并退出
    {
        Ans=mid;
        break;
    }
    else if(a[mid]<k) l=mid+1;//找到的数字比k小，说明答案在右半部分，收缩区间左端点
    else if(a[mid]>k) r=mid-1;//找到的数字比k大，说明答案在左半部分，收缩区间右端点
}
```

这个代码的时间复杂度是$O(logn)$，如果带上询问次数$q(1 \le q \le 10^6)$的话，所有询问的复杂度为$O(qlogn)$，完全够用的复杂度。

问题三

稍微变更一下问题，如果原来的数组的特性从**单调递增**改为**单调不减**(即允许出现相同的数字)呢？

给定**升序**排列的数列$a_n(1 \le n \le 10^6)$，询问数字k在数列中**第一次出现**的位置，保证k一定在数列中出现。

代码：

```
int l=1,r=n,Ans;//Ans就是要找的答案
while(l<=r)
{
    int mid=(l+r)/2;
    if(a[mid]==k)//找到的数字等于k，说明答案可能是当前位置或者更左边的位置，但一定不是右半区间
    {
        r=mid-1;
        Ans=mid;
    }
    else if(a[mid]<k) l=mid+1;//找到的数字比k小，说明答案在右半部分，收缩区间左端点
    else if(a[mid]>k) r=mid-1;//找到的数字比k大，说明答案在左半部分，收缩区间右端点
}
```

上面的代码可以稍作简化，改成如下所示，一样能解决问题：

```
int l=1,r=n,Ans;//Ans就是要找的答案
while(l<=r)
{
    int mid=(l+r)/2;
    if(a[mid]<k) l=mid+1;
    else if(a[mid]>=k) r=mid-1,Ans=mid;
}
```

这段更简短的代码实际上是解决**“查找单调不减序列中，第一个大于等于k的数字的位置”**的问题(与问题三相比，这里的k可以不在数列中出现)，这是二分查找中比较常用的代码模板之一。

```
int l=1,r=n,Ans;//Ans就是要找的答案
while(l<=r)
{
    int mid=(l+r)/2;
    if(a[mid]>k) r=mid-1;
    else if(a[mid]<=k) l=mid+1,Ans=mid;
}
```

而这段代码模板就是解决**“查找单调不减序列中，最后大于等于k的数字的位置”**的问题。

尽量理解上面两段代码的差异，最好自己拿数据代入模拟一下。

二分答案

前文提到，二分法的作用前提是查找区间具备**单调性**。

当然，具备单调性的不仅仅局限于一个数字序列。当我们考虑一个比较复杂的问题时，如果问题中预估的答案存在一个区间，我们同样可以对这个答案区间做二分，找到最优解。

二分答案的技巧常用于一些求最大或最小符合要求的答案的题目，这种题目通常正向去求往往较为困难的。这个技巧的思路一般是：

1. 二分法找到一个可能答案；
2. 用这个可能答案代入题干，去验证其是否符合要求；
3. 根据第2步的结果，收缩答案区间，并记录答案。

二分答案的技巧较为灵活，最好根据题目理解。接下来，请看题。

例题一

木材加工

木材厂有 $n(1\le n\le 10^5)$ 根原木，每根木头的长度是$L[i](1\le L[i]\le 10^8)$，现在想把这些木头切割成 $k(1\le k\le 10^8)$ 段长度**均**为 $l(1\le l\le 10^8)$ 的小段木头（木头有可能有剩余）。当然，我们希望得到的小段木头越长越好，请求出 l 的最大值。

木头长度的单位是 cm，原木的长度都是正整数，我们要求切割得到的小段木头的长度也是正整数。例如有两根原木长度分别为 11 和 21，要求切割成等长的 6 段，很明显能切割出来的小段木头长度最长为 5。

我们假设 $l=l'$ ，那么倒推回去，我们让每根木头以每小段长度为 l' 地切割，就可以轻松地得出小段木头数量上限 k' ，如果$k'\ge k$ ，那就意味着 l' 是个合法的 l 。显然，所有合法的 l 应当构成一段答案区间，我们要找的就是这个答案区间的最大值。

直接枚举 l 肯定不行，范围太大会超时。

不过这题的特点符合二分法的要求，合法答案是一段区间，存在单调性，因此我们完全可以二分答案，去验证答案合法性，最后得出我们想要的最大合法答案。

代码：

```
bool check(int mid)//检验mid是否合法
{
    int cnt=0;
    for(int i=1;i<=n;++i) cnt+=L[i]/mid;
    if(cnt>=k) return 1;
    else return 0;
}
int work()
{
    int l=1,r=1e8,Ans;
    while(l<=r)
    {
        int mid=(l+r)/2;
        if(check(mid)) Ans=mid,l=mid+1;
        else r=mid-1;
    }
    return Ans;
}
```

时间复杂度为$O(nlog(10^8))$，可以通过本题。

例题二

跳石头

直线上一个起点和一个终点，中间有n个点，告诉你这些点之间的距离，请问去掉任意m个点以后，剩下的点之间的距离的最小值最大是多少。
$(1 \leq n,m \leq 5*10^5,1 \leq L \leq 10^9\text{，这里的}L\text{指代相邻两点间距离})$

用二分法思路，尝试假设答案，然后去反向推理。

在考虑任意两点间距离最小值的问题中，我们完全可以看成考虑任意相邻两点间距离最小值。假设剩下的点之间距离最小值为 k ，我们要去检验 k 的合法性。如果我们想要"剩下的点之间距离最小值为 k "这个命题成立，就需要不断找到相邻距离小于 k 的两点，择其一删除，删到最后任意相邻两点间距离都大于等于 k 。

这里有个小问题，怎么删点?

显然存在许多删点方案符合要求，但是我们需要找到其中删点数量最少的方案，这样得出的"下限"去和限制条件中的 m 比较，才能正确检验 k 的合法性。由于起点和终点固定且不能删除，所以我们从起点开始，一个个删除与起点距离小于 k 的点，直到碰到第一个与起点的距离大于 k 的点。以此类推，到终点时改为逐个删除终点前面的点即可。这样能保证删点数量最少，记该删点数为 m' ，比较 m' 与 m 的关系即可。

代码:

```
bool check(int mid)
{
    int cnt=0;
    int i=0,now=0;
    while(i<n+1)//i表示点的编号，0代表起点，n+1代表终点,now表示当前点的编号
    {
        ++i;
        if(a[i]-a[now]<mid) ++cnt;//距离太小，删除第i个点
        else now=i;
    }
    if(cnt<=m) return 1;
    else return 0;
}
int work()
{
    int l=0,r=n,Ans;
    while(l<=r)
    {
        int mid=(l+r)/2;
        if(check(mid)) Ans=mid,l=mid+1;
        else r=mid-1;
    }
    return Ans;
}
```

总结

二分答案问题，往往涉及到**最大值最小**和**最小值最大**两类要求，同时也普遍存在着正面很难解决，反过来却比较好做的特点。

尝试用二分答案解决问题时，要思考答案是否具备**单调性**和**检验**答案的方法。

最短距离最大化问题：保证任意区间距离要比最短距离 mid 大或相等（这样，mid 才是最短距离）即：区间的距离 $\geq mid$

最长距离最小化问题：保证任意区间距离要比最大距离 mid 小或相等（这样，mid 才是最大距离）即：区间的距离 $\leq mid$

训练题单

https://vjudge.net/article/7448