> **A题解题报告：思维+前缀和+lower_bound优化**
>
> 题意：n个糖罐，每个糖罐坚固度为i。固定力量值d，每次攻击使糖罐坚固度变为x/d向下取整。有m把钥匙可打开对应坚固度bi的糖罐，多次询问区间内在打开最多糖罐的前提下携带钥匙数。

首先思考一个问题：给定的钥匙数有没有重复的？
假如有两把钥匙□□，分别能打开坚固度为2和5的糖罐。 力量值假设为2。
我们发现，型号为5的钥匙经过攻击操作，5/2向下取整后的值和另一把钥匙的型号相同。
也就是说，所有能够用5号钥匙打开的糖罐都能用2号钥匙代替。
（注意到，能够被5号钥匙打开，指的是经过有限次攻击操作坚固度降为5的那些糖罐；此时再进行一次攻击操作，坚固度变为2，可用2号钥匙打开）
所以说，我们的钥匙里用一部分是重复的，或者说无用的，为了后续计算的方便，我们应该首先把这些钥匙剔除出去。
考虑到钥匙的数量最多也只有60把，这里我们可以暴力剔除。

```
void move_same(){
 sort(key+1,key+1+key_num);//先排个序，小的肯定没法被大的代替
 for(int i=1;i<=key_num;++i){
  if(!flag[i]){
   ll temp;
   for(int j=i+1;j<=key_num;++j){
    temp=key[j];
//key[j]经过有限次攻击操作后和key[i]相同 while(temp>=key[i]){
//可以认为，钥匙j可被钥匙i完全替代          if(temp==key[i]){
      flag[j]=true;
      break;
     }
     temp/=power;
    }
   }
  }
 }
 for(int i=1;i<=key_num;++i){
  if(!flag[i]) Key[++tot]=key[i];
 }//去重后得到的新的钥匙集合
}
```

对钥匙预处理后，很明显我们有以下结论：如果可以被打开的话，每个糖罐只与一把钥匙匹配 ~~（要不然我们刚刚的操作是干啥的）~~。
那我们先把这个匹配情况统计出来，下面的计算会用到的。

```
void match_Key(){
 ll lb=Key[1],ub=Key[tot];
 ll temp1,temp2;
 for(int i=1;i<=n;++i){
  temp1=sugar[i];
  while(temp1>ub){
   temp1/=power;
   //先减少到进入钥匙范围内再说
  }
  while(temp1>=lb){
   temp2=lower_bound(Key+1,Key+tot+1,temp1)-Key;
   //Key钥匙集合是已经排过序的
   //利用lower_bound函数可以降低一定的时间复杂度
   if(Key[temp2]==temp1) {
    match[i]=temp2;
    break;
   }
   temp1/=power;
  }
 }
}
```

通过统计各个钥匙的前缀和（有使用就加1），就可以知道指定区间内的各个钥匙使用的情况了。
询问区间钥匙使用情况时，就可以遍历去重后的钥匙集合，统计那些区间内使用过的钥匙，加起来就是我们所求的结果。

```
//这个代码就非常的简单了
void figure_sum(){
 for(int i=1;i<=n;++i){
  sum_key[match[i]][i]=++cnt[match[i]];
  for(int j=1;j<=tot;++j){
   sum_key[j][i]=cnt[j];
  }
 }
}
```

> **时间复杂度O(n*logn*logn+n*m):**
>
> **钥匙去重因为数据量很小的原因可以直接忽略，求前缀和直接看代码就能看出来最多60*n*,*单次查询就最坏查全部，一共查m次。匹配钥匙的时候每个糖罐都要匹配，一共n个糖罐；匹配的时候要不断除以力量值，log级别，查询是否匹配用lower_bound函数，也是log级别的，乘起来就是nlognlogn。总的时间复杂度就是(n*lognlogn+nm)。*