题目限制

2000ms 28M

题意描述

水王有一个由 n 个整数组成的多重集。他希望你处理以下两种操作：

```
- 将整数 k 加入集合;

- 删除集合中第 k 小的数。
```

如果当前操作需要删除一个在集合中多次出现的元素，则只删除其中的一个。

在处理所有操作之后，如果它是空的，输出 "0" ； 否则输出集合中最小的数字。

输入格式

第一行包含两个整数 n 和 $q(1 \leq n, q \leq 10^6)$ ——初始集合中的元素数和操作数。

第二行包含 n 个整数 $a_1, a_2，\ \ldots，\ a_n(1 \leq a_1 \leq a_2 \leq \ldots\ldots \leq a_n \leq n)$ 表示多重集中的元素。

第三行包含 q 个整数$k_1, k_2，\ \ldots，\ k_q$，每个 k 表示一个操作:

-如果 $1 \leq k_i \leq n$ ，则第 i 个操作为“在集合中加入 k_i”

-如果 $k_i < 0$ ，则第 i 个操作是“从集合中删除第 $|k_i|$ 小的数”。对于这个操作，可以保证 $|k_i|$ 不大于集合当前的大小。

输出格式

如果所有操作后的集合为空，则打印" 0 "。

否则，否则输出集合中最小的数字。

样例输入1

```
5 5
1 2 3 4 5
-1 -1 -1 -1 -1
```

样例输出1

```
0
```

样例输入2

```
5 4
1 2 3 4 5
-5 -1 -3 -1
```

样例输出2

```
3
```

样例输入3

```
6 2
1 1 1 2 3 4
5 6
```

样例输出3

```
6
```

题解

题意解释

给定一个大小为n的初始集合，里面的元素范围是1到n。接下来有q次操作，操作有两种：将整数k加入集合或者删除集合里第k小的数字。

处理所有操作之后，如果集合为空则输出0，否则输出集合中的最小的数字。

第一个样例，每次修改后的集合为：

2 3 4 5

3 4 5

4 5

5

空

因为集合最后为空，所以输出0

第二个样例，每次修改后的集合为：

1 2 3 4

2 3 4

2 3

3

最后集合剩下了3，所以输出3

第三个样例，每次修改后的集合为：

1 1 1 2 3 4 5

1 1 1 2 3 4 5 6

最后集合内还剩8个数字，因为要输出最小的数字，所以输出1

知识点提炼

数据结构或者二分答案

核心解题思路

思路1：直接输出0骗分(5pts)

注意到如果集合为空需要输出0，所以直接输出0骗分，有可能能拿到分。

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    cout<<0;
    return 0;
}
```

时间复杂度$O(1)$。期望得分$0-5$分。

思路2：直接加数字，sort排序后删数字(35pts)

考虑用无序的数组维护集合，即不时时维护数组的有序。

对于加数字操作，可以直接对n加一，然后把k放在a_n的位置。复杂度为$O(1)$。

对于删数字操作，因为不需要维护有序性，可以对数组排序，然后删除a_k。删除a_k可以将a_k和a_n交换，然后对n减一来完成。复杂度为$O(nlogn)$。

操作完成后，如果$n=0$则输出0，否则排序后输出a_1。

```
#include<bits/stdc++.h>
using namespace std;
int n,q,a[2000010],k;
int main()
{
    scanf("%d%d",&n,&q);
    for(int i=1;i<=n;i++)
        scanf("%d",&a[i]);
    for(int i=1;i<=q;i++)
    {
        scanf("%d",&k);
        if(k>0)
        {
            n++;
            a[n]=k;//直接将k放在最后
        }
        else
        {
            k=-k;
            sort(a+1,a+1+n);//对集合排序
            swap(a[k],a[n]);//将a[n]调入a[k]的位置
            n--;//删掉原来的a[k]
        }
    }
    if(n==0)
```

```
        cout<<0;
    else
    {
        sort(a+1,a+1+n);//因为无序，所以需要排序后输出
        cout<<a[1];
    }
    return 0;
}
```

时间复杂度$O(qnlogn)$。期望得分40分。

思路3：维护有序集合(60pts)

在思路2的基础上，我们发现操作1的复杂度较小，而操作2的复杂度过大，考虑如何平衡二者的复杂度来获取更优的时间复杂度。
一种方法是在所有操作前，先对数组排序。以后时时刻刻维护一个有序的数组。

加数字时，对n加一，将k暂时放置在a[n]的位置上。然后将它不断往前调整位置，直到数组重新变的有序，即可加入数字k并且不改变有序性。复杂度为O(n)。

删数字时，可以将k位置后面的数字不断向前移动，然后对n减一，即可删除第k小的数字并且不改变有序性。复杂度为O(n)。

操作完成后，如果n=0则输出0，否则不需排序，直接输出a[1]。

```
#include<bits/stdc++.h>
using namespace std;
int n,q,a[2000010],k;
int main()
{
    scanf("%d%d",&n,&q);
    for(int i=1;i<=n;i++)
        scanf("%d",&a[i]);
    sort(a+1,a+1+n);//先进行一次排序，以后每次修改前后都是有序的
    for(int i=1;i<=q;i++)
    {
        scanf("%d",&k);
        if(k>0)
        {
            n++;
            a[n]=k;//将k暂时安放在an
            for(int j=n;j>1;j--)
                if(a[j]<a[j-1])//然后将k向前调整
                    swap(a[j],a[j-1]);
                else//不需要调整时break
                    break;
        }
        else
        {
            k=-k;//取反获得真正的k
            for(int i=k;i<n;i++)
                a[i]=a[i+1];//删除a[k]
            n--;//集合大小减一
        }
    }
    if(n==0)
```

```
        cout<<0;
    else
        cout<<a[1];//因为是有序的，所以直接输出即可
    return 0;
}
```

时间复杂度$O(qn)$，期望得分60分。

思路4：使用权值线段树维护集合(100pts)

注意到初始集合与k都是1到n范围内的，可以很方便的使用权值线段树，维护区间数字出现的次数继而维护整个集合。

刚开始的初始集合可以认为是n次加入数字的操作，使用线段树可以做到复杂度$O(nlogn)$。

对于加数操作，是线段树上的单点修改，复杂度$O(logn)$。

对于删除第k小，我们可以使用线段树上二分进行删除。具体的思路等同于找第k小的操作：如果左子树里元素数量大于等于k，则第k小元素位于左子树，是左子树的第k小；否则第k小元素位于右子树，是右子树里的**k-左子树元素数量**小。递归地处理即可。在寻找第k小的过程中，对区间数字出现次数减一即可删除第k小。

也可以先找到第k小，再执行删除操作。复杂度都是$O(logn)$的。

修改操作结束后，根节点的sum即为剩余集合元素数量，如果为0直接输出0即可。否则我们需要输出最小的元素，可以再次使用寻找第k小的函数即寻找第一小，也可以在线段树上贪心：如果左子树里有元素，则最小的元素存在于左子树，否则存在于右子树，递归地寻找即可。复杂度为$O(logn)$。

```
#include<bits/stdc++.h>
using namespace std;
int c[4000010],n,m,q;
void add(int x,int l,int r,int d)//单点加
{
    c[x]++;//d位于当前区间，将总数++
    if(l==r)//到了叶子节点，返回
        return ;
    int mid=(l+r)/2;
    if(d<=mid)//如果d在左子树里，则进入左子树
        add(x*2,l,mid,d);
    else//否则进入右子树
        add(x*2+1,mid+1,r,d);
}
void jian(int x,int l,int r,int k)
{
    c[x]--;//第k小位于当前区间，将总数--
    if(l==r)//如果到了叶子节点，返回
        return ;
    int mid=(l+r)/2;
    if(c[x*2]>=k)//如果左子树里元素数量大于等于k，则证明第k小位于左子树
        jian(x*2,l,mid,k);
    else//否则当前区间第k小是右子树里的第(k-c[x*2])小，进入右子树
        jian(x*2+1,mid+1,r,k-c[x*2]);
}
void print(int x,int l,int r)
{
    if(l==r)//如果到了叶子节点则找到了最小值，输出并返回
```

```
    {
        printf("%d",l);
        return ;
    }
    int mid=(l+r)/2;
    if(c[x*2])//如果左子树里有点，则最小值位于左子树
        print(x*2,l,mid);
    else//否则位于右子树
        print(x*2+1,mid+1,r);
}
int main()
{
    n=read();q=read();
    for(int i=1;i<=n;i++)
        add(1,1,n,read());//进行n次加点操作
    for(;q;q--)
    {
        int k=read();
        if(k>=1)
            add(1,1,n,k);//加点操作
        else
            jian(1,1,n,-k);//删除第k小
    }
    if(c[1]==0)//如果集合大小为0
        printf("0");
    else
        print(1,1,n);//贪心地寻找最小值
}
```

思路5：二分答案(100pts)

考虑读入初始数组和修改操作后二分答案，计算$\leq mid$的元素在q次操作后的集合的个数。我们要找到第一个大于0的位置。

二分的理由很容易理解，要思考的是$count(x)$函数的具体实现。

对于初始集合，我们统计$a[i] \leq x$的数量。

对于加数操作，如果加的数字是小于等于x的，则数量加一。

对于删数操作，如果k小于等于当前数量，则证明被删的数是小于等于x的，我们要对数量减一。否则证明被删的数字是大于x的，不用理会。

正解对我们的启示：对于题目进行深度思考，由题目特性出发，得到专用算法，一般会极大的减少代码难度。

```
#include<bits/stdc++.h>
using namespace std;
int read()
{
    int x;scanf("%d",&x);return x;
}
int n,q,a[1000010],k[1000010];
int count(int x)//统计q次操作后小于等于x的元素的数量
{
    int cnt=0;
```

```
    for(int i=1;i<=n;i++)
        if(a[i]<=x)
            cnt++;//统计初始时小于等于x的元素的数量
    for(int i=1;i<=q;i++)
    {
        if(k[i]>0&&k[i]<=x)//加数且加的数字小于等于x
            cnt++;//cnt加一
        if(k[i]<0&&-k[i]<=cnt)//删数且k小于等于当前数量
            cnt--;//cnt减一
    }
    return cnt;
}

int main()
{
    freopen("set.in","r",stdin);
    freopen("set.out","w",stdout);
    n=read();q=read();
    for(int i=1;i<=n;i++)
        a[i]=read();
    for(int i=1;i<=q;i++)
        k[i]=read();
    if(count(1000000000)==0)//如果小于等于1e9的数字数量仍为0，证明q次修改后集合元素数量为0
    {
        printf("0");
        return 0;
    }
    int l=0,r=1000000;
    while(r-l>1)//二分过程
    {
        int mid=(l+r)/2;
        if(count(mid)>0)
            r=mid;
        else
            l=mid;
    }
    cout<<r;
    return 0;
}
```

本题易错点

本题在考虑部分分时容易不深入思考，选择$O(qnlogn)$的方法而不均摊时间复杂度来达到$O(qn)$的复杂度。

在编写线段树方案时容易忽略空间限制，线段树空间开得太大而拿不到满分。

在编写线段树方案时如果对线段树上二分的做法不熟练，有可能选择二分套线段树使得复杂度变为$qlog^2n$，代码更难实现并且复杂度更高。

在编写二分答案方案时容易在二分的实现上出现失误，造成边界处的错误例如答案多1、少1或者应该输出0而没有输出0。

标准代码

C++

```cpp
#include<bits/stdc++.h>
using namespace std;
int read()
{
    int x;scanf("%d",&x);return x;
}
int n,q,a[1000010],k[1000010];
int count(int x)
{
    int cnt=0;
    for(int i=1;i<=n;i++)
        if(a[i]<=x)
            cnt++;
    for(int i=1;i<=q;i++)
    {
        if(k[i]>0&&k[i]<=x)
            cnt++;
        if(k[i]<0&&-k[i]<=cnt)
            cnt--;
    }
    return cnt;
}

int main()
{
    freopen("set.in","r",stdin);
    freopen("set.out","w",stdout);
    n=read();q=read();
    for(int i=1;i<=n;i++)
        a[i]=read();
    for(int i=1;i<=q;i++)
        k[i]=read();
    if(count(1000000000)==0)
    {
        printf("0");
        return 0;
    }
    int l=0,r=1000000;
    while(r-l>1)
    {
        int mid=(l+r)/2;
        if(count(mid)>0)
            r=mid;
        else
            l=mid;
    }
    cout<<r;
    return 0;
}
```