```cpp
#include<bits/stdc++.h>
#define int int_fast64_t
using namespace std;
int a[int(1e6+5)],n,s;
class szsz{
public:
    void update(int x,int y){
        for(int i=x;i<=int(1e6);i+=lowbit(i)){
            tr[i]+=y;
        }
    }
    int query(int x){
        if(x>int(1e6))x=int(1e6);//进食后人，树状数组只需要维护 1~1e6 的数值，但是 m 为 1e12，在下面的查询可能会查到 m，直接导致数组越界 RE，开 1e12 的数组也是不可能的，直接 MLE。我们树状数组的作用就是统计 ≤ 某个数的 a[i] 个数或和，值域上限是 1e6，即所有 a[i] 都 ≤ 1e6，不管你查多大，统计的都是全部的 a[i]，结果和直接查 1e6 完全一样。
        int ans=0;
        for(int i=x;i>=1;i-=lowbit(i)){
            ans+=tr[i];
        }
        return ans;
    }
private:
    int lowbit(int x){
        return x&(-x);
    }
    int tr[int(1e6+5)];
}cnt,sum;
bool check(const int& x/*候选篮子数*/,const int& m/*篮子数*/,const int& k/*目标糖果数*/){
    //最少能放 k 个糖的状态
```

```
    int sc=n-cnt.query(x);//计算 a[i] > x 的元素个数（这些人必须给前 x 个篮子各放 1 颗糖）
    int ss=sum.query(x);//计算 a[i] ≤ x 的所有元素的总和（这些人可以把糖全放前 x 个篮子）
    int st=sc*x+ss;//计算前 x 个篮子的最小总糖数（无论怎么放，最少能拿到的糖）
    bool sf=(st>=k*x);//如果最小总糖数 ≥ 目标总糖，则“最少能放 k 个糖的状态”条件成立

    //最多能分多少糖
    int dn=m-x;//后篮子的数量 = 总篮子 - 前 x 个篮子
    int dc,ds;
    if(dn>=0){//后篮子的数量不能为负
        dc=n-cnt.query(dn);//a_i > 后篮子的数量的元素个数（这些人必须先放满后篮子的数量个篮子）
        ds=sum.query(dn);//a_i ≤ 后篮子的数量的所有元素的总和（这些人可以把糖全放后篮子的数量个篮子）
    }else{
        dc=n,ds=0;
    }
    int dt=s-ds-(dc*dn);//计算前 x 个篮子的最大总糖数（无论怎么放，最多能拿到的糖）
    bool df=(dt<=k*x);//如果最大总糖数 ≤ 目标总糖，则“最多能放 k 个糖的状态”条件成立

    return (sf&&df);//需要两个条件同时成立，候选的篮子数才合法。
}
int ef(const int& m,const int& k){
    int l=-1,r=m+1;
    while(l+1<r){
        int mid=l+r>>1;
```

```
        if(check(mid,m,k))l=mid;
        else r=mid;
    }
    return l;
}
int32_t main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr),cout.tie(nullptr);
    int c,q;
    cin>>c>>n>>q;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        cnt.update(a[i],1);
        sum.update(a[i],a[i]);
        s+=a[i];
    }
    while(q--){
        int op;
        cin>>op;
        if(op==1){
            int x,y;
            cin>>x>>y;
            cnt.update(a[x],-1);
            sum.update(a[x],-a[x]);
            s=s-a[x]+y;
            a[x]=y;
            cnt.update(a[x],1);
            sum.update(a[x],a[x]);
        }else{
            int m,k;
            cin>>m>>k;
            cout<<ef(m,k)<<'\n';
        }
    }
    return 0;
}
```