个人对整体二分的理解是这样子的，首先对于修改，即把a[xi]=yi（1<=yi<=C）的时候，我们可以把修改的操作划分成两部分，一部分是yi<=mid,另一部分是yi>mid。首先我们忽略掉yi>mid的操作，将yi<=mid以及询问操作按照输入的顺序执行一遍，这样我们就可以知道<=mid的操作对所有询问的贡献，接下来我们就要根据贡献对修改操作划分成两部分，一部分是答案在<=mid里面的，另外一部分是答案在>mid里面的，对于<=mid的，显然我们可以递归求解（因为>mid的操作对那些<=mid是没有影响的），而对于>mid的来说，<=mid的操作是有影响的，但是<=mid所造成的影响已经算过一遍了，所以对于>mid的来说，其实也是独立的。所以如果我们把答案二分的范围的大小设为C，操作的总数设为n，那么划分到<=mid的操作是n1,>mid的操作是n2时（n1+n2=n）

int n,m;

int ans[maxn];

struct Bit{

int n,treenum[maxn];

void init(int k){

n=k;

memset(treenum,0,sizeof treenum);

}

int lowbit(int x){

return x&(-x);

}

void add(int i,int add){

while(i<=n){

treenum[i]+=add;

i+=lowbit(i);

}

}

int sum(int i){

int sum=0;

while(i){

sum+=treenum[i];

i-=lowbit(i);

}

return sum;

}

}bit;

struct Node{

int x,y,k;

int id,type;

}q[maxn],ql[maxn],qr[maxn];

int a[maxn];

void solve(int iql,int iqr,int l,int r){

if(iql>iqr)

return;

if(l==r){

for(int i=iql; i<=iqr; i++)

if(q[i].type==2)

ans[q[i].id]=l;

return;

}

int m=(l+r)>>1;

int p1=0,p2=0;

for(int i=iql; i<=iqr; i++){

if(q[i].type==1){

if(q[i].x<=m){

bit.add(q[i].id,q[i].y);

ql[p1++]=q[i];

}

else

qr[p2++]=q[i];

}

else{

int cnt=bit.sum(q[i].y)-bit.sum(q[i].x-1);

if(cnt>=q[i].k){

ql[p1++]=q[i];

}

else{

q[i].k-=cnt;

qr[p2++]=q[i];

}

}

}

for(int i=0; i<p1; i++)

if(ql[i].type==1)

bit.add(ql[i].id,-ql[i].y);

for(int i=0; i<p1; i++)

q[iql+i]=ql[i];

for(int i=0; i<p2; i++)

q[iql+p1+i]=qr[i];

solve(iql,iql+p1-1,l,m);

solve(iql+p1,iqr,m+1,r);

}

int main(){

int T; scanf("%d",&T);

while(T--){

scanf("%d%d",&n,&m);

bit.init(n);

int cnt=0,cntq=0;

for(int i=1; i<=n; i++){

scanf("%d",&a[i]);

q[++cnt]=Node{a[i],1,inf,i,1};

}

for(int i=1; i<=m; i++){

int x,y,k; char op[20];

scanf("%s%d%d",op,&x,&y);

if(op[0]=='Q'){

scanf("%d",&k);

q[++cnt]=Node{x,y,k,++cntq,2};

}

else{

q[++cnt]=Node{a[x],-1,inf,x,1};

a[x]=y;

q[++cnt]=Node{a[x],1,inf,x,1};

}

}

solve(1,cnt,-inf,inf);

for(int i=1; i<=cntq; i++)

printf("%d\n",ans[i]);

}

return 0;

}