格式：不要使用换行符，会引入一个空格。注意公式和两边的汉字之间空一格。

内容：挺好的。

线段树基础

简介

考虑这样一个问题：假设我们有一个长度为 n 的整数数列，每次给出 一个区间 $[L,R]$□要求求出第 L 个数到第 R 个数中的最小值□RMQ 问题） 最简单粗暴的方法就是每次将区间 $[L,R]$ 扫一遍，然后求出最小值， 假设有 m 个询问，显然这样的复杂度是 $O(nm)$ 的。 线段树是这样一种数据结构：每一个点代表一个区间，存储的是这个 区间内的信息（对于本题，区间最小值）。这个点的左儿子存储它表示的区 间的左半边，右儿子存储它表示的区间的右半边，以此类推，直至叶子节点（保存单个数的值）。

思想

在查询的时候我只要查询 若干个点的最小值就可以了。注意，我们只需要让查询的点不重不漏包含整 个区间 $[L,R]$ 即可，而并不是要找到该区间包含的所有节点。我们递归定 义查找过程：

1. 若当前节点区间被查询区间所包含，返回这个区间的值

2. 若查询区间与左子树区间有交集，递归查询左子树

3. 若查询区间与右子树区间有交集，递归查询右子树

4. 合并并返回递归查询的值

操作2、3 保证了我们每次查到的节点都和查询区间有交集。由于我们 查询的区间是连续的，所以至多有一个点P□它的左右儿子与查询区间都 有交集且不是完全覆盖。也就是说我们只有至多两个递归是一直递归到叶子 的（在 P 处分叉），所以很简单地证明了查询算法的单次复杂度是 $O(\log n)$

假设 RMQ 问题中，我们每次可以修改一个数，我们只需要更新线 段树上从根到叶子的一条路径上的所有节点的信息即可，复杂度仍旧是 $O(\log n)$ □如果更改一下问题，每次可以修改一个连续的区间，又该怎么操 作以保证复杂度呢？ 我们引入懒标记的概念。懒标记是指在某一个节点上打上标记，表示以 这个节点为根的子树的信息都是一样的，查询至此无需继续递归。如果要继 续对这棵子树的一部分进行其他修改时，我们需要将懒标记下放给它的左 右儿子。所以，进行段修改的算法流程和段查询是完全一致的，只需要在每 个需要修改的节点上打上懒标记即可。 打懒标记有一个原则：在打标记的同时应该将打上标记的那个节点的 信息全部更新完全，而不应该是查询时根据标记现算，那样会大幅增加编 程和思维难度。

注意到如果我们对一个节点的子树上的点进行了修改，那么我们需要 使用它的左右儿子的信息对这个点的信息进行更新，这就需要我们维护的 信息满足可合并性，例如区间的最小值（一个区间的最小值显然是这个区间 前半边的最小值和后半段的最小值中的较小者）。类似地，可合并性被定义 为一个区间的某种信息可以通过它前半段的这种信息与后半段的这种信息 通过简单的计算得到。类似区间众数、区间次大值等就不满足可合并性。

模板

一、有一个 n 个数的序列 $\{a_n\}$ □要求实现三种操作：

1.将 a_x 改为 y；

2.将 a_x 加上 y；

3.查询 $\sum_{i=l}^ra_i$ 的值

$1\le n,q\le 10^5$

单点修改，区间查询

```
#include<bits/stdc++.h>
using namespace std;
const int N=500005;
typedef long long ll;
ll tr[N*4],a[N];
inline int ls(int o){
    return o<<1;
}
inline int rs(int o){
    return o<<1|1;
}
void push_up(int o){
    tr[o]=tr[ls(o)]+tr[rs(o)];
}
void build(int o,int l,int r){
    if(l==r){
        tr[o]=a[l];
        return;
    }
    int mid=l+r>>1;
    build(ls(o),l,mid);
    build(rs(o),mid+1,r);
    push_up(o);
}
void xg(int o,int pos,int l,int r,ll k,int op){
    if(l==r){
        if(op==1) tr[o]+=k;
        else tr[o]=k;
        return;
    }
    int mid=l+r>>1;
    if(pos<=mid) xg(ls(o),pos,l,mid,k,op);
    else xg(rs(o),pos,mid+1,r,k,op);
    push_up(o);
}
ll cx(int o,int nl,int nr,int l,int r){
    if(nl<=l&&r<=nr) return tr[o];
    int mid=l+r>>1;
    ll ret=0;
    if(nl<=mid) ret+=cx(ls(o),nl,nr,l,mid);
```

```
    if(nr>mid) ret+=cx(rs(o),nl,nr,mid+1,r);
    return ret;
}
int main(){
    int n,m;
    scanf("%d %d",&n,&m);
    for(int i=1;i<=n;i++){
        scanf("%d",&a[i]);
    }
    build(1,1,n);
    for(int i=1;i<=m;i++){
        int op;
        scanf("%d",&op);
        if(op==1||op==3){
            int pos;
            ll k;
            scanf("%d %lld",&pos,&k);
            xg(1,pos,1,n,k,op);
        }
        else{
            int l,r;
            scanf("%d %d",&l,&r);
            printf("%lld\n",cx(1,l,r,1,n));
        }
    }
    return 0;
}
```

二、有一个 n 个数的序列 $\{a_n\}$ 要求实现两种操作：

1.将 $a_x(l\le x \le r)$ 加上 y ;

2.查询 $\sum_{i=l}^ra_i$ 的值

$1\le n,q\le 10^5$

区间修改，区间查询

打标记

```
#include<bits/stdc++.h>
using namespace std;
const int N=100005;
typedef long long ll;
ll tr[N*4],tag[N*4],a[N];
inline int ls(int o){
    return o<<1;
}
inline int rs(int o){
    return o<<1|1;
```

```
}
void push_up(int o){
    tr[o]=tr[ls(o)]+tr[rs(o)];
}
void build(int o,int l,int r){
    if(l==r){
        tr[o]=a[l];
        return;
    }
    int mid=l+r>>1;
    build(ls(o),l,mid);
    build(rs(o),mid+1,r);
    push_up(o);
}
void f(int o,int l,int r,ll k){
    tag[o]+=k;
    tr[o]+=(r-l+1)*k;
}
void push_down(int o,int l,int r){
    int mid=l+r>>1;
    f(ls(o),l,mid,tag[o]);
    f(rs(o),mid+1,r,tag[o]);
    tag[o]=0;
}
void xg(int o,int nl,int nr,int l,int r,ll k){
    if(nl<=l&&r<=nr){
        tag[o]+=k;
        tr[o]+=(r-l+1)*k;
        return;
    }
    push_down(o,l,r);
    int mid=l+r>>1;
    if(nl<=mid) xg(ls(o),nl,nr,l,mid,k);
    if(nr>mid) xg(rs(o),nl,nr,mid+1,r,k);
    push_up(o);
}
ll cx(int o,int nl,int nr,int l,int r){
    if(nl<=l&&r<=nr) return tr[o];
    push_down(o,l,r);
    int mid=l+r>>1;
    ll ret=0;
    if(nl<=mid) ret+=cx(ls(o),nl,nr,l,mid);
    if(nr>mid) ret+=cx(rs(o),nl,nr,mid+1,r);
    return ret;
}
int main(){
    int n,m;
    scanf("%d %d",&n,&m);
    for(int i=1;i<=n;i++){
        scanf("%d",&a[i]);
```

```
    }
    build(1,1,n);
    for(int i=1;i<=m;i++){
        int op;
        scanf("%d",&op);
        if(op==1){
            int x,y;ll k;
            scanf("%d %d %lld",&x,&y,&k);
            xg(1,x,y,1,n,k);
        }
        else{
            int x,y;
            scanf("%d %d",&x,&y);
            printf("%lld\n",cx(1,x,y,1,n));
        }
    }
    return 0;
}
```

三、有一个 n 个数的序列 $\{a_n\}$ 要求实现三种操作：

1.将 $a_x(l\le x \le r)$ 改为 y ;

2.将 $a_x(l\le x \le r)$ 加上 y ;

3.将 $a_x(l\le x \le r)$ 乘上 y ;

4.查询 $\sum_{i=l}^ra_i$ 的值

$1\le n,q\le 10^5$

标记下放的优先级：赋值>乘法>加法

```
#include<bits/stdc++.h>
using namespace std;
const int N=100005;
typedef long long ll;
int n,m,p;
ll tr[N*4],mul[N*4],add[N*4],asi[N*4],a[N];
inline int ls(int o){
    return o<<1;
}
inline int rs(int o){
    return o<<1|1;
}
void push_up(int o){
    tr[o]=tr[ls(o)]+tr[rs(o)];
}
void build(int o,int l,int r){
    mul[o]=1;
```

```
    asi[o]=-1;
    if(l==r){
        tr[o]=a[l];
        return;
    }
    int mid=l+r>>1;
    build(ls(o),l,mid);
    build(rs(o),mid+1,r);
    push_up(o);
}
void f(int o,int l,int r,ll k){
    tr[o]=(tr[o]+(r-l+1)*k)%p;
    add[o]=(add[o]+k)%p;
}
void g(int o,int l,int r,ll k){
    tr[o]=tr[o]*k%p;
    mul[o]=mul[o]*k%p;
    add[o]=add[o]*k%p;
}
void h(int o,int l,int r,ll k){
    add[o]=0;
    mul[o]=1;
    asi[o]=k;
    tr[o]=k*(r-l+1)%p;
}
void push_down(int o,int l,int r){
    int mid=l+r>>1;
    if(asi[o]!=-1){
        h(ls(o),l,mid,asi[o]);
        h(rs(o),mid+1,r,asi[o]);
        asi[o]=-1;
    }
    if(mul[o]!=1){
        g(ls(o),l,mid,mul[o]);
        g(rs(o),mid+1,r,mul[o]);
        mul[o]=1;
    }
    if(add[o]){
        f(ls(o),l,mid,add[o]);
        f(rs(o),mid+1,r,add[o]);
        add[o]=0;
    }
}
void xg(int o,int nl,int nr,int l,int r,ll k,int op){
    if(nl<=l&&r<=nr){
        if(op==1){
            mul[o]=mul[o]*k%p;
            add[o]=add[o]*k%p;
            tr[o]=tr[o]*k%p;
        }
```

```
        else if(op==2){
            add[o]=(add[o]+k)%p;
            tr[o]=(tr[o]+(r-l+1)*k)%p;
        }
        else if(op==3){
            add[o]=0;
            mul[o]=1;
            tr[o]=k*(r-l+1)%p;
            asi[o]=k;
        }
        return;
    }
    push_down(o,l,r);
    int mid=l+r>>1;
    if(nl<=mid) xg(ls(o),nl,nr,l,mid,k,op);
    if(nr>mid) xg(rs(o),nl,nr,mid+1,r,k,op);
    push_up(o);
}
ll cx(int o,int nl,int nr,int l,int r){
    if(nl<=l&&r<=nr) return tr[o];
    push_down(o,l,r);
    int mid=l+r>>1;
    ll ret=0;
    if(nl<=mid) ret=(ret+cx(ls(o),nl,nr,l,mid))%p;
    if(nr>mid) ret=(ret+cx(rs(o),nl,nr,mid+1,r))%p;
    return ret;
}
int main(){
    scanf("%d %d %d",&n,&m,&p);
    for(int i=1;i<=n;i++) scanf("%lld",&a[i]);
    build(1,1,n);
    for(int i=1;i<=m;i++){
        int op,x,y;
        scanf("%d %d %d",&op,&x,&y);
        if(op==1||op==2||op==3){
            ll k;
            scanf("%lld",&k);
            xg(1,x,y,1,n,k,op);
        }
        else printf("%lld\n",cx(1,x,y,1,n));
    }
    return 0;
}
```

模板题

P3372 【模板】线段树 1

P3373 【模板】线段树 2

P3374 【模板】树状数组 1

P3368 【模板】树状数组 2

P5057 [CQOI2006]简单题

P1531 I Hate It

P2023 [AHOI2009] 维护序列

比模板题难一点的题

P1908 逆序对

法一：借助归并排序过程

法二：线段树（树状数组）

本题我们不关系数字具体多大，只关心各数之间的相对大小，因此可以离散化处理降低空间复杂度（必须离散化，否则空间不够）。然后遍历这些数，遍历过程中，依次插入线段树（单点修改），顺便同时求该数逆序对，即求比它大的数的个数（区间查询）。

P1637 三元上升子序列

本题跟上题类似

更难的题

~~注意本专题是线段树基础~~