**此章收录算法模板**

**感谢acwing，感谢y总**

**高精度加法**

vector<int> add(vector<int> &A, vector<int> &B) *// C = A + B, A >= 0, B >= 0*

{

if (A.size() < B.size()) return add(B, A);

vector<int> C;

int t = 0;

for (int i = 0; i < A.size(); i ++ )

{

t += A[i];

if (i < B.size()) t += B[i];

C.push\_back(t % 10);

t /= 10;

}

if (t) C.push\_back(t);

return C;

}

**高精度减法**

vector<int> sub(vector<int> &A, vector<int> &B) *// C = A - B, 满足A >= B, A >= 0, B >= 0*

{

vector<int> C;

for (int i = 0, t = 0; i < A.size(); i ++ )

{

t = A[i] - t;

if (i < B.size()) t -= B[i];

C.push\_back((t + 10) % 10);

if (t < 0) t = 1;

else t = 0;

}

while (C.size() > 1 && C.back() == 0) C.pop\_back();

return C;

}

**高精度乘低精度**

vector<int> mul(vector<int> &A, int b) // C = A \* b, A >= 0, b >= 0

{

vector<int> C;

int t = 0;

for (int i = 0; i < A.size() || t; i ++ )

{

if (i < A.size()) t += A[i] \* b;

C.push\_back(t % 10);

t /= 10;

}

while (C.size() > 1 && C.back() == 0) C.pop\_back();

return C;

}

**高精度除以低精度**

vector<int> div(vector<int> &A, int b, int &r) // A / b = C ... r, A >= 0, b > 0

{

vector<int> C;

r = 0;

for (int i = A.size() - 1; i >= 0; i -- )

{

r = r \* 10 + A[i];

C.push\_back(r / b);

r %= b;

}

reverse(C.begin(), C.end());

while (C.size() > 1 && C.back() == 0) C.pop\_back();

return C;

}

**lowbit运算**

int lowbit(int x) *// 返回末尾的1*

{

return x & -x;

}

**马拉车算法**

void init() *// a[]为原串，b[]为插入'#'后的新串*

{

int k = 0;

b[k ++ ] = '$', b[k ++ ] = '#';

for (int i = 0; i < n; i ++ ) b[k ++ ] = a[i], b[k ++ ] = '#';

b[k ++ ] = '^';

n = k;

}

void manacher() *// 马拉车算法，b[]为插入'#'后的新串*

{

int mr = 0, mid;

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

{

if (i < mr) p[i] = min(p[mid \* 2 - i], mr - i);

else p[i] = 1;

while (b[i - p[i]] == b[i + p[i]]) p[i] ++ ;

if (i + p[i] > mr)

{

mr = i + p[i];

mid = i;

}

}

}

**归并排序**

void merge\_sort(int q[], int l, int r) *// 归并排序*

{

if (l >= r) return;

int mid = l + r >> 1;

merge\_sort(q, l, mid);

merge\_sort(q, mid + 1, r);

int k = 0, i = l, j = mid + 1;

while (i <= mid && j <= r)

if (q[i] <= q[j]) tmp[k ++ ] = q[i ++ ];

else tmp[k ++ ] = q[j ++ ];

while (i <= mid) tmp[k ++ ] = q[i ++ ];

while (j <= r) tmp[k ++ ] = q[j ++ ];

for (i = l, j = 0; i <= r; i ++, j ++ ) q[i] = tmp[j];

}

**DLX重复覆盖**

int l[N], r[N], u[N], d[N], col[N], row[N], s[N], idx;

int ans[N], top; *// 记录选择了哪些行*

bool st[M]; *// N为节点数，M为列数*

void init() *// 初始化十字链表*

{

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

{

l[i] = i - 1, r[i] = i + 1;

u[i] = d[i] = i;

s[i] = 0, col[i] = i;

}

l[0] = m, r[m] = 0;

idx = m + 1;

}

void add(int& hh, int& tt, int x, int y) *// 在十字链表中插入节点*

{

row[idx] = x, col[idx] = y, s[y] ++ ;

u[idx] = y, d[idx] = d[y], u[d[y]] = idx, d[y] = idx;

r[hh] = l[tt] = idx, r[idx] = tt, l[idx] = hh;

tt = idx ++ ;

}

int h() *// IDA\*的启发函数*

{

int res = 0;

memset(st, 0, sizeof st);

for (int i = r[0]; i; i = r[i])

{

if (st[col[i]]) continue;

res ++ ;

st[col[i]] = true;

for (int j = d[i]; j != i; j = d[j])

for (int k = r[j]; k != j; k = r[k])

st[col[k]] = true;

}

return res;

}

void remove(int p)

{

for (int i = d[p]; i != p; i = d[i])

{

r[l[i]] = r[i];

l[r[i]] = l[i];

}

}

void resume(int p)

{

for (int i = u[p]; i != p; i = u[i])

{

r[l[i]] = i;

l[r[i]] = i;

}

}

bool dfs(int k)

{

if (k + h() > top) return false;

if (!r[0])

{

top = k;

return true;

}

int p = r[0];

for (int i = r[0]; i; i = r[i])

if (s[i] < s[p])

p = i;

for (int i = d[p]; i != p; i = d[i])

{

ans[k] = row[i];

remove(i);

for (int j = r[i]; j != i; j = r[j]) remove(j);

if (dfs(k + 1)) return true;

for (int j = l[i]; j != i; j = l[j]) resume(j);

resume(i);

}

return false;

}

**DLX精确覆盖**

int l[N], r[N], u[N], d[N], col[N], row[N], s[N], idx;

int ans[N], top; *// 记录选择了哪些行*

void init() *// 初始化十字链表*

{

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

{

l[i] = i - 1, r[i] = i + 1;

u[i] = d[i] = i;

}

l[0] = m, r[m] = 0;

idx = m + 1;

}

void add(int& hh, int& tt, int x, int y) *// 在十字链表中添加节点*

{

row[idx] = x, col[idx] = y, s[y] ++ ;

u[idx] = y, d[idx] = d[y], u[d[y]] = idx, d[y] = idx;

r[hh] = l[tt] = idx, r[idx] = tt, l[idx] = hh;

tt = idx ++ ;

}

void remove(int p)

{

r[l[p]] = r[p], l[r[p]] = l[p];

for (int i = d[p]; i != p; i = d[i])

for (int j = r[i]; j != i; j = r[j])

{

s[col[j]] -- ;

d[u[j]] = d[j], u[d[j]] = u[j];

}

}

void resume(int p)

{

for (int i = d[p]; i != p; i = d[i])

for (int j = r[i]; j != i; j = r[j])

{

s[col[j]] ++ ;

d[u[j]] = j, u[d[j]] = j;

}

r[l[p]] = p, l[r[p]] = p;

}

bool dfs()

{

if (!r[0]) return true;

int p = r[0];

for (int i = r[0]; i; i = r[i])

if (s[i] < s[p])

p = i;

if (!s[p]) return false;

remove(p);

for (int i = d[p]; i != p; i = d[i])

{

ans[ ++ top] = row[i];

for (int j = r[i]; j != i; j = r[j]) remove(col[j]);

if (dfs()) return true;

for (int j = r[i]; j != i; j = r[j]) resume(col[j]);

top -- ;

}

resume(p);

return false;

}

**并查集 + 路径压缩**

int find(int x) *// 并查集*

{

if (p[x] != x) p[x] = find(p[x]);

return p[x];

}

**字符串哈希**

ULL get(int l, int r) *// 计算子串 str[l ~ r] 的哈希值*

{

return h[r] - h[l - 1] \* p[r - l + 1];

}

**Trie插入**

int son[N][26], cnt[N], idx;

void insert(char \*str) *// 插入字符串*

{

int p = 0;

for (int i = 0; str[i]; i ++ )

{

int u = str[i] - 'a';

if (!son[p][u]) son[p][u] = ++ idx;

p = son[p][u];

}

cnt[p] ++ ;

}

int query(char \*str) *// 查询字符串出现次数*

{

int p = 0;

for (int i = 0; str[i]; i ++ )

{

int u = str[i] - 'a';

if (!son[p][u]) return 0;

p = son[p][u];

}

return cnt[p];

}

**邻接链表（无权）**

void add(int a,int b)

{

e[idx] = b, next[idx] = h[a], h[a] = idx++;

}

**邻接链表（带权）**

void add(int a,int b,int c)

{

e[idx] = b, next[idx] = h[a], w[idx] = c, h[a] = idx++;

}

**dijkstra算法**

int dijkstra() *// 求1号点到n号点的最短路距离，如果从1号点无法走到n号点则返回-1*

{

memset(dist, 0x3f, sizeof dist);

dist[1] = 0;

priority\_queue<PII, vector<PII>, greater<PII>> heap;

heap.push({0, 1});

while (heap.size())

{

auto t = heap.top();

heap.pop();

int ver = t.second, distance = t.first;

if (st[ver]) continue;

st[ver] = true;

for (int i = h[ver]; i != -1; i = ne[i])

{

int j = e[i];

if (dist[j] > dist[ver] + w[i])

{

dist[j] = dist[ver] + w[i];

heap.push({dist[j], j});

}

}

}

if (dist[n] == 0x3f3f3f3f) return -1;

return dist[n];

}

void dijkstra() *// 求1号点到n号点的最短路距离*

{

memset(dist, 0x3f, sizeof dist);

dist[1] = 0;

priority\_queue<PII, vector<PII>, greater<PII>> heap;

heap.push({0, 1});

while (heap.size())

{

auto t = heap.top();

heap.pop();

int ver = t.second, distance = t.first;

if (st[ver]) continue;

st[ver] = true;

for (int i = h[ver]; i != -1; i = ne[i])

{

int j = e[i];

if (dist[j] > dist[ver] + w[i])

{

dist[j] = dist[ver] + w[i];

heap.push({dist[j], j});

}

}

}

}

**匈牙利算法（NTR算法）**

bool find(int x)

{

for (int i = h[x]; i != -1; i = ne[i])

{

int j = e[i];

if (!st[j])

{

st[j] = true;

if (match[j] == 0 || find(match[j]))

{

match[j] = x;

return true;

}

}

}

return false;

}

**spfa算法（最短路）**

int spfa() *// 求1号点到n号点的最短路距离，如果从1号点无法走到n号点则返回-1*

{

int hh = 0, tt = 0;

memset(dist, 0x3f, sizeof dist);

dist[1] = 0;

q[tt ++ ] = 1;

st[1] = true;

while (hh != tt)

{

int t = q[hh ++ ];

if (hh == N) hh = 0;

st[t] = false;

for (int i = h[t]; i != -1; i = ne[i])

{

int j = e[i];

if (dist[j] > dist[t] + w[i])

{

dist[j] = dist[t] + w[i];

if (!st[j]) *// 如果队列中已存在j，则不需要将j重复插入*

{

q[tt ++ ] = j;

if (tt == N) tt = 0;

st[j] = true;

}

}

}

}

if (dist[n] == 0x3f3f3f3f) return -1;

return dist[n];

}

**spfa算法（判断负环）**

bool spfa() *// 如果存在负环，则返回true，否则返回false。*

{

*// 不需要初始化dist数组*

*// 原理：如果某条最短路径上有n个点（除了自己），那么加上自己之后一共有n+1个点，*

*// 由抽屉原理一定有两个点相同，所以存在环。*

int hh = 0, tt = 0;

for (int i = 1; i <= n; i ++ ) q[tt ++ ] = i, st[i] = true;

while (hh != tt)

{

int t = q[hh ++ ];

if (hh == N) hh = 0;

st[t] = false;

for (int i = h[t]; ~i; i = ne[i])

{

int j = e[i];

if (dist[j] > dist[t] + w[i])

{

dist[j] = dist[t] + w[i];

cnt[j] = cnt[t] + 1;

if (cnt[j] >= n) return true;

if (!st[j])

{

st[j] = true;

q[tt ++ ] = j;

if (tt == N) tt = 0;

}

}

}

}

return false;

}

**括扑排序**

void topsort()

{

int hh = 0, tt = -1;

*// d[i] 存储点i的入度*

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

if (!d[i])

q[ ++ tt] = i;

while (hh <= tt)

{

int t = q[hh ++ ];

for (int i = h[t]; i != -1; i = ne[i])

{

int j = e[i];

if (-- d[j] == 0)

q[ ++ tt] = j;

}

}

}

**欧拉函数**

int phi(int x) *// 欧拉函数*

{

int res = x;

for (int i = 2; i <= x / i; i ++ )

if (x % i == 0)

{

res = res / i \* (i - 1);

while (x % i == 0) x /= i;

}

if (x > 1) res = res / x \* (x - 1);

return res;

}

**线性筛 + 欧拉函数**

void get\_eulers(int n) *// 线性筛法求1~n的欧拉函数*

{

euler[1] = 1;

for (int i = 2; i <= n; i ++ )

{

if (!st[i])

{

primes[cnt ++ ] = i;

euler[i] = i - 1;

}

for (int j = 0; primes[j] <= n / i; j ++ )

{

int t = primes[j] \* i;

st[t] = true;

if (i % primes[j] == 0)

{

euler[t] = euler[i] \* primes[j];

break;

}

euler[t] = euler[i] \* (primes[j] - 1);

}

}

}

**欧几里得算法**

int gcd(int a, int b) *// 欧几里得算法*

{

return b ? gcd(b, a % b) : a;

}

**扩展欧几里得算法**

int exgcd(int a, int b, int &x, int &y) *// 扩展欧几里得算法, 求x, y，使得ax + by = gcd(a, b)*

{

if (!b)

{

x = 1; y = 0;

return a;

}

int d = exgcd(b, a % b, y, x);

y -= (a / b) \* x;

return d;

}

**判定质数**

bool is\_prime(int x) *// 判定质数*

{

if (x < 2) return false;

for (int i = 2; i <= x / i; i ++ )

if (x % i == 0)

return false;

return true;

}

**线性质数筛**

void get\_primes(int n) *// 线性筛质数*

{

for (int i = 2; i <= n; i ++ )

{

if (!st[i]) primes[cnt ++ ] = i;

for (int j = 0; primes[j] <= n / i; j ++ )

{

st[primes[j] \* i] = true;

if (i % primes[j] == 0) break;

}

}

}

**快速幂算法**

int quick\_power(int a, int k, int p) *// 求a^k mod p*

{

int res = 1 % p;

while (k)

{

if (k & 1) res = (LL)res \* a % p;

a = (LL)a \* a % p;

k >>= 1;

}

return res;

}

**线段树基本操作**

```cpp

#pragma GCC optimize(2)

#include <iostream>

#include <algorithm>

const int N = 200010;

int m, p;

struct Node

{

int l, r;

int v; //区间[l, r]中的最大值

}tr[N \* 4];

//计算单个节点

void push\_up(int u)

{

tr[u].v = std::max(tr[u << 1].v, tr[u << 1 | 1].v);

}

//对线段树进行搭建

void build(int u, int L, int R)

{

//将左右端点存入

tr[u] = { L,R };

//如果当前节点为叶子节点，退出

if (L == R) return;

//计算当前区间中点

int mid = L + R >> 1;

//u << 1 等价于 u \* 2，u << 1|1 等价于 u \* 2 + 1

//向左递归建树，向右递归建树

//相当于将区间不断二分直至区间变点

//如此就能从递归终点向上处理区间属性，从而解决区间问题

build(u << 1, L, mid);

build(u << 1 | 1, mid + 1, R);

/\*

一般会在此时进行push\_up操作

push\_up(u);

\*/

}

//询问线段[L，R] 之间的属性，例如区间最值、区间和等等，一般从头节点开始

int query(int u, int l, int r)

{

if (tr[u].l >= l && tr[u].r <= r) return tr[u].v;

int mid = tr[u].l + tr[u].r >> 1;

int v = 0;

if (l <= mid) v = query(u << 1, l, r);

if (r > mid) v = std::max(query(u << 1 | 1, l, r), v);

return v;

}

void modify(int u, int x, int v)

{

if (tr[u].l == x && tr[u].r == x) tr[u].v = v;

else

{

int mid = tr[u].l + tr[u].r >> 1;

if (x <= mid) modify(u << 1, x, v);

else modify(u << 1 | 1, x, v);

push\_up(u);

}

}

int main()

{

std::ios::sync\_with\_stdio(false);

std::cin.tie(0);

std::cout.tie(0);

int n = 0, lasta = 0;

std::cin >> m >> p;

build(1, 1, m);

int x;

char op;

while (m--)

{

std::cin >> op >> x;

if (op == 'Q')

{

lasta = query(1, n - x + 1, n);

std::cout << lasta << std::endl;

}

else

{

modify(1, n + 1, ((long long)lasta + x) % p);

n++;

}

}

}

```

```cpp

#pragma GCC optimize(2)

#include <iostream>

#include <cstring>

#include <algorithm>

const int N = 500010;

int q[N];

struct Tree

{

int l,r;

int sum,lmax,rmax,tmax;

}tr[4 \* N];

//用于方便查询操作需要的重构函数

void pushup(Tree& u,Tree& l,Tree& r)

{

u.sum = l.sum + r.sum;

u.lmax = std::max(l.lmax, l.sum + r.lmax);

u.rmax = std::max(r.rmax, r.sum + l.rmax);

u.tmax = std::max(std::max(l.tmax,r.tmax),l.rmax + r.lmax);

}

void pushup(int u)

{

pushup(tr[u],tr[u<<1],tr[u<<1|1]);

}

void build(int u,int l,int r)

{

tr[u] = {l,r,q[l],q[l],q[l],q[l]};

if(l>=r) return;

int center = l + r >> 1;

build(u<<1,l,center),build(u<<1|1,center + 1,r);

pushup(u);

}

Tree query(int u,int l,int r)

{

if(tr[u].l>=l&&tr[u].r<=r) return tr[u];

int center = tr[u].l + tr[u].r >> 1;

if(r<=center) return query(u<<1,l,r);

else if(l>center) return query(u<<1|1,l,r);

//当发生第三种情况时我们需要重新处理[l,r] 区间的值，毕竟我们的线段树不存在跨越两区间之间的节点

else

{

auto left = query(u<<1,l,r);

auto right = query(u<<1|1,l,r);

Tree res;

pushup(res,left,right);

return res;

}

}

void modify(int u,int x,int v)

{

if(tr[u].l==x&&tr[u].r==x) tr[u] = {x,x,v,v,v,v};

else

{

int center = tr[u].l + tr[u].r >> 1;

if(x<=center) modify(u<<1,x,v);

else modify(u<<1|1,x,v);

pushup(u);

}

}

int n,m;

int main()

{

std::ios::sync\_with\_stdio(false);

std::cin.tie(0);

std::cout.tie(0);

std::cin>>n>>m;

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

{

std::cin>>q[i];

}

build(1,1,n);

while (m -- ){

int op;

int l,r;

std::cin>>op>>l>>r;

if(op==1)

{

if(l>r) std::swap(l,r);

auto v = query(1,l,r);

std::cout<<v.tmax<<std::endl;

}

else

{

modify(1,l,r);

}

}

}

```

```cpp

#pragma GCC optimize(2)

#include <iostream>

#include <cstring>

#include <algorithm>

#include <cmath>

using ll = long long;

const int N = 500010;

ll q[N];

struct Tree

{

int l,r;

//sum 是差分数组的和，假设tr[1,l] 那么tr[1,l].sum = q[l]

//d 存储区间[l,r]最大公约数

ll sum,d;

}tr[4\*N];

ll gcd(ll a,ll b)

{

return b?gcd(b,a%b):a;

}

void pushup(Tree& u,Tree& l,Tree& r)

{

u.sum = l.sum + r.sum;

u.d = gcd(l.d,r.d);

}

void pushup(int u)

{

pushup(tr[u],tr[u<<1],tr[u<<1|1]);

}

void build(int u,int l,int r)

{

if(l==r) tr[u] = {l,r,q[r]-q[r-1],q[r]-q[r-1]};

else

{

tr[u].l = l,tr[u].r = r;

int center = l + r >> 1;

build(u<<1,l,center),build(u<<1|1,center + 1,r);

pushup(u);

}

}

void modify(int u,int x,ll v)

{

if(tr[u].l==x&&tr[u].r==x) tr[u] = {x,x,tr[u].sum+v,tr[u].sum+v};

else

{

int center = tr[u].l + tr[u].r >> 1;

if(x<=center) modify(u<<1,x,v);

else modify(u<<1|1,x,v);

pushup(u);

}

}

Tree query(int u,int l,int r)

{

if(tr[u].l>=l&&tr[u].r<=r) return tr[u];

int center = tr[u].l + tr[u].r >> 1;

if(r<=center) return query(u<<1,l,r);

else if(l>center) return query(u<<1|1,l,r);

else

{

auto left = query(u<<1,l,r);

auto right = query(u<<1|1,l,r);

Tree res;

pushup(res,left,right);

return res;

}

}

int n,m;

int main()

{

std::ios::sync\_with\_stdio(false);

std::cin.tie(0);

std::cout.tie(0);

std::cin>>n>>m;

for(int i = 1;i<=n;i++) std::cin>>q[i];

build(1,1,n);

int l,r;

char op;

ll x;

while (m -- )

{

std::cin>>op>>l>>r;

if(op=='Q')

{

auto left = query(1,1,l);

Tree right {0,0,0,0};

//注意越界

if(l+1<=r) right = query(1,l+1,r);

std::cout<<std::abs(gcd(left.sum,right.d))<<std::endl;

}

else

{

std::cin>>x;

modify(1,l,x);

//注意越界

if(r+1<=n) modify(1,r+1,-x);

}

}

}

```

```cpp

#pragma GCC optimize(2)

#include <iostream>

#include <cstring>

#include <algorithm>

using ll = long long;

const int N = 100010;

struct Tree

{

int l,r;

ll sum;

ll add;

}tr[4 \* N];

ll q[N];

void pushup(int u)

{

tr[u].sum = tr[u<<1].sum + tr[u<<1|1].sum;

}

void pushdown(int u)

{

auto& root = tr[u];auto& left = tr[u<<1];auto& right = tr[u<<1|1];

if(root.add)

{

left.add += root.add, left.sum += (ll)(left.r - left.l + 1) \* root.add;

right.add += root.add, right.sum += (ll)(right.r - right.l + 1) \* root.add;

root.add = 0;

}

}

void build(int u,int l,int r)

{

if(l>=r) tr[u] = {l,r,q[r],0};

else

{

tr[u] = {l,r};

int center = l + r >> 1;

build(u<<1,l,center),build(u<<1|1,center + 1, r);

pushup(u);

}

}

ll query(int u,int l,int r)

{

if(tr[u].l>=l&&tr[u].r<=r) return tr[u].sum;

int center = tr[u].l + tr[u].r >> 1;

ll v = 0;

//记得下沉标记

pushdown(u);

if(l<=center) v = query(u<<1,l,r);

if(r>center) v += query(u<<1|1,l,r);

return v;

}

void modify(int u,int l,int r,int v)

{

//修改时我们先把被标记区间的sum值更改成正确的值

if(tr[u].l>=l&&tr[u].r<=r) {

tr[u].sum += (ll)(tr[u].r - tr[u].l + 1) \* v;

tr[u].add += v;

}

else

{

//记得下沉标记

pushdown(u);

int center = tr[u].l + tr[u].r >> 1;

if(l<=center) modify(u<<1,l,r,v);

if(r>center) modify(u<<1|1,l,r,v);

pushup(u);

}

}

int n,m;

int main()

{

std::ios::sync\_with\_stdio(false);

std::cin.tie(0);

std::cout.tie(0);

std::cin>>n>>m;

for(int i = 1;i<=n;i++) std::cin>>q[i];

build(1,1,n);

while (m -- )

{

char op;

int l,r;

ll v;

std::cin>>op>>l>>r;

if(op=='Q')

{

std::cout<<query(1,l,r)<<std::endl;

}

else

{

std::cin>>v;

modify(1,l,r,v);

}

}

}

```

```cpp

#pragma GCC optimize(2)

#include <iostream>

#include <cstring>

#include <algorithm>

#include <vector>

#include <iomanip>

using ll = long long;

const int N = 10010;

std::vector<double> ys;

struct Tree

{

int l,r;

int cnt;

double len;

}tr[8\*N];

struct Segment

{

double x;

double y1,y2;

int k;

bool operator<(const Segment& t) const

{

return x<t.x;

}

}seg[2\*N];

int find(double y)

{

return std::lower\_bound(ys.begin(),ys.end(),y) - ys.begin();

}

void pushup(int u)

{

if(tr[u].cnt)

{

tr[u].len = ys[tr[u].r+1] - ys[tr[u].l];

}

else if(tr[u].l!=tr[u].r)

{

tr[u].len = tr[u<<1].len + tr[u<<1|1].len;

}else

{

tr[u].len = 0;

}

}

void build(int u,int l,int r)

{

tr[u] = {l,r,0,0};

if(l!=r)

{

int center = l + r >> 1;

build(u<<1,l,center), build(u<<1|1,center + 1,r);

}

}

void modify(int u,int l,int r,int k)

{

if(tr[u].l>=l&&tr[u].r<=r)

{

tr[u].cnt += k;

}

else

{

int center = tr[u].l + tr[u].r >> 1;

if(l<=center) modify(u<<1,l,r,k);

if(r>center) modify(u<<1|1,l,r,k);

}

pushup(u);

}

int n;

int main()

{

std::ios::sync\_with\_stdio(false);

std::cin.tie(0);

std::cout.tie(0);

int T = 1;

while(std::cin>>n,n)

{

ys.clear();

for(int i = 0,j=0;i<n;i++)

{

double x1,y1,x2,y2;

std::cin>>x1>>y1>>x2>>y2;

seg[j++] = {x1,y1,y2,1};

seg[j++] = {x2,y1,y2,-1};

ys.push\_back(y1),ys.push\_back(y2);

}

std::sort(ys.begin(),ys.end());

ys.erase(std::unique(ys.begin(),ys.end()),ys.end());

build(1,0,ys.size()-2);

std::sort(seg,seg + 2\*n);

double res = 0;

for(int i = 0;i<2\*n;i++)

{

if(i) res += tr[1].len \* (seg[i].x - seg[i-1].x);

modify(1,find(seg[i].y1),find(seg[i].y2)-1,seg[i].k);

}

std::cout<<"Test case #"<<T++<<std::endl;

std::cout<<"Total explored area: "<<std::fixed<<std::setprecision(2)<<res<<std::endl<<std::endl;

}

}

```

```cpp

#pragma GCC optimize(2)

#include <iostream>

#include <cstring>

#include <algorithm>

using ll = long long;

const int N = 100010;

ll q[N];

int n,m;

ll p;

struct Tree

{

int l,r;

ll sum,add,mul;

}tr[4\*N];

void eval(Tree& t,ll add,ll mul)

{

t.sum = (t.sum \* mul + (t.r-t.l + 1) \* add) % p;

t.mul = t.mul \* mul % p;

t.add = (t.add \* mul + add) % p;

}

void pushup(int u)

{

tr[u].sum = (tr[u<<1].sum + tr[u<<1|1].sum) % p;

}

void build(int u,int l,int r)

{

if(l>=r) tr[u] = {l,r,q[l],0,1};

else

{

tr[u] = {l,r,0,0,1};

int center = l + r >> 1;

build(u<<1,l,center), build(u<<1|1,center + 1,r);

pushup(u);

}

}

void pushdown(int u)

{

eval(tr[u<<1],tr[u].add,tr[u].mul);

eval(tr[u<<1|1],tr[u].add,tr[u].mul);

tr[u].add = 0,tr[u].mul = 1;

}

ll query(int u,int l,int r)

{

if(tr[u].l>=l&&tr[u].r<=r) return tr[u].sum;

pushdown(u);

int center = tr[u].l + tr[u].r >> 1;

ll res = 0;

if(l<=center) res = query(u<<1,l,r);

if(r>center) res =(res + query(u<<1|1,l,r)) % p;

return res;

}

void modify(int u,int l,int r,ll add,ll mul)

{

if(tr[u].l>=l&&tr[u].r<=r)

{

eval(tr[u],add,mul);

}

else

{

pushdown(u);

int center = tr[u].l + tr[u].r >> 1;

if(l<=center) modify(u<<1,l,r,add,mul);

if(r>center) modify(u<<1|1,l,r,add,mul);

pushup(u);

}

}

int main()

{

std::ios::sync\_with\_stdio(false);

std::cin.tie(0);

std::cout.tie(0);

std::cin>>n>>p;

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

{

std::cin>>q[i];

}

build(1,1,n);

std::cin>>m;

while (m -- )

{

int op;

int l,r;

ll add,mul;

std::cin>>op>>l>>r;

if(op==1)

{

std::cin>>mul;

modify(1,l,r,0,mul);

}

else if(op == 2)

{

std::cin>>add;

modify(1,l,r,add,1);

}else

{

std::cout<<query(1,l,r)<<std::endl;

}

}

}

```