#include <cstdio>

#include <cstring>

typedef long long LL;

const LL MOD = 1e9 + 7;

//matrix quick power

struct Matrix

{

LL a[4][4];

LL r, c;

Matrix(int r, int c):r(r), c(c){memset(a, 0, sizeof(a));}

/\*Matrix operator\* (const Matrix& rhs) const{

Matrix res = Matrix(r, rhs.c);

for(LL i = 1;i <= r;i++)

for(LL j = 1;j <= res.c;j++)

for(LL k = 1;k <= c;k++)

res.a[i][j] += a[i][k] \* rhs.a[k][j];

return res;

}\*/

};

Matrix matrix\_multiply(Matrix& m1, Matrix& m2)

{

Matrix ans = Matrix(m1.r, m2.c);

ans.r = m1.r;

ans.c = m2.c;

for(LL i = 1;i <= m1.r;i++)

for(LL j = 1;j <= m2.c;j++)

for(LL k = 1;k <= m1.c;k++)

ans.a[i][j] = (ans.a[i][j] + m1.a[i][k] \* m2.a[k][j]) % MOD;

return ans;

}

LL matrix\_qpow(LL expo)

{

if(expo < 1) return 1;

Matrix ans = Matrix(3, 3), base = Matrix(3, 3);

ans.a[1][1] = ans.a[2][2] = ans.a[3][3] = 1;

base.a[1][1] = base.a[3][1] = base.a[1][2] = base.a[2][3] = 1;

while(expo)

{

if(expo & 1)

ans = matrix\_multiply(ans, base);

expo >>= 1;

base = matrix\_multiply(base, base);

}

Matrix first = Matrix(1, 3);

first.a[1][1] = first.a[1][2] = first.a[1][3] = 1;

ans = matrix\_multiply(first, ans);

return ans.a[1][1] % MOD;

}

//union check set

int pre[maxn];

int find(int u)

{

int v = u;

while(v != pre[v])

v = pre[v];

return pre[u] = v;

}

void join(int u, int v)

{

pre[find(u)] = find(v);

}

//big number

struct BigNum{

char num[100];

int len;

};

void swap(char \*a,char \*b){char t=\*a;\*a=\*b;\*b=t;}

void BigNumInitate(struct BigNum \*a,int b){

a->len=0;

while(b){

a->num[a->len]=b%10;

b/=10;

a->len++;

}

}

void BigNumPrint(struct BigNum a){

for(int i=a.len-1;i>=0;i--) printf("%c",a.num[i]+'0');

}

void BigNumAdd(struct BigNum \*a,struct BigNum \*b,struct BigNum \*c){

int i;

c->len=a->len>b->len?a->len:b->len;

for(i=a->len;i<c->len;i++) a->num[i]=0;

for(i=b->len;i<c->len;i++) b->num[i]=0;

a->num[c->len]=b->num[c->len]='\0';

int carry=0;

for(i=0;i<c->len;i++){

c->num[i]=a->num[i]+b->num[i];

if(carry){c->num[i]++;carry=0;}

if(c->num[i]>9){c->num[i]-=10;carry=1;}

}

if(carry){c->num[c->len++]=1;c->num[c->len]='\0';}

}

void BigNumSubtract(struct BigNum \*a,struct BigNum \*b,struct BigNum \*c){

int i;

c->len=a->len>b->len?a->len:b->len;

for(i=a->len;i<c->len;i++) a->num[i]=0;

for(i=b->len;i<c->len;i++) b->num[i]=0;

a->num[c->len]=b->num[c->len]='\0';

int carry=0;

for(i=0;i<c->len;i++){

c->num[i]=a->num[i]+b->num[i];

if(carry){c->num[i]++;carry=0;}

if(c->num[i]>9){c->num[i]-=10;carry=1;}

}

if(carry){c->num[c->len++]=1;c->num[c->len]='\0';}

}

void BigNumMultiply(struct BigNum \*a,struct BigNum \*b,struct BigNum \*c){

int i;

memset(c->num,0,sizeof(c->num));c->len=1;

for(i=0;i<b->len;i++){

for(int j=0;j<a->len;j++){

c->num[i+j]+=(b->num[i]\*a->num[j]);

c->len=i+j+1;

if(c->num[i+j]>=10){

c->num[i+j+1]+=(c->num[i+j]/10);

c->num[i+j]%=10;

c->len++;

}

}

}

}

void BigNumDivide(struct BigNum a,int b,struct BigNum \*c){

for(int i=0;i<a.len/2;i++)

swap(a.num+i,a.num+a.len-1-i);

int remainder=0;

c->len=0;

for(int i=0;i<a.len;i++)

{

remainder=remainder\*10+a.num[i];

if(c->len||remainder>=b)

{

c->num[c->len]=remainder/b;

remainder=remainder%b;

c->len++;

}

}

for(int i=0;i<c->len/2;i++)

swap(c->num+i,c->num+c->len-1-i);

}

void BigNumDivide2(struct BigNum a,struct BigNum b,struct BigNum \*c,struct BigNum \*d){

int i;

struct BigNum CONST\_10,a\_now;

BigNumInitate(d,0);

BigNumInitate(&CONST\_10,10);

for(i=0;i<a.len/2;i++)

swap(a.num+i,a.num+a.len-1-i);

memset(c->num,0,sizeof(c->num));

c->len=0;

for(i=0;i<a.len;i++){

BigNumInitate(&a\_now,a.num[i]);

BigNumMultiply(\*d,CONST\_10,d);

BigNumAdd(\*d,a\_now,d);

if(c->len||BigNumCmp(\*d,b)>=0){

while(BigNumCmp(\*d,b)>=0){

BigNumSubtract(\*d,b,d);

c->num[c->len]++;

}

c->len++;

}

}

if(!c->len) c->len=1;

for(i=0;i<c->len/2;i++)

swap(c->num+i,c->num+c->len-1-i);

}

//extend gcd

LL exGcd(LL a, LL b, LL &x, LL &y)

{

if(b == 0)

{

x = 1, y = 0;

return a;

}

LL r = exGcd(b, a%b, x, y);

LL temp = y;

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

x = temp;

return r;

}

//inverse

LL inv[maxn];

void inverse(int n, int p)

{

inv[1] = 1;

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

inv[i] = (p - p/i) \* inv[p % i] % p;

}

//segment tree

#define ROOT 1

const LL maxn = 1e5 + 7;

struct Node{LL l, r, w, f;}tree[4\*maxn];

LL cmd, x, y, d, ans, cnt = 0, a[maxn];

void down(LL k)

{

LL f = tree[k].f;

tree[2\*k].f += f;

tree[2\*k+1].f += f;

tree[2\*k].w += f\*(tree[2\*k].r - tree[2\*k].l + 1);

tree[2\*k+1].w += f\*(tree[2\*k+1].r - tree[2\*k+1].l + 1);

tree[k].f = 0;

}

void operate(LL k)

{

if(x <= tree[k].l && tree[k].r <= y)

{

if(cmd == 2) ans += tree[k].w;//query

if(cmd == 1) tree[k].w += d\*(tree[k].r - tree[k].l + 1);

if(cmd == 1) tree[k].f += d;//add

return;

}

if(tree[k].f) down(k);

LL m = (tree[k].l+tree[k].r)/2;

if(x <= m) operate(2\*k);

if(m < y) operate(2\*k+1);

if(cmd == 1) tree[k].w = tree[2\*k].w + tree[2\*k+1].w;

}

void build(LL l, LL r, LL k)

{

tree[k].l = l;

tree[k].r = r;

tree[k].f = 0;

if(l == r)

{

scanf("%lld", &a[++cnt]);

tree[k].w = a[cnt];

return;

}

LL m = (l+r)/2;

build(l, m, 2\*k);

build(m+1, r, 2\*k+1);

tree[k].w = tree[2\*k].w + tree[2\*k+1].w;

}

//prime sifter

bool isP[maxn];

int P[maxn];

int total = 0;

void SiftPrime(int maxm)

{

memset(isP, true, sizeof(isP));

memset(P, 0, sizeof(P));

isP[0] = isP[1] = false;

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

{

if(isP[i]) P[++total]=i;

for(int j = 1;j <= total && i\*P[j] <= maxm;j++)

{

isP[i\*P[j]]=false;

if(!(i%P[j])) break;

}

}

//Euler function

int phi[maxm];//caution

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

phi[i] = i;

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

for(int j = 1;P[i]\*j < maxm;j++)

phi[P[i]\*j] = phi[P[i]\*j]\*(P[i]-1)/P[i];

}

//quick power

LL qpow(LL base,LL expo)

{

base %= k;

LL ans = 1;

while(expo)

{

if(expo & 1)

ans = ans % k \* base % k;

expo >>= 1;

base = base % k \* base % k;

}

return ans % k;

}

//fft

const double PI = acos(-1.0);

int l, r[MAXN];

struct cp{

double real, imag;

cp(double real = 0, double imag = 0):real(real), imag(imag){}

cp operator + (const cp& rhs) const{return cp(real + rhs.real, imag + rhs.imag);}

cp operator - (const cp& rhs) const{return cp(real - rhs.real, imag - rhs.imag);}

cp operator \* (const cp& rhs) const{return cp(real \* rhs.real - imag \* rhs.imag, real \* rhs.imag + imag \* rhs.real);}

}A[MAXN], B[MAXN];

cp omega(int limits, int k)

{

return cp(cos(PI \* k / limits), sin(PI \* k / limits));

}

void fft(cp \*a, int limits, int type)

{

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

if(i < r[i]) swap(a[i], a[r[i]]);

for(int mid = 1;mid < limits;mid <<= 1)

for(int R = mid << 1, j = 0;j < limits;j += R)

for(int k = 0;k < mid;k++)

{

cp w = omega(mid, type \* k) \* a[j + k + mid];

a[j + k + mid] = a[j + k] - w;

a[j + k] = a[j + k] + w;

}

}

void get\_reverse()

{

int limits = 1;

while(limits <= m + n) limits <<= 1, l++;

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

r[i] = (r[i>>1]>>1) | ((i&1)<<(l-1));

}