// convex hull

struct Point

{

double x, y;

Point(double x = 0, double y = 0):x(x), y(y){}

}p[MAXN];

Point hull[MAXN];

int top = 0, n, coord\_x, coord\_y;

double dist(Point p1, Point p2){return sqrt((p1.x-p2.x)\*(p1.x-p2.x) + (p1.y-p2.y)\*(p1.y-p2.y));}

double cross(Point p1, Point p2){return p1.x \* p2.y - p1.y \* p2.x;}

double cross(double x1, double y1, double x2, double y2){return x1 \* y2 - y1 \* x2;}

double area(Point p1, Point p2, Point p3){return fabs(cross(Point(p2.x-p1.x, p2.y-p1.y), Point(p3.x-p2.x, p3.y-p2.y)));}

bool eq(double a, double b){return fabs(a - b) < EPS;}

bool eq(Point p1, Point p2){return eq(p1.x, p2.x) && eq(p1.y, p2.y);}

bool geq(double a, double b){return a > b || eq(a, b);}

bool oneline(Point p1, Point p2){return eq(cross(p1, p2), 0);}

bool cmp\_angle(Point p1, Point p2){return (cross(p1, p2) > 0) || (oneline(p1, p2) && (p1.x > p2.x || p1.y > p2.y));}

bool is\_convex(Point p1, Point p2, Point p3){return cmp\_angle(Point(p2.x-p1.x, p2.y-p1.y), Point(p3.x-p2.x, p3.y-p2.y));}

Point calc\_intersection(Point p1, Point p2, Point p3, Point p4) // p1: (0, 0)

{

double k1 = p2.y / p2.x;

if(eq(p3.x, p4.x)) return Point(p3.x, k1\*p3.x);

double k2 = (p4.y - p3.y) / (p4.x - p3.x);

double b = p3.y - k2\*p3.x;

if(eq(k1, k2)) return Point(2\*p4.x-p3.x, 2\*p4.y-p3.y); // a point that isn't between p3 and p4

return Point(b/(k1-k2), k1\*b/(k1-k2));

}

double x\_min, y\_min;

void graham()

{

y\_min = 0, x\_min = 0;

x\_max = -INF, y\_max = -INF;

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

scanf("%lf%lf", &p[i].x, &p[i].y);

// find y\_min

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

p[i].x -= x\_min,

p[i].y -= y\_min;

p[n++] = Point(x\_max, -y\_min);

p[n++] = Point(-x\_min, y\_max);

swap(p[n-1], p[0]); // the point that is the most left: p[n-1]

sort(p+1, p+n, cmp\_angle);

hull[top++] = p[0];

hull[top++] = p[1];

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

{

if(oneline(hull[top-1], p[i])) continue;

while(top && !is\_convex(hull[top-2], hull[top-1], p[i]))

top--;

if(is\_convex(hull[top-2], hull[top-1], p[i]))

hull[top++] = p[i];

}

hull[top++] = hull[0];

}

bool in\_section(Point p1, Point p2, Point p3)

{

if(eq(p1, p2) || eq(p1, p3)) return true;

if(geq(0, (p1.x-p2.x)\*(p1.x-p3.x)) && geq(0, (p1.y-p2.y)\*(p1.y-p3.y))) return true;

return false;

}

//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 BN

{

char num[100];// the digits are stored reversely

int len;

};

void swap2(char \*a,char \*b){char t=\*a;\*a=\*b;\*b=t;}

void BNInit(struct BN \*a, int b)

{

a->len = 0;

while(b)

{

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

b /= 10;

a->len++;

}

}

// lhs > rhs : return 1

// lhs == rhs : return 0

// lhs < rhs : return -1

int BNCmp(struct BN \*lhs, struct BN \*rhs)

{

int i;

if(lhs->len != rhs->len)

return lhs->len > rhs->len ? 1 : -1;

for(i = lhs->len - 1;i >= 0;i--)

if(lhs->num[i] != rhs->num[i])

return lhs->num[i] > rhs->num[i] ? 1 : -1;

return 0;

}

struct BN BNScanf()

{

int i;

char str[100];

struct BN ans;

scanf("%s", str);

ans.len = strlen(str);

for(i = 0;i < ans.len;i++)

ans.num[i] = str[ans.len-1-i] - '0';

return ans;

}

void BNPrint(struct BN a)

{

if(a.len == 0) putchar('0');

for(int i = a.len-1;i >= 0;i--)

putchar(a.num[i] + '0');

putchar('\n');

}

void BNTrim(struct BN \*a)

{

while(a->len && a->num[a->len - 1] == 0)

a->len--;

}

void BNAdd(struct BN a,struct BN b,struct BN \*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;

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';}

}

// subtract returns the absolute value

void BNSubtract(struct BN \*a,struct BN \*b,struct BN \*c)

{

int i;

if(BNCmp(a, b) < 0){BNSubtract(b, a, c);return;}

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] < 0){c->num[i] += 10;carry = 1;}

}

BNTrim(c);

}

void BNMultiply(struct BN a,struct BN b,struct BN \*c)

{

int i, j;

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

c->len = 1;

for(i = 0;i < b.len;i++)

{

for(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 BNDiv(struct BN a, int b, struct BN \*c, int \*r)

{

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

swap2(a.num+i,a.num+a.len-1-i);

\*r = 0;

BNInit(c, 0);

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

{

(\*r) = (\*r)\*10 + a.num[i];

if(c->len|| (\*r) >= b)

{

c->num[c->len] = (\*r) / b;

(\*r) %= b;

c->len++;

}

}

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

swap2(c->num+i,c->num+c->len-1-i);

}

void BNDiv2(struct BN a,struct BN b,struct BN \*c,struct BN \*r){

int i;

struct BN CONST\_10, a\_digit, r\_temp;

BNInit(r,0);

BNInit(&CONST\_10, 10);

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

swap2(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++){

BNInit(&a\_digit, a.num[i]);

memcpy(&r\_temp, r, sizeof(struct BN));

BNMultiply(r\_temp, CONST\_10, r);

memcpy(&r\_temp, r, sizeof(struct BN));

BNAdd(r\_temp, a\_digit, r);

if(c->len || BNCmp(r, &b) >= 0)

{

while(BNCmp(r, &b) >= 0)

{

memcpy(&r\_temp, r, sizeof(struct BN));

BNSubtract(&r\_temp, &b, r);

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

}

c->len++;

}

}

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

swap2(c->num+i,c->num+c->len-1-i);

}

// the part of number theory

LL a[MAXN], b[MAXN];

LL euclid(LL a, LL b)

{

return b == 0 ? a : euclid(b, a % b);

}

LL LCM(LL a, LL b)

{

return a / euclid(a, b) \* b;

}

LL extend\_euclid(LL a, LL b, LL &x, LL &y)

{

if(b == 0)

{

x = 1, y = 0;

return a;

}

LL r = extend\_euclid(b, a%b, x, y);

LL temp = y;

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

x = temp;

return r;

}

LL qmul(LL a,LL b, LL MOD)

{

a %= MOD;

LL ans = 0;

while(b)

{

if(b & 1)

ans = (ans + a) % MOD;

b >>= 1;

a = (a + a) % MOD;

}

return ans % MOD;

}

LL extend\_crt(LL n)// x === bi(mod ai)

{

LL M, ans;

M = a[1];

ans = b[1] % a[1];

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

{

LL t, temp\_y, gcd;

gcd = extend\_euclid(M, a[i], t, temp\_y); // t \* M[k-1] + a[k] \* temp\_y = 1

LL c = (b[i] - ans % a[i] + a[i]) % a[i];

if(c % gcd) return -1;

t = qmul(t, c / gcd, a[i]); //caution!

ans += (t\*M);

M = LCM(M, a[i]);

ans = (ans % M + M) % M;

}

return (ans % M + M) % M;

}

//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 sum\_mu[maxn], sum\_phi[maxn]

int total = 0;

void sift\_prime()

{

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

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

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

sum\_mu[1] = sum\_phi[1] = 1;

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

{

if(isP[i])

{

P[++total]=i;

sum\_mu[i] = -1;

sum\_phi[i] = i-1;

}

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

{

isP[i\*prime] = false;

if(!(i%prime))

{

sum\_phi[i\*prime] = prime \* sum\_phi[i];

break;

}

sum\_phi[i\*prime] = sum\_phi[prime] \* sum\_phi[i];

sum\_mu[i\*prime] = -sum\_mu[i];

}

}

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

sum\_phi[i] += sum\_phi[i-1],

sum\_mu[i] += sum\_mu[i-1];

}

// the part of combination

void init\_fac(LL n)

{

fac[0] = 1;

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

fac[i] = fac[i-1] \* i % MOD;

}

LL inv(LL num)

{

return qpow(num, MOD - 2) % MOD;

}

LL C(LL n, LL m)

{

if(m > n) return 0;

return fac[n] \* inv(fac[m] \* fac[n-m]) % MOD;

}

LL lucas(LL n, LL m)

{

if(n <= MOD && m <= MOD) return C(n, m);

return lucas(n / MOD, m / MOD) \* lucas(n % MOD, m % MOD) % MOD;

}

//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));

}

//closest distance

struct Point{double x, y;}Points[maxn], temp[maxn];

bool cmp\_x(Point &lhs, Point &rhs){return lhs.x < rhs.x;}

bool cmp\_y(Point &lhs, Point &rhs){return lhs.y < rhs.y;}

double dist(Point &lhs, Point &rhs)

{

return sqrt((lhs.x - rhs.x)\*(lhs.x - rhs.x) + (lhs.y - rhs.y)\*(lhs.y - rhs.y));

}

double merge(int low, int mid, int high, double d)

{

int len = 0;

double mid\_x = Points[mid].x;

for(int i = mid;i >= low && (Points[i].x >= mid\_x - d);i--)

temp[len++] = Points[i];

for(int i = mid+1;i <= high && (Points[i].x <= mid\_x + d);i++)

temp[len++] = Points[i];

sort(temp, temp + len, cmp\_y);

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

for(int j = 1;i + j < len && (temp[i+j].y - temp[i].y <= d);j++)

d = min(d, dist(temp[i], temp[i+j]));

return d;

}

double closest(int low, int high)

{

if(high == low) return INF;

if(high - low == 1) return dist(Points[low], Points[low+1]);

if(high - low == 2)

{

double d1 = dist(Points[low], Points[low+1]);

double d2 = dist(Points[low], Points[low+2]);

double d3 = dist(Points[low+1], Points[low+2]);

if(d1 > d2) swap(d1, d2);

if(d2 > d3) swap(d2, d3);

if(d1 > d2) swap(d1, d2);

return d1;

}

int mid = (low + high) / 2;

double d1 = closest(low, mid);

double d2 = closest(mid + 1, high);

double d = min(d1, d2);

return merge(low, mid, high, d);

}

int main() // sort(Points\_x + 1, Points\_x + n + 1, cmp\_x)