**// ========= FFT 傅里叶=========**

#include <algorithm>

#include <cmath>

using namespace std;

const double PI = acos(-1.0);

struct complex {

double r,i;

complex(double \_r = 0.0,double \_i = 0.0)

{r = \_r; i = \_i;}

complex operator +(const complex &b)

{return complex(r+b.r,i+b.i);}

complex operator -(const complex &b)

{return complex(r-b.r,i-b.i);}

complex operator \*(const complex &b)

{return complex(r\*b.r-i\*b.i,r\*b.i+i\*b.r);}

};

void change(complex y[],int len) {

int i,j,k;

for(i = 1, j = len/2;i < len-1; i++)

{

if(i < j)swap(y[i],y[j]);

k = len/2;

while( j >= k) {j -= k;k /= 2;}

if(j < k) j += k;

}

}

void fft(complex y[],int len,int on)

//on==-1 IDFT

{

change(y,len);

for(int h = 2; h <= len; h <<= 1)

{

complex wn(cos(-on\*2\*PI/h),sin(-on\*2\*PI/h));

for(int j = 0;j < len;j+=h)

{

complex w(1,0);

for(int k = j;k < j+h/2;k++)

{

complex u = y[k];

complex t = w\*y[k+h/2];

y[k] = u+t;

y[k+h/2] = u-t;

w = w\*wn;

}

}

}

if(on == -1)

for(int i = 0;i < len;i++) y[i].r /= len;

}

const int MAXN = 200011;

complex x[MAXN \* 4];

LL num[MAXN \* 4]; int a[MAXN];

int main() {

memset(num, 0, sizeof(num));

for (int i = 0;i < N; i++) {

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

num[a[i]]++;

}

sort(a, a+N);

int len\_tmp = a[N-1]+1, len = 1;

while (len < len\_tmp\*2)

len <<= 1;

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

x[i] = complex(num[i],0);

fft(x, len, 1);//DFT

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

x[i] = x[i]\*x[i];

fft(x, len, -1);//IDFT

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

num[i] = (LL)round(x[i].r);

//可能要：求组合而不是求排列

num[i] /= 2;

//可能要：扣除a[i]+a[i]的情况

num[a[i]+a[i]]--;

//可能要：扣除带0的特殊情况

Cnt -= (LL)Cnt0 \* (N-1) \* 2LL;// 0+ai=ai && ai+0=ai

printf("%lld\n", Cnt); }

**// ============ Catalan ============**

Catalan数

Cat[n]=C[2\*n][n]/(n+1)

组合性质

Cat[n+1]=sum(Cat[i]\*Cat[n-i] for i from 0 to n)

Cat[0]=1, Cat[n+1]=2\*(2\*n+1)/(n+2)\*Cat[n]

Cat[n]为长度为2\*n的Dyck词总数(长度为2\*n的Dyck词由n个'X'和n个'Y'组成, 对于其任意前缀, 有count('X')>=count('Y'))

Cat[n]为给长度(n+1)的序列打上括号的不同方案数

Cat[n]为有(n+1)片叶子的不同完全二叉树数

Cat[n]为n\*n网格线从左下角到右上角不经过左上部分的最短路径数

Cat[n]为用不相交直线将凸(n+2)边形划分为n个三角形的方案数

Cat[n]为有n个非叶子节点的不同二叉树数

**// =========== Combination ===========**

ll fast\_pow(ll x, ll k, ll p);

int Combination(int m, int n, int p){

ll nom=1, den=1;

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

{nom\*=i; nom%=p; }

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

{den\*=i; den%=p; }

den=fast\_pow(den, p-2, p);

return (nom\*den)%p;

}

ll C[maxn][maxn];

int Combination\_table(int n, ll MOD){

memset(C, 0, sizeof(C));

C[0][0]=1;

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

C[i][0]=1;

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

C[i][j]=(C[i-1][j-1]+C[i-1][j])%MOD;

}

}

**// ====== cont\_frac连分数逼近 ======**

ll a[maxn];

/\* 连分数逼近由欧几里德算法求解

n/d=a[0]+1/(a[1]+1/(a[2]+1/(a[3]+1/(...+1/ a[len-1])))) \*/

int cont\_frac(ll n, ll d){

ll r;

int len=0;

while(d){

a[len++]=n/d;

r=n%d; n=d; d=r;

}

return len;

}

**// ========== Euler\_Phi ==========**

int euler\_phi(int n){

int m=int(sqrt(n+0.5)), res=n;

for(int i=2; i<=m; i++) if(n%i==0){

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

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

}

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

return res;

}

int phi[maxn];

void phi\_table(int n){

memset(phi, 0, sizeof(phi));

phi[1]=1;

for(int i=2; i<=n; i++) if(!phi[i])

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

if(!phi[j]) phi[j]=j;

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

}

}

**// ======== Fibonacci数 ========**

F[0]=F[1]=1;

F[n]=F[n-1]+F[n-2];

组合性质

F[n]=sum(C[n-k-1][k] for k=0 to floor((n-1)/2)) C[i][j]表示组合数

sum(F[i] for i=1 to n)=F[n+2]-1

sum(F[2\*i+1] for i=0 to n-1)=F[2\*n]

sum(F[2\*i] for i=1 to n)=F[2\*n+1]-1

sum(F[i]\*F[i] for i=1 to n)=F[n]\*F[n+1]

Catalan性质:

F[n]\*F[n]-F[n-r]\*F[n+r]=(-1)^(n-r)\*F[r]\*F[r]

Vajda性质: F[n+i]\*F[n+j]-F[n]\*F[n+i+j]=

(-1)^n\*F[i]\*F[j]

d'Ocagne性质:

F[k\*n+c]=sum

(C[k][i]\*F[c-i]\*F[n]^i\*F[n+1]^(k-i) for i=0 to k)

母函数: s(x)=sum

(F[k]\*x^k for k=0 to infinity)=x/(1-x-x^2)

数论性质:

gcd(F[m], F[n])=F[gcd(m, n)]

整数N为Fibonacci数的充要条件为5\*N^2+4或5\*N^2-4为完全平方数

p|F[p-(5/p)] (此处括号为Legendre标记)

如果从1开始计数, 则除了F[4]=3以外, 若下标n为合数则F[n]也为合数

除了1以外Fibonacci数中仅有8和144为整次方数

除了1, 8, 144, 所有Fibonacci数都有至少一个质因数不在所有比其下标更小的Fibonacci数的质因数的集合中

若构造一数列A[i]=F[i]%n, n为任意正整数, 则数列A有周期性且其周期不超过6\*n

**// ============ primes ============**

Legendre符号 (p为质数)

(a|p)=0 if a%p=0

(a|p)=1 if a%p!=0 and 存在整数x x^2=a (mod p)

(a|p)=-1 otherwise

Euler准则: 若p为奇质数且p不能整除d 则d^((p-1)/2)=(d|p) (mod p)

Legendre符号是完全积性函数

二次互反律: 若p, q为奇质数, 则(q|p)=(p|q)\*(-1)^((p-1)/2\*(q-1)/2)

Mersenne数

M[n]=2^n-1

Euclid-Euler定理: 若M[p]为素数, 则(2^p-1)\*2^(p-1)为完全数

若p为奇质数, 则M[p]的所有质因子模2\*p同余1

若p为奇质数, 则M[p]的所有质因子模8同余+/-1

M[m]与M[n]互质的充要条件为m与n互质

若p与2\*p+1皆为素数且p模4同余3, 则(2\*p+1)为M[p]的因子

Wilson定理

大于1的自然数n为素数的充要条件为(n-1)!=-1 (mod n)

Fermat多边形数定理

每一个正整数最多可以表示为n个n边形数之和

Euler引理

对于任意奇素数p, 同余方程x^2+y^2+1=0 (mod p) 必有一组正整数解(x, y)满足0<=x<p/2, 0<=y<p/2

Lagrange的四平方和定理

每个正整数均可以表示为4个整数的平方和

**// ============ log\_mod ============**

//解方程a^x=b (mod n) n为素数

int shank(int a, int b, int n){

int m, v, e=1, i;

m=int(sqrt(n+0.5));//复杂度为O((m+n/m)logm) 所以m==n/m时最快

v=inv(fast\_pow(a, m, n), n);//fast\_pow(a, m, n)=(a^m)%n

map<int, int> x;//x[j]=min(i|e[i]==j)

x[1]=0;

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

e=a\*e%n;//e=(a^i)%n

if(!x.count(e)) x[e]=i;

}

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

//a^(im)到a^(im+m-1)

if(x.count(b)) return i\*m+x[b];

b=b\*v%n;//递推更新b

}

return -1;//无解

}

**// ============ matrix ============**

struct parametre{int c, r;};

struct Matrix{

long long matrix[maxn][maxn];

parametre DIM;

Matrix(){}

Matrix(int c, int r){

DIM={c, r};

memset(matrix, 0, sizeof(matrix));

}

Matrix operator\*(Matrix &A){

Matrix C(DIM.c, A.DIM.r);

memset(C.matrix, 0, sizeof(C.matrix));

for(int i=0; i<DIM.c; i++)

for(int j=0; j<A.DIM.r; j++)

for(int k=0; k<DIM.r; k++)

C.matrix[i][j]=(C.matrix[i][j]+A.matrix[i][k]\* matrix[k][j])%MOD;

return C;

}

Matrix operator+(Matrix &A){

Matrix C(A.DIM.c, A.DIM.r);

for(int i=0; i<DIM.c; i++)

for(int j=0; j<DIM.r; j++)

C.matrix[i][j]=A.matrix[i][j]+matrix[i][j];

return C;

}

void print(){

for(int i=0; i<DIM.c; i++){

for(int j=0; j<DIM.r; j++) cout<<matrix[i][j]<<'\t';

cout<<endl;

}

}

};

Matrix BigMatrixExpo(Matrix &A, long long n){

Matrix B=A;

Matrix C(A.DIM.c, A.DIM.r);

for(int i=0; i<C.DIM.c; i++)

for(int j=0; j<C.DIM.r; j++)

C.matrix[i][j]=i==j;

while(n){

if(n&1) C=C\*B;

B=B\*B;

n>>=1;

}

return C;

}

//定义新矩阵 Matrix a(3, 5); a.matrix={{},{},{}};

//乘法 c=a\*b; 注意a的第一个parametre等于b的第二个parametre;

//加法 c=a+b; //输出 c.print();

**// ============ 莫比乌斯 ============**

//完全积性函数 mo[i\*j]=mo[i]\*mo[j]

//sum(mo[d] for d|n)=(n==1)

//反演公式

//若f(n)=sum(g(d) for d|n) 则 g(n)=sum(mo[n/d]\*f(d) for d|n)=sum(mo[d]\*f(n/d) for d|n)

//If f(i)=sum(g(d\*i) for d from 1 to floor(n/i)) then g(i)=sum(f(d\*i)\*mo[d] for d from 1 to floor(n/i))

bool vis[maxn+123];

int mo[maxn+123], primes[maxn+123], a[maxn+123], pcnt, N;

void mobius(){//预处理

mo[1]=1;

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

if(!vis[i])

{ mo[i]=-1; primes[pcnt++]=i; }

for(int j=0; j<pcnt&&ll(i)\*primes[j]<=maxn; j++){

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

if(i%primes[j]) mo[i\*primes[j]]=-mo[i];

else{

mo[i\*primes[j]]=false;

break;

}

}

}

for(int i=2; i<=maxn; i++) mo[i]+=mo[i-1];//mo记录前缀和

}

//O(sqrt(n)+sqrt(m))

ll cnt\_gcd(ll n, ll m, ll k){//for i from 1 to n for j from 1 to m cnt gcd(i, j)=k

if(n>m) swap(n, m);

ll res=0;

n/=k, m/=k;

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

j=min(n/(n/i), m/(m/i));

res+=ll(mo[j]-mo[i-1])\*(n/i)\*(m/i);//前缀和Mobius

}

return res;

}

**// =========== 高斯消元 ============**

typedef int Matrix[maxn][maxn];

void exgcd(int a, int b, int& d, int& x, int& y){

!b?(d=a, x=1, y=0):(exgcd(b, a%b, d, y, x), y-=x\*(a/b));

}

int inv(int a){

int d, x, y;

exgcd(a, MOD, d, x, y);

return d==1?(x+MOD)%MOD:-1;

}

int gauss\_jordan(Matrix A, int n, int m){//A是增广矩阵, n个未知数, m个方程, MOD是模, 如果MOD不是质数的话每次inv完先检测是否是-1

int i=0, j=0;

while(i<m&&j<n){

int row=i;

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

if(A[k][j]){

row=k;

break;

}

}

if(row!=i) for(int k=0; k<=n; k++)

swap(A[i][k], A[row][k]);

if(!A[i][j]){

j++; continue;

}

for(int k=0; k<m; k++){

if(!A[k][j]||i==k) continue;

int cur=A[k][j]\*inv(A[i][j])%MOD;

for(int t=j; t<=n; t++)

A[k][t]=( (A[k][t]-cur\*A[i][t])

%MOD+MOD)%MOD;

}

i++;

}

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

if(A[k][n]) return -1;//无解

if(i<n) return 0;//无限解

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

A[k][n]=A[k][n]\*inv(A[k][k])%MOD;

//解存在A[k][n]里面

return 1;

}

**// ======= pell equation 佩尔方程======**

//用于求解标准型Pell方程的第(k+1)组非平凡解 (x^2-n\*y^2=1)

//输入n, k和MOD

//递推关系为 x[i+1]=x[0]\*x[i]+n\*y[0]\*y[i];

//y[i+1]=y[0]\*x[i]+x[0]\*y[i];

//上述递推关系可由sqrt(n)的连分数表示推出

typedef pair<ll, ll> pii;

pii res;//(xk, yk)

ll MOD;//模<必须是全局变量>

void Find(ll n, ll& x, ll& y){

//暴力寻找特解(x0, y0)

y=1;

while(true){

x=sqrt(y\*y\*n+1);

if(x\*x-n\*y\*y==1) break;

y++;

}

}

struct parameter{int c, r;};

struct Matrix{

ll matrix[maxn][maxn];

parameter DIM;

Matrix(){}

Matrix(int c, int r);

Matrix operator\*(Matrix &A);//带模乘法

Matrix operator+(Matrix &A);

void print();

};

Matrix BigMatrixExpo(Matrix &A, ll n);//带模快速幂

bool Pell(ll n, ll k){//k为第k组解, 从0开始数

ll t=sqrt(n)+0.5, x, y;

if(t\*t==n) return false;//仅有平凡解 (1, 0) 和 (-1, 0)

Matrix A(2, 2);

Find(n, x, y);

A.matrix[0][0]=A.matrix[1][1]=x;

A.matrix[0][1]=n\*y;

A.matrix[1][0]=y;

A=BigMatrixExpo(A, k-1);

res=make\_pair((A.matrix[0][0]\*x+A.matrix[0][1]\*y)%MOD, (A.matrix[1][0]\*x+A.matrix[1][1]\*y)%MOD);

return true;

}

**// ============ CRT===========**

typedef long long ll;

//n个方程为x=a[i] (mod m[i])

ll china(int n, int\* a, int\* m){

ll M=1, d, y, x=0;//M是等价以后的模

for(int i=0; i<n; i++) M\*=m[i];

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

ll w=M/m[i];

exgcd(m[i], w, d, d, y);

x=(x+y\*w\*a[i])%M;

}

return (x+M)%M;

}

**// ============ Fraction ===========**

ll gcd(ll a, ll b){ return !b?a:gcd(b, a%b); }

struct fraction{

ll num, den;

fraction(){ num=0; den=1; }

fraction(ll a, ll b)

{num=a; den=b; simplify();}

inline void reset(){ num=0; den=1;}

void simplify(){

ll d=gcd(num, den);

num/=d;

den/=d;

if(den<0){num=-num;den=-den;}

}

inline ll convert(){return num/den;}

fraction& operator = (int rhs){

(\*this).num=rhs;

(\*this).den=1;

return \*this;

}

fraction operator + (const fraction &rhs) const{

fraction res;

res.den=lcm(den, rhs.den);

res.num=res.den/den\*num+res.den/rhs.den\*rhs.num;

res.simplify();

return res;

}

fraction operator += (const fraction &rhs)

{return \*this=\*this+rhs;}

fraction operator + (const int &rhs) const{

fraction r(rhs, 1);

return \*this+r;

}

fraction operator += (const int &rhs)

{return \*this=\*this+rhs;}

fraction operator - (const fraction &rhs) const{

fraction res;

res=\*this+fraction(-1, 1)\*rhs;

res.simplify();

return res;

}

fraction operator -= (const fraction &rhs)

{return \*this=\*this-rhs;}

fraction operator - (const int &rhs) const

{fraction r(rhs, 1);return \*this-r;}

fraction operator -= (const int &rhs)

{return \*this=\*this-rhs;}

fraction operator \* (const fraction &rhs) const{

fraction res;

res.num=num\*rhs.num;

res.den=den\*rhs.den;

res.simplify();

return res;

}

fraction operator \*= (const fraction &rhs)

{return \*this=(\*this)\*rhs;}

fraction operator \* (const int &rhs) const

{fraction r(rhs, 1); return (\*this)\*r;}

fraction operator \*= (const int &rhs)

{return \*this=(\*this)\*rhs;}

fraction operator / (const fraction &rhs) const{

fraction res;

res.num=num\*rhs.den;

res.den=den\*rhs.num;

res.simplify();

return res;

}

fraction operator /= (const fraction &rhs)

{return \*this=(\*this)/rhs;}

fraction operator / (const int &rhs) const

{ fraction r(rhs, 1); return (\*this)/r; }

fraction operator /= (const int &rhs)

{return \*this=(\*this)/rhs;}

bool operator == (const fraction &rhs) const

{return num\*rhs.den==den\*rhs.num;}

bool operator == (const int &rhs) const

{return num==den\*rhs;}

bool operator != (const fraction &rhs) const

{return !(\*this==rhs);}

bool operator != (const int &rhs) const

{return !(\*this==rhs);}

bool operator < (const fraction &rhs) const

{return num\*rhs.den<den\*rhs.num;}

bool operator < (const int &rhs) const

{return num<den\*rhs;}

bool operator > (const fraction &rhs) const

{return num\*rhs.den>den\*rhs.num;}

bool operator > (const int& rhs) const

{return num>den\*rhs;}

bool operator <= (const fraction &rhs) const

{return \*this==rhs||\*this<rhs;}

bool operator <= (const int& rhs) const

{return \*this==rhs||\*this<rhs;}

bool operator >= (const fraction &rhs) const

{return \*this>rhs||\*this==rhs;}

bool operator >= (const int &rhs) const

{return \*this>rhs||\*this==rhs;}

};

**// ========= 辛普森积分 =========**

double simpson(double a, double b) {

double c = (a + b) / 2.0;

return (F(a)+4\*F(c)+F(b)) \* (b-a) / 6.0;

} // 这里F为自定义函数

//given A as the simpson Value for the whole interval [a,b]

double asr(double a, double b, double eps, double A) {

double c = (a + b) / 2.0;

double L = simpson(a, c)；

double R = simpson(c, b);

if (fabs(L+R-A) <= 15\*eps)

return L + R + (L+R-A)/15.0;

return asr(a, c, eps/2, L) + asr(c, b, eps/2, R);

}

double asr(double a, double b, double eps)

{ return asr(a, b, eps, simpson(a, b)); } //接口

// int main(): 调用asr(left, right, 1e-5)

// 得到F(x) 在[left, right]上的积分 eps也可改为1e-6

**// ============ 凸包 ============**

const double eps=1e-10;

const double PI=acos(-1);

struct Point{ double x, y;

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

} p[maxn], ch[maxn];

typedef Point Vector;

Vector operator + (Vector A, Vector B)

{ return Vector(A.x+B.x, A.y+B.y); }

Vector operator - (Vector A, Vector B)

{ return Vector(A.x-B.x, A.y-B.y); }

Vector operator \* (Vector A, double p)

{ return Vector(A.x\*p, A.y\*p); }

Vector operator / (Vector A, double p)

{ return Vector(A.x/p, A.y/p); }

int dcmp(double x) {

if(fabs(x) < eps) return 0;

else return x < 0 ? -1 : 1; }

bool operator == (const Point& a, const Point& b)

{ return dcmp(a.x-b.x) == 0 && dcmp(a.y-b.y) == 0; }

bool operator < (const Point& a, const Point& b)

{ return a.x < b.x || (a.x == b.x && a.y < b.y); }

double cross(Vector A, Vector B)

{ return A.x\*B.y - A.y\*B.x; }

double torad(double deg)

{ return deg / 180 \* PI; }

double PolygonArea(Point\* p, int n){

double area=0;

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

area+=cross(p[i]-p[0], p[i+1]-p[0]);

return area/2;

}

int convexhull(Point\* p, int n, Point\* ch) {

sort(p, p+n);

int m = 0;

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

while(m > 1 && cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;

ch[m++] = p[i];

}

int k = m;

for(int i = n-2; i >= 0; i--) {

while(m > k && cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;

ch[m++] = p[i];

}

if(n > 1) m--; return m;

}

Vector Rotate(Vector A, double rad){

// 这里rad是逆时针旋转的角度

return Vector(A.x\*cos(rad)-A.y\*sin(rad),

A.x\*sin(rad)+A.y\*cos(rad));

}

// int main()

Point o(tmpx, tmpy);

p[point\_cnt++]=o; ... ...

int m=convexhull(p, point\_cnt, ch);

double convex\_area=PolygonArea(ch, m);

// Rotate vector(10,10) clockwise by 90 degree

// new\_o = o + Rotate(Vector(10.0,10.0),-torad(90.0));

**// ========== 点在多边形内 ==========**

bool PNPoly(int u, int deg) {

if (! (vertxmin <= x[u] <= vertxmax) || ! ( vertymin <= y[u] <= vertymax ) ) return 0;

bool is\_in = 0; int i,j;

for(i=0;i<deg;i++) {

if(!i) j= deg-1;

else j= i-1;

if ( ((poy[i] > y[u]) != (poy[j] > y[u])) && (x[u] < (pox[j] - pox[i]) \* (y[u] - poy[i]) / (poy[j] - poy[i]) + pox[i]) )

is\_in = ! is\_in;

}

return is\_in;

}