//Number Theory

1. **BigMod**

#define MOD 1000000007 //the value through which I will mod

ll bigMod(ll n,ll power, ll mod) //(n^power)%mod

{

if(power==1) return n%MOD; // returning the base case

else if(power%2 == 0)

{

ll v = bigMod(n,power/2,mod);

return ((v%mod)\*(v%mod))%mod; // (v\*v)

}

else {

ll v = bigMod(n,power-1,mod)%mod;

return ((v%mod)\*(n%mod))%mod; // n \* bigMod(n,p-2,p);

}

}

**2. GCD**

//caluculating the GCD

ll gcd(ll a, ll b)

{

if(a==0) return b;

if(b==0) return a;

ll t;

while(b != 0)

{

t = b;

b = a % b;

a = t;

}

return a;

}

3. SIEVE Function

//Sieve function

//MAX\_SIEVE is the range till I will calculate my prime

//boolean array numbers[] and primes[] real primes

//primeCounter is the calculation of prime numbers

bool numbers[MAX\_SIEVE+5];

int primes[MAX\_SIEVE];

int primeCounter;

void sieve() // the function begins

{

for(int i=0;i<MAX\_SIEVE;i++)

{

numbers[i]=true; // initialization everyone is prime

}

for(int i=4;i<MAX\_SIEVE; i+= 2)

{

numbers[i] = false; // even numbers are not prime

}

int sqrtRoot = sqrt(MAX\_SIEVE); // square root and it's an integer

for(int i=3;i<sqrtRoot; i+= 2)

{

if(numbers[i] == true) // if it's a prime

{

for(int j = i \* i; j <= MAX\_SIEVE ; j += 2 \* i)

{

numbers[j] = false; // all the multiples are not prime

}

}

}

primeCounter = 0;

for(int i=2; i<=MAX\_SIEVE ; i++)

{

if(numbers[i] == true) // if it's true it's a prime

{

primeCounter++;

primes[primeCounter] = i;

}

}

}

4. Prime Factorization

//This function factorize every number to their prime

//vector primeFactor saves the primes which divides the numbers

//vector primeFactorPower saves the prime powers

vector<int>primeFactor[MAX\_SIEVE+5];

vector<int>primeFactorPower[MAX\_SIEVE+5];

void primeFactorization()

{

for(int i=2;i<=MAX\_SIEVE;i++) // from 2 to MAX\_SIEVE number will be factorized

{

int v=i; // temporary saved to v

for(int j=1;j<=sqrt(v) && primes[i]<=primeCounter ; j++) // we will run loop till sqrt

{

int counter = 0; // will count if prime[j] divides

if(v%primes[j] == 0) // divisible

{

while(v%primes[j]==0)

{

counter++;

v /= primes[j];

}

primeFactor[i].push\_back(primes[j]);

primeFactorPower[i].push\_back(counter);

}

}

if(v != 1) //the number itself is prime

{

primeFactor[i].push\_back(v);

primeFactorPower[i].push\_back(1);

}

}

}

5.Bitwise Sieve

//the function for bitwise sieve

#define BITWISE\_MAX MAX/32+1

int number[BITWISE\_MAX+1];

int prime\_counter;

unsigned prime[5761456];

int bit\_set(int n,int pos){

return (n | (1<<pos));

}

int check(int n, int pos){

return (n & (1<<pos));

}

void bitwise\_sieve(){

memset(number,0,sizeof(number));

number[0]=bit\_set(number[0],0); // 0 means prime 1 means not prime

number[0]=bit\_set(number[0],1);

int v=sqrt(MAX);

for(int i=3;i<=v;i += 2){

for(int j=i\*i;j<=MAX;j += 2\*i){

assert((j/32)<=BITWISE\_MAX);

number[j/32] = bit\_set(number[j/32],j%32);

}

}

prime\_counter=0;

v=MAX/32;

prime\_counter++;

prime[prime\_counter]=2;

for(int i=3;i<=MAX;i+=2){

int val=check(number[i/32],i%32);

if(val == 0){

prime\_counter++;

prime[prime\_counter]=i;

}

}

//cout<<prime\_counter<<endl;

}

**6. Sieve With prime factorization**

#include <bits/stdc++.h>

using namespace std;

#define MAX 1000000

#define INF 1000000000

typedef long long int ll;

bool number[MAX+1];

vector<int>factor[MAX+1];

void sieve\_factorization(){

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

number[i]=true;

}

number[0]=false;

number[1]=false;

int sz=0;

for(int i=4;i<=MAX;i += 2){

number[i]=false;

factor[i].push\_back(2);

//int v=factor[i].size();

//sz=max(sz,v);

}

//cout<<"size " << factor[42].size()<<endl;

int v=sqrt(MAX);

for(int i=3;i<=MAX;i+=2){

for(int j= 2\*i;j<=MAX;j=j+i){

if(number[i]){

number[j]=false;

factor[j].push\_back(i);

int v=factor[j].size();

/\*if(j==42) {

cout<<"size " << factor[42].size()<<endl;

}

//sz=max(sz,v);\*/

}

else {

break;

}

}

}

for(int i=2;i<=MAX;i+= 1){

if(number[i]){

factor[i].push\_back(i);

}

}

//cout<<sz<<endl;

}

ll not\_one\_gcd(ll low,ll high,ll number){

int sz = (1<<factor[number].size())-1;

/\*cout<<sz<<" "<<endl;

for(int i=0;i<factor[number].size();i++){

cout<<factor[number][i]<<" ";

}

cout<<endl;\*/

ll answer=0;

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

int v = i;

int cnt = -1;

ll temp = 1;

int one\_cnt = 0;

while(v != 0){

//cout<<"v= " << v<< endl;

int rem=v%2;

v /= 2;

//cout<<"rem = " <<rem<<" v = " << v<< endl;

cnt++;

if(rem == 1){

temp=temp\*factor[number][cnt];

one\_cnt++;

}

//cout<<"v = " << v << endl;

}

//cout<<"bair"<<endl;

ll res = (high)/temp - ((low-1)/temp);

if(one\_cnt%2) {

answer += res;

}

else{

answer -= res;

}

}

return (high-low+1)-answer;

}

ll \_binary\_search(ll x,ll p, ll k){

ll low=x+1;

ll high=INF;

ll mid;

while(low<=high){

mid=(low+high)/2;

ll cnt=not\_one\_gcd(low,mid,p);

if(cnt>=k){

high=mid;

if(low==high) return low;

}

else{

low=mid+1;

k=k-cnt;

}

}

}

void list\_of\_integer(){

int t,T;

sieve\_factorization();

scanf("%d",&T);

for(t=1;t<=T;t++){

ll p,x,k;

scanf("%lld %lld %lld",&x,&p,&k);

/\*cout<<"size = " <<factor[p].size()<<endl;

for(int i=0;i<factor[p].size();i++){

cout<<factor[p][i]<<" ";

}

cout<<endl;\*/

ll ans = \_binary\_search(x,p,k);

cout<<ans<<endl;

}

return;

}

int main(int argc,char \*\*argv){

list\_of\_integer();

return 0;

}