//Prime Factors, Divisors, Sieve

#define MAX 1e5

bitset<10000000>isPrime;

vector<long long>primes;

//Only generates a number is prime or not

void sieve(unsigned long long N)

{

isPrime.set();

//0 and 1 are not prime

isPrime[0] = isPrime[1] = 0;

unsigned long long lim = sqrt(N) + 5;

for(unsigned long long i = 2; i <= lim; i++) {

if(isPrime[i]) {

for(unsigned long long j = i\*i; j <= N; j+= i)

isPrime[j] = 0;

}

}

}

//Generates a number is prime or not, and also makes an array of prime numbers

void sieveGen(unsigned long long N)

{

isPrime.set();

//0 and 1 are not prime

isPrime[0] = isPrime[1] = 0;

//Note, N isn't square rooted!

for(unsigned long long i = 2; i <= N; i++) {

if(isPrime[i]) {

for(unsigned long long j = i\*i; j <= N; j+= i)

isPrime[j] = 0;

primes.push\_back(i);

}

}

}

vector<int> primeFactor(long long n) //returns vector of co-efficient of prime factor

{

if(isPrime[n]) {

vector<int>factor(n+1, 0);

factor[n] = factor[1] = 1;

return factor;

}

vector<int>factor(sqrt(n)+1, 0); //the size of vector must be at most sqrt(n)+1

for(long long i = 0; i < (int)primes.size() && primes[i] <= n; i++) {

while(n%primes[i] == 0) { //divide 1 - n with primes 1 - n

factor[primes[i]]++; //counts how many prime in the number

n/=primes[i]; //cuts out the prime

}

}

return factor;

}

// Returns the divisors

vector<unsigned long long>divisor;

void divisors(unsigned long long n) {

unsigned long long lim = sqrt(n);

//deal with 1 and n manually

for(unsigned long long i = 2; i <= lim; i++) {

if(n % i == 0) {

unsigned long long tmp = n/i;

divisor.push\_back(tmp);

if(i != tmp)

divisor.push\_back(i);

}

}

}

//prime factorization of factorials (n!)

vector<pair<long long, long long> > factorialFactorization(long long n)

{

vector<pair<long long, long long> >V;

for(long long i = 0; i < (int)primes.size() && primes[i] <= n; i++) {

long long tmp = n, power = 0;

while(tmp/primes[i]) {

power += tmp/primes[i];

tmp /= primes[i];

}

if(power != 0)

V.push\_back(make\_pair(primes[i], power));

}

return V;

}

long long numPF(long long n) //returns number of prime factors

{

long long num = 0;

for(long long i = 0; primes[i] \* primes[i] <= n; i++) {

while(n % primes[i] == 0) {

n /= primes[i];

num++;

}

}

//there might left a prime number which is bigger than primes[i]

if(n > 1)

num++;

return num;

}

long long numDIFPF(long long n) // returns number of different prime factors

{

long long diff\_num = 0;

for(long long i = 0; primes[i] \* primes[i] <= n; i++) {

bool ok = 0;

while(n % primes[i] == 0) {

n /= primes[i];

ok = 1;

}

if(ok)

diff\_num++;

}

if(n > 1)

diff\_num++;

return diff\_num;

}

unsigned long long sumPF(long long n) //returns sum of prime factors

{

unsigned long long sum = 0;

for(long long i = 0; primes[i] \* primes[i] <= n; i++)

while(n % primes[i] == 0)

{

n /= primes[i];

sum+=primes[i];

}

if(n > 1)

sum+= n;

return sum;

}

int main()

{

return 0;

}