/\*\* sieve prime \*\*/

#define M 100000000

int marked[M/64 + 2];

#define on(x) (marked[x/64] & (1<<((x%64)/2)))

#define mark(x) marked[x/64] |= (1<<((x%64)/2))

void sieve(int n)

{ for (int i = 3; i \* i < n; i += 2) {

if (!on(i)) {

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

mark(j); } } } }

/\*\* sieve normal\*\*/

bool marked[M];

void sieve2(int n){

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

if (marked[i] == false) // i is a prime {

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

marked[j] = true; } } } }

/\*\* Divisor \*\*/

vector<int> primes; // we'll preload primes once at the beginning

int countDivisor(int n){

int divisor = 1;

for (int i = 0; i < primes.size(); i++) {

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

int cnt = 1;

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

n /= primes[i];

cnt++; }

divisor \*= cnt; } }

return divisor; }

/\*\* GCD \*\*/

int gcd(int a, int b){

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

/\*\*extended euclid where ax+by = gcd (a,b) \*\*/

typedef pair<int, int> pii;

#define fs first

#define se second

pii extendedEuclid(int a, int b) // returns x, y | ax + by = gcd(a,b){ if(b == 0) return pii(1, 0); else { pii d = extendedEuclid(b, a % b); return pii(d.se, d.fs - d.se \* (a / b)); } }

/\*\* LCM \*\*/

int lcm(int a, int b){

return (a / gcd(a, b)) \* b;}

/\*\*Modular Inverse \*\*/

int modularInverse(int a, int n) {

pii ret = extendedEuclid(a, n);

return ((ret.fs % n) + n) % n;}

/\*\* euler toient function \*\*/

int phi[M];

void calculatePhi(){

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

phi[i] = i; }

for (int p = 2; p < M; p++) {

if (phi[p] == p) // p is a prime {

for (int k = p; k < M; k += p) {

phi[k] -= phi[k] / p; } } } }

/\*\* euler toient function 2\*\*/

int phi(int n){

int ret = n;

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

if (n % i == 0) {

while (n % i == 0) {

n /= i; }

ret -= ret / i; } } /\*\* this case will happen if n is a prime number in that case we won't find any prime that divides n that's less or equal to sqrt(n) \*\*/

if (n > 1) ret -= ret / n; return ret; }

/\*\*n factorial mod m\*\*/

int nFactModm(int n,int m) {

int factorial = 1;

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

factorial \*= i;

factorial %= m; }

return factorial;}

/\*\* n sum mod m\*\*/

int nSumModm(int n,int m) {

int sum = 0;

for (int i = 1; i <= n; i++) { sum += i; sum %= m; } return sum; }