quwsarohi@gmail.com Section: Mathematics Page : 1

**// Sieve, Prime, Factorization**

bitset<10000000>isPrime;

vector<long long>primes;

void sieve(unsigned long long N) { // Only generates a number is prime or not

isPrime.set();

isPrime[0] = isPrime[1] = 0; // 0 and 1 are not prime

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;

} } }

void sieveGen(unsigned long long N) { // Generates a number is prime or not, and also makes an array of prime numbers

isPrime.set();

isPrime[0] = isPrime[1] = 0; // 0 and 1 are not prime

for(unsigned long long i = 2; i <= N; i++) { //Note, N isn't square rooted!

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]) { // v[x] contains the co-efficient of x

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 without sieve sqrt(n)**

vector<unsigned long long>divisor;

void divisors(unsigned long long n) {

unsigned long long lim = sqrt(n);

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);

} } }

quwsarohi@gmail.com Section: Mathematics Page : 2

**// 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++;

} }

if(n > 1) num++; //there might left a prime number which is bigger than primes[i]

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;

}

quwsarohi@gmail.com Section: Mathematics Page : 3

**// Binomial Coefficient C(n, k)**

**// Complexity : O(k)**

long long binomialCoeff(long long n, long long k) {

long long res = 1;

if ( k > n - k ) // Since C(n, k) = C(n, n-k)

k = n – k;

for (long long i = 0; i < k; ++i) { // Calculate value of [n \* (n-1) \*---\* (n-k+1)] / [k \* (k-1) \*----\* 1]

res \*= (n – i); **// Note: divide first then multiply to avoid overflow, decimal can be taken**

res /= (i + 1); **// After every calculation round up the value**

}

return res;

}

**// Catalan Number**

**// Use this with Binomial Coefficient**

long long catalan(int n) { //Cat(n) = C(2\*n, n)/(n+1);

long long c = bincomialCoeff(2\*n, n);

return c/(n+1); }

**// Euler’s Toitent**

/\* Euler’s Totient function Φ(n) for an input n is count of numbers in {1, 2, 3, …, n}

\* that are relatively prime to n, i.e., the numbers whose GCD (Greatest Common Divisor) with n is 1.

\* Phi(4) : GCD(1, 4) = 1, GCD(3, 4)

\* so, Phi(4) = 2

\*/

int Phi(int n) { // Computes phi of n

int result = n; // Initialize result as n

for (int p=2; p\*p<=n; ++p) { // Consider all prime factors of n and subtract their multiples from result

if (n % p == 0) { // p is a prime factor of n

while (n % p == 0) // Eliminate all p factors from n

n /= p;

result -= result / p;

} }

if (n > 1) // If n is still greater than 1, then it is also a prime

result -= result / n;

return result;

}

long long phi[MAX];

void computeTotient(int n) { // Computes phi or Euler Phi 1 to n

for (int i=1; i<=n; i++) // Initialize

phi[i] = i;

for (int p=2; p<=n; p++) { // Computation

if (phi[p] == p) { // If phi is not computed

phi[p] = p-1; // p is prime and phi(prime) = prime-1;

for (int i = 2\*p; i<=n; i += p) { // Sieve like implementation

phi[i] = (phi[i]/p) \* (p-1); // Add contribution of p to its multiple i by multiplying with (1 - 1/p)

} } } }

quwsarohi@gmail.com Section: Mathematics Page : 4

**// Prime Probability**

**// Algorithm : Miller-Rabin primality test**

**// This function can be used as power or mod power**

int powMod(int x, unsigned int y, int p) { **// If pow(x, y) needed, change lines according to the comments**

int res = 1;

x = x % p; // Remove this line

while (y > 0)

if (y & 1)

res = (res\*x) % p; // res = res \* x;

y = y>>1;

x = (x\*x) % p; // x = x \* x;

}

return res;

}

// This function is called for all k trials. It returns false if n is composite and returns false if n is probably prime.

// d is an odd number such that d\*2<sup>r</sup> = n-1 for some r >= 1

bool miillerTest(int d, int n) {

int a = 2 + rand() % (n – 4); // Pick a random number in [2..n-2] . Corner cases make sure that n > 4

int x = powMod(a, d, n); // Compute a^d % n

if (x == 1 || x == n-1)

return true;

while (d != n-1) { // Keep squaring x while one of the following doesn't happen

x = (x \* x) % n; // (i) d does not reach n-1

d \*= 2; // (ii) (x^2) % n is not 1

if (x == 1) return false; // (iii) (x^2) % n is not n-1

if (x == n-1) return true; }

return false; // Return composite

}

**// Note : Use k = 10 to avoid WA**

bool isPrime(int n, int k) { // It returns false if n is composite and returns true if n is probably prime. k is an input

if (n <= 1 || n == 4) return false; // parameter that determines accuracy level. Higher value of k indicates more accuracy.

if (n <= 3) return true; // Corner cases

int d = n – 1; // Find r such that n = 2^d \* r + 1 for some r >= 1

while (d % 2 == 0)

d /= 2;

for (int i = 0; i < k; i++) // Iterate given nber of 'k' times

if (miillerTest(d, n) == false)

return false;

return true;

}

main() {…….

if(isPrime(3, 10))

cout << “This number is prime” << endl;

………}

quwsarohi@gmail.com Section: Mathematics Page : 5

**// Pascle’s Triangle**

long long p[55][54];

void buildPascle() { //Building Pascle of 50 rows where p[pascle\_line][no\_of\_element] has every element values

p[0][0] = 1; // Base Case

p[1][0] = p[1][1] = 1;

for(int i = 2; i <= 50; i++)

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

if(j == 0 || j == i)

p[i][j] = 1;

else

p[i][j] = p[i-1][j-1] + p[i-1][j];

}

/\* Uncomment this if you want to see the full triangle

for(int i = 0; i <= 20; i++) {

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

printf("%lld ", p[i][j]);

printf("\n");

} \*/

return;

}

**// Horner Polynomial Equation Solver O(n log n)**

**// Naive Approach Complexity: O(n^2),**

// Evaluate value of 2x3 - 6x2 + 2x – 1= 0 for x = 3

// Input: co\_efficient[] = {2, -6, 2, -1}, x = 3

// Output: 5 Algorithm Calculation : ((((2) x – 6) x + 2) x - 1)

int co\_efficient[1000]; // Contains the co-efficients

long long horner(long long x, long long n) { // Critical case : Check if number of co-efficient is equal to

long long ans = co\_efficient[0]; // (max power of x) + 1

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

ans = ans\*x + co\_efficient[i]; }

return ans;

}

**// Extended Euclid**

int x y, d;

void extendedEuclid(int a, int b) {

if(b == 0) {x = 1; y = 0; d = a; return;}

extendedEuclid(b, a%b);

int x1 = y;

int y1 = x - (a/b) \* y;

x = x1;

y = y1;

}

quwsarohi@gmail.com Section: Mathematics Page : 6

**// Linear Diophantine for solving equation**

float ansX, ansY; // Contains answer of x and y respectively

void linear\_diophantine(int a, int b, int c) { // Solving linear Diophantine equations in two variables

extendedEuclid(a, b); // ax + by = c

int g = c / \_\_gcd(a, b); // x = x0 + b \* n where n is an integer

float x0 = x\*g, y0 = y\*g; // y = y0 - a \* n where n is an integer

float low\_n = - x0 / (b/d), hi\_n = y0 / (a/d);

low\_n = ceil(low\_n), hi\_n = floor(hi\_n); // If low\_n != hi\_n, then there exists

ansX = x0 + b \* low\_n; // More than one solution for low\_n <= n <= hi\_n

ansY = y0 - a \* low\_n; // Only getting the first solution

}

**// Some important Functions**

int mod(int a, int b) { // Actual mod is (x % m) biggest multiple of m which is less than x

return ((a%b) + b) % b; // -15 mod 60 = 45 (works like clock)

} // (a + b) % m = ((a % m) + (b % m)) % m (a \* b) % m = (( a % m) \* (b % m)) % m

int gcd(int a, int b) {

while (b) {

int tmp = a%b;

a = b; b = tmp; }

return a;

}

int lcm(int a, int b) {

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

}

int mod\_inverse(int a, int n) { // Computes b such that ab = 1 (mod n), returns -1 on failure

int x, y;

int g = extendedEuclid(a, n); // Use extendedEuclid function

if (g > 1) return -1;

return mod(x, n);

}