**Fast Exponentiation**

Implementation

int exp\_mod(int a, int b, int m) {

a %= m;

b %= (m - 1); // only if m is prime

int r = 1;

while (b) {

if (b & 1) r = r \* a % m; // if long long, use 1ll

a = a \* a % m;

b >>= 1; // if long long, use 1ll

}

return r;

}

Details

1. Time complexity
   1. O(log b) per exponentiation

**Base conversion**

Implementation (Any base to base 10)

stoll("101101", 0, base); // substitute the original base

Implementation (Base 10 to any base)

string convert(ll value, int base) {

string o = “”;

do {

char d = value % base;

if (d < 10) d += ‘0’;

else d = d + ‘A’ - 10;

o.pb(d);

value /= base;

} while (value > 0);

reverse(o.begin(), o.end());

return o;

}

**GCD and LCM**

Implementation (gcd)

\_\_gcd(a, b);

Implementation (lcm)

a / \_\_gcd(a, b) \* b;

Details

1. Time complexity
   1. O(log min(a, b)) query

**Find divisors**

vector<int> divisors(int n) {

vector<int> v;

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

if (n % i == 0) {

if (n / i == i) v.pb(i);

else v.pb(i), v.pb(n / i);

}

}

return v;

}

Details

1. Time complexity
   1. O(sqrt N) query

**Prime factorisation (without sieve)**

vector<int> primeFactor(int n) {

vector<int> v;

for (; !(n & 1); n >>= 1) v.pb(2);

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

for (; n % i == 0; n /= i) v.pb(i);

if (n > 2) v.pb(2);

return v;

}

Details

1. Time complexity
   1. O(sqrt N) query

**Prime factorisation (with sieve)**

int spf[1000005]; // spf[i] is the smallest prime factor of i

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

spf[i] = ((i & 1) || i == 2 ? i : 2);

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

if (spf[i] == i) { // i is prime

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

if (spf[j] == j) spf[j] = i;

vector<int> primeFactor(int n) {

vector<int> v;

while (n != 1) {

v.pb(spf[n]);

n /= spf[n];

}

return v;

}

Details

1. Time complexity
   1. O(N log log N) initialise
   2. O(log N) query

**Binomial coefficients**

Formula: n choose k =

Modulo M:

=

For small n, calculate factorials of n and k -> O(N)

For small k and **prime** M, use this code (it works because the terms in the numerator and denominator cancel out) -> O(K)

int N, K, R = 1, T = 1;

const int M = 1000000007;

int exp\_mod(int a, int b) {

int r = 1ll;

while (b) {

if (b & 1ll) r = r \* a % M;

a = a \* a % M;

b >>= 1ll;

}

return r;

}

main() {

ios::sync\_with\_stdio(0);

cin.tie(0);

cin >> N >> K;

for (int i = N; i >= N - K + 1; i--) R = R \* i % M;

for (int i = K; i >= 1; i--) T = T \* i % M;

cout << R \* exp\_mod(T, M - 2) % M << '\n';

}

For small and **prime** m, use this code -> O(M logM N) (Credits <https://cp-algorithms.com/algebra/factorial-modulo.html>)

int inverse(int n) { return exp\_mod(n, MOD - 2, MOD); }

int factmod(int n, int p) { // n! modulo p

int res = 1;

while (n > 1) {

res = (res \* ((n / p) & 1 ? p - 1 : 1)) % p;

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

res = (res \* i) % p;

n /= p;

}

return res % p;

}

int nck(int n, int k) { // call this function

return factmod(n, MOD) \* inverse(factmod(k, MOD)) % MOD \* inverse(factmod(n - k, MOD)) % MOD;

}