```
ll phi(ll n) {
  ll phiN = n;
  for(int i = 2; i * i <= n; i++) {
    if(n % i == 0) {
      phiN = phiN * (i - 1) / i; // for unique prime
      while (n % i == 0) n /= i;
    }
  }
  if(n > 1) phiN = phiN * (n - 1) / n;
  return phiN;
}
```

Find Combination(nCr): => O(r*log(n))

Ex: 5C2 = 10, 13C5= 1287;

```
void nCr( ll n, ll r) {
  ll p= 1, k=1, m;
  if (n – r < r)  r = n - r;
  if (r != 0) {
    while(r) {
      p*=n, k*=r;
      m=__gcd(p, k);
      p/=m, k/=m;
      n--, r--;
    }
  }
  else p=1;
  cout << p << endl;
}
```

Find Permutation (nPr): => O(n)

Ex: 5P2= 20, 6P3= 120;

```
ll fact(ll n) {
    if(n <= 1) return 1;
    return n * fact(n - 1);
}
ll nPr(ll n, ll r) {
    return fact(n) / fact(n - r);
}
// nCr and nPr using Modulo
const int Max = 2e5 + 5, mod = 998244353;
ll fact[Max], factInv[Max];
void build_fact() {
  fact[0] = 1;
  for(int i = 1; i < Max; i++) {
    fact[i] = 1LL * fact[i - 1] * i % mod;
  }
  factInv[Max - 1] = Pow(fact[Max - 1], mod - 2);
  for(int i = Max - 2; i >= 0; i--) {
    factInv[i] = 1LL * factInv[i + 1] * (i + 1) %
mod;
  }
  return;
}
int nCr_mod(int n, int r) {
  if(n < r or n < 0 or r < 0) return 0;
  return 1LL * fact[n] * factInv[r] % mod *
factInv[n - r] % mod;
}
int nPr_mod(int n, int r) // nPr = nCr * r!
{
  if(n < r or n < 0 or r < 0) return 0;
  return (1LL * nCr_mod(n, r) * fact[r]) % mod;
}
```

Principle of Inclusion and Exclution:

```
void solve()
{
  ll n, m;
  cin >> n >> m;
  vector<int> v(m);
  for (int i = 0; i < m; i++)
  {
    cin >> v[i];
    if (v[i] == 1)
    {
      cout << 0 << endl;
      return;
    }
  }
  long long ans = 0;
  for (int i = 1; i < (1 << m); i++) // loop from 1 to
2^m
  {
    vector<int> subset;
    int cnt = 0;
    for (int j = 0; j < m; j++) // loop through
binary representation of number(1 to 2^n)
    {
      if (i & (1 << j)) // checking ith bit is set(1) or
not
      {
        subset.push_back(v[j]);
        cnt++;
      }
    }
    int NumOfDiv, lcm = 1;
    for (auto it : subset) lcm = lcm * it / (gcd(lcm,
it));
    NumOfDiv = n / lcm;
   if (cnt & 1) // principle of inclusion and
exclution(A U B U C = n(A) + n(B) + n(C)-n(AUB)-
n(AUC)-
    n(BUC)+n(AUBUC));
      ans += NumOfDiv;
    else ans -= NumOfDiv;
  }
  cout << n - ans << endl;
}
```

// Modula Inverse using Extended Euclid (it does not matter mod is prime or not)

```
#define x first
#define y second
```