```cpp
#include "../bits/stdc++.h"
/**
 * DFT(Discrete Fourier Transform, 離散フーリエ変換)
 * 関数f の離散フーリエ変換は以下であらわされる(i : 虚数単位)
 * F(k) = sum_{j=0}^{N-1}{f(j)*exp(-2*pi*i*j*k/N)} (k = 0,1,...,N-1)
 *
 * IDFT(Inverse Discrete Fourier Transform, 逆離散フーリエ変換)
 * F に対する逆離散フーリエ変換は以下であらわされる
 * f(j) = sum_{k=0}^{N-1}{F(k)*exp(2*pi*i*j*k/N)} (j = 0,1,...,N-1)
 *
 * 畳み込み演算
 * (f*g)(k) = sum_j{f(j)g(k-j)} (k = 0,1,...,N-1)
 *
 * DFT の畳み込み定理
 * N点のfのDFTを F_N[f] とする.
 * F_N[f*g] = F_N[f]×F_N[g] (×: ベクトルごとの積)
 *
 * 以下では Cooley-Tukey のアルゴリズムを用いる
 */
// https://github.com/Suikaba/procon-lib/blob/master/math/FFT.hpp
// verified: https://atc001.contest.atcoder.jp/submissions/4269359
const double pi = std::acos(-1.0);

using type = std::complex<double>;

std::vector<type> fft(std::vector<type> v, bool inv = false)
{
    int const n = v.size();
    double theta = 2 * pi / n * (inv ? -1 : 1);
    for (int m = n; m >= 2; m >>= 1)
    {
        int mh = m >> 1;
        type theta_i(0, theta);
        for (int i = 0; i < mh; i++)
        {
            type w = std::exp((type)i * theta_i);
            for (int j = i; j < n; j += m)
            {
                type x = v[j] - v[j + mh];
                v[j] = v[j] + v[j + mh];
                v[j + mh] = w * x;
            }
        }
        theta *= 2;
    }
    int i = 0;
    for (int j = 1; j < n - 1; j++)
    {
        for (int k = n >> 1; k > (i ^= k); k >>= 1)
            ;
        if (j < i)
            std::swap(v[i], v[j]);
    }
    if (inv)
    {
        for (auto &x : v)
            x /= n;
    }
    return v;
}

std::vector<type> convolution(std::vector<type> x, std::vector<type> y)
{
    int sz = 1;
    int t = x.size() + y.size() - 1;
    while (sz < t)
    {
        sz <<= 1;
    }
    x.resize(sz);
    y.resize(sz);
    x = fft(std::move(x));
    y = fft(std::move(y));
    for (int i = 0; i < sz; i++)
    {
        x[i] *= y[i];
    }
    x = fft(std::move(x), true);
    return x;
}

```