FFT

```
#include <bits/stdc++.h>
using namespace std;
const int D = 18;
const double PI = acos(-1);

struct Complex {
    double x, y;
    Complex() {x = 0, y = 0;}
    Complex(double _x) {x = _x, y = 0;}
    Complex(double _x, double _y) {x = _x, y = _y;}
    Complex inv();
};
Complex operator + (Complex a, Complex b) { return {a.x + b.x, a.y + b.y}; }
Complex operator - (Complex a, Complex b) { return {a.x - b.x, a.y - b.y}; }
Complex operator * (Complex a, Complex b) { return {a.x * b.x - a.y * b.y, a.x * b.y +
a.y * b.x}; }
Complex operator * (Complex a, double b) { return {a.x * b, a.y * b}; }
Complex operator - (Complex a) { return a * -1; }
Complex Complex::inv() {
    return Complex(x, -y) * (1 / (x * x + y * y));
}

namespace Poly {
    typedef vector <Complex> poly;
    vector <Complex> gpower[D];
    Complex fftf[1 << D | 10];
    int rev[1 << D | 10];
    int len(const poly &x) {return x.size();}
    void init() {
        for (int i = 0; i < D; i++) {
            gpower[i].resize(2 << i);
            for (int j = 0; j < 2 << i; j++) {
                gpower[i][j] = Complex(cos(PI / (1 << i) * j), sin(PI / (1 << i) * j));
            }
        }
    }
    void get_rev(int l) {
        static int lstl = -1;
        if (l == lstl) return;
        lstl = l;
        for (int i = 1; i < 1 << l; i++) rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << l -
1);
    }
    void NTT(Complex *a, int n, int type) {
        for (int i = 0; i < n; i++) if (rev[i] < i) swap(a[i], a[rev[i]]);
        for (int p = 1, d = 0; p < n; p <<= 1, d++) {
            for (int s = 0; s < n; s += p << 1) {
                Complex *w = gpower[d].data();
                for (int i = s; i < s + p; i++) {
                    Complex h1 = *w++ * a[i + p];
                    a[i + p] = a[i] - h1;
                    a[i] = a[i] + h1;
                }
            }
        }
        if (type == -1) {
            for (int i = 0; i < n; i++) a[i] = a[i] * (1.0 / n);
            reverse(a + 1, a + n);
        }
    }
    void NTT(poly &a, int l, int type) {
        get_rev(l);
        for (int i = 0; i < 1 << l; i++) fftf[i] = a[i];
        NTT(fftf, 1 << l, type);
        for (int i = 0; i < 1 << l; i++) a[i] = fftf[i];
    }
    poly operator * (poly a, poly b) {
        int n = len(a), m = len(b);
        int l = 0;
        for (; 1 << l < n + m - 1; l++);
        a.resize(1 << l), b.resize(1 << l);
        NTT(a, l, 1), NTT(b, l, 1);
        for (int i = 0; i < 1 << l; i++) a[i] = a[i] * b[i];
        NTT(a, l, -1);
        a.resize(n + m - 1);
        return a;
    }
}
using Poly :: poly;
using Poly :: operator *;
int main() {
    Poly :: init(); // 必写
    int n, m;
    scanf("%d%d", &n, &m);
    Poly :: poly x, y;
    x.resize(n), y.resize(m);
    for (auto &i : x) cin >> i.x;
    for (auto &i : y) cin >> i.x;
    x = x * y;
    for (auto i : x) cout << i.x << ' ';
    cout << endl;
}
```