#include <iostream>

#include <cmath>

#include <vector>

**using** **namespace** std;

**typedef** **long** **long** **int** entero;

**double** PI = acos(0) \* 2;

**struct** complex{

**double** a, b;

complex(**double** \_a = 0.0, **double** \_b = 0.0) : a(\_a), b(\_b) {}

**const** complex **operator** + (**const** complex& otro) **const**{

**return** complex(a + otro.a, b + otro.b);

}

**const** complex **operator** - (**const** complex& otro) **const**{

**return** complex(a - otro.a, b - otro.b);

}

**const** complex **operator** \* (**const** complex& otro) **const**{

**return** complex(a \* otro.a - b \* otro.b, a \* otro.b + b \* otro.a);

}

};

**struct** FFT{

vector<complex> datos;

vector<complex> raices;

vector<**int**> bit\_rev;

**int** n, p;

**void** init(**int** potencia){

p = potencia;

n = (1 << p);

bit\_rev.resize(n);

datos.resize(n);

raices.resize(n + 1);

*//permutacion de los bits*

**int** pos\_bit\_mayor = -1, pos\_bit\_prender = p;

bit\_rev[0] = 0;

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

**if**((i & (-i)) == i){ *//i es una pot de 2*

pos\_bit\_mayor++;

pos\_bit\_prender--;

}

bit\_rev[i] = bit\_rev[i ^ (1 << pos\_bit\_mayor)];

bit\_rev[i] |= (1 << pos\_bit\_prender);

}

*//precalcula raices complejas*

raices[0] = complex(1, 0);

complex fact = complex(cos(2 \* PI / n), sin(2 \* PI / n));

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

raices[i] = raices[i - 1] \* fact;

}

}

**void** permutacion\_bits(){

vector<complex> aux(n);

**for**(**int** i = 0; i < n; i++){

aux[i] = datos[bit\_rev[i]];

}

aux.swap(datos);

}

**void** transforma(**bool** inversa = false){

permutacion\_bits();

vector<complex> aux = datos;

**int** paso\_raices = n / 2;

**for**(**int** k = 1; k < n; k <<= 1){

aux.swap(datos);

**int** paso = 2 \* k;

**for**(**int** ini = 0; ini < n; ini += paso){

**for**(**int** i = ini; i < ini + k; i++){

complex t = raices[(i - ini) \* paso\_raices] \* aux[i + k];

datos[i] = aux[i] + t;

datos[i + k] = aux[i] - t;

}

}

paso\_raices >>= 1;

}

**if**(inversa){

aux[0] = datos[0];

aux[0].a /= n;

aux[0].b /= n;

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

aux[i] = datos[n - i];

aux[i].a /= n;

aux[i].b /= n;

}

aux.swap(datos);

}

}

};

*//dados dos vectores de "entero" de tam N, regresa un vector de "entero" de tam 2 \* N - 1*

vector<entero> convolucion(vector<entero>& A, vector<entero>& B){

**int** N = A.size();

**int** tam\_res = 2 \* N - 1;

**int** maxpow = 1, potencia = 0;

**while**(maxpow < tam\_res){

maxpow <<= 1;

potencia++;

}

FFT pA, pB;

pA.init(potencia);

pB.init(potencia);

**for**(**int** i = 0; i < N; i++){

pA.datos[i] = complex(A[i], 0);

pB.datos[i] = complex(B[i], 0);

}

**for**(**int** i = N; i < maxpow; i++){

pA.datos[i] = complex(0, 0);

pB.datos[i] = complex(0, 0);

}

pA.transforma();

pB.transforma();

**for**(**int** i = 0; i < maxpow; i++){

pA.datos[i] = pA.datos[i] \* pB.datos[i];

}

pA.transforma(true);

vector<entero> resp(tam\_res);

**for**(**int** i = 0; i < tam\_res; i++){

resp[i] = (entero) round(pA.datos[i].a);

}

**return** resp;

}

**int** main(){

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

cin.tie(0);

**int** N;

cin >> N;

vector<entero> A(N), B(N);

**for**(**int** i = 0; i < N; i++){

cin >> A[i];

}

**for**(**int** i = 0; i < N; i++){

cin >> B[i];

}

vector<entero> resp = convolucion(A, B);

**for**(**int** i = 0; i < 2 \* N - 1; i++){

cout << resp[i] << " ";

} cout << endl;

**return** 0;

}