

## FFT递归

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 4 * 1e6 + 10;
inline int read() {
    char c = getchar(); int x = 0, f = 1;
    while (c < '0' || c > '9') {if (c == '-') f = -1; c = getchar();}
    while (c >= '0' && c <= '9') {x = x * 10 + c - '0'; c = getchar();}
    return x * f;
}
const double Pi = acos(-1.0);
struct CP {
    double x, y;
    CP (double xx = 0, double yy = 0) {x = xx, y = yy;}
} a[MAXN], b[MAXN];
CP operator + (CP a, CP b) { return CP(a.x + b.x, a.y + b.y);}
CP operator - (CP a, CP b) { return CP(a.x - b.x, a.y - b.y);}
CP operator * (CP a, CP b) { return CP(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x);}//不懂的看复数的运算那部分
void fast_fast_tle(int limit, CP *a, int type) {
    if (limit == 1) return ; //只有一个常数项
    CP a1[limit >> 1], a2[limit >> 1];
    for (int i = 0; i <= limit; i += 2) a1[i >> 1] = a[i], a2[i >> 1] = a[i + 1];
    fast_fast_tle(limit >> 1, a1, type);
    fast_fast_tle(limit >> 1, a2, type);
    CP wn = CP(cos(2.0 * Pi / limit), type * sin(2.0 * Pi / limit)), w = CP(1, 0);
    for (int i = 0; i < (limit >> 1); i++, w = w * wn){ //这里的w相当于公式中的k
        a[i] = a1[i] + w * a2[i];
        a[i + (limit >> 1)] = a1[i] - w * a2[i];
    }
}
int main() {
    int N = read(), M = read();
    for (int i = 0; i <= N; i++) a[i].x = read();
    for (int i = 0; i <= M; i++) b[i].x = read();
    int limit = 1; while (limit <= N + M) limit <= limit;
    fast_fast_tle(limit, a, 1);
    fast_fast_tle(limit, b, 1);
    for (int i = 0; i <= limit; i++)
        a[i] = a[i] * b[i];
    fast_fast_tle(limit, a, -1);
    for (int i = 0; i <= N + M; i++) printf("%d ", (int)(a[i].x / limit + 0.5));
    //按照我们推倒的公式，这里还要除以n
    return 0;
}
```

