Fast Fourier Transform (FFT) Implementation

FFT Implementation

```
const int root = binpow(3, 119);
const int root_1 = binpow(root, mod - 2);
const int root_pw = 1 << 23;

void fft(vector<int>& a, bool invert) {
    int n = a.size();
    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;
        if (i < j)
            swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <<= 1) {
        int wlen = invert ? root_1 : root;
        for (int i = len; i < root_pw; i <<= 1)
            wlen = (int)(1LL * wlen * wlen % mod);

        for (int i = 0; i < n; i += len) {
            int w = 1;
            for (int j = 0; j < len / 2; j++) {
                int u = a[i + j], v = (int)(1LL * a[i + j + len / 2] * w % mod);
                a[i + j] = u + v < mod ? u + v : u + v - mod;
                a[i + j + len / 2] = u - v >= 0 ? u - v : u - v + mod;
                w = (int)(1LL * w * wlen % mod);
            }
        }
    }
    if (invert) {
        int n_1 = binpow(n, mod - 2);
        for (int& x : a)
            x = (int)(1LL * x * n_1 % mod);
    }
}
```

Polynomial Multiplication

```
vector<int> multiply(vector<int>& a, vector<int>& b) {
    vector<int> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] = (1LL * fa[i] * fb[i]) % mod;
    fft(fa, true);

    vector<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = fa[i];
    result.resize(a.size() + b.size() - 1);
    return result;
}
```