Numerical (1)

FFT

Description: Applies the discrete Fourier transform to a sequence of numbers modulo MOD.
Time: $\mathcal{O}(n \log n)$

```
int rev[N], root[N];

void init(int n) {
    static int last_init = -1;
    if (n == last_init) return;
    last_init = n;
    for (int i = 1; i < n; ++i) {
        rev[i] = (rev[i >> 1] >> 1) | (i & 1) * (n >> 1);
    }
    const int root_n = binpow(ROOT, (MOD - 1) / n);
    int cur = 1;
    for (int i = 0, cur = 1; i < n / 2; ++i) {
        root[i + n / 2] = cur;
        cur = mul(cur, root_n);
    }
    for (int i = n / 2 - 1; i >= 0; --i) {
        root[i] = root[i << 1];
    }
}

void dft(int* f, int n, bool inverse = false) {
    init(n);
    for (int i = 0; i < n; ++i) {
        if (i < rev[i]) swap(f[i], f[rev[i]]);
    }
    for (int k = 1; k < n; k <<= 1)
        for (int i = 0; i < n; i += (k << 1))
            for (int j = 0; j < k; ++j) {
                int z = mul(f[i + j + k], root[j + k]);
                f[i + j + k] = sub(f[i + j], z);
                f[i + j] = add(f[i + j], z);
            }
    if (inverse) {
        reverse(f + 1, f + n);
        const int inv_n = inv(n);
        for (int i = 0; i < n; ++i) f[i] = mul(f[i], inv_n);
    }
}
```

Berlekamp-Massey

Description: Returns the polynomial of a recurrent sequence of order n from the first $2n$ terms.
Usage: `berlekamp_massey({0, 1, 1, 3, 5, 11}) // {1, -1, -2}`
Time: $\mathcal{O}(n^2)$

```
vector<int> berlekamp_massey(vector<int> s) {
    int n = sz(s), L = 0, m = 0;
    vector<int> c(n), b(n), t;
    c[0] = b[0] = 1;
    int eval = 1;
    for (int i = 0; i < n; ++i) {
        m++;
        int delta = 0;
        for (int j = 0; j <= L; ++j) {
            delta = add(delta, mul(c[j], s[i - j]));
        }
        if (delta == 0) continue;
        t = c;
        int coef = mul(delta, inv(eval));
        for (int j = m; j < n; ++j) {
            c[j] = sub(c[j], mul(coef, b[j - m]));
        }
        if (2 * L > i) continue;
        L = i + 1 - L, m = 0, b = t, eval = delta;
    }
    c.resize(L + 1);
    return c;
}
```

Miscellaneous (2)

Integrate

Description: Function integration over an interval using Simpson's rule. The error is proportional to h^4.

```
double integrate(double a, double b, auto&& f, int n = 1000) {
    double h = (b - a) / 2 / n, rs = f(a) + f(b);
    for (int i = 1; i < n * 2; ++i) {
        rs += f(a + i * h) * (i & 1 ? 4 : 2);
    }
    return rs * h / 3;
}
```

Fractional binary search

Description: Finds the smallest fraction $p/q \in [0,1]$ s.t. $f(p/q)$ is true and $p, q \leq N$.
Time: $\mathcal{O}(\log N)$

```
struct frac { ll p, q; };

frac fracBS(auto&& f, ll N) {
    bool dir = true, A = true, B = true;
    frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search (0, N]
    if (f(lo)) return lo;
    assert(f(hi));
    while (A || B) {
        ll adv = 0, step = 1;
        for (int si = 0; step; (step *= 2) >>= si) {
            adv += step;
            frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q};
            if (abs(mid.p) > N || mid.q > N || dir == !f(mid)) {
                adv -= step; si = 2;
            }
        }
        hi.p += lo.p * adv;
        hi.q += lo.q * adv;
        dir = !dir;
        swap(lo, hi);
        A = B; B = !!adv;
    }
    return dir ? hi : lo;
}
```