MTHSC 985 Fall 2001 Clemson University
Fast Fourier Transforms and
sparse linear systems over finite fields Instructor: Shuhong Gao
Lecture 3, September 6,11,13 Scribe: Jeff Farr

3. Fast Fourier Transforms

Recall the interpolation problem.

Problem 3.1 (Interpolation)**.** *Given n pairs $(a_i, b_i) \in \mathbb{F}^2$, where $a_1, \ldots a_n$ are distinct, find a polynomial $f \in \mathbb{F}[x]$ of degree less than n such that $f(a_i) = b_i$, $1 \le i \le n$.*

The inverse of the interpolation problem is that of evaluation of polynomials.

Problem 3.2 (Evaluation)**.** *Given n points $a_1, \ldots, a_n \in \mathbb{F}$ and a polynomial $f \in \mathbb{F}[x]$ of degree less than m, compute the values $f(a_1), \ldots, f(a_n)$ as fast as possible.*

For fixed $a_1, \ldots, a_n \in \mathbb{F}$, the transform from $f \in \mathbb{F}[x]$ to its values $f(a_1), \ldots, f(a_n)$ is called the *discrete Fourier transform*, denoted by $\mathrm{DFT}(f)$, i.e.,

$$\mathrm{DFT}(f) = (f(a_1), \ldots, f(a_n)).$$

Here and hereafer we shall identify a polynomial $f = f_0 + f_1 x + \cdots + f_{n-1}x^{n-1} \in \mathbb{F}[x]$ of degree $< n$ with its coefficient vector $(f_0, f_1, \ldots, f_{n-1})$. Hence, for any fixed elements $a_1, \ldots, a_n \in \mathbb{F}$, DFT is a bijection from $\mathbb{F}^n$ to $\mathbb{F}^n$. The inverse transform (i.e. interpolation) is called the *inverse discrete Fourier transform*, denoted by iDFT or DFT^{-1}.

The interpolation problem can be solved by Garner's algorithm from last lecture. For the evaluation problem, there is a simple method, called *Horner's Rule*.

Horner's Rule. Consider the small example in which $f = f_0 + f_1 x + f_2 x^2 + f_3 x^3$. Then

$$f(a) = ((f_3 \cdot a + f_2) \cdot a + f_1) \cdot a + f_0.$$

The following algorithm applies to an arbitrary polynomial.

Algorithm 1 : Horner's Rule

Input: $f = \sum_{i=0}^{m-1} f_i x^i \in \mathbb{F}[x]$ and $a \in \mathbb{F}$
Output: $f(a)$.

1. $u := f_{m-1}$;
2. **for** $i = m-2$ downto 0 **do** $u := u \cdot a + f_i$;
3. return u.

This algorithm uses at most $(m-1)$ multiplications and $(m-1)$ additions in $\mathbb{F}$. Hence Problem 3.2 can be solved using at most $n \cdot m$ multiplications and $n \cdot m$ additions in $\mathbb{F}$.

Exercise. Find the number of operations used by Algorithm 4 from the last lecture (Using Horner's rule for evaluation).

We shall focus on special cases when the set of points $a_1, \ldots, a_n$ has special structures:

(i) $n = 2^k$ and $a_1, \ldots, a_n$ form a multiplicative group of order n.
(ii) $n = 3^k$ and $a_1, \ldots, a_n$ form a multiplicative group of order n.
(iii) $n = 2^k$ and $a_1, \ldots, a_n$ form an additive subgroup of $\mathbb{F}_{2^e}$ for some e.

In the first two cases, the elements $a_1, \ldots, a_n$ are the roots of $x^n - 1$. For an integer n, we say $\mathbb{F}$ *supports* DFT for $x^n - 1$ if $x^n - 1$ has n distinct roots in $\mathbb{F}$, that is, $\mathbb{F}$ contains all n^{th} roots of unity and the characteristic of $\mathbb{F}$ is relatively prime to n. In the following we shall discuss fast algorithms for computing DFT, and any of these *algorithms* is called a *fast Fourier transform* (FFT). An FFT algorithm usually uses some divide-and-conquer technique, that is, reducing a problem of large size to several similar problems of smaller sizes, and apply the procedure recursively to the latter. For convenience, we shall use $\mathrm{DFT}(f, n)$ to indicate the length of the input/output length n.

3.1. **Binary FFT.** First consider (i). Suppose for a moment $n = 2t$ where $t \geq 1$ is an arbitrary integer. Then

$$x^n - 1 = x^{2t} - 1 = (x^t + 1)(x^t - 1).$$

Assume that $\mathbb{F}$ supports DFT for $x^n - 1$, hence $\mathbb{F}$ must have odd characteristic. Let $\omega_1, \ldots, \omega_t \in \mathbb{F}$ be the t distinct roots of $x^t - 1$ (in any order). Then

$$x^t - 1 = \prod_{i=1}^{t} (x - \omega_i). \tag{22}$$

Let $\omega \in \mathbb{F}$ be any root of $x^t + 1$. Then

$$x^t + 1 = \prod_{i=1}^{t} (x - \omega \cdot \omega_i). \tag{23}$$

The reason is that $w^t = -1$ and

$$x^t + 1 = x^t - \omega^t = \omega^t \cdot \left(\left(\frac{x}{\omega} \right)^t - 1 \right) = \omega^t \cdot \prod_{i=1}^{t} \left(\frac{x}{\omega} - \omega_i \right) = \prod_{i=1}^{t} (x - \omega \cdot \omega_i).$$

This means that all the roots of $x^{2n} - 1$ are

$$\omega_1, \ \ldots, \ \omega_t, \ \omega \cdot \omega_1, \ \ldots, \ \omega \cdot \omega_t.$$

Let $f = \sum_{i=0}^{n-1} f_i x^i \in \mathbb{F}[x]$. We want to compute

$$f(\omega_1), \ldots, f(\omega_t), f(\omega \cdot \omega_1), \ldots, f(\omega \cdot \omega_t). \tag{24}$$

Write

$$f = \sum_{i=0}^{t-1} f_i x^i + x^t \cdot \sum_{i=0}^{t-1} f_{t+i} x^i = g + x^t \cdot h = (g, h),$$

where g denotes the lower half of f, and h the higher half. Then

$$f \equiv g + h \pmod{x^t - 1}, \qquad f \equiv g - h \pmod{x^t + 1}.$$

Now $g + h$ has degree less than t, and has the same values as f at ω_i, $1 \le i \le t$. Similarly, $g - h$ has the same value as f at $\omega \cdot \omega_i$, $1 \le i \le t$. To see the values of $g - h$, let $g - h = \sum_{j=0}^{t-1} u_j x^j = u$. Then

$$u(\omega \cdot \omega_i) = \sum_{j=0}^{t-1} (u_j \cdot \omega^j)\omega_i^j, \quad 1 \le i \le t.$$

Define

$$\mathrm{wt}(u, \omega) = \sum_{j=0}^{t-1} (u_j \cdot \omega^j)x^j = (u_0 \cdot 1, u_1 \cdot \omega, \ldots, u_i\omega^i, \ldots, u_{t-1} \cdot \omega^{t-1}),$$

which is called the *weighted polynomial* of u at ω. Then we have

$$\mathrm{DFT}(f, 2t) = (\mathrm{DFT}(g_1, t), \mathrm{DFT}(h_1, t)) \tag{25}$$

where

$$g_1 = g + h, \quad h_1 = \mathrm{wt}(g - h, \omega), \tag{26}$$

and g and h denote the lower and higher halves of f, respectively. So a problem of size $2t$ is reduced to two similar problems of size t each. When t is even, one can apply this reduction again to each of the smaller problems. In the special case when $n = 2^k$, $k = \log_2 n$ reductions will reduce the problem to n problems of size 1 each, hence obtaining the values in (24).

We illustrate by an example. Consider $\mathbb{F} = \mathbb{F}_{17}$. Note that $w = 3$ is a primitive 16^{th} root of unity. So we can perform FFT for polynomials of degree < 16 over $\mathbb{F}_{17}$. The whole process of FFT is illustrated by the diagram in Figure 2.

Certainly, the diagram in Figure 2 works over any field of odd characteristic that contains a primitive 16th root of unity, or over any ring that has characteristic not divisible by 2 and that contains a primitive 16th root of unity.

To see the total numbers of operations used, note that each of the $k = \log_2 n$ reductions needs at most $2t = n$ additions and $t = n/2$ multiplications in $\mathbb{F}$. Hence the next theorem follows immediately.

Theorem 3.1. *Let n be a power of 2. Assume that $\mathbb{F}$ supports DFT for $x^n - 1$. Then DFT(f, n) over $\mathbb{F}$ can be computed using at most $n \log_2 n$ additions and $\frac{1}{2} n \log_2 n$ multiplications in $\mathbb{F}$.*

We summarize the algorithm as follows. For any array $f = (f_0, f_1, \ldots, f_{n-1})$ and integers $j \ge i \ge 0$, we make the convention that $f[i \ldots j] = (f_i, f_{i+1}, \ldots, f_j)$.

Algorithm 2: Binary FFT (recursive)

Input: $n = 2^k$, $\omega \in \mathbb{F}$ of order n, and $f = (f_0, f_1, \ldots, f_{n-1}) \in \mathbb{F}^n$.

Output: $\mathrm{DFT}(f, n, \omega)$, the values of f at ω^i, $0 \le i \le n - 1$.

1. If $n = 1$ then return f.
2. $t := n/2$, $g := f[0 \ldots t - 1]$, and $h := f[t \ldots n - 1]$.
3. Compute

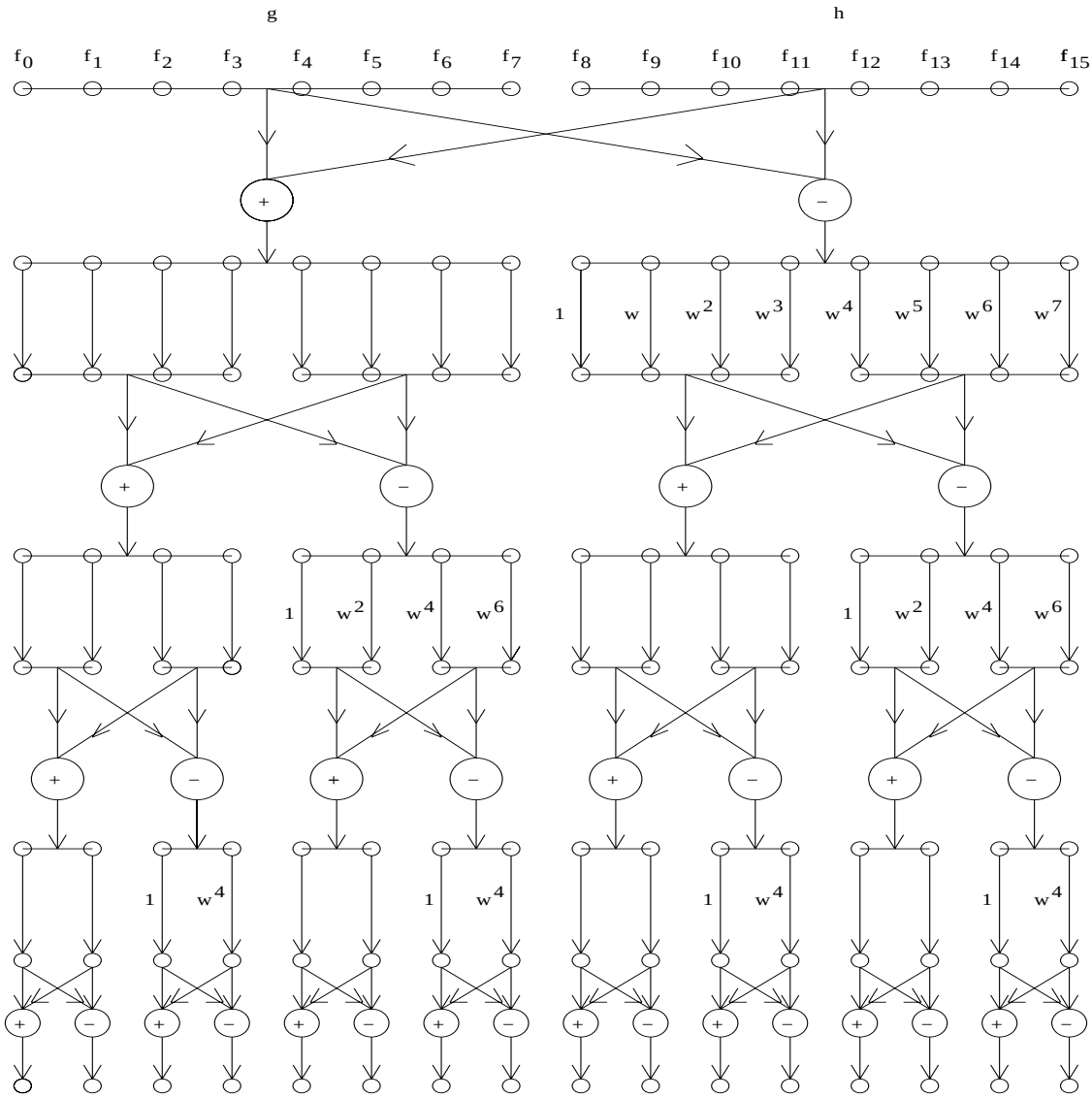

FIGURE 2. FFT for $n = 16$

$g_1 := g + h$,
$h_1 := g - h$, and $h_1 := \mathrm{wt}(h_1, \omega)$.

4. Return $(\mathrm{DFT}(g_1, t, \omega^2), \mathrm{DFT}(h_1, t, \omega^2))$.

We can also present the algorithm without using recursion as follows.

Algorithm 3: Binary FFT (nonrecursive)

Input: $n = 2^k$ $(k \geq 1)$, $\omega \in \mathbb{F}$ of order n, and $f = (f_0, f_1, \ldots, f_{n-1}) \in \mathbb{F}^n$.

Output: the values of f at ω^i, $0 \leq i \leq n-1$.

1. $T := f$, $m := n$, $s := 1$, $a = \omega$. (At each reduction level i below, T is the current array to be updated, m is the current block length for DFT and s is the number of blocks, so $ms = n$ always holds.)
2. **for** i from $k-1$ downto 0 **do**
   2.1. $t := m/2$.
   2.2. **for** j from 0 to $s-1$ **do**

$$\begin{aligned} g &:= T[jm \ldots jm+t-1], \\ T[jm \ldots jm+t-1] &:= g + T[jm+t \ldots jm+m-1], \\ T[jm+t \ldots jm+m-1] &:= g - T[jm+t \ldots jm+m-1], \\ T[jm+t \ldots jm+m-1] &:= \mathrm{wt}(T[jm+t \ldots jm+m-1], a). \end{aligned}$$

   2.3. $m := m/2$, $s := 2 \cdot s$, $a := a^2$.
3. Return T.

In terms of the diagram in Figure 2, the outer loop i corresponds to the level from top to bottom, and the inner loop j runs horizontally at each level from left to right. The steps in the inner loop correspond to the computation in (26).

Remark. We have been vague about the order of values output by the above FFT algorithms. It must be stressed that they are not in the (natural?) order

$$f(\omega^0),\ f(\omega^1),\ \ldots,\ f(\omega^i),\ \ldots,\ f(\omega^{n-1}). \tag{27}$$

To state the correct order, we need to define a permutation σ on the set $\{0, 1, \ldots, n-1\}$ where $n = 2^k$. For each integer $0 \le i < n$, write i in binary form

$$i = i_0 + i_1 \cdot 2 + \cdots + i_{k-1} \cdot 2^{k-1} = (i_0, i_1, \ldots, i_{k-1})_2.$$

We insist that i has k bits, padding zeros if necessary. Define

$$\sigma(i) = (i_{k-1}, \ldots, i_1, i_0)_2,$$

i.e., $\sigma(i)$ obtained from i by reversing its k bits. For example, if $n = 2^5$ and $i = 11 = (1,1,0,1,0)_2$, then $\sigma(i) = (0,1,0,1,1)_2 = 26$. Also, $\sigma(i) = i$ for $i = 0$, or $n-1$.

Exercise. Show that

$$\mathrm{DFT}(f, n, \omega) = (f(\omega^{\sigma(0)}), f(\omega^{\sigma(1)}), \ldots, f(\omega^{\sigma(i)}), \ldots, f(\omega^{\sigma(n-1)})). \tag{28}$$

In practice, if the order (27) is desirable, one must permute (or scramble) the data by σ at the end of the above algorithms.

In the following, we list several families of primes p for which the field $\mathbb{F}_p$ supports DFT for various length $n = 2^k$. Recall $\mathbb{F}_p$ has a primitive n^{th} root of unity if and only if $n|(p-1)$.

(a) Fermat primes $F_i = 2^{2^i} + 1$ for $i = 0, 1, 2, 3$ or 4. That is,

$$p = 3, 5, 17, 257, 65537.$$

For each of these primes p, $\omega = 3$ is a primitive element in $\mathbb{F}_p$. Fermat conjectured that F_i is prime for all i, but he was completely wrong, since it is now known to be composite for all $5 \le i \le 32$. In fact, the next number $2^{2^5} + 1 = 641 \cdot 6700417$, and complete factorization

of F_i is known for all $i \le 11$. Currently $i = 33$ is the smallest for which the compositeness of F_i is unknown (by August 29, 2001; see http://www.prothsearch.net/fermat.html).

(b) $p = 3 \cdot 2^k + 1$. All values of $k \le 2000$ that give primes of this form are

$$k = 1, 2, 5, 6, 8, 12, 18, 30, 36, 41, 66, 189, 201, 209, 276, 353, 408, 438, 534.$$

(c) $p = 5 \cdot 2^k + 1$. All values of $k \le 2000$ that give primes of this form are

$$k = 1, 3, 13, 15, 25, 39, 55, 75, 85, 127, 1947.$$

(d) $p = 7 \cdot 2^k + 1$. All values of $k \le 2000$ that give primes of this form are

$$k = 2, 4, 6, 14, 20, 26, 26, 50, 52, 92, 120, 174, 180, 190, 290, 320, 390, 432, 616, 830, 1804.$$

(e) $p = 9 \cdot 2^k + 1$. All values of $k \le 2000$ that give primes of this form are

$$\begin{aligned} k \;=\; & 1, 2, 3, 6, 7, 11, 14, 17, 33, 42, 43, 63, 65, 67, 81, 134, 162, 206, \\ & 211, 366, 663, 782, 1305, 1411, 1494. \end{aligned}$$

(f) Mersenne primes: $p = 2^k - 1$. All known Mersenne primes come from

$$\begin{aligned} k \;=\; & 2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281, 3217, \\ & 4253, 4423, 9689, 9941, 11213, 19937, 21701, 23209, 44497, 86243, 110503, \\ & 132049, 216091, 756839, 859433, 1257787, 1398269, 2976221, 3021377, 6972593. \end{aligned}$$

For a Mersenne prime $p = 2^k - 1$,

$$p^2 - 1 = (p+1)(p-1) = 2^{k+1}(2^{k-1} - 1)$$

is divisible by $n = 2^{k+1}$. So one can apply FFT over $\mathbb{F}_{p^2}$. Since $p \equiv 3 \pmod 4$, we know $x^2 + 1$ is irreducible over $\mathbb{F}_p$ and

$$\mathbb{F}_{p^2} = \mathbb{F}_p[i] = \{a + bi : a, b \in \mathbb{F}_p\},$$

where $i^2 = -1$. For explicit construction of elements of order $n = 2^{k+1}$ in $\mathbb{F}_{p^2}$, see the paper

> Ian F. Blake, Shuhong Gao and Ronald C. Mullin, "Explicit factorization of $x^{2^k} + 1$ over F_p with prime $p \equiv 3 \bmod 4$," *Applicable Algebra in Engineering, Communication and Computation* **4** (1993), 89–94.

3.2. **3-adic FFT.** Next consider (ii), in which $n = 3^k$. Momentarily let $n = 3t$ where $t \ge 1$ is an arbitrary integer. Assume $\mathbb{F}$ supports DFT for $x^n - 1$, hence $\mathbb{F}$ must have characteristic not divisible by 3. Let $f \in \mathbb{F}[x]$. We want to compute $f(a)$ for all roots a of $x^n - 1$. Let $\omega \in \mathbb{F}$ be a primitive $(3t)$th root of unity. Then $\omega_0 = \omega^t$ is a primitive third root of unity, hence

$$x^3 - 1 = (x - 1)(x - \omega_0)(x - \omega_0^2),$$

and

$$x^n - 1 = x^{3t} - 1 = (x^t - 1)(x^t - \omega_0)(x^t - \omega_0^2).$$

Suppose

$$x^t - 1 = \prod_{i=1}^{t}(x - \omega_i).$$

Then

$$x^t - \omega_0 = \prod_{i=1}^{t}(x - \omega \cdot \omega_i), \quad x^t - \omega_0^2 = \prod_{i=1}^{t}(x - \omega^2 \cdot \omega_i).$$

The reason for the second equation is that

$$x^t - \omega_0^2 = x^t - \omega^{2t} = \omega^{2t}\left(\left(\frac{x}{\omega^2}\right)^t - 1\right) = \prod_{i=1}^{t}(x - \omega^2 \cdot \omega_i),$$

similarly for the first. Hence, for any polynomial $u \in \mathbb{F}[x]$, $\mathrm{DFT}(u)$ for $x^t - \omega_0$ is the same as $\mathrm{DFT}(u_1)$ for $x^t - 1$ where $u_1 = \mathrm{wt}(u, \omega)$, the weighted polynomial of u at ω, similarly $\mathrm{DFT}(u)$ for $x^t - \omega_0^2$ is equal to $\mathrm{DFT}(u_2)$ for $u_2 = \mathrm{wt}(u, \omega^2)$.

To evaluate f, write

$$f = f_0 + f_1 \cdot x^t + f_2 \cdot x^{2t}, \quad f_i \in \mathbb{F}[x], \quad \deg f_i(x) < t, \qquad i = 0, 1, 2.$$

Then

$$\begin{aligned} f \equiv \quad & f_0 + f_1 + f_2 && (\mathrm{mod}\ x^t - 1), \\ f \equiv \quad & f_0 + \omega_0 f_1 + \omega_0^2 f_2 && (\mathrm{mod}\ x^t - \omega_0), \\ f \equiv \quad & f_0 + \omega_0^2 f_1 + \omega_0 f_2 && (\mathrm{mod}\ x^t - \omega_0^2). \end{aligned}$$

Computing

$$\begin{aligned} g_0 =& f_0 + f_1 + f_2, \\ g_1 =& f_0 + \omega_0 f_1 + \omega_0^2 f_2, && g_1 = \mathrm{wt}(g_1, \omega), \\ g_2 =& f_0 + \omega_0^2 f_1 + \omega_0 f_2, && g_2 = \mathrm{wt}(g_2, \omega^2), \end{aligned}$$

we have

$$\mathrm{DFT}(f, 3t) = (\mathrm{DFT}(g_0, t), \mathrm{DFT}(g_1, t), \mathrm{DFT}(g_2, t)). \tag{29}$$

So a DFT of size $3t$ (for $x^{3t} - 1$) is reduced to three DFTs of size t (for $x^t - 1$). Pictorially, this reduction looks like the following:

When t is divisible by 3, one can apply this reduction again to each of the smaller problems. In the case when $n = 3^k$, the problem of size n can be reduced to n problems of size 1 each in $k = \log_3 n$ steps, hence obtaining the values of f. To see the numbers of operations used in total, we note that each reduction uses $6t = 2n$ additions in $\mathbb{F}$, $4t = \frac{4}{3}n$ multiplications by ω_0, and $2t = \frac{2}{3}n$ multiplications by powers of ω. Since ω_0 is very special and is fixed in each reduction, we may assume that multiplication by ω_0 is equivalent to one addition in $\mathbb{F}$ (e.g., when $\mathbb{F} = \mathbb{F}_{p^2}$ where p is of the form $3^k \cdot \ell - 1$). Hence each reduction uses at most $\frac{10}{3}n$ additions in $\mathbb{F}$ and $\frac{2}{3}n$ multiplications by powers of ω.

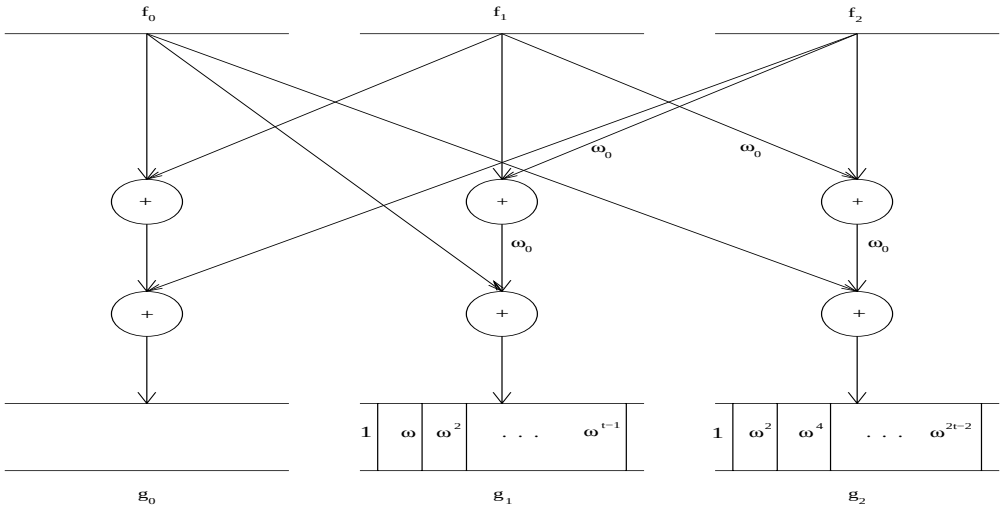

FIGURE 3. 3-adic FFT

Theorem 3.2. *Let n be a power of 3. Assume that $\mathbb{F}$ supports DFT for $x^n - 1$. Then DFT over $\mathbb{F}$ can be computed using at most $\frac{10}{3} n \log_3 n$ additions and $\frac{2}{3} n \log_3 n$ multiplications in $\mathbb{F}$.*

We omit the detailed description of the algorithm, but the reader is encouraged to supply one here. We briefly compare 3-adic FFT with binary FFT. Note that

$$\frac{10}{3} n \log_3 n > 2.1 n \log_2 n, \quad \frac{2}{3} n \log_3 n < 0.421 n \log_2 n.$$

Hence 3-adic FFT uses at least twice as many additions in $\mathbb{F}$ as binary FFT, but slightly fewer multiplications in $\mathbb{F}$.

Exercise. Develop a 4-adic FFT. Let $i \in \mathbb{F}$ has order 4, so $i^2 = -1$. Assume that multiplication by i in $\mathbb{F}$ is equivalent to one addition (e.g. $\mathbb{F} = \mathbb{F}_{p^2}$ where p is a Mersenne prime). Show that your FFT uses at most $\frac{9}{8} n \log_2 n$ additions and $\frac{3}{8} n \log_2 n$ multiplications in $\mathbb{F}$. Hence 4-adic FFT can be more efficient than binary FFT.

Exercise. Let $n = \ell \cdot h$ where $\ell, h > 1$. Suppose $\mathbb{F}$ contains a primitive nth root ω of unity and has characteristic not dividing n. Denote $\omega_\ell = \omega^h$ and $\omega_h = \omega^\ell$, which have orders ℓ and h, respectively. Let $f \in \mathbb{F}[x]$ of degree $< n$. Write f as

$$f = \sum_{i=0}^{\ell-1} \sum_{j=0}^{h-1} f_{ij} \, x^{\ell j + i}.$$

For $0 \le k \le n-1$, write k as $k = u + hv$ where $0 \le u \le h-1$ and $0 \le v \le \ell - 1$. Verify that

$$f(\omega^k) = \sum_{i=0}^{\ell-1} \left(\sum_{j=0}^{h-1} f_{ij} \omega_h^{ju} \right) \cdot \omega^{iu} \cdot \omega_\ell^{iv}.$$

Use this equation to compute $\mathrm{DFT}(f, \omega)$ using DFTs of lengths h and ℓ.

3.2.1. *Fast Interpolation.* We have seen that interpolation is nothing but the inverse of DFT. Fast algorithms for interpolation can be obtained by using the inverse reductions discussed above. In terms of the diagrams, one essentially just needs to reverse the arrows and use ω^{-1} in place of ω. We examine more carefully at the binary FFT, and leave the other FFTs to the reader.

The reduction in (26) from (g, h) to (g_1, h_1) is

$$g_1 := g + h, \quad h_1 := g - h, \quad h_1 := \mathrm{wt}(h_1, \omega).$$

The reverse reduction is then

$$h := \mathrm{wt}(h_1, \omega^{-1}), \quad g := (g_1 + h)/2 \quad h := (g_1 - h)/2.$$

Here the extra part is the division by 2. One may omit this division at each reduction step, but divide by $2^k = n$ at the end of the algorithm. Then the algorithm looks exactly the same, except now from smaller blocks to large ones, and the number of operations are still the same (plus n divisions by n). Algorithm 3 is now easily modified to compute the inverse FFT. The map σ is the same as defined in (28).

Algorithm 4: Interpolation: inverse FFT (binary)

Input: $n = 2^k$ $(k \geq 1)$, $\zeta = \omega^{-1} \in \mathbb{F}$ of order n, and $b = (b_0, b_1, \ldots, f_{n-1}) \in \mathbb{F}^n$.

Output: $\mathrm{iDFT}(b) = f$ such that $f(\omega^{\sigma(i)}) = b_i$, $0 \leq i \leq n-1$.

1. $T := b$, $m := 1$, $s := n$. (At each reduction level i below, T is the current array to be updated, m is the current block length for DFT and s is the number of blocks, so $ms = n$ always holds.)
2. **for** i from 0 to $k-1$ **do**
   2.1. $s := s/2, \quad t := m, \quad m := 2 \cdot m, \quad a := \zeta^{2^{k-1-i}}$.
   2.2. **for** j from 0 to $s-1$ **do**

$$\begin{aligned} T[jm + t \ldots jm + m - 1] &:= \mathrm{wt}(T[jm + t \ldots jm + m - 1], a), \\ g &:= T[jm \ldots jm + t - 1], \\ T[jm \ldots jm + t - 1] &:= g + T[jm + t \ldots jm + m - 1], \\ T[jm + t \ldots jm + m - 1] &:= g - T[jm + t \ldots jm + m - 1]. \end{aligned}$$

3. Return T/n.

There is another elegant method for interpolation. Suppose that

$$(b_0, b_1, \ldots, b_{n-1}) = \mathrm{DFT}((f_0, f_1, \ldots, f_{n-1}), \omega, \sigma),$$

where σ indicates that the scramble σ of data is performed after the FFT. Then we have

$$f(\omega^i) = b_i, \quad 0 \leq i \leq n-1,$$

or equivalently

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^{2\cdot 2} & \cdots & \omega^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{n-1} & \omega^{(n-1)\cdot 2} & \cdots & \omega^{(n-1)\cdot(n-1)} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{pmatrix}.$$

The inverse of the above matrix is known, in fact,

$$\begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{n-1} \end{pmatrix} = \frac{1}{n} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-2\cdot 2} & \cdots & \omega^{-2\cdot(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-(n-1)\cdot 2} & \cdots & \omega^{-(n-1)\cdot(n-1)} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{pmatrix}.$$

Hence

$$(f_0, f_1, \ldots, f_{n-1}) = \mathrm{DFT}((b_0, b_1, \ldots, b_{n-1}), \omega^{-1}, \sigma)/n.$$

Therefore, interpolation is nothing more than a DFT with a data scrambling!