Fast Fourier Transform

Last Updated: September 30th, 2023

1 Introduction

Like Strassen's algorithm, the Fast Fourier Transform (FFT) is considered one of the more suprising and interesting known divide-and-conquer algorithms. It finds important use in the field of signal and image processing but is perhaps best understood as a means for efficiently multiplying two polynomials which we present in this lecture.

2 Review of Complex Numbers

Definition 2.1. A **complex number** is a number of the form $a + bi$, where $a, b \in \mathcal{R}$ are real numbers, and $i = \sqrt{-1}$. The **conjugate** of a complex number $a + bi$, denoted, $\overline{a + bi}$ is the complex number $a - bi$.

Definition 2.2. Let $a + bi$ and $c + di$ be complex numbers. Then the following are the defined operations on complex numbers.

Addition $(a + bi) + (c + di) = (a + c) + (b + d)i$

Subtraction $(a + bi) - (c + di) = (a - c) + (b - d)i$

Multiplication $(a + bi) \cdot (c + di) = (ac - bd) + (ad + bc)i$

Division $(a + bi)/(c + di) = \frac{ac+bd}{c^2+d^2} + \frac{bc-ad}{c^2+d^2}i$

The **modulus** or **length** of complex number $c = a + bi$, denoted $|c|$, is defined as

$$|c| = c \cdot \overline{c} = \sqrt{a^2 + b^2}.$$

With this definition we may rewrite division as

$$c_1/c_2 = \frac{c_1 \cdot \overline{c_2}}{|c_2|^2},$$

where $c_2 \neq 0$.

Proposition 2.3. The following are some identities for complex numbers.

Conjugation When viewed as a function that maps complex number c to $\bar{c}$, conjugation may be viewed as an automorphism over the field of complex numbers:

$$\overline{c_1 + c_2} = \overline{c_1} + \overline{c_2} \text{ and } \overline{c_1 c_2} = \overline{c_1} \cdot \overline{c_2}.$$

Euler's Identity $e^{i\theta} = \cos\theta + i\sin\theta$

$e^{2n\pi i} = 1$ for all integers n.

2.1 Roots of Unity

For each $j = 0, \ldots, n-1$, $e^{\frac{2\pi ij}{n}}$ is a **complex nth root of unity**, meaning that

$$e^{(\frac{2\pi ij}{n})^n} = e^{2\pi ij} = \cos(2\pi j) + i\sin(2\pi j) = 1.$$

Example 2.4. Determine the a) complex 4th roots of unity, and b) complex 6th roots of unity.

Solution.

The next proposition shows that $e^{\frac{2\pi ij}{n}}$, $j = 0, \dots, n-1$, are the only unique powers of $e^{\frac{2\pi i}{n}}$.

Proposition 2.5. If integers j and k satisfy $j \equiv k \bmod n$, then

$$e^{\frac{2\pi ij}{n}} = e^{\frac{2\pi ik}{n}}.$$

Proof of Proposition. Assume $j \equiv k \bmod n$. Then $k = nq + j$, for some integer q. Then

$$e^{\frac{2\pi ik}{n}} = e^{\frac{2\pi i(j+nq)}{n}} = e^{\frac{2\pi ij}{n}} e^{\frac{2\pi inq}{n}} = e^{\frac{2\pi ij}{n}} e^{2\pi iq} = e^{\frac{2\pi ij}{n}} \cdot 1 = e^{\frac{2\pi ij}{n}}.$$

Proposition 2.5 allows us to define the abelian group whose members are the nth roots of unity, with multiplication serving as the group addition. In other words,

$$e^{\frac{2\pi ij}{n}} \cdot e^{\frac{2\pi ik}{n}} = e^{\frac{2\pi i(j+k)}{n}}.$$

Moreover, the addition is associative since multiplying two roots of unity is identical to adding the two integers j and k, and integer addition is associative. Also, 1 is the additive identity, and the (additive) inverse of $e^{\frac{2\pi ij}{n}}$ is $e^{\frac{2\pi i(n-j)}{n}}$. Another way of writing the inverse of $e^{\frac{2\pi ij}{n}}$ is $e^{\frac{-2\pi ij}{n}}$. This is valid, since $n - i \equiv -i \bmod n$.

For simplicity, we let ω_n^j denote the j th root of unity, and ω_n^{-j} denotes its inverse. In general, for any integer k, ω_n^k is defined as being equal to ω_n^j, where $j \equiv k \bmod n$.

Example 2.6. For the 6th roots of unity, determine the inverse of each group element, and verify that $(a + bi)(a + bi)^{-1} = 1$ through direct multiplication.

Proposition 2.7. The following are some properties of roots of unity.

1. If n is even, then ω_n^j and $-\omega_n^j$ are both roots of unity. In other words, roots of unity come in additive-inverse pairs. Furthermore, if $0 \le j < n/2$, then $\omega_n^{j+n/2} = -\omega_n^j$.
2. If n is even, then the squares of the nth roots of unity yield the $n/2$ roots of unity.

Proof of Proposition.

1. By the sum-of-angle formulas for cosine and sine, we have
$$e^{(\theta+\pi)i} = \cos(\theta+\pi) + i\sin(\theta+\pi) = -\cos\theta - \sin\theta i = -e^{\theta i}.$$
Therefore,
$$-\omega_n^j = e^{(\frac{2\pi ij}{n}+\pi i)} = e^{(\frac{2\pi ij}{n}+\frac{2\pi i(n/2)}{n})} = e^{\frac{2\pi i(j+n/2)}{n}} = \omega_n^{j+(n/2)}$$
which is a root of unity.
2. For $0 \le j < n/2$, we have
$$(\omega_n^j)^2 = \omega_n^{2j} = e^{\frac{2\pi i(2j)}{n}} = e^{\frac{2\pi ij}{n/2}},$$
which is an $n/2$ root of unity. Note also that, for $n/2 \le j < n$, $e^{\frac{2\pi ij}{n}}$ is just the negative of ω_n^j, and thus its square yields the same $n/2$ root of unity as its additive-inverse counterpart.

□

3 Polynomial Multiplication and the Fast Fourier Transform

Given two polynomials

$$A(x) = a_0 + a_1 x + \cdots + a_d x^d$$

and

$$B(x) = b_0 + b_1 x + \cdots + b_d x^d,$$

our goal is to compute the product $C(x) = A(x)B(x)$ where $C(x)$ is a degree-$2d$ polynomial whose k th term c_k, $k = 0, 1, \ldots, d$, is computed as

$$c_k = \sum_{i=0}^{k} a_i b_{k-i}.$$

Thus, using the above formula we see that computing the first $d+1$ coefficients of $C(x)$ requires

$$1 + 2 + 3 + 4 + \cdots + d + (d+1) = \Theta(d^2)$$

steps.

The following algorithm provides an alternative way to compute $C(x)$.

Alternative Polynomial Multiplication Algorithm

Input: Coefficients of polynomials $A(x)$ and $B(x)$.

Output: Coefficients of $C(x) = A(x)B(x)$.

Pick points: $x_0, x_1, \ldots, x_{n-1}$, for some $n \geq 2d+1$.

Evaluate A and B: compute $A(x_0), \ldots, A(x_{n-1})$ and $B(x_0), \ldots, B(x_{n-1})$.

Evaluate C: compute $C(x_0) = A(x_0)B(x_0), \ldots, C(x_{n-1}) = A(x_{n-1})B(x_{n-1})$.

Interpolate: determine the unique coefficients $c_0, c_1, \ldots, c_{2d}$ for which, for all $i = 0, 1, \ldots, n-1$,

$$C(x_i) = c_0 + c_1 x_i + \cdots + c_{2d} x_i^{2d}.$$

Return $c_0, c_1, \ldots, c_{2d}$.

On the surface, it appears that this method will also require $O(d^2)$ steps, since evaluating a d-degree polynomial on some input x_i generally requires $\Theta(d)$ steps via Horner's algorithm. Moreover, interpolation also requires $O(d^2)$ steps since, as we'll see, it involves the inverting a $2d \times 2d$ Vandermonde matrix. However, by choosing to evaluate A and B with the points $1, \omega_n, \omega_n^2, \ldots, \omega_n^{n-1}$ (i.e. the nth roots of unity) and evaluating a polynomial via a divide-and-conquer approach, we can reduce the total number of evaluation and interpolation steps to $O(n \log n)$.

3.1 A Divide and Conquer approach to polynomial evaluation

In what follows we assume that n is a power of two. Consider the polynomial

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}.$$

Then $A(x)$ may be written as

$$A(x) = A_e(x^2) + x A_o(x^2),$$

where $A_e(y)$ and $A_o(y)$ are the polynomials

$$A_e(y) = a_0 + a_2 y + a_4 y^2 + \cdots + a_{n-2} y^{\frac{n-2}{2}},$$

and

$$A_o(y) = a_1 + a_3 y + \cdots + a_{n-1} y^{\frac{n-2}{2}}.$$

Thus, we may evaluate $(n-1)$-degree polynomial $A(x)$ by evaluating two $(\frac{n-2}{2})$-degree polynomials at x^2. In other words, we've taken the problem and divided it into two subproblems, each of which is one-half the size.

Now, for a single evaluation $A(x)$, the above divide-and-conquer method does *not* improve the running time. In fact, recurrence for the number of steps $T(n)$ is

$$T(n) = 2T(n/2) + n,$$

which implies $T(n) = \Theta(n \log n)$ which is *worse* than linear! However, suppose instead the problem is to evaluate n complex points $\pm x_1, \pm x_2, \ldots, \pm x_{\frac{n}{2}}$ consisting of $n/2$ additive-inverse pairs. Then, since $(-x_i)^2 = x_i^2$, we see that the problem may again be divided into two subproblems, each of size $n/2$, and in both cases whose $n/2$ points that require evaluation are $x_1^2, \ldots, x_{\frac{n}{2}}^2$. This works so long as these $n/2$ squares may be represented as $n/4$ additive-inverse pairs. Of course, this would not be possible if these squares were real numbers (since the squares would all be positive), but *is* possible if our n points are equal to the nth roots of unity. Let's check this.

1. By part 1 of Proposition 2.7, since we assume n a power of two, the roots of unity may in fact be partitioned into additive-inverse pairs, with ω_n^i being paired with $\omega_n^{\frac{n}{2}+i}$, for all $i = 0, 1, \ldots, n/2 - 1$.
2. Moreover, by part two of the same proposition, the squares of the nth roots of unity yield precisely the $\frac{n}{2}$-th roots of unity and, since $n/2 \geq 2$ is even, once again these numbers may be partitioned into additive-inverse pairs. Therefore the two subproblems, $(A_e, \{x_1^2, \ldots, x_{\frac{n}{2}}^2\})$ and $(A_o, \{x_1^2, \ldots, x_{\frac{n}{2}}^2\})$ are in fact two (smaller by one half) instances of the original problem.

The above divide-and-conquer algorithm leads us to the following definition.

Definition 3.1. Given complex coefficients $c_0, \dots, c_{n-1}$, let $p(x)$ be the polynomial

$$p(x) = \sum_{k=0}^{n-1} c_k x^k.$$

Then the n**th order discrete Fourier transform** is the function

$$\mathrm{DFT}_n(c_0, \dots, c_{n-1}) = (y_0, \dots, y_{n-1}),$$

where $y_j = p(\omega_n^j)$, $j = 0, \dots, n-1$.

In words the nth order discrete Fourier transform, takes as input the complex coefficients of a degree $n-1$ polynomial p, and returns the n-dimensional vector whose components are the evaluation of p at each of the nth roots of unity. Another way to write $\mathrm{DFT}_n(c_0, \dots, c_{n-1})$ is $\mathrm{DFT}_n(p)$, where p is the polynomial of degree $n-1$ whose coefficients are $c_0, \dots, c_{n-1}$.

Example 3.2. Compute $\mathrm{DFT}_4(0, 1, 2, 3)$.

3.2 Fast Fourier Transform

We may now write our divide-and-conquer algorithm in terms of DFT_n. In what follows we define

$$(u_1, \ldots, u_n) \odot (v_1, \ldots, v_n) = (u_1 v_1, \ldots, u_n v_n),$$

which we call the **scaling of** v **with** u.

Fast Fourier Transform

> Input: polynomial $A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$, where n is a power of two.
>
> Output: $\mathrm{DFT}_n(A)$.
>
> If $n = 1$, then return (a_0).
>
> $Y_0 = \mathrm{DFT}_{\frac{n}{2}}(A_e)$.
>
> $Y_0 = Y_0 \circ Y_0$. //Concatenate vector Y_0 with itself.
>
> $Y_1 = \mathrm{DFT}_{\frac{n}{2}}(A_o)$.
>
> $Y_1 = Y_1 \circ Y_1$. //Concatenate vector Y_1 with itself.
>
> $Y_1 = \vec{\omega_n} \odot Y_1$. //Scale Y_1 with the length-n vector of nth roots of unity.
>
> Return $Y_0 + Y_1$. //Return the vector sum of Y_0 with Y_1.

We see that the running time for FFT is $\Theta(n \log n)$, since its running time satisfies

$$T(n) = 2T(n/2) + n.$$

Thus, we have found a way to evaluate a polynomial at n points using only a log-linear number of steps!

Example 3.3. Compute $\mathrm{DFT}_4(0, 1, 2, 3)$ using the FFT algorithm.

Solution.

4 Solving Interpolation with the Inverse DFT

Returning to the alternative polynomial multiplication algorithm, the FFT algorithm allows us to compute $C(\omega_n^j)$, for each $j = 0, 1, \ldots, n-1$. To finish the algorithm, we must find coefficients $c_0, c_1, \ldots, c_{n-1}$ for which, for each $j = 0, 1, \ldots, n-1$,

$$C(\omega_n^j) = c_0 + c_1\omega_n^j + \cdots + c_{n-1}\omega_n^{j(n-1)}.$$

Furthermore, we can write these n equations in matrix form as follows.

$$\begin{pmatrix} C(\omega_n^0) \\ C(\omega_n^1) \\ \vdots \\ C(\omega_n^{n-1}) \end{pmatrix} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & \cdots & \omega_n^{1(n-1)} \\ \vdots & \vdots & \cdots & \vdots \\ 1 & \omega_n^{n-1} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{pmatrix}$$

Letting F_n denote the $n \times n$ matrix in the above equation, we leave it as an exercise to show that its inverse is

$$F_n^{-1} = \frac{1}{n} \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega_n^{-1} & \cdots & \omega_n^{-1(n-1)} \\ \vdots & \vdots & \cdots & \vdots \\ 1 & \omega_n^{-(n-1)} & \cdots & \omega_n^{-(n-1)(n-1)} \end{pmatrix}.$$

Thus, we may compute the coefficients of $C(x)$ as

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{pmatrix} = \frac{1}{n} \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega_n^{-1} & \cdots & \omega_n^{-1(n-1)} \\ \vdots & \vdots & \cdots & \vdots \\ 1 & \omega_n^{-(n-1)} & \cdots & \omega_n^{-(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} C(\omega_n^0) \\ C(\omega_n^1) \\ \vdots \\ C(\omega_n^{n-1}) \end{pmatrix}.$$

Thus, for all $j = 0, 1, \ldots, n-1$, we have

$$c_j = \frac{1}{n}(C(\omega_n^0) + C(\omega_n^1)\omega_n^{-j} + \cdots + C(\omega_n^{n-1})\omega_n^{-j(n-1)}).$$

Notice that this equation is essentially the evaluation of polynomial

$$\frac{1}{n}(C(\omega_n^0) + C(\omega_n^1)x + \cdots + C(\omega_n^{n-1})x^{n-1})$$

on input $x = \omega_n^{-j}$. This suggests the following definition.

Definition 4.1. Given complex coefficients $y_0, \ldots, y_{n-1}$, let $p(x)$ be the polynomial

$$p(x) = \sum_{k=0}^{n-1} y_k x^k.$$

Then the n**th order inverse discrete Fourier transform** is the function

$$\mathrm{DFT}_n^{-1}(y_0, \ldots, y_{n-1}) = (c_0, \ldots, c_{n-1}),$$

where $c_j = \frac{1}{n} p(\omega_n^{-j})$, $j = 0, \ldots, n-1$.

In words the nth order inverse discrete Fourier transform, takes as input the complex coefficients of a degree $n-1$ polynomial p, and returns the n-dimensional vector whose components are the evaluation of $\frac{1}{n}p(x)$ at each of the inverses of the nth roots of unity.

4.1 The Inverse Fast Fourier Transform

We may provide a similar divide-and-conquer algorithm for computing DFT_n^{-1} which we call the **Inverse Fast Fourier Transform (IFFT)**.

Inverse Fast Fourier Transform

> Input: polynomial $A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$, where n is a power of two.
>
> Output: $\mathrm{DFT}_n^{-1}(A)$.
>
> If $n = 1$, then return (a_0).
>
> $Y_0 = \mathrm{DFT}_{\frac{n}{2}}^{-1}(A_e)$.
>
> $Y_0 = Y_0 \circ Y_0$. //Concatenate vector Y_0 with itself.
>
> $Y_1 = \mathrm{DFT}_{\frac{n}{2}}^{-1}(A_o)$.
>
> $Y_1 = Y_1 \circ Y_1$. //Concatenate vector Y_1 with itself.
>
> $Y_1 = \vec{\omega}_n^{-1} \odot Y_1$. //Scale Y_1 with the respective inverses of the nth roots of unity.
>
> Return $\frac{1}{2}(Y_0 + Y_1)$. //Return the vector sum of Y_0 with Y_1.

Notice that in the final line we must scale the vector by $1/2$. This is because both $\mathrm{DFT}_{\frac{n}{2}}^{-1}(A_e)$ and $\mathrm{DFT}_{\frac{n}{2}}^{-1}(A_o)$ give the polynomial evaluations divided by $n/2$. However, we want both to be divided by n. So we must multiply by $n/2$ to undo the division by $n/2$, and then divide by n, which has the net effect of multiplying by $1/2$.

Example 4.2. Compute $\mathrm{DFT}_4^{-1}(0, 1, -1, 2)$ by a) using the definition of $\mathrm{DFT}_4^{-1}(0, 1, -1, 2)$, and b) using the IFFT algorithm on $\mathrm{DFT}_4^{-1}(0, 1, -1, 2)$.

4.2 Summary

$DFT_n(p)$ The *discrete Fourier transform* that evaluates an $(n-1)$-degree polynomial p at each of the nth roots of unity and returns a vector of these evaluations.

FFT An *algorithm* for computing $DFT_n(p)$ in $O(n \log n)$ steps when n is assumed a power of 2.

$DFT_n^{-1}(p)$ The *inverse discrete Fourier transform* that evaluates an $(n-1)$-degree polynomial p at each multiplicative inverse of each nth root of unity, and returns a vector of these evaluations scaled by $\frac{1}{n}$. Moreover if the coefficients of p are the values $q(\omega_n^0), q(\omega_n^1), \ldots, q(\omega_n^{n-1})$, for some $(n-1)$-degree polynomial q, then $DFT_n^{-1}(p)$ outputs the coefficients of q, meaning that it solves the problem of *polynomial interpolation* with respect to q

IFFT An *algorithm* for computing $DFT_n^{-1}(p)$ in $O(n \log n)$ steps when n is assumed a power of 2.

Exercises

1. Prove that for any two complex numbers c and d, $\overline{cd} = \overline{c}\overline{d}$
2. Determine the complex cube roots of unity.
3. Determine the complex 8th roots of unity.
4. For the 8th roots of unity, determine the inverse of each group element, and verify that $(a + bi)(a + bi)^{-1} = 1$ through direct multiplication.
5. Let $n \geq 1$, $d > 0$, and k be integers. Prove that $\omega_{dn}^{dk} = \omega_n^k$. This is called the **cancellation rule**.
6. Let n be an even positive integer. Prove that the square of each of the nth roots of unity yields the $n/2$ roots of unity. Moreover, each $n/2$ root of unity is associated with two different squares of nth roots of unity.
7. Show that $\omega_n^{n/2} = -1$, for all even $n \geq 2$.
8. For positive integer n and for integer j not divisible by n, prove that

$$\sum_{k=0}^{n-1} \omega_n^{jk} = 0.$$

   Hint: use the geometric series formula

$$\sum_{k=0}^{n-1} a^k = \frac{a^n - 1}{a - 1},$$

   which is valid when a is a complex number.

9. Find the equation of the quadratic polynomial whose graph passes through the points $(2, 13)$, $(-1, 10)$, and $(3, 26)$.
10. Find the equation of the cubic polynomial whose graph passes through the points $(0, -1)$, $(1, 0)$, $(-1, -4)$, and $(2, 5)$.
11. Compute $\text{DFT}_4(1, -1, 2, 4)$ using the definition.
12. Compute $\text{DFT}_4(-1, 3, 4, 10)$ using the definition.
13. Compute $\text{DFT}_4^{-1}(0, 0, -4, 0)$ using the definition.
14. Compute $\text{DFT}_4^{-1}(2, 1 - i, 0, 1 + i)$ using the definition.
15. Show the sequence of polynomials that are evaluated when evaluating $p(x) = x^3 - 3x^2 + 5x - 6$ using Horner's algorithm. Use the algorithm to evaluate $p(-2)$.
16. Show the sequence of polynomials that are evaluated when evaluating $p(x) = 2x^4 - x^3 + 2x^2 + 3x - 5$ using Horner's algorithm. Use the algorithm to evaluate $p(5)$.

17. Use the FFT algorithm to compute $\mathrm{DFT}_4(1, -1, 2, 4)$.
18. Use the FFT algorithm to compute $\mathrm{DFT}_4(-1, 3, 4, 10)$.
19. Compute $\mathrm{DFT}_4^{-1}(0, 0, -4, 0)$ using the definition.
20. Compute $\mathrm{DFT}_4^{-1}(2, 1-i, 0, 1+i)$ using the definition.
21. Use the IFFT algorithm to compute $\mathrm{DFT}_4^{-1}(0, 0, -4, 0)$.
22. Use the IFFT algorithm to compute $\mathrm{DFT}_4^{-1}(2, 1-i, 0, 1+i)$.

Exercise Solutions

1. Let $c = a + bi$, and $d = e + fi$. Then
$$\overline{cd} = \overline{(ae - bf) + i(af + be)} = (ae - bf) - i(af + be).$$
On the other hand,
$$overlinec\overline{d} = (a - bi)(e - fi) = (ae - bf) + i(-af - be) = (ae - bf) - i(af + be),$$
which proves the claim.

2. For $j = 0$,
$$e^{\frac{(2\pi)(0)i}{3}} = 1.$$
For $j = 1$,
$$e^{\frac{2\pi i}{3}} = -1/2 + \frac{\sqrt{3}i}{2}.$$
For $j = 2$,
$$e^{\frac{4\pi i}{3}} = -1/2 - \frac{\sqrt{3}i}{2}.$$

3. For $j = 0$,
$$e^{\frac{(2\pi)(0)i}{3}} = 1.$$
For $j = 1$,
$$e^{\frac{\pi i}{4}} = \frac{\sqrt{2}}{2} + \frac{\sqrt{2}i}{2}.$$
For $j = 2$,
$$e^{\frac{\pi i}{2}} = i.$$
For $j = 3$,
$$e^{\frac{3\pi i}{4}} = \frac{-\sqrt{2}}{2} + \frac{\sqrt{2}i}{2}.$$
For $j = 4$,
$$e^{\pi i} = -1.$$

For $j = 5$,
$$e^{\frac{5\pi i}{4}} = \frac{-\sqrt{2}}{2} + \frac{-\sqrt{2}i}{2}.$$
For $j = 6$,
$$e^{\frac{3\pi i}{2}} = -i.$$
For $j = 7$,
$$e^{\frac{7\pi i}{4}} = \frac{\sqrt{2}}{2} + \frac{-\sqrt{2}i}{2}.$$

4. For example, $\omega_8^2 = i$ while $\omega_8^{-2} = \omega_8^6 = -i$, since $(i)(-i) = 1$. Similarly, $\omega_8^4 = -1$ while $\omega_8^{-4} = \omega_8^4 = -1$, since $(-1)(-1) = 1$.

5. By definition,
$$\omega_{dn}^{dk} = e^{\frac{2\pi i dk}{dn}} = e^{\frac{2\pi i k}{n}} = \omega_n^k.$$

6. For $j = 0, \ldots, n-1$,
$$(\omega_n^j)^2 = \omega_n^j \omega_n^j = \omega_n^{2j} = \omega_{n/2}^j,$$
where the last equality is due to the cancellation rule from Exercise 5. Thus the square of an nth root of unity is indeed an $n/2$ root of unity. Moreover, notice that j ranges from 0 to $n-1$. By definition, when j ranges from 0 to $n/2-1$, we obtain each $n/2$ root of unity. Then, due to the cyclic nature of the roots unity, when j ranges from $n/2$ to $n-1$, we once again obtain each $n/2$ root of unity. Therefore, each $n/2$ root of unity $\omega_{n/2}^j$ is the square of exactly two different nth-roots of unity, namely $(\omega_{n/2}^j)^2$ and $(\omega_{n/2}^{j+n/2})^2$.

7. We have, for even $n \geq 2$,
$$\omega_n^{n/2} = e^{(2\pi i/n)n/2} = e^{\pi i} = \cos\pi + i \sin\pi = -1.$$

8. Using the geometric series formula
$$\sum_{k=0}^{n-1} a^k = \frac{a^n - 1}{a - 1},$$
we have
$$\sum_{k=0}^{n-1} (\omega_n^j)^k = \sum_{k=0}^{n-1} \omega_n^{jk} =$$
$$\frac{\omega_n^{jn} - 1}{\omega_n^j - 1} = \frac{\omega_1^j - 1}{\omega_n^j - 1} = \frac{1 - 1}{\omega_n^j - 1} = 0,$$
where the first equality is due to the cancellation rule, and the 2nd to last equality is due to the fact that $\omega_1^1 = 1$. Notice also that the denominator is not equal to zero, since we assumed j is not divisible by n; i.e. $j \not\equiv 0 \bmod n$.

9. We desire a polynomial of the form $c_0 + c_1 x + c_2 x^2$. The three points imply the following system of equations.
$$c_0 + 2c_1 + 4c_2 = 13$$
$$c_0 - c_1 + c_2 = 10$$
$$c_0 + 3c_1 + 9c_2 = 26$$
Solving this system gives the polynomial $5 - 2x + 3x^2$.

10. We desire a polynomial of the form $c_0 + c_1x + c_2x^2 + c_3x^3$. The four points imply the following system of equations.

$$c_0 = -1$$

$$c_0 + c_1 + c_2 + c_3 = 0$$

$$c_0 - c_1 + c_2 - c_3 = -4$$

$$c_0 + 2c_1 + 4c_2 + 8c_3 = 5$$

Solving this system gives the polynomial $-1 + x - x^2 + x^3$.

11. $\mathrm{DFT}_4(1, -1, 2, 4) = (6, -1 - 5i, 0, -1 + 5i)$

12. $\mathrm{DFT}_4(-1, 3, 4, 10) = (16, -5 - 7i, -10, -5 + 7i)$

13. $\mathrm{DFT}_4^{-1}(0, 0, -4, 0) = (-1, 1, -1, 1)$

14. $\mathrm{DFT}_4^{-1}(2, 1 - i, 0, 1 + i) = (1, 0, 0, 1)$

15. $p_0(x) = 1$, $p_1(x) = xp_0(x) - 3 = x - 3$, $p_2(x) = xp_1(x) + 5 = x^2 - 3x + 5$, $p_3(x) = xp_2(x) - 6 = x^3 - 3x^2 + 5x - 6$. $p_0(-2) = 1$, $p_1(-2) = -2(1) - 3 = -5$, $p_2(-2) = -2(-5) + 5 = 15$, $p_3(-2) = -2(15) - 6 = -36$.

16. $p_0(x) = 2$, $p_1(x) = xp_0(x) - 1 = 2x - 1$, $p_2(x) = xp_1(x) + 2 = 2x^2 - x + 2$, $p_3(x) = xp_2(x) + 3 = 2x^3 - x^2 + 2x + 3$, $p_4(x) = xp_3(x) - 5 = 2x^4 - x^3 + 2x^2 + 3x - 5$. $p_0(5) = 2$, $p_1(5) = 5(2) - 1 = 9$, $p_2(5) = 5(9) + 2 = 47$, $p_3(5) = 5(47) + 3 = 238$, $p_4(5) = 5(238) - 5 = 1185$.

17. $p_0(x) = 1 + 2x$, $\mathrm{DFT}_2(1 + 2x) = (3, -1)$. Thus,

$$Y_0 = (3, -1, 3, -1).$$

Also, $p_1(x) = -1 + 4x$, and $\mathrm{DFT}_2(-1 + 4x) = (3, -5)$. Thus,

$$Y_1 = (3, -5, 3, -5).$$

Furthermore, $Y_{1j} \leftarrow \omega_4^j Y_{1j}$ gives

$$Y_1 = (3, -5i, -3, 5i).$$

Finally, $\mathrm{DFT}_4(1, -1, 2, 4) = Y_0 + Y_1 = (6, -1 - 5i, 0, -1 + 5i)$.

18. $p_0(x) = -1 + 4x$, $\mathrm{DFT}_2(-1 + 4x) = (3, -5)$. Thus,

$$Y_0 = (3, -5, 3, -5).$$

Also, $p_1(x) = 3 + 10x$, and $\mathrm{DFT}_2(3 + 10x) = (13, -7)$. Thus,

$$Y_1 = (13, -7, 13, -7).$$

Furthermore, $Y_{1j} \leftarrow \omega_4^j Y_{1j}$ gives

$$Y_1 = (13, -7i, -13, 7i).$$

Finally, $\mathrm{DFT}_4(-1, 3, 4, 10) = Y_0 + Y_1 = (16, -5 - 7i, -10, -5 + 7i)$.

19. Input $(0,0,-4,0)$ corresponds with polynomial $p(x)=-4x^2$. Moreover,

$$p(\omega_4^{(-1)(0)})=p(1)=-4,$$
$$p(\omega_4^{-1})=p(-i)=4,$$
$$p(\omega_4^{-2})=p(-1)=-4,$$

and

$$p(\omega_4^{-3})=p(i)=4.$$

Thus,

$$\mathrm{DFT}_4^{-1}(0,0,-4,0)=\frac{1}{4}(-4,4,-4,4)=(-1,1,-1,1),$$

and so $\mathrm{DFT}_4^{-1}(0,0,-4,0)=(-1,1,-1,1)$, which corresponds with polynomial $-1+x-x^2+x^3$.

20. Input $(2,1-i,0,1+i)$ corresponds with polynomial $p(x)=2+(1-i)x+(1+i)x^3$. Moreover,

$$p(\omega_4^{(-1)(0)})=p(1)=4,$$
$$p(\omega_4^{-1})=p(-i)=0,$$
$$p(\omega_4^{-2})=p(-1)=0,$$

and

$$p(\omega_4^{-3})=p(i)=4.$$

Thus, $\mathrm{DFT}_4^{-1}(2,1-i,0,1+i)=(1,0,0,1)$,, which corresponds with polynomial $1+x^3$.

21. $p_0(x)=-4x$, $\mathrm{DFT}_2^{-1}(-4x)=\frac{1}{2}(-4,4)=(-2,2)$. Thus,

$$C_0=(-2,2,-2,2).$$

Also, $p_1(x)=0$, and $\mathrm{DFT}_2^{-1}(0)=(0,0)$. Thus,

$$C_1=(0,0,0,0).$$

Furthermore, $C_{1j}\leftarrow\omega_4^{-j}C_{1j}$ gives

$$C_1=(0,0,0,0).$$

Finally, $\mathrm{DFT}_4^{-1}(0,0,-4,0)=\frac{1}{2}(C_0+C_1)=\frac{1}{2}(-2,2,-2,2)=(-1,1,-1,1)$, which corresponds with polynomial $-1+x-x^2+x^3$.

22. $p_0(x)=2$, $\mathrm{DFT}_2^{-1}(2)=\frac{1}{2}(2,2)=(1,1)$. Thus,

$$C_0=(1,1,1,1).$$

Also, $p_1(x)=(1-i)+(1+i)x$, and $\mathrm{DFT}_2^{-1}((1-i)+(1+i)x)=\frac{1}{2}(2,-2i)=(1,-i)$. Thus,

$$C_1=(1,-i,1,-i).$$

Furthermore, $C_{1j}\leftarrow\omega_4^{-j}C_{1j}$ gives

$$C_1=(1,-1,-1,1).$$

Finally, $\mathrm{DFT}_4^{-1}(2,1-i,0,1+i)=\frac{1}{2}(C_0+C_1)=(1,0,0,1)$, which corresponds with polynomial $1+x^3$.