15-451/651: Algorithms November 14, 2019

Lecture Notes: Fast Fourier Transform

Lecturer: Gary Miller *Scribe:*

[1]

1 Introduction-Motivation

A polynomial of the variable x over an algebraic field $\mathcal{F}$ is defined as:

$$P(x) = \sum_{j=0}^{n-1} p_j x^j. \tag{1}$$

The values $p_0, p_1, \ldots, p_n$ are called the **coefficients** of the polynomial. The polynomial A is said to have degree k if its highest non-zero coefficient is a_k. Any integer strictly greater than the degree of A is a **degree-bound** of A. The degree of a polynomial of degree-bound n can be any integer between 0 and $n-1$, inclusive.

The most common operations including polynomials are addition and multiplication. Given two polynomials of degree-bound n, $A(x)$:

$$A(x) = \sum_{j=0}^{n-1} a_j x^j, \tag{2}$$

and $B(x)$:

$$B(x) = \sum_{j=0}^{n-1} b_j x^j, \tag{3}$$

their **polynomial addition** is also a degree-bound n polynomial $C(x)$, which is defined as follows:

$$C(x) = \sum_{j=0}^{n-1} c_j x^j, \tag{4}$$

where

$$c_j = a_j + b_j, j = 0, 1, \ldots, n-1. \tag{5}$$

On the other hand, the **polynomial multiplication** of the degree-bound n polynomials $A(x)$ and $B(x)$ is a degree-bound $2n-1$ polynomial $D(x)$, which is defined as:

$$D(x) = \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} a_j b_k x^{j+k} = \sum_{j=0}^{2n-2} d_j x^j \tag{6}$$

where

[1] Originally 15-750 notes by David Witmer and Dimitris Konomis

$$d_j = \sum_{k=0}^{j} a_k b_{j-k}, j = 0, 1, \ldots, 2n-2. \tag{7}$$

2 Two Representations of Polynomials

Polynomials are usually represented in coefficient form or point-value form. We present the two representations and prove that they are equivalent: every polynomial in coefficient form has a unique counterpart in point-value form and vice-versa.

2.1 Coefficient Representation

The **coefficient representation** of the degree-bound n polynomial P, as defined by equation 1 is simply the vector of coefficients $p_0, p_1, \ldots, p_{n-1}$. Now consider the coefficient representation of degree-bound n polynomials A and B, (equations 2 and 3), i.e. the vectors $a = (a_0, a_1, \ldots, a_{n-1})$ and $b = (b_0, b_1, \ldots, b_{n-1})$. Clearly, the coefficient representation of their addition, C (equation 4), i.e. the vector $c = (c_0, c_1, \ldots, c_{n-1})$ can be computed in $\Theta(n)$ time, whereas the coefficient representation of their multiplication, D (equation 6), i.e. the vector $d = (d_0, d_1, \ldots d_{2n-2})$ requires $\Theta(n^2)$ time, as every coefficient in vector a has to be multiplied by every coefficient in vector b.

Remark 2.1. For $x, y \in \mathbb{R}^n$, the convolution, z of x, y, denoted by $z = x \otimes y$ is the vector $z \in \mathbb{R}^{2n-1}$, whose j-th entry is defined as:

$$z_j = \sum_{k=0}^{j} x_k y_{j-k}, j = 0, 1, \ldots 2n-2. \tag{8}$$

2.2 Point-value Representation

The **point-value represesentation** of the degree-bound n polynomial P is a set of n point-value pairs:

$$\{(x_0, P(x_0)), (x_1, P(x_1)), \ldots, , (x_{n-1}, P(x_{n-1}))\}, \tag{9}$$

where all x_i are distinct.

It is clear that a degree-bound n polynomial has infinitely many different point-value representations, as any set of n distinct points can be chosen as a basis for the representation.

2.3 Converting between coefficient and point-value representations

2.3.1 Evaluation: coefficient to point-value

The process of computing the point-value representation of a degree-bound n polynomial $P(x)$ given its coefficient representation, $p \in \mathbb{R}^n$, is known as **evaluation**. This problem is relatively straightforward: we pick n distinct points, $x_0, x_1, \ldots, x_{n-1}$ and for each x_i, $i = 0, 1, \ldots, n-1$, we compute $P(x_i)$. Using Horner's method, we can evaluate the value of an arbitrary degree-bound n polynomial P at an arbitrary given point x in $\Theta(n)$, by making the following crucial observation:

$$P(x) = p_0 + x(p_1 + x(p_2 + \cdots + x(p_{n-2} + x(p_{n-1}))\ldots)). \tag{10}$$

The Horner's algorithm is defined precisely as follows:

Algorithm 1 Horner's algorithm

```
1: function P(p, x)
2:     y = 0
3:     for i = n − 1 downto 0 do
4:         y = p_i + xy
5:     end for
6:     return y
7: end function
```

Therefore, naively computing the point-value representation of a degree-bound n polynomial P takes $\Theta(n^2)$ time.

2.3.2 Interpolation: point-value to coefficient

The process of computing the coefficient representation of a degree-bound n polynomial P given its point-value representation, $\{(x_0, P(x_0)), (x_1, P(x_1)), \ldots, , (x_{n-1}, P(x_{n-1}))\}$, is known as **interpolation**.

Theorem 2.2 (Uniqueness of an interpolating polynomial)**.** *For any set $\{(x_0, y_0), (x_1, y_1), \ldots, , (x_{n-1}, y_{n-1})\}$ of n point-value pairs such that all the x_i values are distinct, there exists a unique polynomial $P(x)$ of degree-bound n such that $y_i = P(x_i)$, for $i = 0, 1, \ldots, n-1$.*

Proof. The n equations $y_i = P(x_i)$, for $i = 0, 1, \ldots, n-1$ can be organized as the following linear system $V_n(x_0, x_1, \ldots, x_{n-1})p = y$:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-2} & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-2} & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-2} & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-2} & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} \tag{11}$$

We know from Linear Algebra, that the above square system has a unique solution $p \in \mathbb{R}^n$ if and only if the matrix V_n is invertible, which happens if and only if its determinant, $|V_n|$ is non-zero. The determinant of V_n is clearly a function of $x_0, x_1, \ldots, x_{n-1}$. Hence let $|V_n| = |V_n(x_0, x_1, \ldots, x_{n-1})|$. For $i = n-2, n-1, \ldots, 1$ we multiply i-th column C_i by x_0 and subtract it from $i+1$-th column C_{i+1}:

$$\begin{aligned} &|V_n(x_0, x_1, \ldots, x_{n-1})| = \\ &= \begin{vmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-2} & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-2} & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-2} & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-2} & x_{n-1}^{n-1} \end{vmatrix} \\ &= \begin{vmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-2} & 0 \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-2} & (x_1 - x_0)x_1^{n-2} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-2} & (x_2 - x_0)x_2^{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-2} & (x_{n-1} - x_0)x_{n-1}^{n-2} \end{vmatrix} \qquad (C_{n-1} = C_{n-1} - x_0 C_{n-2}) \end{aligned}$$

$$
\begin{aligned}
&= \begin{vmatrix} 1 & x_0 & x_0^2 & \cdots & 0 & 0 \\ 1 & x_1 & x_1^2 & \cdots & (x_1-x_0)x_1^{n-3} & (x_1-x_0)x_1^{n-2} \\ 1 & x_2 & x_2^2 & \cdots & (x_2-x_0)x_2^{n-3} & (x_2^-x_0)x_2^{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & (x_{n-1}-x_0)x_{n-1}^{n-3} & x_{n-1}^{n-1}-x_0x_{n-1}^{n-2} \end{vmatrix} && (C_{n-2} = C_{n-2} - x_0C_{n-3}) \\
&= \vdots && (C_i = C_i - x_0C_{i-1}) \\
&= \begin{vmatrix} 1 & x_0 & 0 & \cdots & 0 & 0 \\ 1 & x_1 & (x_1-x_0)x_1 & \cdots & (x_1-x_0)x_1^{n-3} & (x_1-x_0)x_1^{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_{n-1} & (x_{n-1}-x_0)x_{n-1} & \cdots & (x_{n-1}-x_0)x_{n-1}^{n-3} & (x_{n-1}-x_0)x_{n-1}^{n-2} \end{vmatrix} && (C_2 = C_2 - x_0C_2) \\
&= \begin{vmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 1 & (x_1-x_0) & (x_1-x_0)x_1 & \cdots & (x_1-x_0)x_1^{n-3} & (x_1-x_0)x_1^{n-2} \\ 1 & (x_2-x_0) & (x_1-x_0)x_2 & \cdots & (x_2-x_0)x_2^{n-3} & (x_2-x_0)x_2^{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & (x_{n-1}-x_0) & (x_{n-1}-x_0)x_{n-1} & \cdots & (x_{n-1}-x_0)x_{n-1}^{n-3} & (x_{n-1}-x_0)x_{n-1}^{n-2} \end{vmatrix} && (C_1 = C_1 - x_0C_0) \\
&= \begin{vmatrix} (x_1-x_0) & (x_1-x_0)x_1 & \cdots & (x_1-x_0)x_1^{n-3} & (x_1-x_0)x_1^{n-2} \\ (x_2-x_0) & (x_2-x_0)x_1 & \cdots & (x_2-x_0)x_2^{n-3} & (x_2-x_0)x_2^{n-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ (x_{n-1}-x_0) & (x_{n-1}-x_0)x_{n-1} & \cdots & (x_{n-1}-x_0)x_{n-1}^{n-3} & (x_{n-1}-x_0)x_{n-1}^{n-2} \end{vmatrix} && \text{(cofactors of first row)} \\
&= (x_1-x_0)\begin{vmatrix} 1 & x_1 & \cdots & x_1^{n-3} & x_1^{n-2} \\ 1 & (x_2-x_0)x_2 & \cdots & (x_2-x_0)x_2^{n-3} & (x_2-x_0)x_2^{n-2} \\ 1 & \vdots & \vdots & \ddots & \vdots \\ 1 & (x_{n-1}-x_0)x_{n-1} & \cdots & (x_{n-1}-x_0)x_{n-1}^{n-3} & (x_{n-1}-x_0)x_{n-1}^{n-3} \end{vmatrix} && \text{(factor out}(x_1-x_0)\text{)} \\
&= (x_1-x_0)(x_2-x_0)\begin{vmatrix} 1 & x_1 & \cdots & x_1^{n-3} & x_1^{n-2} \\ 1 & x_2 & \cdots & x_2^{n-3} & x_2^{n-2} \\ 1 & \vdots & \vdots & \ddots & \vdots \\ 1 & (x_{n-1}-x_0)x_{n-1} & \cdots & (x_{n-1}-x_0)x_{n-1}^{n-3} & (x_{n-1}-x_0)x_{n-1}^{n-3} \end{vmatrix} && \text{(factor out}(x_2-x_0)\text{)} \\
&\ \vdots \\
&= (x_1-x_0)(x_2-x_0)\cdots(x_{n-1}-x_0)\begin{vmatrix} 1 & x_1 & \cdots & x_1^{n-3} & x_1^{n-2} \\ 1 & x_2 & \cdots & x_2^{n-3} & x_2^{n-2} \\ 1 & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & \cdots & x_{n-1}^{n-3} & x_{n-1}^{n-3} \end{vmatrix} && \text{(factor out}(x_{n-1}-x_0)\text{)} \\
&= (x_1-x_0)(x_2-x_0)\cdots(x_{n-1}-x_0)|V_{n-1}(x_1,x_2,\ldots,x_{n-1})|
\end{aligned}
$$

We have just shown that $|V_n(x_0,x_1,\ldots,x_{n-1})| = (x_1-x_0)(x_2-x_0)\cdots(x_{n-1}-x_0)|V_{n-1}(x_1,x_2,\ldots,x_{n-1})|$. A simple induction argument yields that

$$|V_n(x_0,x_1,\ldots,x_{n-1})| = \prod_{0\le j<k\le n-1}(x_k-x_j) \tag{12}$$

which is non-zero, since the x_i, $i = 0,1,\ldots,n-1$ are distinct.

Therefore, $p = V_n(x_0, x_1, \ldots, x_{n-1})^{-1}y$ is the unique coefficient representation of the degree-bound n polynomial $P(x)$ that satisfies $y_i = P(x_i)$, for $i = 0, 1, \ldots, n-1$. □

Now consider the point-value representations of degree-bound n polynomials A and B, (equations 2 and 3), i.e. the sets

$$\{(x_0, A(x_0)), (x_1, A(x_1)), \ldots, (x_{n-1}, A(x_{n-1}))\} \tag{13}$$

and

$$\{(x_0, B(x_0)), (x_1, B(x_1)), \ldots, (x_{n-1}, B(x_{n-1}))\}. \tag{14}$$

Clearly, the point-value representation of their addition, the degree-bound n polynomial C (equation 4), i.e. the set

$$\{(x_0, C(x_0) = A(x_0)+B(x_0)), (x_1, C(x_1) = A(x_1)+B(x_1)), \ldots, (x_{n-1}, C(x_{n-1}) = A(x_{n-1})+B(x_{n-1}))\}, \tag{15}$$

can be computed in $\Theta(n)$ time.

The point-value representation of their multiplication, D (equation 6), is a little bit trickier. This happens because $degree(D) = degree(A) + degree(B)$ and therefore given that A and B are degree-bound n, it follows that D is of degree-bound $2n$. However, we need $2n$ distinct points to interpolate a polynomial of degree-bound $2n$. We have to therefore extend the point-value representations of A and B so that they consist of $2n$ points. Let us then consider the extended point-value representations of degree-bound n polynomials A and B i.e. the sets

$$\{(x_0, A(x_0)), (x_1, A(x_1)), \ldots, (x_{2n-1}, A(x_{2n-1}))\} \tag{16}$$

and

$$\{(x_0, B(x_0)), (x_1, B(x_1)), \ldots, (x_{2n-1}, B(x_{2n-1}))\}. \tag{17}$$

Clearly, the point-value representation of their multiplication, the degree-bound $2n$ polynomial D (equation 6), i.e. the set

$$\{(x_0, D(x_0) = A(x_0)B(x_0)), (x_1, D(x_1) = A(x_1)B(x_1)), \ldots, (x_{2n-1}, D(x_{2n-1})A(x_{2n-1})B(x_{2n-1}))\} \tag{18}$$

can be computed in $\Theta(n)$ time.

We can see that given a fast technique to obtain the point-value representation of a degree-bound n polynomial P from its coefficient representation and vice versa, we can use it to multiply two degree-bound n polynomials, A and B in time faster than $\Theta(n^2)$. We can achieve this by first evaluating the polynomials at a set of $2n$ distinct points, multiplying them in $\Theta(n)$ and then interpolating the resulting polynomial D to obtain its coefficient representation. As we will see shortly, it is the particular choice of the points at which the polynomials are going to be evaluated that allows the FFT algorithm to perform both evaluation and interpolation in $\Theta(n \log(n))$ time. The overall procedure then will clearly have time complexity $\Theta(n \log(n)) + \Theta(n) + \Theta(n \log(n)) = \Theta(n \log(n))$. These points are the $2n$-th roots of unity, and are discussed next.

3 Complex Roots of Unity

A complex root of unity is a complex number ω such that:

$$\omega^n = 1. \tag{19}$$

There are exactly n complex n-th roots of unity: $e^{\frac{i2k\pi}{n}}$, $k = 0, 1, \ldots, n-1$. This formula is easily interpreted by using the definition of the exponential of a complex number:

$$e^{ix} = \cos(x) + i\sin(x) \tag{20}$$

where $i^2 = -1$.

The following figure illustrates that the n complex n-th roots of unity are equally spaced around the circle of unit radius at the origin of the complex plane. The quantity $\omega_n = e^{\frac{i2\pi}{n}}$ is the principal n-th root of unity and all the complex n-th roots of unity are powers of ω_n.

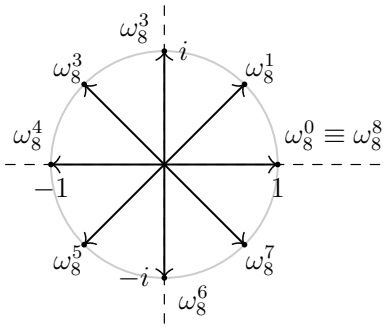

Figure 1: The 8 8-th roots of unity on the complex plane.

Lemma 3.1 (Cancellation Lemma)**.** *For any integers $n \geq 0$, $k \geq 0$, and $d \geq 0$, $\omega_{dn}^{dk} = \omega_n^k$.*

Proof. The lemma follows immediately from the definition of the n-th complex roots of unity:

$$\begin{aligned} \omega_{dn}^{dk} &= (e^{\frac{i2\pi}{dn}})^{dk} \\ &= (e^{\frac{i2\pi}{n}})^{k} \\ &= \omega_n^k \end{aligned}$$

□

Corollary 3.2. *For any even integer $n \geq 0$, $\omega_n^{\frac{n}{2}} = \omega_2 = -1$.*

Proof.

$$\begin{aligned} \omega_n^{\frac{n}{2}} &= \omega_{2\frac{n}{2}}^{1\frac{n}{2}} \\ &= \omega_2^1 && \text{(Lemma 3.1)} \\ &= e^{\frac{i2\pi}{2}} \\ &= e^{i\pi} \end{aligned}$$

$$= cos(\pi) + isin(\pi)$$
$$= -1 + i0 = -1$$

□

Lemma 3.3 (Halving Lemma)**.** *If $n \geq 0$ is even, the squares of the n complex n-th roots of unity are the $n/2$ complex $n/2$-th roots of unity.*

Proof. It suffices to show that if we square all of the complex n-th roots of unity, we will get each $n/2$-th root of unity exactly twice. In particular, we prove that the roots ω_n^k and $\omega_n^{k+\frac{n}{2}}$ have the same square. Indeed,

$$\begin{aligned}
(\omega_n^k)^2 &= \omega_n^{2k} \\
&= \omega_n^{2k}\omega_n^n && (\omega_n^n = 1) \\
&= \omega_n^{2k+n} \\
&= (\omega_n^{k+\frac{n}{2}})^2
\end{aligned}$$

□

We will soon see that the halving lemma is crucial to the divide-and-conquer technique for converting between coefficient and and point-value representations of polynomials, since it ensures that the recursive subproblems are half as large as the original problem.

Lemma 3.4 (Summation Lemma)**.** *For any integer $n \geq 1$ and nonzero integer k not divisible by n:*

$$\sum_{j=0}^{n-1}(\omega_n^k)^j = 0.$$

Proof. The given sum is just a geometric series with a common ratio $r = \omega_n^k$. Therefore, it can be computed precisely as follows:

$$\begin{aligned}
\sum_{j=0}^{n-1}(\omega_n^k)^j &= \frac{(\omega_n^k)^n - 1}{\omega_n^k - 1} \\
&= \frac{(\omega_n^n)^k - 1}{\omega_n^k - 1} \\
&= \frac{(1)^k - 1}{\omega_n^k - 1} \\
&= 0.
\end{aligned}$$

Note that because $k \bmod n \neq 0$, $\omega_n^k \neq 1$ and the division by $\omega_n^k - 1$ is allowed.

□

4 The Fast Fourier Transform

4.1 Evaluation: $y = DFT(p)$

Assume we are given a degree-bound n polynomial P in coefficient form, i.e the vector $p \in \mathbb{C}^n$:

$$p = \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{n-1} \end{bmatrix}, \tag{21}$$

and we want to obtain its point-value representation by evaluating it at the n distinct n-th complex roots of unity $\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$.

The vector $y \in \mathbb{C}^n$, defined as:

$$y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} P(\omega_n^0) \\ P(\omega_n^1) \\ \vdots \\ P(\omega_n^{n-1}) \end{bmatrix} = \begin{bmatrix} \sum_{j=0}^{n-1} p_j(\omega_n^0)^j \\ \sum_{j=0}^{n-1} p_j(\omega_n^1)^j \\ \vdots \\ \sum_{j=0}^{n-1} p_j(\omega_n^{n-1})^j \end{bmatrix}, \tag{22}$$

is called the **Discrete Fourier Transform** of p. The above relationship between y and p is usually abbreviated as $y = DFT_n(p)$.

Expanding equation 22, we get $y = F_n p$, where F_n is defined as:

$$F_n = \begin{bmatrix} (\omega_n^0)^0 & (\omega_n^0)^1 & (\omega_n^0)^2 & \cdots & (\omega_n^0)^{n-2} & (\omega_n^0)^{n-1} \\ (\omega_n^1)^0 & (\omega_n^1)^1 & (\omega_n^1)^2 & \cdots & (\omega_n^1)^{n-2} & (\omega_n^1)^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ (\omega_n^{n/2-1})^0 & (\omega_n^{n/2-1})^1 & (\omega_n^{n/2-1})^2 & \cdots & (\omega_n^{n/2-1})^{n-2} & (\omega_n^{n/2-1})^{n-1} \\ (\omega_n^{n/2})^0 & (\omega_n^{n/2})^1 & (\omega_n^{n/2})^2 & \cdots & (\omega_n^{n/2})^{n-2} & (\omega_n^{n/2})^{n-1} \\ (\omega_n^{n/2+1})^0 & (\omega_n^{n/2+1})^1 & (\omega_n^{n/2+1})^2 & \cdots & (\omega_n^{n/2+1})^{n-2} & (\omega_n^{n/2+1})^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ (\omega_n^{n-1})^0 & (\omega_n^{n-1})^1 & (\omega_n^{n-1})^2 & \cdots & (\omega_n^{n-1})^{n-2} & (\omega_n^{n-1})^{n-1} \end{bmatrix} \tag{23}$$

The Fast Fourier Transform (FFT) is an algorithm that takes advantage of the properties of the complex roots of unity and allows computing $y = DFT_n(p)$ in $\Theta(n \log(n))$ time, as opposed to the standard $\Theta(n^2)$ time algorithm. For the rest of this analysis, we assume that $n = 2^k$ for some $k > 0$.

The FFT is a divide-and-conquer algorithm, which uses the even-indexed and odd-indexed coefficients of P separately to define the 2 new polynomials of degree $n/2$:

$$P_{even}(x) = p_0 + p_2 x + p_4 x^2 + \cdots + p_{n-2} x^{\frac{n}{2}-1} \tag{24}$$

$$P_{odd}(x) = p_1 + p_3 x + p_5 x^2 + \cdots + p_{n-1} x^{\frac{n}{2}-1} \tag{25}$$

It is very easy to see that $\forall x \in \mathbb{C}$, one can compute $P(x)$ combining the values $P_{even}(x)$ and $P_{odd}(x)$:

$$P(x) = P_{even}(x^2) + xP_{odd}(x^2) \tag{26}$$

The problem of evaluating the n-degree P at $\omega_n^0, \omega_n^1, \ldots, \omega_n^k$ is decomposed to

1. evaluating the $n/2$-degree $P_{even}(x)$ and $P_{odd}(x)$ at $(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_n^{n-1})^2$,
2. combining the results using equation 26

From the Halving Lemma, we know that the list of the squares of the n-th complex roots of unity $(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_n^{n-1})^2$ consists exactly of the list of the $n/2$-th complex roots of unity $\omega_{\frac{n}{2}}^0, \omega_{\frac{n}{2}}^1, \ldots, \omega_{\frac{n}{2}}^{n-1}$, but with each value appearing exactly twice.

The FFT algorithm is precisely described by the following pseudocode:

Algorithm 2 FFT algorithm

```
1:  function FFT(p)
2:      n = p.length
3:      if n == 1 then
4:          return p
5:      end if
6:      ω_n = e^(i2π/n)
7:      ω = 1
8:      p_even = (p_0, p_2, ..., p_{n-2})
9:      p_odd = (p_1, p_3, ..., p_{n-1})
10:     y_even = FFT(p_even)
11:     y_odd = FFT(p_odd)
12:     for k = 0 to n/2 − 1 do
13:         y[k] = y_even[k] + ω y_odd[k]
14:         y[k + n/2] = y_even[k] − ω y_odd[k]
15:         ω = ω ω_n
16:     end for
17:     return y
18: end function
```

It is clear from the analysis above that the run-time complexity of the FFT algorithm is described by the recurrence

$$T(n) = 2T(n/2) + \Theta(n) \tag{27a}$$
$$T(1) = 1 \tag{27b}$$

The solution to this very well-known recurrence is $T(n) = \Theta(n \log(n))$.

Another beautiful way to reason about the runtime complexity of the FFT algorithm is the following. Recall that $y = F_n p$:

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n/2-1} \\ y_{n/2} \\ y_{n/2+1} \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} (\omega_n^0)^0 & (\omega_n^0)^1 & (\omega_n^0)^2 & \cdots & (\omega_n^0)^{n-2} & (\omega_n^0)^{n-1} \\ (\omega_n^1)^0 & (\omega_n^1)^1 & (\omega_n^1)^2 & \cdots & (\omega_n^1)^{n-2} & (\omega_n^1)^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ (\omega_n^{n/2-1})^0 & (\omega_n^{n/2-1})^1 & (\omega_n^{n/2-1})^2 & \cdots & (\omega_n^{n/2-1})^{n-2} & (\omega_n^{n/2-1})^{n-1} \\ (\omega_n^{n/2})^0 & (\omega_n^{n/2})^1 & (\omega_n^{n/2})^2 & \cdots & (\omega_n^{n/2})^{n-2} & (\omega_n^{n/2})^{n-1} \\ (\omega_n^{n/2+1})^0 & (\omega_n^{n/2+1})^1 & (\omega_n^{n/2+1})^2 & \cdots & (\omega_n^{n/2+1})^{n-2} & (\omega_n^{n/2+1})^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ (\omega_n^{n-1})^0 & (\omega_n^{n-1})^1 & (\omega_n^{n-1})^2 & \cdots & (\omega_n^{n-1})^{n-2} & (\omega_n^{n-1})^{n-1} \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{n/2-1} \\ p_{n/2} \\ p_{n/2+1} \\ \vdots \\ p_{n-1} \end{bmatrix} \tag{28}$$

At this point we make the following crucial observation: the matrix F_n can be re-arranged so that the columns that correspond to even powers of the roots of unity come before the columns that correspond to odd powers of the roots of unity. We also re-arrange the vector p so that the coefficients that correspond to even powers of the polynomial come before the coefficients that correspond to odd powers of the polynomial. This recursive procedure can be described by a permutation $\pi_n : [n-1] \to [n-1]$, which we have to determine.

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n/2-1} \\ y_{n/2} \\ y_{n/2+1} \\ \vdots \\ y_{n-1} \end{bmatrix} = \left[\begin{array}{ccc|ccc} (\omega_n^0)^{\pi_n(0)} & \cdots & (\omega_n^0)^{\pi_n(n/2-1)} & (\omega_n^0)^{\pi_n(n/2)} & \cdots & (\omega_n^0)^{\pi_n(n-1)} \\ (\omega_n^1)^{\pi_n(0)} & \cdots & (\omega_n^1)^{\pi_n(n/2-1)} & (\omega_n^1)^{\pi_n(n/2)} & \cdots & (\omega_n^1)^{\pi_n(n-1)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ (\omega_n^{n/2-1})^{\pi_n(0)} & \cdots & (\omega_n^{n/2-1})^{\pi_n(n/2-1)} & (\omega_n^{n/2-1})^{\pi_n(n/2)} & \cdots & (\omega_n^{n/2-1})^{\pi_n(n-1)} \\ \hline (\omega_n^{n/2})^{\pi_n(0)} & \cdots & (\omega_n^{n/2})^{\pi_n(n/2-1)} & (\omega_n^{n/2})^{\pi_n(n/2)} & \cdots & (\omega_n^{n/2})^{\pi_n(n-1)} \\ (\omega_n^{n/2+1})^{\pi_n(0)} & \cdots & (\omega_n^{n/2+1})^{\pi_n(n/2-1)} & (\omega_n^{n/2+1})^{\pi_n(n/2)} & \cdots & (\omega_n^{n/2+1})^{\pi_n(n-1)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ (\omega_n^{n-1})^{\pi_n(0)} & \cdots & (\omega_n^{n-1})^{\pi_n(n/2-1)} & (\omega_n^{n-1})^{\pi_n(n/2)} & \cdots & (\omega_n^{n-1})^{\pi_n(n-1)} \end{array}\right] \left[\begin{array}{c} p_{\pi_n(0)} \\ p_{\pi_n(1)} \\ \vdots \\ p_{\pi_n(n/2-1)} \\ \hline p_{\pi_n(n/2)} \\ p_{\pi_n(n/2+1)} \\ \vdots \\ p_{\pi_n(n-1)} \end{array}\right] \tag{29}$$

Equivalently, we have:

$$\left[\begin{array}{c} y_{n,0} \\ \hline y_{n,1} \end{array}\right] = \left[\begin{array}{c|c} F_{n,00} & F_{n,01} \\ \hline F_{n,10} & F_{n,11} \end{array}\right] \left[\begin{array}{c} p_{n,0} \\ \hline p_{n,1} \end{array}\right] \tag{30}$$

where:

$$y_{n,0} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n/2-1} \end{bmatrix} \tag{31a}$$

$$y_{n,1} = \begin{bmatrix} y_{n/2} \\ y_{n/2+1} \\ \vdots \\ y_{n/2+n/2-1} \end{bmatrix}, \tag{31b}$$

$$F_{n,00} = \begin{bmatrix} (\omega_n^0)^{\pi_n(0)} & (\omega_n^0)^{\pi_n(1)} & (\omega_n^0)^{\pi_n(2)} & \dots & (\omega_n^0)^{\pi_n(n/2-1)} \\ (\omega_n^1)^{\pi_n(0)} & (\omega_n^1)^{\pi_n(1)} & (\omega_n^1)^{\pi_n(2)} & \dots & (\omega_n^1)^{\pi_n(n/2-1)} \\ (\omega_n^2)^{\pi_n(0)} & (\omega_n^2)^{\pi_n(1)} & (\omega_n^2)^{\pi_n(2)} & \dots & (\omega_n^2)^{\pi_n(n/2-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (\omega_n^{n/2-1})^{\pi_n(0)} & (\omega_n^{n/2-1})^{\pi_n(1)} & (\omega_n^{n/2-1})^{\pi_n(2)} & \dots & (\omega_n^{n/2-1})^{\pi_n(n/2-1)} \end{bmatrix} \tag{32a}$$

$$F_{n,01} = \begin{bmatrix} (\omega_n^0)^{\pi_n(n/2)} & (\omega_n^0)^{\pi_n(n/2+1)} & (\omega_n^0)^{\pi_n(n/2+2)} & \dots & (\omega_n^0)^{\pi_n(n-1)} \\ (\omega_n^1)^{\pi_n(n/2)} & (\omega_n^1)^{\pi_n(n/2+1)} & (\omega_n^0)^{\pi_n(n/2+2)} & \dots & (\omega_n^1)^{\pi_n(n-1)} \\ (\omega_n^2)^{\pi_n(n/2)} & (\omega_n^2)^{\pi_n(n/2+1)} & (\omega_n^0)^{\pi_n(n/2+2)} & \dots & (\omega_n^2)^{\pi_n(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (\omega_n^{n/2-1})^{\pi_n(n/2)} & (\omega_n^{n/2-1})^{\pi_n(n/2+1)} & (\omega_n^{n/2-1})^{\pi_n(n/2+1)} & \dots & (\omega_n^{n/2-1})^{\pi_n(n-1)} \end{bmatrix} \tag{32b}$$

$$F_{n,10} = \begin{bmatrix} (\omega_n^{n/2})^{\pi_n(0)} & (\omega_n^{n/2})^{\pi_n(1)} & (\omega_n^{n/2})^{\pi_n(2)} & \dots & (\omega_n^{n/2})^{\pi_n(n/2-1)} \\ (\omega_n^{n/2+1})^{\pi_n(0)} & (\omega_n^{n/2+1})^{\pi_n(1)} & (\omega_n^{n/2+1})^{\pi_n(2)} & \dots & (\omega_n^{n/2+1})^{\pi_n(n/2-1)} \\ (\omega_n^{n/2+2})^{\pi_n(0)} & (\omega_n^{n/2+2})^{\pi_n(1)} & (\omega_n^{n/2+2})^{\pi_n(2)} & \dots & (\omega_n^{n/2+2})^{\pi_n(n/2-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (\omega_n^{n-1})^{\pi_n(0)} & (\omega_n^{n-1})^{\pi_n(1)} & (\omega_n^{n-1})^{\pi_n(2)} & \dots & (\omega_n^{n-1})^{\pi_n(n/2-1)} \end{bmatrix} \tag{32c}$$

$$F_{n,11} = \begin{bmatrix} (\omega_n^{n/2})^{\pi_n(n/2)} & (\omega_n^{n/2})^{\pi_n(n/2+1)} & (\omega_n^{n/2})^{\pi_n(n/2+2)} & \dots & (\omega_n^{n/2})^{\pi_n(n-1)} \\ (\omega_n^{n/2+1})^{\pi_n(n/2)} & (\omega_n^{n/2+1})^{\pi_n(n/2+1)} & (\omega_n^{n/2+1})^{\pi_n(n/2+2)} & \dots & (\omega_n^{n/2+1})^{\pi_n(n-1)} \\ (\omega_n^{n/2+2})^{\pi_n(n/2)} & (\omega_n^{n/2+2})^{\pi_n(n/2+1)} & (\omega_n^{n/2+2})^{\pi_n(n/2+2)} & \dots & (\omega_n^{n/2+2})^{\pi_n(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (\omega_n^{n-1})^{\pi_n(n/2)} & (\omega_n^{n-1})^{\pi_n(n/2+1)} & (\omega_n^{n-1})^{\pi_n(n/2+2)} & \dots & (\omega_n^{n/2-1})^{\pi_n(n-1)} \end{bmatrix}, \tag{32d}$$

and

$$p_{n,0} = \begin{bmatrix} p_{\pi_n(0)} \\ p_{\pi_n(1)} \\ \vdots \\ p_{\pi_n(n/2-1)} \end{bmatrix} \tag{33a}$$

$$p_{n,1} = \begin{bmatrix} p_{\pi_n(n/2)} \\ p_{\pi_n(n/2+1)} \\ \vdots \\ p_{\pi_n(n/2+n/2-1)} \end{bmatrix}. \tag{33b}$$

Careful thought shows that the right permutation $\pi_n : [n-1] \to [n-1]$ is:

$$\pi_n(i) = \textsc{Dec}(\textsc{Rev}(\textsc{Bin}(i))), \tag{34}$$

where $\textsc{Bin}(k)$ returns the binary representation of k, $\textsc{Dec}(k)$ returns the decimal representation of k and $\textsc{Rev}(k)$ reverses the bits of (binary) k. This permutation is also know as the **bit-reverse permutation**.

Lemma 4.1 (Bit-Reversal Permutation)**.** *The bit-reverse permutation, as defined by equation 34, satisfies the following properties:*

1. $\pi_n(j) \bmod 2 = 0, j = 0, 1, \ldots, n/2 - 1$.
2. $\pi_n(j) \bmod 2 = 1, j = n/2, n/2 + 1, \ldots, n - 1$.
3. $\pi_n(n/2 + j) = \pi_n(j) + 1, j = 0, 1, \ldots, n/2 - 1$.
4. $\pi_{n/2}(j) = \pi_n(j)/2$.

Proof. Homework. (Hint: use binary representation and complex roots of unity's properties). □

Let us also define the following two matrices:

$$D_{n/2} = \begin{bmatrix} \omega_n^0 & 0 & \cdots & 0 \\ 0 & \omega_n^1 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \omega_n^{n/2-1} \end{bmatrix} \tag{35a}$$

$$E_{n/2} = \begin{bmatrix} \omega_n^{n/2} & 0 & \cdots & 0 \\ 0 & \omega_n^{n/2+1} & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \omega_n^{n-1} \end{bmatrix} \tag{35b}$$

Lemma 4.2 (FFT matrix)**.** *Let F_n denote the $n \times n$ FFT matrix and $F_{n,00}, F_{n,01}, F_{n,10}, F_{n,11}$ its $n/2 \times n/2$ submatrices, as given by equations 32. Also, let $D_{n/2}, E_{n/2}$ be the $n/2 \times n/2$ diagonal matrices defined just above. Then, the following are true:*

1. $E_{n/2} = -D_{n/2}$.
2. $F_{n,01} = D_{n/2}F_{n,00}$ *and similarly* $F_{n,11} = E_{n/2}F_{n,10} = -D_{n/2}F_{n,10}$.
3. $F_{n,10} = F_{n,00}$.
4. $F_{n,00} = F_{n/2}$.
5. $$F_n = \left[\begin{array}{c|c} F_{n/2} & D_{n/2}F_{n/2} \\ \hline F_{n/2} & -D_{n/2}F_{n/2} \end{array}\right]$$

Proof. 1. This follows directly from Corollary 3.2. Specifically,

$$\begin{aligned} E_{n/2}[i,i] &= \omega_n^{n/2+i} \\ &= \omega_n^{n/2}\omega_n^i \\ &= -\omega_n^i \\ &= -D_{n/2}[i,i]. \end{aligned}$$

2. From Lemma 4.1 we know that $\pi_{n/2+i} = \pi_i + 1, i = 0, 1, \ldots, n/2-1$. We substitute this in the definition of $F_{n,01}$ and obtain:

$$
\begin{aligned}
F_{n,01} &= \begin{bmatrix} (\omega_n^0)^{\pi_n(n/2)} & (\omega_n^0)^{\pi_n(n/2+1)} & \cdots & (\omega_n^0)^{\pi_n(n-1)} \\ (\omega_n^1)^{\pi_n(n/2)} & (\omega_n^1)^{\pi_n(n/2+1)} & \cdots & (\omega_n^1)^{\pi_n(n-1)} \\ (\omega_n^2)^{\pi_n(n/2)} & (\omega_n^2)^{\pi_n(n/2+1)} & \cdots & (\omega_n^2)^{\pi_n(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ (\omega_n^{n/2-1})^{\pi_n(n/2)} & (\omega_n^{n/2-1})^{\pi_n(n/2+1)} & \cdots & (\omega_n^{n/2-1})^{\pi_n(n-1)} \end{bmatrix} \\
&= \begin{bmatrix} (\omega_n^0)^{\pi_n(0)+1} & (\omega_n^0)^{\pi_n(1)+1} & \cdots & (\omega_n^0)^{\pi_n(n/2-1)+1} \\ (\omega_n^1)^{\pi_n(0)+1} & (\omega_n^1)^{\pi_n(1)+1} & \cdots & (\omega_n^1)^{\pi_n(n/2-1)+1} \\ (\omega_n^2)^{\pi_n(0)+1} & (\omega_n^2)^{\pi_n(1)+1} & \cdots & (\omega_n^2)^{\pi_n(n/2-1)+1} \\ \vdots & \vdots & \ddots & \vdots \\ (\omega_n^{n/2-1})^{\pi_n(0)+1} & (\omega_n^{n/2-1})^{\pi_n(1)+1} & \cdots & (\omega_n^{n/2-1})^{\pi_n(n/2-1)+1} \end{bmatrix} \\
&= \begin{bmatrix} \omega_n^0 & 0 & \cdots & 0 \\ 0 & \omega_n^1 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \omega_n^{n/2-1} \end{bmatrix} \begin{bmatrix} (\omega_n^0)^{\pi_n(0)} & (\omega_n^0)^{\pi_n(1)} & \cdots & (\omega_n^0)^{\pi_n(n/2-1)} \\ (\omega_n^1)^{\pi_n(0)} & (\omega_n^1)^{\pi_n(1)} & \cdots & (\omega_n^1)^{\pi_n(n/2-1)} \\ (\omega_n^2)^{\pi_n(0)} & (\omega_n^2)^{\pi_n(1)} & \cdots & (\omega_n^2)^{\pi_n(n/2-1)} \\ \vdots & \vdots & \ddots & \vdots \\ (\omega_n^{n/2-1})^{\pi_n(0)} & (\omega_n^{n/2-1})^{\pi_n(1)} & \cdots & (\omega_n^{n/2-1})^{\pi_n(n/2-1)} \end{bmatrix} \\
&= D_{n/2} F_{n,00}
\end{aligned}
$$

In the exact same fashion we obtain $F_{n/2,11} = E_{n/2} F_{n,01} = -D_{n/2} F_{n,01}$.

3. For $k = 0, 1, \ldots, n/2-1$ and $l = 0, 1, \ldots, n/2-1$ we have:

$$
\begin{aligned}
F_{n,10}[k,l] &= (\omega_n^{n/2+k})^{\pi_n(l)} \\
&= (\omega_n^{n/2})^{\pi_n(l)} (\omega_n^k)^{\pi_n(l)} \\
&= (-1)^{\pi_n(l)} (\omega_n^k)^{\pi_n(l)} && \text{(Corollary 3.2)} \\
&= (\omega_n^k)^{\pi_n(l)} && \text{(Lemma 4.1)} \\
&= F_{n,00}[k,l].
\end{aligned}
$$

4. For $k = 0, 1, \ldots, n/2-1$ and $l = 0, 1, \ldots, n/2-1$ we have:

$$
\begin{aligned}
F_{n,00}[k,l] &= (\omega_n^k)^{\pi_n(l)} \\
&= (\omega_n^{\pi_n(l)})^k \\
&= (\omega_{n/2}^{\pi_n(l)/2})^k && \text{(Lemma 3.1)} \\
&= (\omega_{n/2}^{\pi_{n/2}(l)})^k && \text{(Lemma 4.1)} \\
&= (\omega_{n/2}^k)^{\pi_{n/2}(l)} \\
&= F_{n/2}[k,l].
\end{aligned}
$$

5. Follows immediately from (1), (2), (3) and (4).

□

4.2 Interpolation: $p = DFT^{-1}(y)$

We showed that the evaluation at the n n-th roots of unity of a degree-bound n polynomial P from its coefficient representation, i.e the vector $p \in \mathbb{C}^n$ is equivalent to computing the DFT of the p vector: $y = DFT(p)$, with $\Theta(n \log(n))$ time complexity. Now we prove that the interpolation of a degree-bound n polynomial P from its point-value representation at the n n-th roots of unity, which is equivalent to computing DFT^{-1} of the $y \in \mathbb{C}^n$ vector: $p = DFT^{-1}(y)$ can also be achieved with $\Theta(n \log(n))$ time complexity.

Recall that $y = F_n p$ (equation 22) and hence $p = F_n^{-1} y$. Expanding the last equation yields:

$$\begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{n/2-1} \\ p_{n/2} \\ p_{n/2+1} \\ \vdots \\ p_{n-1} \end{bmatrix} = \begin{bmatrix} (\omega_n^0)^0 & (\omega_n^0)^1 & (\omega_n^0)^2 & \cdots & (\omega_n^0)^{n-2} & (\omega_n^0)^{n-1} \\ (\omega_n^1)^0 & (\omega_n^1)^1 & (\omega_n^1)^2 & \cdots & (\omega_n^1)^{n-2} & (\omega_n^1)^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ (\omega_n^{n/2-1})^0 & (\omega_n^{n/2-1})^1 & (\omega_n^{n/2-1})^2 & \cdots & (\omega_n^{n/2-1})^{n-2} & (\omega_n^{n/2-1})^{n-1} \\ (\omega_n^{n/2})^0 & (\omega_n^{n/2})^1 & (\omega_n^{n/2})^2 & \cdots & (\omega_n^{n/2})^{n-2} & (\omega_n^{n/2})^{n-1} \\ (\omega_n^{n/2+1})^0 & (\omega_n^{n/2+1})^1 & (\omega_n^{n/2+1})^2 & \cdots & (\omega_n^{n/2+1})^{n-2} & (\omega_n^{n/2+1})^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ (\omega_n^{n-1})^0 & (\omega_n^{n-1})^1 & (\omega_n^{n-1})^2 & \cdots & (\omega_n^{n-1})^{n-2} & (\omega_n^{n-1})^{n-1} \end{bmatrix}^{-1} \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n/2-1} \\ y_{n/2} \\ y_{n/2+1} \\ \vdots \\ y_{n-1} \end{bmatrix} \tag{36}$$

Theorem 4.3 (FFT Inverse Matrix)**.** *$F_n^{-1}[k,l] = \frac{\omega_n^{-lk}}{n}$, for $k = 0, 1, \ldots, n-1$ and $l = 0, 1, \ldots, n-1$.*

Proof. We prove that $F_n^{-1} F_n = I_n$, where I_n denotes the $n \times n$ identity matrix. Let us consider the (k,l)-th entry of the product $F_n^{-1} F_n$:

$$\begin{aligned} F_n^{-1} F_n[k,l] &= \sum_{t=0}^{n-1} (\frac{\omega_n^{-tk}}{n}) \omega_n^{tl} \\ &= \frac{1}{n} \sum_{t=0}^{n-1} \omega_n^{t(l-k)} \\ &= \begin{cases} 1, & l = k \\ 0, & l \neq k \end{cases} \qquad \text{(Lemma 3.4)} \end{aligned}$$

□

We have therefore proved that:

$$p = \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{n-1} \end{bmatrix} = \begin{bmatrix} \sum_{j=0}^{n-1} \frac{1}{n} y_j (\omega_n^{-0})^j \\ \sum_{j=0}^{n-1} \frac{1}{n} y_j (\omega_n^{-1})^j \\ \vdots \\ \sum_{j=0}^{n-1} \frac{1}{n} y_j (\omega_n^{-(n-1)})^j \end{bmatrix}, \tag{37}$$

Comparing equations 22 and 37, we notice that by modifying the FFT algorithm to switch the roles of p and y, replacing ω_n by ω_n^{-1} and dividing each element of the result by n, we are able to compute the inverse of DFT in time $\Theta(n \log(n))$ as well.

5 A fast algorithm for multiplication of polynomials in coefficient form

Given the FFT algorithm, the following algorithm multiplies two degree-bound n polynomials A and B, in the coefficient representation form, in time $\Theta(n \log(n))$:

1. **Double degree-bound**: create coefficient representations of A and B as degree-bound $2n$ polynomials by adding n high-order zero coefficients to each.
2. **Evaluate**: compute point-value representations of A and B of length $2n$ by applying the FFT algorithm of order $2n$ on each polynomial. These representations contain the values of the two polynomials at the $2n$-th roots of unity.
3. **Point-wise multiply**: compute a point-value representation for the polynomial $D = A * B$ by multiplying the above values pointwise. This representation contains the value of D at each $2n$-th root of unity.
4. **Interpolate**: create the coefficient representation of the polynomial C by applying the FFT algorithm on $2n$ point-value pairs to compute the inverse DFT.

Steps (1) and (3) take $\Theta(n)$ time, whereas steps (2) and (4) take $\Theta(n \log(n))$ time.