1 FFT Intro

We will use $\omega_n$ to denote the first $n$-th root of unity $\omega_n = e^{2\pi i/n}$. The most important fact about roots of unity for our purposes is that the squares of the $2n$-th roots of unity are the $n$-th roots of unity.

Fast Fourier Transform! The Fast Fourier Transform FFT$(p, n)$ takes arguments $n$, some power of 2, and $p$ is some vector $[p_0, p_1, \ldots, p_{n-1}]$.

Treating $p$ as a polynomial $P(x) = p_0 + p_1 x + \ldots + p_{n-1} x^{n-1}$, the FFT computes the following matrix multiplication in $O(n \log n)$ time:

$$
\begin{bmatrix}
P(1) \\
P(\omega_n) \\
P(\omega_n^2) \\
\vdots \\
P(\omega_n^{n-1})
\end{bmatrix} =
\begin{bmatrix}
1 & 1 & 1 & \ldots & 1 \\
1 & \omega_n^1 & \omega_n^2 & \ldots & \omega_n^{n-1} \\
1 & \omega_n^2 & \omega_n^4 & \ldots & \omega_n^{2(n-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \ldots & \omega_n^{(n-1)(n-1)}
\end{bmatrix}
\cdot
\begin{bmatrix}
p_0 \\
p_1 \\
p_2 \\
\vdots \\
p_{n-1}
\end{bmatrix}
$$

If we let $E(x) = p_0 + p_2 x + \ldots + p_{n-2} x^{n/2-1}$ and $O(x) = p_1 + p_3 x + \ldots + p_{n-1} x^{n/2-1}$, then $P(x) = E(x^2) + xO(x^2)$, and then FFT$(p, n)$ can be expressed as a divide-and-conquer algorithm:

1. Compute $E' = FFT(E, n/2)$ and $O' = FFT(O, n/2)$.
2. For $i = 0 \ldots n-1$, assign $P(\omega_n^i) \leftarrow E((\omega_n^i)^2) + \omega_n^i O((\omega_n^i)^2)$

(a) Let $p = [p_0]$. What is FFT$(p, 1)$?

(b) Use the FFT algorithm to compute FFT$([1, 4], 2)$ and FFT$([3, 2], 2)$.

(c) Use your answers to the previous parts to compute FFT$([1, 3, 4, 2], 4)$.
(d) Describe how to multiply two polynomials \( p(x), q(x) \) in coefficient form of degree at most \( d \).

2 Cubed Fourier

(a) Cubing the 9\(^{th}\) roots of unity gives the 3\(^{rd}\) roots of unity. Next to each of the third roots below, write down the corresponding 9\(^{th}\) roots which cube to it. The first has been filled for you. We will use \( \omega_9 \) to represent the primitive 9\(^{th}\) root of unity, and \( \omega_3 \) to represent the primitive 3\(^{rd}\) root.

\[
\begin{align*}
\omega_3^0 & : \omega_9^0, \\
\omega_3^1 & : , , \\
\omega_3^2 & : , , 
\end{align*}
\]

(b) You want to run FFT on a degree-8 polynomial, but you don’t like having to pad it with 0s to make the (degree+1) a power of 2. Instead, you realize that 9 is a power of 3, and you decide to work directly with 9\(^{th}\) roots of unity and use the fact proven in part (a). Say that your polynomial looks like \( P(x) = a_0 + a_1x + a_2x^2 + \ldots + a_8x^8 \). Describe a way to split \( P(x) \) into three pieces so that you can make an FFT-like divide-and-conquer algorithm.

3 Predicting a Weighted Average

You have a time-series dataset \( y_0, y_1, \ldots, y_{n-1} \) where all \( y_t \in \mathbb{R} \). You are given fixed coefficients \( c_0, \ldots, c_{n-2} \), which give the following prediction for day \( t \geq 1 \):

\[
p_t = \sum_{k=0}^{t-1} c_k y_{t-1-k}
\]

You would like to evaluate the accuracy of this prediction on the dataset by computing the mean squared error, given by

\[
\frac{1}{n-1} \sum_{t=1}^{n-1} (p_t - y_t)^2
\]

Find an \( O(n \log n) \) time algorithm to compute the mean squared error, given dataset \( y_0, y_1, \ldots, y_{n-1} \) and coefficients \( c_0, \ldots, c_{n-2} \).

**Hint:** Recall that if \( p(x) = p_0 + p_1x + p_2x^2 + \ldots + p_{n-1}x^{n-1} \) and \( q(x) = q_0 + q_1x + q_2x^2 + \ldots + q_{n-1}x^{n-1} \), then their product is \( p(x) \cdot q(x) = r(x) = r_0 + r_1x + \ldots + r_{2n-2}x^{2n-2} \), where

\[
r_j = \sum_{k=0}^{j} p_k q_{j-k}
\]
4 Extra Divide and Conquer Practice: Sorted Array

Given a sorted array \( A \) of \( n \) (possibly negative) distinct integers, you want to find out whether there is an index \( i \) for which \( A[i] = i \). Devise a divide-and-conquer algorithm that runs in \( O(\log n) \) time.

5 Extra Divide and Conquer Practice: Quantiles

Let \( A \) be an array of length \( n \). The boundaries for the \( k \) quantiles of \( A \) are \( \{a^{(n/k)}, a^{(2n/k)}, \ldots, a^{((k-1)n/k)}\} \) where \( a^{(\ell)} \) is the \( \ell \)-th smallest element in \( A \).

Devise an algorithm to compute the boundaries of the \( k \) quantiles in time \( O(n \log k) \). For convenience, you may assume that \( k \) is a power of 2.

*Hint:* Recall that \( \text{QUICKSELECT}(A, \ell) \) gives \( a^{(\ell)} \) in \( O(n) \) time.