Note: Your TA may not get to all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. The discussion worksheet is also a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 FFT Intro

We will use ω_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* $\text{FFT}(p, n)$ 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 $\mathcal{O}(n \log n)$ 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 $E' = \text{FFT}(E, n/2)$ and $O' = \text{FFT}(O, n/2)$.
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 $\text{FFT}(p, 1)$?

Solution: Notice the FFT matrix is just $[1]$, so $\text{FFT}(p, 1) = [p_0]$.

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

Solution: $\text{FFT}([1, 4], 2) = [5, -3]$ and $\text{FFT}([3, 2], 2) = [5, 1]$.

We show how to compute $\text{FFT}([1, 4], 2)$, and $\text{FFT}([3, 2], 2)$ is similar.

First we compute $\text{FFT}([1], 1) = [1] = E'$ and $\text{FFT}([4], 1) = [4] = O'$ by part (a). Notice that $E' = [E(1)] = 1$ and $O' = [O(1)] = [4]$, so when we need to use these values later they have already been computed in E' and O'.

Let P be our result. We wish to compute $P(\omega_2^0) = P(1)$ and $P(\omega_2^1) = P(-1)$.

$$P(1) = E(1) + 1 \cdot O(1) = 1 + 4 = 5$$
$$P(-1) = E(1) + -1 \cdot O(1) = 1 - 4 = -3$$

So our answer is $[5, -3]$.

(c) Use your answers to the previous parts to compute $\text{FFT}([1, 3, 4, 2], 4)$.

Solution: $\omega_4 = i$. The following table is good to keep handy:

ω_4^i	1	i	-1	$-i$
$(\omega_4^i)^2$	1	-1	1	-1

Let $E' = \text{FFT}([1,4],2) = [5,-3]$ and $O' = \text{FFT}([3,2],2) = [5,1]$. Notice that $E' = [E(1), E(-1)] = [5,-3]$ = and $O' = [O(1), O(-1)] = [5,1]$, so when we need to use these values later they have already been computed in the divide step. Let R be our result, we wish to compute $R(1), R(i), R(-1), R(-i)$.

$$R(1) = E(1) + 1 \cdot O(1) = 5 + 5 = 10$$
$$R(i) = E(-1) + i \cdot O(-1) = -3 + i$$
$$R(-1) = E(1) - 1 \cdot O(1) = 5 - 5 = 0$$
$$R(-i) = E(-1) - i \cdot O(-1) = -3 - i$$

So our answer $[10, -3+i, 0, -3-i]$.

(d) Describe how to multiply two polynomials $p(x), q(x)$ in coefficient form of degree at most d.

Solution: The idea is to take the FFT of both p and q, multiply the evaluations, and then take the inverse FFT. Note that $p \cdot q$ has degree at most $2d$, which means we need to pick n as the smallest power of 2 greater than $2d$, call this 2^k. We can zero-pad both polynomials so they have degree $2^k - 1$.

Then $M = \text{FFT}(p, 2^k) \cdot \text{FFT}(q, 2^k)$ (with multiplication elementwise) computes $pq(\omega_{2^k}^i)$ for all $i = 0, \ldots, 2^k - 1$.

We take the inverse FFT of M to get back to $p \cdot q$ in coefficient form.

2 Cubed Fourier

(a) Cubing the 9th roots of unity gives the 3rd roots of unity. Next to each of the third roots below, write down the corresponding 9th roots which cube to it. The first has been filled for you. *We will use ω_9 to represent the primitive 9th root of unity, and ω_3 to represent the primitive 3rd root.*

$\omega_3^0 : \omega_9^0,$,

$\omega_3^1 :$, ,

$\omega_3^2 :$, ,

(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 9th roots of unity and use the fact proven in part (a). Say that your polynomial looks like $P(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_8 x^8$. Describe a way to split $P(x)$ into three pieces so that you can make an FFT-like divide-and-conquer algorithm.

Solution:

(a) $\omega_3^0 : \omega_9^0, \omega_9^3, \omega_9^6$
$\omega_3^1 : \omega_9^1, \omega_9^4, \omega_9^7$
$\omega_3^2 : \omega_9^2, \omega_9^5, \omega_9^8$

(b) Let $P(x) = P_1(x^3) + xP_2(x^3) + x^2 P_3(x^3)$
where $P_1(x^3) = a_0 + a_3 x^3 + a_6 x^6$.
and $P_2(x^3) = a_1 + a_4 x^3 + a_7 x^6$.
and $P_3(x^3) = a_2 + a_5 x^3 + a_8 x^6$.

3 Predicting a Weighted Average

You have a time-series dataset $y_0, y_1, \ldots, y_{n-1}$ where all $y_i \in \mathbb{R}$. 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 $\mathcal{O}(n \log n)$ 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 $p(x) = p_0 + p_1 x + p_2 x^2 + \ldots p_{n-1} x^{n-1}$ and $q(x) = q_0 + q_1 x + q_2 x^2 + \ldots q_{n-1} x^{n-1}$, then their product is $p(x) \cdot q(x) = r(x) = r_0 + r_1 x + \ldots + r_{2n-2} x^{2n-2}$, where

$$r_j = \sum_{k=0}^{j} p_k q_{j-k}$$

Solution:
To compute the mean-squared-error, the main thing we need to focus on is computing all of the p_ts. Once they have been computed, the MSE just takes time $\mathcal{O}(n)$ time to compute.

Create polynomials $p(x) = c_0 + c_1 x + \ldots + c_{n-2} x^{n-2}$ and $q(x) = 0 + y_0 x + y_1 x^2 + \ldots + y_{n-1} x^n$, and compute their product $r = p \cdot q$ using FFT-based multiplication in $\mathcal{O}(n \log n)$ time. Notice that we shifted all the coefficients in q by one place, so that the prediction on the tth day can only use the stock prices until the $t-1$st day.

Then notice that

$$r_t = \sum_{k=0}^{t} c_k q_{t-k} = \sum_{k=0}^{t-1} c_k y_{t-1-k}$$

So the coefficients $r_1, \ldots, r_n$ correspond to the predictions $p_1, \ldots p_n$.