1 Complex numbers review

A complex number is a number that can be written in the rectangular form \( a + bi \) (\( i \) is the imaginary unit, with \( i^2 = -1 \)). The following famous equation (Euler’s formula) relates the polar form of complex numbers to the rectangular form:

\[
re^{i\theta} = r (\cos \theta + i \sin \theta)
\]

In polar form, \( r \geq 0 \) represents the distance of the complex number from 0, and \( \theta \) represents its angle. Note that since \( \sin(\theta) = \sin(\theta + 2\pi) \), \( \cos(\theta) = \cos(\theta + 2\pi) \), we have \( re^{i\theta} = re^{i(\theta + 2\pi)} \) for any \( r, \theta \).

The \( n \)-th roots of unity are the \( n \) complex numbers satisfying \( \omega^n = 1 \). They are given by

\[
\omega_k = e^{2\pi ik/n}, \quad k = 0, 1, 2, \ldots, n - 1
\]

(a) Let \( x = e^{2\pi i 3/10}, y = e^{2\pi i 5/10} \) which are two 10-th roots of unity. Compute the product \( x \cdot y \). Is this an \( n \)-th root of unity for some \( n \)? Is it a 10-th root of unity?

What happens if \( x = e^{2\pi i 6/10}, y = e^{2\pi i 7/10} \)?

(b) Show that for any \( n \)-th root of unity \( \omega \neq 1 \), \( \sum_{k=0}^{n-1} \omega^k = 0 \), when \( n > 1 \).

Hint: Use the formula for the sum of a geometric series \( \sum_{k=0}^{n} \alpha^k = \frac{\alpha^{n+1} - 1}{\alpha - 1} \). It works for complex numbers too!

(c) (i) Find all \( \omega \) such that \( \omega^2 = -1 \).

(ii) Find all \( \omega \) such that \( \omega^4 = -1 \).


## 2 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 $\text{FFT}(p, n)$ 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 value of $P(x)$ for all $x$ that are $n$-th roots of unity by doing the following matrix multiplication in $O(n \log n)$ time:

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

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 $\text{FFT}(p, n)$ 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 $P(\omega_n^i) \leftarrow E((\omega_n^i)^2) + \omega_n^i O((\omega_n^i)^2)$

Also observe that:

$$
\frac{1}{n}
\begin{bmatrix}
1 & 1 & 1 & \cdots & 1 \\
1 & \omega_n^{-1} & \omega_n^{-2} & \cdots & \omega_n^{-(n-1)} \\
1 & \omega_n^{-2} & \omega_n^{-4} & \cdots & \omega_n^{-2(n-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega_n^{-(n-1)} & \omega_n^{2(n-1)} & \cdots & \omega_n^{-(n-1)(n-1)}
\end{bmatrix}
\begin{bmatrix}
1 & 1 & 1 & \cdots & 1 \\
1 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{(n-1)} \\
1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega_n^{(n-1)} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)}
\end{bmatrix}^{-1}
$$

(You should verify this on your own!) And so given the values $P(1), P(\omega_n), P(\omega_n^2), \ldots$, we can compute $P$ by doing the following matrix multiplication:

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

This can be done in $O(n \log n)$ time using a similar divide and conquer algorithm.

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

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

---

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 \( d \).

3 Cartesian Sum

Let \( A \) and \( B \) be two sets of integers in the range 0 to 10\( n \). The Cartesian sum of \( A \) and \( B \) is defined as

\[
A + B = \{ a + b \mid a \in A, b \in B \}
\]

i.e. all sums of an element from \( A \) and an element with \( B \). For example, \( \{1, 3\} + \{2, 4\} = \{3, 5, 7\} \).

Note that the values of \( A + B \) are integers in the range 0 to 20\( n \). Design an algorithm that finds the elements of \( A + B \) in \( O(n \log n) \) time, which additionally tells you for each \( c \in A + B \), how many pairs \( a \in A, b \in B \) there are such that \( a + b = c \).

*Hint:* Notice that \((x^1 + x^3) \cdot (x^2 + x^4) = x^3 + 2x^5 + x^7\)
4 Cubed Roots of Unity

(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.

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

$$\omega_3^1 : , ,$$

$$\omega_3^2 : , ,$$

(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 $P(x) = a_0 + a_1x + a_2x^2 + \ldots + a_8x^8$. Describe a way to split $P(x)$ into three pieces (instead of two) so that you can make an FFT-like divide-and-conquer algorithm.

(c) What is the runtime of FFT when we divide the polynomial into three pieces instead of two?