CS 170 Homework 3

Due **9/18/2023, at 10:00 pm (grace period until 11:59pm)**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write "none".

2 Inverse FFT

Recall from lecture that a polynomial of degree d can either be specified by its coefficients or by its values on $d+1$ points. If we choose the points to be roots of unity then we can use FFT to switch between the two representations efficiently.

(a) Consider two polynomials $A(x) = 1 + 5x + 3x^2 + 4x^3$ and $B(x) = 3 + 4x + 2x^3$. Pick an appropriate value of n and write down the values of $A(x)$, $B(x)$ and $C(x) = A(x)B(x)$ at each of the n roots of unity.

Your n should be large enough to allow recovery of the coefficients of C from its evaluation on the n points using inverse FFT. What value of n did you pick?
(Note: n should be a power of 2).

Hint: What will be the degree of C?

Now we will focus recovering the coefficients of C given its evaluation on n points. Recall that in class we defined M_n, the matrix involved in the Fourier Transform, to be the following matrix:

$$M_n = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)(n-1)} \end{bmatrix},$$

where ω is a primitive n-th root of unity.

For the rest of this problem we will refer to this matrix as $M_n(\omega)$ rather than M_n. We will examine some proprties of the inverse of this matrix.

(b) Define

$$M_n(\omega^{-1}) = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \dots & \omega^{-(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \dots & \omega^{-(n-1)(n-1)} \end{bmatrix}$$

Recall that $\omega^{-1} = 1/\omega = \bar{\omega} = \exp(-2\pi i/n)$.

Show that $\frac{1}{n}M_n(\omega^{-1})$ is the inverse of $M_n(\omega)$, i.e. show that

$$\frac{1}{n}M_n(\omega^{-1})M_n(\omega) = I$$

where I is the $n \times n$ identity matrix – the matrix with all ones on the diagonal and zeros everywhere else.

(c) Let A be a square matrix with complex entries. The *conjugate transpose* $A^\dagger$ of A is given by taking the complex conjugate of each entry of A^T. A matrix A is called *unitary* if its inverse is equal to its conjugate transpose, i.e. $A^{-1} = A^\dagger$. Show that $\frac{1}{\sqrt{n}}M_n(\omega)$ is unitary.

(d) Suppose we have a polynomial $C(x)$ of degree at most $n-1$ and we know the values of $C(1), C(\omega), \ldots, C(\omega^{n-1})$. Explain how we can use $M_n(\omega^{-1})$ to find the coefficients of $C(x)$.

(e) Show that $M_n(\omega^{-1})$ can be broken up into four $n/2 \times n/2$ matrices in almost the same way as $M_n(\omega)$. Specifically, suppose we rearrange columns of M_n so that the columns with an even index are on the left side of the matrix and the columns with an odd index are on the right side of the matrix (where the indexing starts from 0). Show that after this rearrangement, $M_n(\omega^{-1})$ has the form:

$$\begin{bmatrix} M_{n/2}(\omega^{-2}) & \omega^{-j}M_{n/2}(\omega^{-2}) \\ M_{n/2}(\omega^{-2}) & -\omega^{-j}M_{n/2}(\omega^{-2}) \end{bmatrix}$$

The notation $\omega^{-j}M_{n/2}(\omega^{-2})$ is used to mean the matrix obtained from $M_{n/2}(\omega^{-2})$ by multiplying the j^{th} row of this matrix by ω^{-j} (where the rows are indexed starting from 0). You may assume that n is a power of two.

3 Counting k-inversions

A *k-inversion* in a bitstring b is when a 1 in the bitstring appears k indices before a 0; that is, when $b_i = 1$ and $b_{i+k} = 0$, for some i. For example, the string 010010 has two 1-inversions (starting at the second and fifth bits), one 2-inversion (starting at the second bit), and one 4-inversion (starting at the second bit).

Devise an algorithm which, given a bitstring b of length n, counts all the k-inversions, for each k from 1 to $n-1$. Your algorithm should run faster than $\Theta(n^2)$ time. You can assume arithmetic on real numbers can be done in constant time.

Give a 3-part solution.

4 Protein Matching

Often times in biology, we would like to locate the existence of a gene in a species' DNA. Of course, due to genetic mutations, there can be many similar but not identical genes that

serve the same function, and genes often appear multiple times in one DNA sequence. So a more practical problem is to find all genes in a DNA sequence that are similar to a known gene.

To model this problem, let g be a length-n string corresponding to the known gene, and let s be a length-m string corresponding to the full DNA sequence, where $m \geq n$. We would like to solve the following problem: find the (starting) location of all length n-substrings of s which match *exactly* match g. For example, using 0-indexing, if $g = ACT$, $s = ACTACTA$, your algorithm should output 0 and 3.

(a) Give a $O(nm)$ time algorithm for this problem.

Hint: if stuck, refer to discussion Q4.

(b) Assume g and s are given as bitstrings, i.e. every character is either 0 or 1. Give a $O(m \log m)$ time algorithm.

Hint: if stuck, refer to discussion Q4.

(c) Assume more generally that we know that g and s combined use at most α distinct characters $c_1, c_2, \ldots c_\alpha$ (for example in the previous part, we had $\alpha = 2$. For DNA sequences, we'd have $\alpha = 4$). Give an $O(m \log m)$ time algorithm to find all substrings of length n in s that match g.

Hint: Represent the strings as complex vectors.

5 Triple sum

We are given an array $A[0..n-1]$ with n elements, where each element of A is an integer in the range $0 \leq A[i] \leq n$ (the elements are not necessarily distinct). We would like to know if there exist indices i, j, k (not necessarily distinct) such that

$$A[i] + A[j] + A[k] = n$$

Design an $\mathcal{O}(n \log n)$ time algorithm for this problem. Note that you do not need to actually return the indices; just yes or no is enough.

Hint 1: elements can be encoded using exponents!

Hint 2: if stuck, refer to discussion Q3.

6 [Coding] FFT & Evaluating Predictions

For this week's homework, you'll implement implement FFT and then apply FFT as a black box to implement some fast algorithms. There are two ways that you can access the notebook and complete the problems:

1. **On Local Machine**: `git clone` (or if you already cloned it, `git pull`) from the coding homework repo,

   `https://github.com/Berkeley-CS170/cs170-fa23-coding`

and navigate to the `hw03` folder. Refer to the `README.md` for local setup instructions.

2. **On Datahub**: Click here and navigate to the `hw03` folder if you prefer to complete this question on Berkeley DataHub.

Notes:

- *Submission Instructions:* Please download your completed `fft.ipynb` file or submission `.zip` file and submit it to the gradescope assignment titled "Homework 3 Coding Portion".
- *OH/HWP Instructions:* Designated coding course staff will provide conceptual and debugging help during office hours and homework parties.
- *Academic Honesty Guideline:* We realize that code for some of the algorithms we ask you to implement may be readily available online, but we strongly encourage you to not directly copy code from these sources. Instead, try to refer to the resources mentioned in the notebook and come up with code yourself. That being said, we **do acknowledge** that there may not be many different ways to code up particular algorithms and that your solution may be similar to other solutions available online.