CS 170 Homework 7

Due **Monday 3/9/2026, at 10:00 pm (grace period until 11:59pm)**

Study Group

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

1 Inverse FFT (Solo Question; 10 points)

Recall that the Fourier transform gives an efficient algorithm to multiply a vector by the matrix

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

where $\omega = \cos\frac{2\pi}{n} + i\sin\frac{2\pi}{n}$ is a primitive nth root of unity.

(a) Show that

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

is the inverse matrix to $M_n(\omega)$, meaning that the product $\frac{1}{n}\cdot M_n(\omega^{-1})\cdot M_n(\omega)$ equals the $n \times n$ identity matrix. Here recall that $\omega^{-1} = \omega^* = \cos\frac{2\pi}{n} - i\sin\frac{2\pi}{n}$.

(b) Using the similarity between the matrices $M_n(\omega)$ and $M_n(\omega^{-1})$, describe (in a sentence) how to modify the FFT algorithm to instead perform the *inverse* Fourier transform. That is, if we apply FFT followed by your inverse FFT, we should get back the original vector.

(c) Recall that the Fourier transform can be interpreted as performing polynomial evaluation. How can we similarly interpret the inverse Fourier transform? Specifically, what do the inputs and outputs correspond to in the language of polynomials?

2 Triple Sum (10 points)

We are given an array $A[0, \ldots, n-1]$ 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 $O(n \log n)$ time algorithm for this problem. Note that you do not need to actually return the indices; just yes or no is enough.

3 Cyclic Shifts (10 points)

In this question, given two length-n vectors u, v, you will show how to efficiently compute the dot product of v with different "shifts" of u.

(a) Let the jth forward shift of u be the vector we obtain from u by moving every entry j steps forward. For instance, the $j = 0, 1, 2, 3$ shifts of $u = (2, 4, 5)$ are $(2, 4, 5)$, $(0, 2, 4)$, $(0, 0, 2)$, $(0, 0, 0)$. Note that the first j entries of the shifted vector are set to 0.

You want to compute the dot product of a vector v with every forward shift of u. Specifically, the output consists of $n + 1$ values, where the jth value is the dot product of v with the jth forward shift of u. Give a $O(n \log n)$ time algorithm for this task.

Hint: Define polynomials whose coefficients are given by the vectors u and v. What happens when we multiply these polynomials?

(b) We define the jth *cyclic* shift of u to again be obtained by moving every entry j steps forward, but now where we loop back to index 1 after hitting index n. For instance, the $j = 0, 1, 2$ cyclic shifts of $u = (2, 4, 5)$ are $(2, 4, 5), (5, 2, 4), (4, 5, 2)$. Modify your algorithm from part (a) to instead compute the dot product of v with every cyclic shift of u.

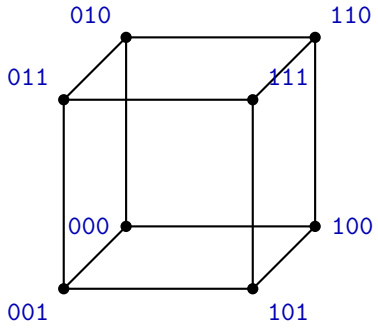
4 Butterflies or Coding (10 points)

For full credit, you should do **one of the following two** questions. (You may solve the other for fun if you want!)

1. **Butterflies**

   Suppose you have $n = 2^d$ processors arranged on the vertices of a d-dimensional boolean hypercube. This graph has n vertices, labeled by all the length-d bit strings, and an edge connects every pair of vertices whose labels differ in exactly one bit.

   For instance, if $d = 3$, then there are $n = 2^3 = 8$ processors on the vertices of a 3-dimensional cube as drawn below:

   Assume each processor can only communicate with its d neighbors. How can you efficiently run the FFT, where each processor is given one of the n inputs, and must compute one of the n outputs? You should use the processors in parallel.

2. **Coding FFT**

   For this weeks coding questions, you'll implement the **Fast Fourier Transform (FFT)** algorithm and apply it to speed up polynomial multiplication. There are two ways that you can access the notebook and complete the problems:

   (a) **On Datahub**: click here and navigate to the `hw07` folder.

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

   https://github.com/Berkeley-CS170/cs170-sp26-coding

   and navigate to the `hw07` folder. Refer to the `README.md` for local setup instructions.

   Notes:

   - *Submission Instructions:* Please download your completed submission `.zip` file and submit it to the Gradescope assignment titled "HW07 (Coding)".
   - *Getting Help:* Conceptual questions are always welcome on Edstem and office hours; *note that support for debugging help during OH will be limited.* If you need

debugging help first try asking on the public Edstem threads. To ensure others can help you, make sure to:

(a) Describe the steps you've taken to debug the issue prior to posting on Ed.

(b) Describe the specific error you're running into.

(c) Include a few small but nontrivial test cases, alongside both the output you expected to receive and your function's actual output.

If staff tells you to make a private Ed post, make sure to include *all of the above items* plus your full function implementation. If you don't provide them, we will ask you to provide them.

- *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.

