# D&C and Polynomial Multiplication
# Problem Set 2 – CS6515 (Spring 2025)

- This problem set is due on **Thursday January 23rd**.
- Submission is via Gradescope.
- Your solution must be a typed pdf (e.g., via LaTeX, Markdown, etc. Anything that allows you to type math notation) – no handwritten solutions.
- Please try to make your solutions as concise and readable as possible. Most problems will have solutions that are no more than a page long. Consider using bullet points and adding space to break up large paragraphs into smaller chunks.
- There are 3 problems. Each problem is graded with 20p. **There is +1p bonus per problem** for stating (i) how long it took you to solve that problem, and (ii) how long it took you to type the answer.

**Remark on algorithm descriptions** To make things easier for the TAs to grade, please use a combination of plain English and mathematical notation. Do not write code (C, Java, etc.). For example, you can say something like "$x := \max_{i=0,\dots,n-1}$ `A[i]`" or "Find the maximum value in the array $A$" instead of writing a for-loop that computes the maximum.

## 4 Sum-tuous Arrays

For given $p, q$ with $0 \le p \le 0.99$ and $0.01 \le q \le 1$, we call an array $A[0 ... n-1]$ $pq$-sum-tuous if it has following recursive properties:

- $A[0, ..., \lfloor p \cdot (n-1) \rfloor\ ]$ is $pq$-sum-tuous,
- $A[\ \lceil q \cdot (n-1) \rceil, ..., n-1]$ is $pq$-sum-tuous, and
- we have $\sum_{i=0}^{\lfloor p(n-1) \rfloor} A[i] < \sum_{i=\lceil q(n-1) \rceil}^{n-1} A[i]$.

Arrays of length 1 are always $pq$-sum-tuous.

**Problem:**

1. Give an algorithm that receives $p, q, A[0, ..., n-1]$ as input, and decides if $A$ is $pq$-sum-tuous.
   (You do not need to prove correctness.)
2. Analyze your algorithm's time complexity. You may assume $p, q$ are both constant, i.e., do not depend on $n$. (Observe that there may be different cases for your algorithm's complexity, depending on the relationship between $p$ and $q$.)

Remark: You may use the Akra-Bazzi theorem from class to analyze the time complexity. Be careful though, as Akra-Bazzi needs $b_i > 1$.

**Remark:** For problems 5 and 6 below, you may assume that polynomials of degree at most $d$ can be multiplied in $O(d \log d)$ time via FFT. Your algorithm can have a line like "$h(x) = f(x) \cdot g(x)$" or "h = Multiply(f,g)" and you can assume this line takes $O(d \log d)$ time when $f, g$ are of degree $\leq d$.

## 5 Vector Distances via Polynomial Multiplication

We are given array $A[0...n-1]$ and another array $B[0...m-1]$ $(m \leq n)$. The arrays contain numbers, i.e., $A[i], B[j] \in \mathbb{R}$ for all $i, j$.

We want to find the position $i$ where $A[i, i+1, ..., i+m-1]$ is close to $B$. Here we use the $\ell_2$-distance[1] to measure closeness.

(a) For two vectors $u, v \in \mathbb{R}^m$, show that $\|u - v\|^2 = \|u\|^2 - 2\langle u, v\rangle + \|v\|^2$.

(b) Construct an algorithm that receives arrays $A$ and $B$, and then returns the index $i$ where

$$\|A[i, i+1, ..., i+m-1] - B\|$$

is as small as possible. You do not need to prove correctness of your algorithm.

(c) Give a complexity analysis, i.e., state and explain the complexity of your algorithm.

Remark: For full points, your algorithm should run in $O(n \log n)$ time.

### 5.1 Programming Option

Instead of submitting a pdf answering (a),(b),(c), you can write a Python program.

**Submission Instructions:** Submission is via gradescope. You must submit your source code.

**Grading:** Gradescope will automatically test your submission. You can resubmit as often as you want (until the due date) and you can see on gradescope how many test cases you currently passed. There are no partial points for passing part of the test cases, because even very wrong solutions can pass some test cases. For partial points, submit a theory/pdf solution instead. While your submission must pass all test cases on gradescope, the final pass/fail decision is made by a TA. (This is to make sure your submission is actually solving the problem described above. For example, hardcoding the output is obviously not a valid solution.) We need this rule because all test cases are public (on Piazza) to help you test/debug your program.

**Input and Output:** There is a template provided on Piazza to help you get started. The template handles how to read the input and write the output, so you can just focus on implementing the algorithm. The template also includes an implementation of FFT/polynomial multiplication that you can use. If you do not want to use those templates, we have a description on how to read the input and write the output below.

The program reads the input from standard input and prints its output to standard output. The input format is as follows: The first line provides two numbers $n$ and $m$, the size of $A$ and $B$ respectively. The lines list $n$ numbers separated by space and represents the entries of $A$. The next line represents the entries of $B$. Example:

```
6 3
1 2 1 5 3 1
1 5 3
```

Here $A = [1, 2, 1, 5, 3, 1]$ is an array of length 6, and $B = [1, 5, 3]$ is an array of length 3.
Your program must print the **index *i*** where $\|A[i, ..., i+m-1] - B\|$ is as small as possible. In the above example, the answer would be 2. **In case of ties, return the smallest index.**

[1] Reminder: for $u, v \in \mathbb{R}^m$ the distance is $\|u - v\| = \sqrt{\sum_i (u_i - v_i)^2}$.

## 6 Polynomial Zeros

In this problem, we are given as input a list of $n$ numbers $x_1, ..., x_n \in \mathbb{R}$ and want to compute the coefficients of a degree $n$ monic[2] polynomial $f(x)$ with roots at $x_1, ..., x_n$.
That is, find $f_0, f_1, ..., f_n \in \mathbb{R}$ where $f_n = 1$, such that for $f(x) := \sum_{k=0}^{n} f_k \cdot x^k$ we have $f(x_i) = 0$ for all $i = 1, ..., n$.

1. Give an algorithm for this problem.
2. Briefly explain why your algorithm is correct.
3. Give a complexity analysis, i.e., state and explain the time complexity of your algorithm.

Remark: For full points, your algorithm should run in $O(n(\log n)^2)$ time.

[2]Monic means that the leading coefficient is 1. So the trivial all-0-polynomial is not a valid solution.

## Ungraded Practice Problems

The following are some additional practice problems that you can use to practice FFT-based algorithms. Do not submit your solutions to gradescope as they will not be graded. There is no extra credit for solving these.

### 102.1 Masked Distances

We are given three arrays $A[0...n-1]$, $B[0...m-1]$, $M[0...m-1]$ with $m \le n$ and $M[i] \in [0,1]$ for all $i$. For vectors $v \in \mathbb{R}^m$ we define the *mask-norm*

$$\|v\|_M^2 := \sum_{k=0}^{m-1} M[k] \cdot v_k^2$$

i.e., the classical L2-norm but we weight entries by $M[k] \in [0,1]$. So array $M$ functions as a "mask" and each entry $M[k]$ specifies how important the entry is on a scale from 0 to 1.

For any index $i$, we interpret $A[i...i+m-1], B[0...m-1] \in \mathbb{R}^m$ as $m$-dimensional vectors. We want to find index $i$ where $A[i...i+m-1]$ is "closest" to $B[0...m-1]$ with respect to the mask-norm. That is, find $i$ that minimizes

$$\|A[i...i+m-1] - B[0...m-1]\|_M^2 = \sum_{k=0}^{m-1} M[k] \cdot (A[i+k] - B[k])^2.$$

(For example, array $B$ may be a picture of some object, array $M$ is a mask that separates the object from the background in picture $B$, and we try to find the object within picture $A$.)

1. Show that for any two vectors $u, v \in \mathbb{R}^m$, we have

$$\|u-v\|_M^2 = \|u\|_M^2 - 2\left(\sum_{k=0}^{m-1} u_k \cdot v_k \cdot M[k]\right) + \|v\|_M^2.$$

2. Design and analyze an algorithm that, given arrays $C[0...n-1]$, $D[0...m-1]$, computes in $O(n \log n)$ time the array $E[0...n-m]$ with

$$E[i] = \sum_{k=0}^{m-1} C[i+k] \cdot D[k]$$

(Hint: Use FFT in $O(n \log n)$ time for polynomial multiplication.)

3. Design and analyze an algorithm that, given arrays $A[0...n-1]$, $B[0...m-1]$, $M[0...m-1]$, returns in $O(n \log n)$ time an index $i$ where $\|A[i...i+m-1] - B[0...m-1]\|_M^2$ is as small as possible. (You may use 2. even if you have not solved it.)

### 102.2 Arithmetic Progressions of Length 3

We are given a set of numbers $S \subset \{0, 1, ..., n\}$. We say that $S$ contains an arithmetic progression of length 3, if there is some triple $i, j, k \in S$ with $j - i = k - j$ (so $i, j, k$ are 3 equally spaced numbers).

We want to count the number of arithmetic progressions, i.e., the number of triples $i, j, k \in S$ with $j - i = k - j$.

(a) Given set $S$, describe a polynomial $f(x)$ where $f(x) \cdot f(x)$ allows us to extract for all $j \in \{0, 1, ..., n-1\}$ the number of pairs $i, k \in S$ with $i + k = 2j$.
Explain how to extract that information from the coefficients of $f(x) \cdot f(x)$.

(b) Construct an algorithm that receives $S$, and then computes the number of arithmetic progressions of length 3. You do not need to prove correctness of your algorithm.