Name:

Student ID #:

- You are expected to work on this assignment on your own
- Use pseudocode, Python-like or English to describe your algorithms. Absolutely no C++/C/Java
- When designing an algorithm, you are allowed to use any algorithm or data structure we explained in class, without giving its details, unless the question specifically requires that you give such details
- Always remember to analyze the time complexity of your algorithms
- Homework has to be submitted electronically on Gradescope by the deadline. No late assignments will be accepted
**Problem 1.** (10 points)

In this problem we are given a text of size $2n$ and a pattern of size $n$ and we want to compute for each position in the text the number of matched symbol with the pattern. A brute force approach would consist in aligning the pattern at each position in the text and computing the number of matches, for an overall $O(n^2)$ time. Here we show that one can solve this problem using the FFT.

Formally, let $t = a_0a_1a_2 \cdots a_{2n−1}$ and $p = b_0b_1b_2 \cdots b_{n−1}$ be two strings over an alphabet $\Sigma$ of size $\sigma$ (here we assume that $\sigma$ is not a constant). We define an array $C$ as follows:

$$C(i) = \sum_{k=0}^{n−1} \text{equal}(a_{i+k}, b_k), \quad i \in \{0, 1, \ldots, n\},$$

where $\text{equal}(\alpha, \beta)$ is 1 if $\alpha = \beta$, zero otherwise. For example, if $t = \text{abracada}$ and $p = \text{abaa}$ then $C(0) = 3, C(1) = 1, C(2) = 1, C(3) = 2, C(4) = 1$. Note that $C(i)$ gives the number of matched symbols when $p$ is aligned with the text at position $i$.

For each $\gamma \in \Sigma$ let us define $C_\gamma(i) = \sum_{k=0}^{n−1} \text{equal}_\gamma(a_{i+k}, b_k)$ for $i \in \{0, 1, \ldots, n\}$, where $\text{equal}_\gamma(\alpha, \beta)$ is 1 if $\alpha = \beta = \gamma$, zero otherwise. Give a $O(n \log n)$ algorithm to compute one of the arrays $C_\gamma$ based on convolution. This implies an efficient algorithm to compute $C$. What is the complexity as a function of $n$ and $\sigma$?

**Answer:**
Problem 2. (10 points)

You are given a list of \( n \) jobs \( j_1, j_2, \ldots, j_n \) each with a time \( t(j) \) to perform a job and a weight \( w(j) \). We want to order the jobs as \( j_{\sigma(1)}, j_{\sigma(2)}, \ldots j_{\sigma(n)} \) in such a way that the quantity

\[
\sum_{i=1}^{n} \left( w(j_{\sigma(i)}) \sum_{k=1}^{i} t(j_{\sigma(k)}) \right)
\]

is minimized. In other words, we want to minimize the weighted sum of the time each job has to wait before being executed. Give an efficient greedy algorithm for this problem, analyze the complexity, and prove its optimality.

Answer:
Problem 3. (10 points)

Consider the following **REVERSE-DELETE** algorithm for the Minimum Spanning Tree (MST) problem. Start with a connected edge-weighted graph \( G = (V, E) \). Consider edges in order of decreasing weight (breaking ties arbitrarily). When considering an edge \( e \), delete \( e \) from \( E \) unless doing so would disconnect the current graph. Analyze the complexity of **REVERSE-DELETE** and prove that it computes an (optimal) MST.

Answer: