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)

A $N \times N$ matrix is called Toeplitz if it has constant entries down its diagonals. For example

$$A = \begin{bmatrix}
    a_0 & a_1 & a_2 & a_3 & \ldots & a_{N-1} \\
    a_{-1} & a_0 & a_1 & a_2 & \ldots & : \\
    a_{-2} & a_{-1} & a_0 & a_1 & \ldots & a_3 \\
    a_{-3} & a_{-2} & a_{-1} & a_0 & \ldots & a_2 \\
    & & & & & \\
    a_{-(N-1)} & \cdots & a_{-3} & a_{-2} & a_{-1} & a_0
\end{bmatrix}$$

Toeplitz matrices occur surprisingly often in several application domains (e.g. time-series analysis, the numerical solution of certain partial differential equations, approximation of functions, etc.).

1. Is the sum of two Toeplitz matrices necessarily Toeplitz? What about the product?

2. Describe how to represent a Toeplitz matrix so that two $n \times n$ Toeplitz matrices can be added in $O(n)$ time.

3. Give a $O(n \log n)$-time algorithm for multiplying an $n \times n$ Toeplitz matrix by a vector of length $n$

Answer:
Problem 2. (10 points)

Suppose you are given a set \( T = \{(s_1, f_1), \ldots, (s_n, f_n)\} \) of \( n \) tasks, where each task \( i \) is defined by the start time \( s_i \) and a finish time \( f_i \). Two tasks \( t_i \) and \( t_j \) are non-conflicting if \( f_i \leq s_j \) or \( f_j \leq s_i \). The activity selection problem asks for the largest number of tasks can be scheduled in a non-conflicting way. The greedy algorithm explained in class considers tasks one by one ordered by the finish time. Ordering by increasing finish time is crucial to prove that this strategy always leads to the optimal solution.

Does the following greedy algorithm compute the optimal solution for activity selection?

1. Compute the number of overlaps for each task
2. Sort the task by the number of overlaps, in increasing order (break ties arbitrarily)
3. Pick the task \( i \) with the smallest number of overlaps, schedule it, and remove from further consideration tasks that are overlapping with \( i \)
4. Repeat step 3 until all tasks are scheduled

Note that the number of overlaps is NOT updated after step 1. If you think the strategy works, prove that the greedy choice property hold. If not, show an example where this strategy gives a suboptimal solution.

Answer:
Problem 3. (10 points)

A server has \( n \) customer waiting to be served. The service time required by each customer is known in advance: it is \( t_i \) minutes for customer \( i \). So if, for example, the customers are served in order of increasing \( i \), then the \( i \)-th customer has to wait \( \sum_{j=1}^{i} t_j \) minutes. We want to minimize the total waiting time:

\[
T = \sum_{i=1}^{n} (\text{time spent waiting by customer } i)
\]

Give a greedy (efficient) algorithm for computing the optimal order in which to process the customers. Prove the correctness of your algorithm by showing greedy choice and optimal substructure.

Answer: