Please read these notes

- This exam is closed book, closed notes and 50 minutes long
- Read the questions carefully
- No electronic equipment allowed (smart phones, tablets, computers, …)
- Write legibly and try to be brief and to the point; what can’t be read will not be graded
- No code: use pseudocode or English to describe your algorithms
- When designing an algorithm, you are allowed to use any algorithm or data structure we explained in class or covered in a CS 218 homework or exam, without giving its details, unless the question specifically requires that you give such details
- Always remember to analyze the time complexity of your solution
- If you have a question, please raise your hand
**Problem 1.** [Greedy] (25 points)

Prove (or disprove) the following statement on the optimal substructure of any shortest path on a weighted graph $G = (V, E)$. **Fact:** Let $p = \{e_1, e_2, \ldots, e_{k-1}\}$ be the shortest path from $v_1 \in V$ to $v_k \in V$ composed of the following $k - 1$ edges $e_1 = (v_1, v_2), e_2 = (v_2, v_3), \ldots, e_{k-1} = (v_{k-1}, v_k)$. Then, $\{e_i, e_{i+1}, \ldots, e_j\}$ must be the shortest path from $v_i$ to $v_{j+1}$ for all choices of $1 \leq i < j < k$. 
Problem 2. [Greedy] (25 points)
You are given two unsorted arrays \( A = \{a_1, a_2, \ldots, a_n\} \) and \( B = \{b_1, b_2, \ldots, b_n\} \) composed of distinct positive integers.

- Give a \( O(n \log n) \)-time greedy algorithm that determines an ordering of the elements of \( A \) and \( B \) such that \( W = \prod_{i=1}^{n} a_i b_i \) is maximized
- Explain why your algorithm runs in \( O(n \log n) \)-time
- Prove the greedy choice property for your algorithm; no need to prove the optimal substructure.
Problem 3. [Dynamic Programming][Design] (25 points)

You are given a directed graph $G = (V, E)$, two vertices $s$ and $t$, and an integer $k$. We want to compute the number of paths in $G$ from $s$ to $t$ that have exactly $k$ edges. The path does not have to be simple, i.e., vertices can be used more than once. Give a dynamic-programming algorithm that runs in time $O((n + m)k)$. Analyze the time- and space-complexity of your algorithm.

**Hint:** Define $M[v, i] = \text{the number of paths from } s \text{ to a vertex } v \in V \text{ that have exactly } i \text{ edges, } 0 \leq i \leq k$.

The recurrence relation is

$$M[v, i] = \begin{cases} 
\text{if } i > 0 \\
\text{if } v = t \text{ and } i = 0 \\
\text{if } v \neq t \text{ and } i = 0 
\end{cases}$$

The space-complexity of this algorithm is

The time-complexity of this algorithm is
Problem 4. [Dynamic Programming][Knowledge] (25 points)

We want to extend the LCS dynamic programming algorithm we covered in class to find the longest common subsequence between three strings $X$, $Y$ and $Z$.

Let $X_i$ be a prefix of string $X$ of length $i$, $Y_j$ be a prefix of string $Y$ of length $j$, and $Z_k$ be a prefix of string $Z$ of length $k$. If we define $C[i, j, k]$ to store the length of the longest common subsequence between $X_i$, $Y_j$ and $Z_k$, then

$$C[i, j, k] = \begin{cases} 
\text{if } i = 0 \text{ or } j = 0 \text{ or } k = 0 & \text{if } i = 0 \text{ or } j = 0 \text{ or } k = 0 \\
\text{if } x_i = y_j \text{ and } x_i = z_k & \text{if } x_i = y_j \text{ and } x_i = z_k \\
C[i-1, j-1, k-1] + 1 & \text{if } x_i = y_j \text{ and } x_i = z_k
\end{cases}$$

The time complexity of this algorithm is

The space complexity of this algorithm is