Problem 1. (30 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 $f_i$. Each task has to be performed on one machine, and each machine can execute only one task at a time. Two tasks $t_i$ and $t_j$ are non-conflicting if $f_i \leq s_j$ or $f_j \leq s_i$. The task scheduling problem asks for the minimum number of machines that can schedule all the given tasks in a non-conflicting way. The greedy algorithm explain in class, assigns the tasks to the machines by considering the tasks one by one ordered by the finish time. The ordering by the 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 task scheduling?

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.
Problem 2. (35 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.

Problem 3. (35 points)

Let \( G = (V, E) \) be a weighted undirected graph. We define the bottleneck of a path \( p \) in \( G \) as the minimum weight of any edge on \( p \). We define the maximum bottleneck of any \( s, t \)-path as the maximum, over all paths \( p \) from \( s \) to \( t \), of the bottleneck of \( p \).

Prove or disprove: given any connected, undirected, edge-weighted graph, algorithm BOTTLENECK below produces a tree \( T \) such that the bottleneck of the path \( p \) from \( s \) to \( t \) in \( T \) is the maximum bottleneck of any \( s, t \)-path in the original graph.

Algorithm BOTTLENECK \((G(V, E) : graph)\)

\[
\begin{align*}
\text{sort } & \text{ the edges } e_1, e_2, \ldots, e_m \text{ in order of decreasing cost} \\
T & \leftarrow \emptyset \\
\text{for } & i \leftarrow 1, 2, \ldots, m \text{ do} \\
& \text{ Add } e_i \text{ to } T \text{ if doing so does not create a cycle in } T \\
\text{return } & T
\end{align*}
\]