For problems 3, 4 and 5, you should:

- Give a polynomial time algorithm to solve the problem (using pseudocode, or other sufficiently precise and readable formalism).
- Prove that your algorithm is correct, i.e., it produces an optimal solution.
- Analyze the running time of your algorithm, showing that it runs in polynomial time.
- Briefly describe the data structures needed to implement your algorithm and their performance. (For well known data structures, it is sufficient to provide its name, e.g., “priority queue”, or “linked list”, etc.) Then, give a more precise bound on the running time of your algorithm when implemented your chosen data structures. (Make a good choice of data structure for full credit.)

Problem 1 (KT 4.2a)

Suppose we are given an instance of the Minimum Spanning Tree problem on a graph $G$, with edge costs $c(e)$ that are all positive and distinct. Let $T$ be a minimum spanning tree for this instance. Now, suppose we replace each edge cost $c(e)$ by its square $c(e)^2$, thereby creating a new instance of the problem with the same graph but different costs. Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counterexample.

"$T$ must still be a minimum spanning tree for this new instance."

Problem 2 (KT 4.2b)

Suppose we are given an instance of the Shortest $s-t$ Path problem on a directed graph $G$, with edge costs $c(e)$ that are all positive and distinct. Let $P$ be a minimum cost $s-t$ path for this instance. Now, suppose we replace each edge cost $c(e)$ by its square $c(e)^2$, thereby creating a new instance of the problem with the same graph but different costs. Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counterexample.

"$P$ must still be a minimum cost $s-t$ path for this new instance."

Problem 3 (KT 17)

Consider the following variant of the Interval Scheduling Problem "on a circle". You have a processor that can operate 24 hours a day, every day. People submit requests to run daily jobs on the processor. Each such job comes with a start time and an end time; if the job is accepted to run on
the processor, it must run continuously, every day, for the period between its start and end times.
(Note that certain jobs can begin before midnight and end after midnight of the following day.)

Given a list of \( n \) such jobs, your goal is to accept as many jobs as possible (regardless of their
length), subject to the constraint that the processor can run at most one job at any given point in
time. You may assume for simplicity that no two jobs have the same start and end times. Provide
an algorithm to solve this problem in time polynomial in \( n \).

**Problem 4**

Consider the following problem. You are designing a business plan for a start-up company. You have
identified \( n \) possible projects for your company, and, for \( i = 1, \ldots, n \), let \( c_i > 0 \) be the minimum
capital required to start the project \( i \), and \( p_i > 0 \) be the profit after the project is completed. You
also know your initial capital \( C_0 > 0 \). You want to perform at most \( k \leq n \) projects before the
IPO and want to maximize your total capital at the IPO. Your company cannot perform the same
project twice. In other words, you want to pick a list of up to \( k' \) distinct projects, \( i_1, \ldots, i_{k'} \), with
\( k' \leq k \).

Your accumulated capital after completing the project \( i_j \) will be \( C_j = C_0 + \sum_{h=1}^{j} p_h \). The
sequence must satisfy the constraint that you have sufficient capital to start the project \( i_{j+1} \) after
completing the first \( j \) projects, i.e., \( C_j \geq c_{i_{j+1}} \) for \( j = 1, \ldots, k' \). You want to maximize the final
amount of capital \( C_{k'} \). Give an algorithm to find an optimal solution.

**Problem 5**

Consider the following problem. You have a collection of \( n \) tasks/jobs \( (t_1, \ldots, t_n) \) and \( n \) ma-
chines/workers \( w_1, \ldots, w_n \). Each task requires a certain amount of work \( t_i > 0 \), and each machine
can perform a certain amount of work \( w_i > 0 \) per unit of time. Each machine/worker can assigned
to at most one task, and each task, if executed, must be performed on a single machine. You want
to maximize the number of tasks executed, under the constraint that they are completed within a
given deadline \( D \), i.e., you can assign task \( t_i \) to machine \( w_j \) if \( t_i / w_j \leq D \). Give an algorithm to
find an optimal assignment.