Algorithms. Assignment 3

Problem 1

Here is a "lazy" version of the Interval Scheduling problem. A set J of n jobs is given, specified by their start times and end times. That is, jobs are given as intervals $[s_i, f_i]$, $i = 1, \ldots, n$.
A worker has to select a subset $X \subseteq J$; these are the jobs he wants to do. Every job in X must be done exactly during its prescribed time interval, and all jobs in X must have pairwise disjoint time intervals.
But the worker is lazy and wants to do as few jobs as possible, that is, minimize the number $|X|$ rather than maximizing it! However the worker cannot simply take $X = \emptyset$, since there is yet another condition to fulfill:

(*) There remains no interval $y \in J$ outside X which is disjoint to all intervals in X (and could therefore be added to the solution X).

Roughly speaking, the worker wants to appear busy all the time while minimizing the actual number of jobs assigned to him. (This goal might appear unethical, but there may be better applications of the same computational problem.)

Now the following greedy algorithm for the lazy worker comes to mind, inspired by Earliest End First for the original Interval Scheduling problem: Let z be the interval with the earliest end. Let $Z \subseteq J$ be the set consisting of z and all intervals that intersect z. It should be clear that some interval from Z must be selected, in order to fulfill the condition (*). Therefore, we select the interval $x \in Z$ with the latest end, and put it in X. (The idea is that x blocks the maximum number of further intervals.) Then, we remove x and all intersecting intervals from the instance J. We apply this rule iteratively, until the instance is empty.

However, it is not that easy to be lazy: Explain why the following instance is a counterexample for the proposed greedy algorithm.
$[1, 2]$, $[1, 4]$, $[3, 8]$, $[5, 6]$, $[7, 8]$.
What is the optimal solution, and what would the greedy algorithm do?

Problem 2

As seen in the warm-up Problem 1, a natural greedy strategy fails. Now we come to the real stuff: your task is to develop a dynamic programming algorithm for the lazy worker problem.

What you should do in detail:

(a) First *define* (not compute!) the objectve function OPT that you intend to use for dynamic programming. It is natural to define a function that expresses how many jobs are completed before a certain time. Still you need to be precise in the details.
Bear in mind that a complete, precise and unambiguous definition is crucial, because the following steps depend on that.

(b) Now give a "recursive" formula for the computation of your OPT function defined in (a).

(c) Explain the correctness of your formula from (b).
Basically, you must argue that no case that may lead to an optimal solution is forgotten.

(d) Analyze the time of the resulting algorithm.
It might be necessary to invoke auxiliary functions or suitable data structures, in order to to achieve a good time bound. In any case, the time must be polynomial in n; superpolynomial bounds are not acceptable. It is also important to remember that dynamic programming works with memoization of optimal values and backtraing,rather than with recursion, even though the OPT formula "looks recursive"!

Some more advice:
It may happen that you realize at some point that your function is inconvenient to compute. In this case, just revise your definition and try again. And do not hesitate to ask for help if you totally get stuck.
The correct algorithm is not unique; there may well exist several ones with different time bounds.