3. [5 points each for parts (a) and (b)]

   Recall the job scheduling problem from Homework 2: Suppose n jobs, numbered $1, \ldots, n$ are to be scheduled on a single machine. Job j takes t_j time units, has an integer deadline d_j, and a profit p_j. The machine can process only one job at a time and each job must be processed without interruption (for its t_j time units). A job j that finishes before its deadline d_j receives profit p_j, while a tardy job receives no profit (and might as well not have been scheduled). Except now do not put any restriction on the number of time units for a job (except that these are positive integers). The objective (of the optimization version) is to come up with a maximum profit set of jobs that can all be scheduled by their deadlines

   (a) Formulate this problem as a decision problem and prove that the decision problem can be solved in polynomial time if and only if the optimization problem can be solved in polynomial time (using the evaluation and certificate versions as intermediaries).

   (b) Prove the the decision problem of part (a), call it **Max Profit Scheduling (MPS)**, is NP-Complete.

Answer:

(a) The decision version of the *Max Profit Scheduling Problem* is as follows -

Given n jobs, numbered $1 \ldots n$, with t_j time units, d_j integer deadlines, p_j positive profit values and a profit value k,
does there exist a set of jobs which when scheduled for the stipulated time within their deadlines yield an overall profit of atleast k?

To prove the biconditional statement : decision problem can be solved in polynomial time iff the optimization version can, we break it into two parts.

Part 1: If optimization problem can be solved in polynomial time, then the decision version can be solved in polynomial time.
From the relationship among problem types, we know that solving an optimization problem yields solution to all the others.

Thus having a polynomial time optimization problem that determines the maximum profit set of jobs, we can evaluate the maximum profit in polynomial time.
Based on this value, we can decide the decision problem's answer, thus solving it in polynomial time.

Part 2: If the decision problem can be solved in polynomial time, then the optimization version can be solved in polynomial time.

To prove polynomial time optimization problem, we need two things:
- polynomial time decision algorithm (*given*)
- polynomial time certificate algorithm

Let MPS-DEC(T,P,D,k) be the boolean-valued decision algorithm.
Consider array $K \ni k_i = \sum p_i$.

Then we have the following certificate algorithm -

```
MPS-CERT(T,P,D,K,low,high):
        if high = low:
                return K[low]

        mid = ⌊(high − low)/2⌋

        if MPS-DEC(T,P,D,K[mid]) = True:
                MPS-CERT(T,P,D,K,mid,high)
        else:
                MPS-CERT(T,P,D,K,low,mid−1)
end
```

The above certificate algorithm (which is similar to *binary search*) will take polynomial time to compute the max profit.
Since both decision and certificate algorithms are polynomial time, the optimization problem can also be solved in polynomial time.

(b) To prove that *Max Profit Scheduling* is NP complete, we need to prove the following -

(1) Prove that $MPS \in NP$

For the decision problem mentioned in (a), let S be the candidate solution.
S is a boolean array, which indicates that the process i is scheduled if $S_i = 1$.
Given the candidate solution S, we can easily find its profit in O(n), and check if the profit $\leq k$.
Hence, the problem is in NP.

(2) Prove that some problem (say SUBSET-SUM) in NPC is polynomial time reducible to *MPS* i.e. SUBSET-SUM $\leq_p$ *MPS*

(a) `Mapping` f `that converts instance of SUBSET-SUM into instance of` *MPS*
The SUBSET-SUM problem is modeled as $X = \{x[1 \dots n], sum\}$
This can be expressed as an *MPS* problem as follows -
$T = P = x[1, \dots, n] \qquad \forall i \; d[i] = sum,$ and $\; k = sum$
This can be done in O(n). Hence you can reduce SUBSET-SUM to *MPS* in polynomial time.

(b) `Prove that certificate for SUBSET-SUM implies certificate for` *MPS* i.e. if there exists a solution for SUBSET-SUM, then the *MPS* instance is solvable as well
If there exists a solution for SUBSET-SUM, such that the variable sum can be obtained, then -
Since, $T = P = x[1, \dots, n] \qquad \forall i \; d[i] = sum$, there will exist a solution for *MPS* as well.
And if there isn't a solution for the SUBSET-SUM problem, then there isn't a solution for its *MPS* version problem as well.

Consider the following example -
$x = \{1, 4, 5, 10\} \qquad\qquad sum = 16$
$\therefore Solution = \{1, 5, 10\}$

This problem will be expressed as *MPS* as :
$T = [1, 4, 5, 10]$
$D = [16, 16, 16, 16]$
$P = [1, 4, 5, 10]$
$k = 16$
The solution for this will be $s = [1, 0, 1, 1]$ which is the same as the solution for the SUBSET-SUM problem.

However if $sum = 17$ for the SUBSET-SUM problem, there is no possible solution. And even its corresponding *MPS* problem does not have a solution.

Q.E.D.

(c) `Prove that certificate for` *MPS* `implies certificate for SUBSET-SUM` i.e. if there exists a solution for *MPS*, then the SUBSET-SUM instance is solvable as well
The *MPS* problem can be expressed as a SUBSET-SUM problem with $x = P$ and $sum = k$.
Hence, solution for *MPS* implies a solution for SUBSET-SUM.

Thus, the given decision problem in (a) is in *NPC* class.