Recall that a dynamic programming algorithm to solve a given problem $P$ involves the following elements:

(a) A definition of a polynomial number of subproblems that will be solved (and from whose solution we will compute the solution to $P$ — see (b) below).

(b) A recursive formula to compute the solution to each subproblem from the solutions to smaller subproblems. This induces a partial order on the subproblems defined in (a).

(c) A way to compute the solution to $P$ from the solutions to the subproblems computed in (b).

Proving the correctness of a dynamic programming algorithm amounts to justifying (i) why the recursive formula in step (b) correctly computes the subproblems defined in step (a), and (ii) why the computation in (c) yields a solution to the given problem. Part (ii) is often immediate from the definition of the subproblems.

Question 1. (15 marks) Let $A = a_1, a_2, \ldots, a_n$ be a sequence of (negative, zero, or positive) integers, $n \geq 1$. A subsequence $A'$ of $A$ is nonconsecutive if it does not contain consecutive elements of $A$; more
precisely $A' = a_{i_1}, a_{i_2}, \ldots, a_{i_k}$, where $1 \leq i_1 < i_2 < \ldots < i_k \leq n$ and for every $t \in [2..k]$, $i_t \not= i_{t-1} + 1$. Give a linear-time dynamic programming algorithm that, given $A$, returns a nonconsecutive subsequence of $A$ whose elements have the maximum possible sum. (The sum of the elements of an empty sequence is defined to be zero.)

Describe your algorithm in pseudocode, explain why it is correct, and analyze its running time.

**Question 2.** (15 marks) We have two identical machines in which to execute $n$ jobs, numbered $1, 2, \ldots, n$. Each machine can execute one job at a time, and once a job is started on a machine, it runs to completion on that machine. Each job $i$ requires time $t_i$ on either machine, where $t_i$ is a positive integer. An assignment $A$ allocates to each machine a subset of the jobs; formally $A$ is simply a subset of $\{1, 2, \ldots, n\}$, interpreted as the set of jobs allocated to Machine 1 to execute, and the remaining jobs $A = \{1, 2, \ldots, n\} - A$ are allocated to Machine 2 to execute. Given assignment $A$, the load of each machine is the total amount of time required by the jobs allocated to that machine; i.e., the load of Machine 1 (resp. Machine 2) is $\sum_{i \in A} t_i$ (resp. $\sum_{i \notin A} t_i$). The maximum of these two quantities is the time by which all $n$ jobs have been processed by the two machines in assignment $A$, and is called the makespan of $A$. We want to find an assignment that has minimum makespan.

Describe an $O(nT)$ dynamic programming algorithm that, given the times $t_1, t_2, \ldots, t_n$ required by the $n$ jobs, determines an assignment with minimum makespan, where $T = \sum_{i=1}^{n} t_i$. Describe your algorithm in pseudocode, explain why it is correct, and analyze its running time.

**Note:** Such an algorithm is not a polynomial-time one; make sure you understand why. If you do come up with a correct polynomial-time algorithm for this problem, don’t keep it secret. You may have just won the one-million dollar (US) Clay Institute prize for settling the P vs. NP question!

**Hint:** The following question has an easy, but useful, answer: If you know that the load on Machine 1 under an assignment is $t$, what is the load on Machine 2 under that assignment? Also note that not every $t \in [1..T]$ is a possible load for a machine.

**Question 3.** (10 marks) Let $G = (V, E)$ be a directed graph and $wt : E \rightarrow \mathbb{Z}$ be an edge-weight function, such that every cycle of $G$ has positive weight (but note that the graph may have negative- or zero-weight edges). Modify the Floyd-Warshall algorithm to obtain an $O(n^3)$ algorithm that, given $G$ and $wt$, returns the array $N[u, v]$ for each pair of nodes $u, v \in V$ such that $N[u, v]$ is the number of minimum-weight $u \rightarrow v$ paths. Justify the correctness of your algorithm. (You don’t need to analyze its running time, but it should be $O(n^3)$.)

**Note:** We want all cycles to have positive weight (rather than merely non-negative) weight because with zero-weight cycles the minimum weight of $u \rightarrow v$ paths is well-defined but the number such paths may be infinite.