1. The 0/1 knapsack problem is described as follows. Given the capacity m of a knapsack and n objects whose weights are $w_1,\ldots,w_n$ and whose profits are $p_1,\ldots,p_n$, find the largest value of $\sum_{1\le i\le n} p_i x_i$ by assigning either 0 or 1 to $x_1,\ldots,x_n$ under the constraint $\sum_{1\le i\le n} w_i x_i \le m$, where $w_1,\ldots,w_n$ and $p_1,..,p_n$ are positive integers. Write a dynamic programming algorithm to solve the 0/1 knapsack problem with the time complexity $O(n\cdot m)$. You should analyze your algorithm to show that its time complexity is indeed $O(n\cdot m)$. (25%)

2. A non-deterministic (ND) algorithm has two phases, the choosing phase and the checking phase, for solving a given problem. The former is for selecting one from a specific set of choices iteration by iteration. The latter is for checking if all selected choices constitute a solution to the problem. If so, the algorithm returns SUCCESS; otherwise, FAILURE. It is assumed that an ND algorithm always selects choices that lead to the return of SUCCESS unless there are no such choices. A problem is called an NP problem if there exists a polynomial time-complexity ND algorithm solving the problem. For example, the famous satisfiability (SAT) problem is an NP problem. The SAT problem is to determine if a given Boolean formula $f(x_1,\ldots,x_n)$ of n Boolean variables $x_1,\ldots,x_n$ is satisfiable or unsatisfiable. A formula $f(x_1,\ldots,x_n)$ is satisfiable (resp., unsatisfiable) if there exists an (resp., no) TRUE-FALSE assignment of the n variables to make the formula TRUE. The following polynomial time-complexity ND algorithm, called ND-SAT, can solve the SAT problem, which is the evidence that the SAT problem is an NP problem.

Algorithm: ND-SAT
Input: a Boolean formula $f(x_1,\ldots,x_n)$ of n variables $x_1,\ldots,x_n$
Output: SUCCESS if f is satisfiable; **FAILURE**, otherwise.
for $i \leftarrow 1$ to n **do**
    $x_i \leftarrow$ choice({TRUE, FALSE}) //Choose TRUE or FALSE to assign to x_i
if $f(x_1,\ldots,x_n)$ == TRUE **then** //Check if $f(x_1,\ldots,x_n)$ is satisfiable or unsatisfiable
    return SUCCESS
else
    return FAILURE

In practice, we can prove a problem to be an NP problem by showing a polynomial time-complexity ND algorithm solving the problem. (a) By this concept, please prove that the exact cover decision problem (ECDP) is an NP problem by showing a polynomial time-complexity ND algorithm solving the ECDP (18%). Note that you should follow the above-mentioned ND algorithm definition and the format of the ND-SAT algorithm. That is, the ND algorithm should contain the input description, the output description, the choosing phase, the checking phase, and return statements; otherwise, you will lose some points. (b) Furthermore, please analyze the time complexity of your ND algorithm in terms of the big O notation to show that it indeed

has a polynomial time complexity (7%). The ECDP is defined as follows. Given a universal set $U=\{u_1,...,u_m\}$ of m elements, and a collection $S=\{S_1,...,S_n\}$ of n sets, where S_i is a non-empty subset of U, $1\le i\le n$, the ECDP is to determine if there exists a collection S^* of sets that is an exact cover of U, where $S^*\subseteq S$. A collection S^* of sets is an exact cover of U if every element u in U appears exactly once in only one set of S^*. For example, suppose $U=\{1,2,3,4,5,6,7\}$ is a universal set of seven elements, and $S=\{A,B,C,D,E\}$ is a collection of five sets, where $A=\{1,2,7\}$, $B=\{1,4\}$, $C=\{4,5\}$, $D=\{3,5,6\}$, and $E=\{4\}$. Then, $S^*=\{A,D,E\}\subseteq S$ is an exact cover of U. In summary, the ECDP with the input of $U=\{1,2,3,4,5,6,7\}$ and $S=\{A=\{1,2,7\}, B=\{1,4\}, C=\{4,5\}, D=\{3,5,6\}, E=\{4\}\}$ will return SUCCESS, since $S^*=\{A,D,E\}\subseteq S$ is an exact cover of U.

3. You have inherited the publishing rights to n songs by the Raucous Rockers. You want to release a boxed set of d compact disks, each of which can hold at most m minutes of music. To satisfy the fans, you must put the songs in chronological order, but you can omit songs (regardless of when they were recorded) if necessary. Of course, no song can be split across a disk. Given a list of the song lengths in chronological order, your task is to figure out the maximum number of songs that can be recorded on the set of disks subject to these criteria. (25%)

4. Given a sequence of objects where each object is associated with a value and a weight, design an algorithm to find a subsequence such that its corresponding value sequence is increasing and its weights' sum is maximized. (25%)