1. (20 Points) Give an adversary argument showing that at least \( \binom{n}{2} \) adjacency questions are necessary (in worst case) to determine whether a graph \( G \) on \( n \) vertices is connected. (Recall by “adjacency” question we mean a question of the form: “is vertex \( u \) adjacent to vertex \( v \”).)

2. (20 Points) Give an adversary argument showing that at least \( \binom{n}{2} \) adjacency questions are necessary (in worst case) to determine whether a graph \( G \) on \( n \) vertices is acyclic.

3. (20 Points) Recall the coin changing algorithm described in class:

\[
\text{CoinChange}(d, N) \quad (\text{Pre: An unlimited supply of coins in denominations } d[1..n] \text{ are available})
\]

1. \( n \leftarrow \text{length}[d] \)
2. for \( i \leftarrow 1 \) to \( n \)
3. \( C[i,0] \leftarrow 0 \)
4. for \( i \leftarrow 1 \) to \( n \)
   for \( j \leftarrow 1 \) to \( N \)
   if \( i = 1 \) and \( j < d[i] \)
      \( C[1, j] \leftarrow \infty \)
   else if \( i = 1 \)
      \( C[1, j] \leftarrow 1 + C[1, j-d[i]] \)
   else if \( j < d[i] \)
      \( C[i, j] \leftarrow C[i-1, j] \)
   else
      \( C[i, j] \leftarrow \min(C[i-1, j], 1 + C[i, j-d[i]]) \)
14. return \( C[n,N] \)

a. (10 Points) Modify this algorithm to allow for the possibility that the supply of coins in some denominations is limited. Let \( L[1..n] \) be an input array giving the limits on each denomination, i.e. at most \( L[i] \) coins of denomination \( d[i] \) are to be used \( (1 \leq i \leq n) \). Note that \( L[i] \) may be 0 or \( \infty \).

b. (10 Points) Write a recursive algorithm which given the filled table \( C[1..n; 0..N] \) prints out a sequence of \( C[n,N] \) coin values which disburse \( N \) monetary units. In the case \( C[n,N] = \infty \), print a message to the effect that no such disbursal is possible.
4. (20 Points) Recall the 0-1 Knapsack Problem described in class. A thief wishes to steal $n$ objects having values $v_i > 0$ and weights $w_i > 0$ ($1 \leq i \leq n$). His knapsack, which will carry the stolen goods, holds at most a total weight $W$. Let $x_i \in \{0,1\}$ to be the indicator variable for this problem, i.e. for $1 \leq i \leq n$, $x_i = 1$ if object $i$ is taken, $x_i = 0$ if object $i$ is not taken. The thief’s goal is then to maximize the total value $\sum_{i=1}^{n} x_i v_i$, subject to the capacity constraint $\sum_{i=1}^{n} x_i w_i \leq W$.

a. (10 Points) Write pseudo-code for a dynamic programming algorithm which solves this problem. Your algorithm should take as input the value and weight arrays $v[1..n]$ and $w[1..n]$, and the weight limit $W$. It should generate a table $V[1..n; 0..W]$ of intermediate results. Each entry $V[i, j]$ will be the maximum value of the objects which can be stolen if the weight limit is $j$, $0 \leq j \leq W$, and if we only include objects in the set $\{1,\ldots,i\}$, $1 \leq i \leq n$. Your algorithm should return the maximum possible value of the goods which can be stolen from the full set of objects, i.e. the value $V[n,W]$. (Alternatively you may write your algorithm to return the whole table.)

b. (10 Points) Write an algorithm which given the filled table generated in part (a), prints out a list of exactly which objects are to be stolen.