1. Consider a DP algorithm which takes as input an array $A$ of $n$ elements. Subproblems of the given problem are contiguous subarrays $[A_i, \ldots, A_j]$ where $1 \leq i \leq j \leq n$. The DP algorithm computes solutions to all possible subproblems before it returns the desired result for the given problem $[A_1, \ldots, A_n]$. The number of subproblems for which solutions are computed by the DP algorithm is then: (Answer: $\Theta(n^2)$)

(a) $\Theta(\log n)$
(b) $\Theta(n \log n)$
(c) $\Theta(n^2)$
(d) $\Theta(n^3)$

Solution
$\Theta(n^2)$. $i$ and $j$ can each vary in the range $[1 \ldots n]$, and hence there are possibly $n \times n$ different subproblems of the kind $[A_i, \ldots, A_j]$. As $i \leq j$ always, we only have to compute approximately $1/2$ of the entries in a table of size $n \times n$. Since $1/2$ is a constant factor, this is still $\Theta(n^2)$.

2. What is the worst-case running time of the DP algorithm for String Reconstruction presented in lecture if we know that the maximum length of any word in the dictionary is $k$, and we implement the DP algorithm to take advantage of this fact? (Answer: $\Theta(kn)$)

(a) $\Theta(kn^2)$
(b) $\Theta(n \log n)$
(c) $\Theta(kn)$
(d) $\Theta(\sqrt{n})$

Solution
The DP algorithm formulates problem recursively using subproblems. The number of subproblems is $n$ and the maximum length of any
word is $k$. The algorithm looks up the dictionary in $\Theta(k)$ for given $n$ subproblems. Therefore, the worst-case running time of the String Reconstruction problem is $\Theta(kn)$.

3. Which of the following algorithms is not Dynamic Programming based? (Answer: Prim’s algorithm)
   (a) Computing the binomial coefficient $C(n,k)$ using Pascal’s Triangle
   (b) Floyd-Warshall algorithm for all-pairs shortest paths
   (c) Bellman-Ford algorithm for single-source shortest paths
   (d) Prim’s algorithm for minimum spanning tree

   **Solution**

   Prim’s algorithm for minimum spanning tree is a greedy algorithm, not a DP algorithm.

4. In the problem of finding the Longest Common Subsequence in two strings $X[1 \ldots m]$ and $Y[1 \ldots n]$ as described in lecture, each subproblem seeks a subsolution $c[i,j]$, the length of the LCS of $X[1 \ldots i]$ and $Y[1 \ldots j]$. The recurrence relation given in lecture is:

   $$c[i,j] = \begin{cases} c[i-1,j-1] + 1 & \text{if } X[i] = Y[j] \\ \max\{c[i,j-1], c[i-1,j]\} & \text{otherwise} \end{cases}$$

   Consider the table of subsolutions computed by the algorithm. What are the initial conditions for this recurrence? (Answer: (c))
   (a) $c[0,0] = 0$, i.e., the top-left element of the table is 0.
   (b) $c[i,i] = 0 \ \forall i$, i.e., the diagonal elements of the table are 0.
   (c) $c[0,j] = 0 \ \forall j$ and $c[i,0] = 0 \ \forall i$, i.e., the 0th row and the 0th column of the table have all 0s.
   (d) $c[0,0] = c[0,1] = c[1,0] = 0$, i.e., the three top-left elements of the table are 0.

   **Solution**

   $c[0,j] = 0 \ \forall j$ and $c[i,0] = 0 \ \forall i$, i.e., the 0th row and the 0th column of the table have all 0s since the recurrence looks up only three previously-computed subproblems’ solutions.

5. Given $n$ types of coins with values $v_1, v_2, \ldots, v_n$ and a target value $C$. (You may assume $v_1 = 1$ so that it is always possible to find a set of coins that exactly achieves any given target value.) A DP algorithm that finds the smallest number of coins required to sum to $C$ is:

   $$M(j) = \begin{cases} 0, & \text{if } j = 0 \\ \min_{i=1,\ldots,n}\{M(j-v_i) + 1\} & \text{else} \end{cases}$$
(In this recurrence, \( M(j) \) is the minimum number of coins required to sum to the value \( j \).) What is the worst-case running time of the given algorithm? (Answer: \( \Theta(Cn) \))

(a) \( \Theta(Cn) \)
(b) \( \Theta(n^2) \)
(c) \( \Theta(Cn^2) \)
(d) \( \Theta(n \log n) \)

**Solution**

\( M \) has \( C \) elements and computing each element takes \( \Theta(n) \) time so the worst-case running time of the given algorithm is \( \Theta(Cn) \).

6. Given a sequence of \( n \) real numbers, \( a_1, a_2, \ldots, a_n \), a DP algorithm that finds the maximum sum of any contiguous (i.e., consecutively occurring) subsequence of elements in the given sequence is:

\[
M(j) = \begin{cases} 
  a_j, & \text{if } j = 1 \\
  \max\{M(j-1) + a_j, a_j\}, & \text{else} 
\end{cases}
\]

(In this recurrence, \( M(j) \) is the maximum sum of contiguous subsequence of elements found within the first \( j \) elements of the given sequence) What is the worst-case running time of the given algorithm? (Answer: \( \Theta(n) \))

(a) \( \Theta(n) \)
(b) \( \Theta(n^2) \)
(c) \( \Theta(n^3) \)
(d) \( \Theta(n \log n) \)

**Solution**

\( M \) is size \( n \) and evaluating each element of \( M \) takes \( \Theta(1) \) time for \( \Theta(n) \) time to create \( M \). Scanning \( M \) also takes \( \Theta(n) \) time for a total time of \( \Theta(n) \).

7. Consider the DP algorithm for the Knapsack problem as described in lecture. The subproblem is defined as \( F_k(y) \), the maximum value possible using only the first \( k \) item types, when the weight limit is \( y \). The recurrence relation in this problem is:

\[
F_k(y) = \max\{F_{k-1}(y), F_k(y - w_k) + v_k\}
\]

where \( k \) ranges from 1 to \( n \), the number of item types, and \( y \) ranges from 1 to \( b \), the weight limit of the knapsack. \( w_k \) and \( v_k \) indicate the weight and value of item type \( k \), respectively. Consider the array of subsolutions \( F \) filled in by the DP algorithm according to this recurrence. Which of the following best describes this array? (Answer: \( (c) \))
(a) The array $F$ is a one-dimensional array of size $b$ (not accounting for the 0th element), in which each value is calculated exactly once using $O(1)$ operations.

(b) The array $F$ is a one-dimensional array of size $n$ (not accounting for the 0th element), in which each value is updated $b$ times.

(c) The array $F$ is a two-dimensional array of dimensions $n \times b$ (not accounting for the 0th row and 0th column), in which each value is calculated exactly once using $O(1)$ operations.

(d) The array $F$ is a two-dimensional array of dimensions $n \times b$ (not accounting for the 0th row and 0th column), in which each value is updated $b$ times.

Solution
The array $F$ is a two-dimensional array of dimensions $n \times b$ (not accounting for the 0th row and 0th column), in which each value is calculated exactly once using $O(1)$ operations since the algorithm looks up two previously-computed subproblems’ solutions; $F_{k-1}(y)$ for the $k^{th}$ item not used and $F_k(y-w_k) + v_k$ for the $k^{th}$ item used once.

8. What is the definition of the subproblem in the DP algorithm for finding All-Pairs Shortest Paths? (Answer: (b))

(a) $c_{ij}^{(m)}$, the shortest distance between any two vertices $i$ and $j$ achieved by paths that use at most $m$ edges.

(b) $c_{ij}^{(m)}$, the shortest distance between any two vertices $i$ and $j$ achieved by paths that pass only through vertices in the set $\{1, 2, \ldots, m\}$.

(c) $c_{ij}$, the shortest distance between any two vertices $i$ and $j$ achieved by paths that use only vertices from an arbitrary set $S \subset V$.

Solution
(b) is the definition of the subproblem for the Floyd-Warshall dynamic programming algorithm.

9. Consider the problem of finding the Minimum Edit Distance between two strings $x[1 \ldots n]$ and $y[1 \ldots m]$. In the DP solution to this problem, the recurrence is formulated as:

$$E[i, j] = \min\{1 + E[i - 1, j], 1 + E[i, j - 1], \text{diff}(i, j) + E[i - 1, j - 1]\}$$

Based upon this recurrence, how much time does computing each subproblem require? (Answer: $\Theta(1)$)

(a) One subsolution requires $\Theta(1)$ to compute.

(b) One subsolution requires $\Theta(m)$ to compute.
(c) One subsolution requires $\Theta(n)$ to compute.

**Solution**

One subsolution requires $\Theta(1)$ to compute since the recurrence looks up only three previously-computed subproblems’ solutions.