1.

Programming approach

1. (a) A weighted directed graph G = (V, E), has the sets V = {1, 2, 3, 4, 5, 6, 7} of nodes and a set of ordered pair of directed edges E = {(1, 2), (1, 3), (2, 5), (3, 2), (3, 5), (4, 6), (5, 4), (6, 5)} with the following weights, or costs c(e); e ∈ E, of the edges:

e	(1,2)	(1,3)	(2,5)	(3,2)	(3,5)	(4,6)	(5,4)	(6,5)
c(e)	2	-1	-5	3	4	-4	2	3

Determine the shortest path from source vertex1 2
to sink vertex5 in the given weighted digraph G, using Bellman-Ford algorithm.

2.

4. (a) MATRIX-CHAIN-MULTIPLY(A, s, i, j)

```
if i == j
        return A[i]
if i + 1 == j
        return A[i] * A[j]
b = MATRIX-CHAIN-MULTIPLY(A, s, i, s[i, j])
c = MATRIX-CHAIN-MULTIPLY(A, s, s[i,j], j)
return b * c
```

The goal of the above recursive algorithm MATRIX-CHAIN-MULTIPLY(A,s,i,j) is to actually perform the optimal matrix-chain multiplication, given the sequence of matrices ⟨A1,A2,...,An⟩, and the "s" table computed by MATRIX-CHAIN-ORDER, and the indices i and j.

Find the error, if any, in the above pseudocode. Give a brief justification of your answer.

3.

2. A 0/1 knapsack problem can be defined as: given n items (cannot be splitted) of known weights $\langle w_1, \cdots, w_n \rangle$ and values/profits $\langle v_1, \cdots, v_n \rangle$, and a knapsack capacity W, then find the most valuable subset of items that fit into the knapsack assuming that $w_i \in Z^+ (1 \le i \le n)$, $W \in Z^+$, and $v_i \in R^+ (1 \le i \le n)$.

(a) Discover the "overlapping sub-problem(s)" property of the dynamic programming for the given knapsack problem. How many sub problems are to be solved to get the final answer when $\langle w_1, w_2, w_3, w_4 \rangle = \langle 7, 3, 4, 5 \rangle$, and $\langle v_1, v_2, v_3, v_4 \rangle = \langle \$42, \$12, \$40, \$25 \rangle$ with the knapsack capacity W = 10.

(b) Design a pseudocode that uses the concept of bottom-up dynamic programming to maximize the value/profit. Analyze the pseudocode to find its time and space complexity in the worst case scenario.

(c) Add a function actual_knapsack_items () to the pseudocode to find out the subset of items selected to fit into the knapsack that maximizes the value. Why the time complexity to find the actual knapsack items is O (n + W)? Justify your answer.

4.

(b) Give an optimal parenthesization to multiply a chain of matrices with dimension vector p = <5, 10, 7, 2, 3>. 2

(c) Describe the sub-problem graph for matrix-chain multiplication with an input chain of length n. How many vertices does it have? How many edges does it have, and which edges are they? 2

a) Give a brief comparison between different versions of Dynamic Programming(DP) strategy such as DP with binary choices, DP with muti-way choices, DP with addition of a variable and DP over intervals. 2

"Optimal substructure is one major property of both dynamic programming and greedy approach." Briefly explain the statement. (You may use example scenarios) 2

Critically compare dynamic programming with divide-and-conquer.

5. Compare the structure of the optimal solutions obtained using Dynamic Programming approach for Matrix chain multiplication and Weighted interval scheduling problem. Why the optimal solution of Matrix chain multiplication needs two variables while that of the weighted interval scheduling problem needs a single variable?
6. Give a fully parenthesization of a chain of matrices structured by the dimension vector P = <2, 4, 6, 8, 6>. Draw the subproblem graph and compare the total number of subproblems solved by:

a. Direct recursive approach
b. Dynamic programming based solution
c. Memoized recursive solution

7. Find the maximum weight subset of jobs that can be processed on the single machine available only for 11 minutes.

Jobs	Weights/value/profit	Running time
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

8. The Fibonacci numbers are defined by recurrence (3.22). Give an $O(n)$-time dynamic-programming algorithm to compute the nth Fibonacci number. Draw the subproblem graph. How many vertices and edges are in the graph?

9.

Find an optimal parenthesization of a matrix-chain product whose sequence of dimensions is $\langle 5, 10, 3, 12, 5, 50, 6 \rangle$.

10.

Give a recursive algorithm MATRIX-CHAIN-MULTIPLY(A, s, i, j) that actually performs the optimal matrix-chain multiplication, given the sequence of matrices $\langle A_1, A_2, \ldots, A_n \rangle$, the s table computed by MATRIX-CHAIN-ORDER, and the indices i and j. (The initial call would be MATRIX-CHAIN-MULTIPLY$(A, s, 1, n)$.)

11.

Describe the subproblem graph for matrix-chain multiplication with an input chain of length n. How many vertices does it have? How many edges does it have, and which edges are they?

12.

Draw the recursion tree for the MERGE-SORT procedure from Section 2.3.1 on an array of 16 elements. Explain why memoization fails to speed up a good divide-and-conquer algorithm such as MERGE-SORT.

13.

As stated, in dynamic programming we first solve the subproblems and then choose which of them to use in an optimal solution to the problem. Professor Capulet claims that we do not always need to solve all the subproblems in order to find an optimal solution. She suggests that we can find an optimal solution to the matrix-chain multiplication problem by always choosing the matrix A_k at which to split the subproduct $A_i A_{i+1} \cdots A_j$ (by selecting k to minimize the quantity $p_{i-1} p_k p_j$) *before* solving the subproblems. Find an instance of the matrix-chain multiplication problem for which this greedy approach yields a suboptimal solution.

14.

Imagine that you wish to exchange one currency for another. You realize that instead of directly exchanging one currency for another, you might be better off making a series of trades through other currencies, winding up with the currency you want. Suppose that you can trade n different currencies, numbered $1, 2, \ldots, n$, where you start with currency 1 and wish to wind up with currency n. You are given, for each pair of currencies i and j, an exchange rate r_{ij}, meaning that if you start with d units of currency i, you can trade for dr_{ij} units of currency j. A sequence of trades may entail a commission, which depends on the number of trades you make. Let c_k be the commission that you are charged when you make k trades. Show that, if $c_k = 0$ for all $k = 1, 2, \ldots, n$, then the problem of finding the best sequence of exchanges from currency 1 to currency n exhibits optimal substructure. Then show that if commissions c_k are arbitrary values, then the problem of finding the best sequence of exchanges from currency 1 to currency n does not necessarily exhibit optimal substructure.