Question 1

a. Consider the size of the matrices given in Table 1.
   i. Perform matrix chain multiplication for $A_1 \cdot A_2 \cdot A_3 \cdot A_4$ to minimize the number of scalar multiplication operations.
   ii. Explain the optimal parenthesis substructure of matrix chain.

Table 1

Matrix	No. of Rows	No. of columns
A_1	5	4
A_2	4	6
A_3	6	2
A_4	2	7

b. Consider the two strings "$BDCB$" and "$BACDB$".
   i. Determine the longest common subsequence (LCS).
   ii. Discuss the three optimal substructure cases of LCS using dynamic programming.

Solution:

a.

$$\begin{array}{cccc} A_1 & A_2 & A_3 & A_4 \\ (5\times4) & (4\times6) & (6\times2) & (2\times7) \\ P_0 \; P_1 & P_1 \; P_2 & P_2 \; P_3 & P_3 \; P_4 \end{array}$$

No. of operations ⟹

$$(5\times4\times6)\times(5\times6\times2)+(5\times2\times7)$$

$$= 120 + 60 + 70$$

$$= 250$$

$$M[i,j] = \begin{cases} 0 & , \text{if } i=j \\ \min M[i,k] + M[k+1,j] + P_{i-1}P_jP_k \end{cases}$$

i\j	1	2	3	4
1	0	120	88	158
2	X	0	48	104
3	X	X	0	84
4	X	X	X	0

$$M[1,2] = M[1,1] + M[2,2] + P_0 P_1 P_2$$
$$\min_{i \le k \le j} \quad = 120$$

$$M[2,3] = M[2,2] + M[3,3] + P_1 P_2 P_3$$
$$= 0 + (4 \times 6 \times 2) = 48$$

$$M[3,4] = M[3,3] + M[4,4] + P_2 P_3 P_4$$
$$= 0 + (6 \times 2 \times 7) = 84$$

$$M[1,3] = M[1,1] + M[2,3] + P_0 P_1 P_3$$
$$= 0 + 48 + (5 \times 4 \times 2)$$
$$= 48 + 40 = 88$$

$$M[2,4] = M[2,2] + M[3,4] + P_1 P_2 P_4$$
$$k = 2 \Rightarrow 0 + 84 + (4 \times 6 \times 7) \text{ ✗}$$
$$k = 3 \Rightarrow M[2,3] + M[4,4] + P_1 P_3 P_4$$
$$= 48 + 56 = 104$$

$$M[1,4] = M[1,3] + M[4,4] + P_0 P_3 P_4$$
$$= 88 + (5 \times 2 \times 7)$$
$$= 158$$

a. ii. Given a sequence of matrices, find the most efficient way to multiply these matrices together. The problem is not actually to perform the multiplications, but merely to decide in which order to perform the multiplications.

We have many options to multiply a chain of matrices because matrix multiplication is associative. In other words, no matter how we parenthesize the product, the result will be the same.

However, the order in which we parenthesize the product affects the number of simple arithmetic operations needed to compute the product, or the efficiency.

Given an array p[] which represents the chain of matrices such that the ith matrix Ai is of dimension $p[i-1]\ x\ p[i]$. We need to write a function MatrixChainOrder() that should return the minimum number of multiplications needed to multiply the chain.

1)**Optimal Substructure**:

A simple solution is to place parenthesis at all possible places, calculate the cost for each placement and return the minimum value. In a chain of matrices of size n, we can place the first set of parentheses in n-1 ways. For example, if the given chain is of 4 matrices. let the chain be ABCD, then there are 3 ways to place first set of parenthesis outer side: (A)(BCD),

(AB)(CD) and (ABC)(D). So, when we place a set of parentheses, we divide the problem into subproblems of smaller size. Therefore, the problem has optimal substructure property and can be easily solved using recursion.
Minimum number of multiplications needed to multiply a chain of size n = Minimum of all n-1 placements (these placements create subproblems of smaller size).

Optimal Substructure Property: If final "optimal" solution of $A_{i\ldots j}$ involves splitting into $A_{i\ldots k}$ and $A_{k+1\ldots j}$ at final step then parenthesization of $A_{i\ldots k}$ and $A_{k+1\ldots j}$ in final optimal solution must also be optimal for the subproblems "standing alone": If parenthisization $A_{i\ldots k}$ of was not optimal we could replace it by a better parenthesization and get a cheaper final solution, leading to a contradiction. Similarly, if parenthisization of $A_{k+1\ldots j}$ was not optimal we could replace it by a better parenthesization and get a cheaper final solution, also leading to a contradiction.

b. i)

	ϕ	B	A	C	D	B
ϕ	0	0	0	0	0	0
B	0	1	1	1	1	1
D	0	1	1	1	2	2
C	0	1	1	2	2	2
B	0	1	1	2	2	$\boxed{3}$

length of LCS = 3

if (A[i] = = B[j])
    LCS[i, j] = 1 + LCS[i-1, j-1]
else
    LCS[i, j] = max(LCS[i-1, j],
                    LCS[i, j-1])

b ii) : Let X =< x1 .... x_m > and Y =< y1 .... y_n > be sequences and let
Z =< z1 ,.............,zk > any LCS of X and Y .

1. If $x_m = y_n$, then $z_k = x_m = yn$ and Z_{k-1} is an LCS of $Xm - 1$ and $Yn - 1$.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of $Xm - 1$ and Y.
3. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X and $Yn - 1$.

Question 2

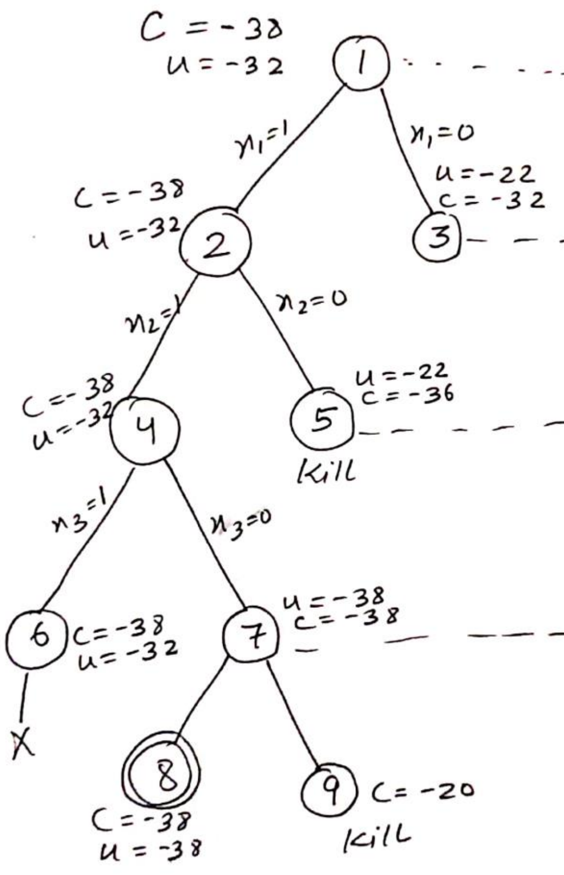
a. Solve the following instance of the $0/1$ Knapsack problem using branch and bound method.
   Number of elements $= 4$, Capacity $= 15$, Weights $= \{2, 4, 6, 9\}$,
   Profit $= \{10, 10, 12, 18\}$
b. Explain the greedy algorithm to determine the optimal solution in a 0/1 Knapsack's problem with a suitable example
c. Derive the time complexity of the quick sort algorithm.
   a.

capacity $= 15$

$n = 4$

upper ~~bound~~ $= \sum_{n=1}^{n} P_i x_i$

Cost $c = \sum_{i=1}^{n} P_i x_i$ (with fraction)



$C = 10+10+12+\frac{\overset{2}{\cancel{18}}}{9}\times 3$
$= 2+4+6$
$= 15$

$C = 10+12+\frac{\overset{2}{\cancel{18}}}{9}\times 5$
$= 4+6$
$= 10$

$C = 10+12+\frac{\overset{2}{\cancel{18}}}{9}$
$= 2+6$
$= 8$

$C = 10+10+\frac{\overset{2}{\cancel{18}}}{9}$
$= 2+4$
$= 6$

Answer $\Rightarrow x = \{1, 1, 0, 1\}$
profit $= 10+10+8 = 38$
weight $= 2+4+9 = 15$

<u>b. Greedy doesn't work for 0-1 Knapsack Problem:</u>

- Example 1: Knapsack Capacity W = 25 and

Item	A	B	C	D
Price	12	9	9	5
Size	24	10	10	7

  - Optimal: B and C. Size=10+10=20. Price=9+9=18
  - Possible greedy approach: take largest Price first (Price=12, not optimal)
  - Possible greedy approach: take smallest size item first (Price=9+5=14, not optimal)
  -

- Example 2: Knapsack Capacity = 30

Item	A	B	C
Price	50	140	60
Size	5	20	10
Ratio	10	7	6

  - $Possible\ greedy\ approach: take\ largest\ ratio: (Solution: A\ and\ B. Size = 5 + 20 = 25. Price = 50 + 140 = 190$
  - $Optimal: B\ and\ C. Size = 20 + 10 = 30. Price = 140 + 60 = 200$
  - $Greedy\ fractional: A, B, and\ half\ of\ C. Size = 5 + 20 + 10 * (5/10) = 30. Price\ 50 + 140 + 60 * (5/10) = 190 + 30 = 220$

- For comparison: DP algorithm gives 18
  - Use 2D array: rows 0..25, columns 0..4
  - Initialize first row and column to 0
  - Solve a row at a time, subtracting off added size as needed
  - What is the best way to fill:
    - With A only: sizes 0…23, **24**, 25
    - With A, B only: sizes 0…9, **10**, 11…23, **24**, 25
    - With A, B, C only: sizes 0…9, **10**, 11...19, **20**, 21...23, **24**, 25
    - With A, B, C, D: sizes 0...6, **7**, 8...9, **10**, 11...16, **17**, 18...19, **20**, 21...23, **24**, 25

c.

Solution for quick sort algorithm:

Worst case: The worst-case time complexity of quicksort is $\Omega(n^2)$.

Proof. The partitioning step: at least, n − 1 comparisons.

- $At\ each\ next\ step\ for\ n \geq 1, the\ number\ of\ comparisons\ is\ one\ less,$

$$so\ that\ T(n) = T(n-1) + (n-1);\ T(1) = 0.$$

- $“Telescoping”\ T(n) - T(n-1) = n - 1:$

$$T(n) + T(n - 1) + T(n - 2) + \ldots + T(3) + T(2)$$

$$-T(n - 1) - T(n - 2) - \ldots - T(3) - T(2) - T(1)$$

$$= (n - 1) + (n - 2) + \ldots + 2 + 1 - 0$$

$$T(n) = (n - 1) + (n - 2) + \ldots + 2 + 1 = (n-1)n/2$$

This yields that $T(n) \in \Omega(n^2)$.

Average case:

In Average Case next pivot splits array into two equal lengths (N/2) by comparing all elements which results time complexity N.

$$T(N) = 2 * T(N/2) + N$$

$$divide\ both\ sides\ by\ N$$

$$T(N) / N = 2 * (T(N/2) / N) + N/N$$

$$T(N)/N = 2 * (T(N/2) / N) + 1$$

$$Therefore, T(N)/N = (T(N/2) /(N/2)) + 1$$

$$T(N/2) / (N/2) = T(N/4)/(N/4) + 1) = T(N/4)/(N/4) + 1$$

$$T(2) / 2 = T(1) / 1 + 1$$

$$As\ T(1) = 0$$

$$T(2) / 2 = 1$$

$$T(4) / 4 = T(2) + T(1)$$

$$T(8) / 8 = T(4) + T(2) + T(1)\ \#\#log(n)\ terms$$

$$Therefore\ T(N)/N = log\ N$$

$$T(N) = NlogN$$

Best case:

$$T(n) = 2 * T(n/2) + n \qquad // T(n/2) = 2 * T(n/4) + (n/2)$$

$$= 2 * [2 * T(n/4) + n/2] + n$$

$$= 2 * 2 * T(n/4) + n + n$$

$$= 2 * 2 * T(n/4) + 2n \qquad // T(n/4) = 2 * T(n/8) + (n/4)$$

$$= 2 * 2 * [2 * T(n/8) + (n/4)] + 2n$$

$$= 2 * 3 * T(n/8) + 22 * (n/4) + 2n$$

$$= 23 * T(n/8) + n + 2n$$

$$= 2 * 3 * T(n/8) + 3n$$

$$a \qquad nd\ so\ on....$$

$$= 2k * T(n/(2k)) + k * n \quad // Keep\ going\ until: n/(2k) = 1 \ <==> n = 2k$$

$$= 2k * T(1) + k * n$$

$$= 2k * 1 + k * n$$

$$= n + k * n$$

$$= n + (log(n)) * n$$

$$= n * (log(n) + 1)$$

$$\sim = n * log(n)$$