DELHI SKILL AND ENTREPRENEURSHIP UNIVERSITY

B.Tech Semester-V (CSE)
End-Semester Examination

Time: 3 Hours

Instructions to Candidates:
1. This paper consists of two sections: Section A and Section B.
2. Section A is compulsory. Attempt all questions.
3. Section B contains descriptive questions.
4. Assume necessary data if not given.

SECTION A – Short Answer Questions (25 Marks)

Attempt all questions. Answers must be concise.

Q1. Define the terms 'Big-Oh' (O) and 'Omega' (Ω) notation in the context of algorithm analysis.

Answer:

- **Big-Oh (O):** Represents the asymptotic upper bound. $f(n) = O(g(n))$ if there exist constants $c > 0, n_0 > 0$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. It denotes the worst-case complexity.
- **Omega (Ω):** Represents the asymptotic lower bound. $f(n) = \Omega(g(n))$ if there exist constants $c > 0, n_0 > 0$ such that $0 \leq c \cdot g(n) \leq f(n)$ for all $n \geq n_0$. It denotes the best-case complexity.

Q2. What is a 'Stable Sorting Algorithm'? Give one example.

Answer: A sorting algorithm is considered **stable** if two objects with equal keys appear in the same order in the sorted output as they appear in the input array.

- **Example:** Merge Sort is stable, whereas Quick Sort is typically unstable.

Q3. Differentiate between Divide and Conquer and Dynamic Programming.

Answer:

1. **Nature of Sub-problems:** Divide and Conquer solves disjoint (independent) sub-problems, whereas Dynamic Programming solves overlapping sub-problems.
2. **Reusability:** DP stores the results of sub-problems (memoization) to avoid re-computation; Divide and Conquer does not store solutions.
3. **Example:** Merge Sort (D&C) vs. Matrix Chain Multiplication (DP).

Q4. State the 'Principle of Optimality'. Which algorithmic paradigm relies on it?

Answer: The Principle of Optimality states that an optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

- Both **Dynamic Programming** and **Greedy Approaches** rely on this principle (Optimal Substructure).

Q5. Briefly explain the 'Spurious Hit' in the Rabin-Karp string matching algorithm.

Answer: A **Spurious Hit** occurs in the Rabin-Karp algorithm when the hash value of the pattern matches the hash value of a current window in the text, but the actual characters do not match. This requires an additional verification step (character-by-character comparison) to confirm the match.

Q6. What is a State Space Tree? In which context is the term 'Promising Node' used?

Answer:

- A **State Space Tree** is a tree representation of all possible states (solutions or partial solutions) of a problem, used in Backtracking and Branch Bound.
- A **Promising Node** is a node in the tree that has the potential to lead to a feasible or optimal solution. If a node violates constraints, it is non-promising and pruned.

Q7. Define NP-Hard and NP-Complete classes.

Answer:

- **NP-Hard:** A problem X is NP-Hard if every problem in NP is reducible to X in polynomial time. It is at least as hard as the hardest problems in NP.
- **NP-Complete:** A problem X is NP-Complete if it is both in NP (verifiable in polynomial time) AND it is NP-Hard.

Q8. Why is Quick Sort preferred over Merge Sort for sorting arrays in practice?

Answer: Quick Sort is generally preferred for arrays because it is an **in-place sort** (requires $O(\log n)$ stack space vs $O(n)$ auxiliary space for Merge Sort) and has good cache locality. Although its worst-case is $O(n^2)$, its average case is $O(n \log n)$ with smaller constant factors.

Q9. Given the recurrence $T(n) = 2T(n/2) + n^2$, find the complexity using Master Method.

Answer: Here $a = 2, b = 2, f(n) = n^2$.

- Calculate $n^{\log_b a} = n^{\log_2 2} = n^1 = n$.
- Compare $f(n)$ with $n^{\log_b a}$: n^2 grows faster than n ($f(n) = \Omega(n^{1+\epsilon})$).
- This matches Case 3 of the Master Theorem. Regularity condition $af(n/b) \leq cf(n) \implies 2(n/2)^2 \leq cn^2 \implies n^2/2 \leq cn^2$ holds for $c < 1$.
- Thus, $T(n) = \Theta(f(n)) = \Theta(n^2)$.

Q10. What is the Vertex Cover Problem? Is it P or NP?

Answer: The Vertex Cover Problem involves finding the smallest set of vertices in a graph such that every edge in the graph is incident to at least one vertex in the set.

- It is an **NP-Complete** problem. There is no known polynomial-time algorithm to solve it exactly for general graphs.

SECTION B – Descriptive Questions (75 Marks)

Detailed answers required. Draw diagrams wherever necessary.

Q11. Explain the **Counting Sort** algorithm. Why is it not a comparison-based sort? Analyze its time complexity.

Answer: **Counting Sort Algorithm:** Counting sort is an integer sorting algorithm that operates by counting the number of objects that have each distinct key value. It uses arithmetic on those counts to determine the positions of each key value in the output sequence.

Steps:

1. Determine the range of input data (k).
2. Initialize a count array of size k with zeros.
3. Iterate through input, incrementing the index in the count array corresponding to the value.
4. Modify the count array such that each element at index i stores the sum of previous counts (cumulative frequency).
5. Place elements into a sorted output array based on count indices.

Comparison-Based?: No. It does not compare elements (like $A[i] > A[j]$). Instead, it uses the actual values of elements as indices into an auxiliary array.

Time Complexity:

- Total Time: $O(n + k)$, where n is number of elements and k is the range of input.
- If $k = O(n)$, it runs in linear time.

Q12. Using the **Master Method**, solve the recurrence relation: $T(n) = 3T(n/4) + n \log n$.

Answer: Given: $T(n) = 3T(n/4) + n \log n$.

- $a = 3$, $b = 4$, $f(n) = n \log n$.
- Calculate critical exponent: $E = \log_b a = \log_4 3$.
- Since $3 < 4$, $\log_4 3 < 1$ (approx 0.79).
- Compare $f(n)$ with $n^{\log_b a}$:
  - $n^{\log_4 3} \approx n^{0.79}$.
  - $f(n) = n \log n$.
  - $f(n)$ is polynomially larger than $n^{\log_4 3}$. specifically $f(n) = \Omega(n^{\log_4 3 + \epsilon})$.
- This corresponds to **Case 3** of the Master Theorem.
- **Regularity Condition check:** $af(n/b) \leq cf(n)$

$$3\left(\frac{n}{4} \log \frac{n}{4}\right) \leq c(n \log n)$$

$$\frac{3}{4} n(\log n - \log 4) \leq cn \log n$$

This holds true for any $c \geq 3/4$.

- **Conclusion:** The solution is dominated by $f(n)$.

$$T(n) = \Theta(n \log n)$$

Q13. Explain the **Knuth-Morris-Pratt (KMP)** algorithm logic. specifically, how is the π (prefix) table computed?

Answer: **Logic:** The KMP algorithm improves string matching by avoiding re-evaluation of characters that have already been matched. It uses a pre-computed array called the π table (or LPS - Longest Prefix Suffix array). When a mismatch occurs, the algorithm uses the π table to shift the pattern intelligently rather than shifting by just one.

Computing the π (Prefix) Table: The value $\pi[q]$ represents the length of the longest proper prefix of the pattern $P[1..q]$ that is also a suffix of $P[1..q]$.

Example: Pattern P = ABABAC

- $i = 0$: $\pi[0] = 0$.
- $i = 1$ ('B'): No prefix matches suffix 'B'. $\pi[1] = 0$.
- $i = 2$ ('A'): Matches first 'A'. $\pi[2] = 1$.
- $i = 3$ ('B'): Pattern "ABAB". Prefix "AB" matches suffix "AB". $\pi[3] = 2$.
- $i = 4$ ('A'): Pattern "ABABA". Prefix "ABA" matches suffix "ABA". $\pi[4] = 3$.
- $i = 5$ ('C'): No prefix matches suffix ending in C. $\pi[5] = 0$.

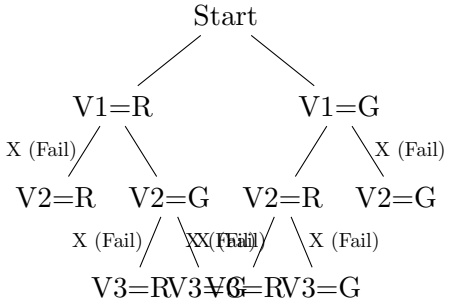
Resulting Table: $[0, 0, 1, 2, 3, 0]$.

Q14. Explain **Graph Coloring** using Backtracking. Draw the state space tree for coloring a triangle graph (3 vertices connected to each other) with 2 colors (R, G).

Answer: **Problem:** Assign colors to vertices of a graph such that no two adjacent vertices share the same color using minimum number of colors (m).

Backtracking Approach: We assign a color to vertex V_1. Then we move to V_2 and try to assign a color valid with respect to V_1, and so on. If we reach a vertex where no color can be assigned (constraint violation), we backtrack to the previous vertex and change its color.

State Space Tree (3 Vertices, 2 Colors R/G): Vertices: 1, 2, 3 (All connected - Triangle).



Result: With only 2 colors, a triangle (K_3) cannot be colored. All branches are pruned or fail at the leaf. We need 3 colors.

Q15. Discuss **Strassen's Matrix Multiplication**. How does it improve over the standard Divide and Conquer method? (Write the complexity comparison).

Answer: **Standard Divide and Conquer:** Multiplying two $n \times n$ matrices involves 8 recursive multiplications of size $n/2 \times n/2$ and 4 additions. Recurrence: $T(n) = 8T(n/2) + O(n^2)$. Complexity: $O(n^{\log_2 8}) = O(n^3)$.

Strassen's Algorithm: Volker Strassen discovered a way to compute the product using only **7 multiplications** instead of 8, at the cost of more additions/subtractions. Recurrence: $T(n) = 7T(n/2) + O(n^2)$.

Complexity Comparison:

- Using Master Theorem on $T(n) = 7T(n/2) + n^2$:
- $\log_2 7 \approx 2.81$.
- Complexity: $O(n^{2.81})$.

Since $2.81 < 3$, Strassen's algorithm is asymptotically faster than the standard method for large n.

Q16. (a) Explain the **Quick Sort** algorithm. Trace the execution of Quick Sort on the array: $A = \{10, 80, 30, 90, 40, 50, 70\}$ using the last element as the pivot.
(b) Derive the Best-Case and Worst-Case time complexity.

Answer: **(a) Quick Sort Algorithm:** Quick Sort is a Divide and Conquer algorithm. It picks an element as a 'pivot' and partitions the given array around the picked pivot.

1. **Partition:** Reorder the array so that all elements with values less than the pivot come before the pivot, and all elements greater come after it.
2. **Recursion:** Recursively apply the above step to the sub-array of elements with smaller values and separate the sub-array of elements with greater values.

Trace: $A = \{10, 80, 30, 90, 40, 50, \mathbf{70}\}$. Pivot = 70.

- Pointers i (initially -1) and j (0 to $n-1$).
- Compare 10 ¡ 70: swap(i+1, j), $i = 0$. Array: $\{10, 80, ...\}$
- Compare 80 ¿ 70: no swap.
- Compare 30 ¡ 70: swap(i+1, j), $i = 1$. Array: $\{10, 30, 80, 90, 40, 50, 70\}$
- Compare 90 ¿ 70: no swap.
- Compare 40 ¡ 70: swap(i+1, j), $i = 2$. Array: $\{10, 30, 40, 90, 80, 50, 70\}$
- Compare 50 ¡ 70: swap(i+1, j), $i = 3$. Array: $\{10, 30, 40, 50, 80, 90, 70\}$
- End of loop. Swap Pivot (70) with $i + 1$ (80).
- **Partitioned:** $\{10, 30, 40, 50, \mathbf{70}, 90, 80\}$.
- Now recursively sort Left $\{10, 30, 40, 50\}$ and Right $\{90, 80\}$.

(b) Complexity Analysis:

- **Best Case:** Partition splits array exactly in half.

$$T(n) = 2T(n/2) + \Theta(n) \implies O(n \log n)$$

- **Worst Case:** Partition picks smallest or largest element (sorted array). One subproblem is size 0, other is $n-1$.

$$T(n) = T(n-1) + \Theta(n) \implies O(n^2)$$

Q17. Solve the **Matrix Chain Multiplication** problem for matrices with dimensions: $A_1(10 \times 20)$, $A_2(20 \times 30)$, $A_3(30 \times 40)$, $A_4(40 \times 30)$.
Show the DP table computation to find the minimum number of scalar multiplications.

Answer: **Objective:** Find min cost $m[i, j] = \min_{i \le k < j}\{m[i, k] + m[k+1, j] + p_{i-1}p_k p_j\}$. Dimension vector $p = \{10, 20, 30, 40, 30\}$. $n = 4$ matrices.

1. Chain Length $L = 1$ (Main Diagonal): $m[1, 1] = m[2, 2] = m[3, 3] = m[4, 4] = 0$.

2. Chain Length $L = 2$:

- $m[1, 2] = 10 \times 20 \times 30 = 6000$.
- $m[2, 3] = 20 \times 30 \times 40 = 24000$.
- $m[3, 4] = 30 \times 40 \times 30 = 36000$.

3. Chain Length $L = 3$:

- $m[1, 3]$:
  - $k = 1$: $m[1, 1] + m[2, 3] + 10 \cdot 20 \cdot 40 = 0 + 24000 + 8000 = 32000$.
  - $k = 2$: $m[1, 2] + m[3, 3] + 10 \cdot 30 \cdot 40 = 6000 + 0 + 12000 = 18000$.

  $\implies m[1, 3] = 18000$.
- $m[2, 4]$:
  - $k = 2$: $m[2, 2] + m[3, 4] + 20 \cdot 30 \cdot 30 = 0 + 36000 + 18000 = 54000$.
  - $k = 3$: $m[2, 3] + m[4, 4] + 20 \cdot 40 \cdot 30 = 24000 + 0 + 24000 = 48000$.

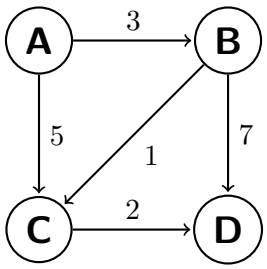
  $\implies m[2, 4] = 48000$.

4. Chain Length $L = 4$ (Final Answer):

- $m[1, 4]$:
  - $k = 1$: $m[1, 1] + m[2, 4] + 10 \cdot 20 \cdot 30 = 0 + 48000 + 6000 = 54000$.
  - $k = 2$: $m[1, 2] + m[3, 4] + 10 \cdot 30 \cdot 30 = 6000 + 36000 + 9000 = 51000$.
  - $k = 3$: $m[1, 3] + m[4, 4] + 10 \cdot 40 \cdot 30 = 18000 + 0 + 12000 =$ **30000**.

Minimum Multiplications: 30,000.

Q18. Explain **Dijkstra's Algorithm** for Single Source Shortest Path. Apply it to the following graph starting from Source 'A'.



Answer: **Algorithm Logic:** Dijkstra's algorithm is a Greedy strategy. It maintains a set of visited vertices and a set of unvisited vertices. It iteratively selects the unvisited vertex with the smallest tentative distance from the source, adds it to the visited set, and updates the distances of its neighbors (Relaxation).

Execution Trace (Source A):

1. **Init:** $dist[A] = 0$, $dist[others] = \infty$. Visited $S = \emptyset$.
2. **Step 1:** Select min node **A** (0). $S = \{A\}$.
   - Relax B: $dist[B] = \min(\infty, 0 + 3) = 3$.
   - Relax C: $dist[C] = \min(\infty, 0 + 5) = 5$.
3. **Step 2:** Select min unvisited from $\{B(3), C(5), D(\infty)\}$. Pick **B**. $S = \{A, B\}$.
   - Relax C via B: $dist[C] = \min(5, 3 + 1) = 4$. (Updated).
   - Relax D via B: $dist[D] = \min(\infty, 3 + 7) = 10$.
4. **Step 3:** Select min unvisited from $\{C(4), D(10)\}$. Pick **C**. $S = \{A, B, C\}$.

- Relax D via C: $dist[D] = \min(10, 4 + 2) = 6$. (Updated).

5. **Step 4:** Select min unvisited **D**. $S = \{A, B, C, D\}$.

Final Distances: A: 0, B: 3, C: 4, D: 6.

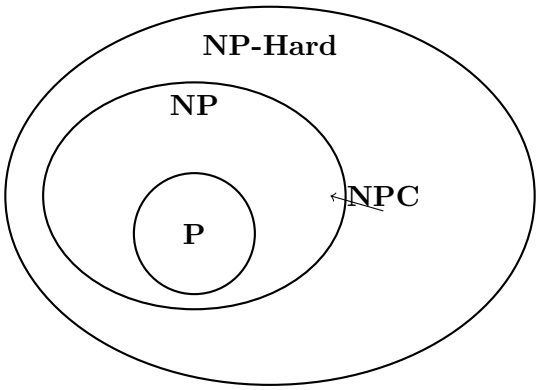
Q19. Explain the concept of **NP-Completeness**. Describe the relationship between P, NP, NP-Hard, and NP-Complete classes using a diagram. What is meant by 'Polynomial Time Reduction'?

Answer: **Classes Definition:**

- **P (Polynomial Time):** Problems solvable by a deterministic Turing machine in polynomial time ($O(n^k)$).
- **NP (Nondeterministic Polynomial):** Decision problems where a 'Yes' solution can be verified in polynomial time. P is a subset of NP.
- **NP-Hard:** Problems that are at least as hard as the hardest problems in NP. They do not necessarily have to be in NP (e.g., Halting Problem).
- **NP-Complete (NPC):** Problems that are both in NP and NP-Hard. These are the hardest problems in NP.

Polynomial Time Reduction: A problem A is polynomial-time reducible to problem B ($A \leq_p B$) if there exists a function f computable in polynomial time that maps instances of A to instances of B such that answer to A is YES iff answer to B is YES. If B can be solved easily, A can be solved easily.

Relationship Diagram:



(Assuming $P \neq NP$)

Q20. (a) Explain the **0/1 Knapsack Problem**.
(b) Solve the following instance using **Dynamic Programming**:
Weights $W = \{2, 3, 4, 5\}$, Values $V = \{3, 4, 5, 6\}$, Capacity $M = 5$.

Answer: **(a) Problem Statement:** Given n items, each with a weight w_i and a value v_i, and a knapsack of capacity M, select a subset of items to maximize total value such that total weight $\leq M$. In 0/1 Knapsack, items cannot be broken (take it or leave it).

(b) DP Solution: Let $DP[i][w]$ be max value using first i items with capacity w. Formula: $DP[i][w] = \max(DP[i-1][w], \quad v_i + DP[i-1][w - w_i])$.

Items: (1: 2kg, \$3), (2: 3kg, \$4), (3: 4kg, \$5), (4: 5kg, \$6). Max W=5.

Item / W	0	1	2	3	4	5
0	0	0	0	0	0	0
1 (2,3)	0	0	3	3	3	3
2 (3,4)	0	0	3	4	4	**7** (3+4)
3 (4,5)	0	0	3	4	5	7
4 (5,6)	0	0	3	4	5	7

Step-by-step for last row:

- Item 2 at $W = 5$: Max of (prev row val at $W = 5$) vs (Value of Item 2 + prev row at $W = 5 - 3 = 2$). $\max(3, 4 + 3) = 7$.
- Item 3 and 4 don't improve the optimal $W = 5$ because their weights plus valid remaining weights don't exceed the value 7 obtained by Items 1 and 2.

Optimal Value: 7. (Items 1 and 2 selected).