| (Affiliated to University of Mumbai) | ***End Semester Examination (R-24/R-25) SH 2026 Answer Key with marking scheme*** |
|---|---|
| Branch: CSE IOT/ECS/IT | Course: Advance Algorithm |
| Year/ Semester: TE/V | Course code: MDMC5052 |
| Time: 03 hours | Marks: 80 |

| | | Marks |
|---|---|---|
| Q. 1 | All questions are compulsory. | 30 |
| A. | Explain Asymptotic notations in detail.<br>Asymptotic notations are mathematical tools used to describe the time and space complexity of an algorithm as the input size (n) increases. They help compare the efficiency of algorithms independently of hardware and programming language.<br>The main asymptotic notations are:<br>**1. Big-O Notation — $O(g(n))$**<br>• Represents the **upper bound** of an algorithm's growth rate.<br>• Commonly used to express worst-case time complexity.<br>• Example: $T(n) = 3n^2 + 2n + 1 = O(n^2)$.<br>**2. Big-Omega Notation — $\Omega(g(n))$**<br>• Represents the **lower bound** of an algorithm's growth rate.<br>• Example: $T(n) = 3n^2 + 2n + 1 = \Omega(n^2)$.<br>**3. Big-Theta Notation — $\Theta(g(n))$**<br>• Represents a **tight bound**, meaning both upper and lower bounds are of the same order.<br>• Example: $T(n) = 3n^2 + 2n + 1 = \Theta(n^2)$. | 05 |
| B. | Differentiate between Las Vegas and Monte Carlo algorithm.<br>**Las Vegas Algorithm** / **Monte Carlo Algorithm**<br>Always produces the correct answer. / May produce an incorrect answer with a small probability.<br>Running time is random or variable. / Running time is usually bounded or predictable.<br>Correctness is guaranteed. / Correctness is probabilistic.<br>Randomness affects execution time. / Randomness can affect the answer's correctness.<br>Example: Randomized Quick Sort. / Example: Miller–Rabin primality test. | 05 |

| | | |
|---|---|---|
| C. | Explain the recurrence relation used to solve the Matrix Chain Multiplication problem.<br>**Matrix Chain Multiplication (MCM)** is a dynamic programming problem that determines the most efficient order of multiplying a sequence of matrices to minimize the number of scalar multiplications.<br>Let $A_1, A_2, \ldots, A_n$ be matrices, where matrix $A_i$ has dimensions $p_{i-1} \times p_i$.<br>Define $m[i, j]$ as the minimum number of scalar multiplications required to multiply matrices $A_i$ through $A_j$.<br>The recurrence relation is:<br>$$m[i,j] = \begin{cases} 0, & i = j \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] \\ \quad + p_{i-1} p_k p_j\}, & i < j \end{cases}$$<br>**Explanation:**<br>1. **Base case:** If $i = j$, only one matrix is present, so no multiplication is required.<br>2. **Splitting:** Divide the chain at position $k$, between $i$ and $j$.<br>3. **Subproblems:** Calculate the minimum costs of multiplying the left and right subchains.<br>4. **Combination:** Add the cost of multiplying the resulting matrices, $p_{i-1} p_k p_j$.<br>5. **Optimization:** Select the split position $k$ that gives the minimum total cost.<br>**Example:** For matrices of dimensions $10 \times 20$, $20 \times 30$, and $30 \times 15$:<br>• $(A_1 A_2) A_3$: $10(20)(30) + 10(30)(15) = 10,500$ multiplications.<br>• $A_1 (A_2 A_3)$: $20(30)(15) + 10(20)(15) = 12,000$ multiplications.<br>Therefore, the optimal parenthesization is $(A_1 A_2) A_3$, requiring **10,500 scalar multiplications**.<br>**Conclusion:** The recurrence relation uses dynamic programming to find the minimum multiplication cost. Its standard time complexity is $O(n^3)$, with space complexity $O(n^2)$. | 05 |
| D. | State and explain the Max-Flow Min-Cut Theorem and its significance.<br>The **Max-Flow Min-Cut Theorem** states that the maximum possible flow from a source vertex $s$ to a sink vertex $t$ in a flow network is equal to the minimum capacity of any cut separating $s$ from $t$.<br>**Key terms:**<br>1. **Flow network:** A directed graph in which each edge has a capacity.<br>2. **Maximum flow:** The greatest amount of flow that can be sent from the source to the sink without violating capacity constraints.<br>3. **Cut:** A partition of vertices into two sets $S$ and $T$, where $s \in S$ and $t \in T$.<br>4. **Cut capacity:** The sum of capacities of edges directed from $S$ to $T$.<br>The theorem is expressed as:<br>$$\boxed{\text{Maximum Flow} = \text{Minimum Cut Capacity}}$$<br>**Example:**<br>Consider a network where the maximum flow from source $s$ to sink $t$ is 10 units. If the minimum cut capacity is also 10 units, then the maximum possible flow is 10 units.<br>**Significance:**<br>• Determines the maximum flow possible in a network.<br>• Identifies bottlenecks that restrict flow.<br>• Helps optimize transportation and communication networks.<br>• Is useful in resource allocation, supply chains, and network routing.<br>• Provides a basis for algorithms such as Ford–Fulkerson and Edmonds–Karp. | 05 |

| | | |
|---|---|---|
| E. | Explain the concept of LP duality and describe the relationship between primal and dual problems. | 05 |

**Linear Programming (LP) duality** is a concept in mathematical optimization in which every linear programming problem, called the *primal problem*, has an associated problem called the *dual problem*.

The primal and dual problems are mathematically related and provide useful information about optimal solutions.

**Example of a primal problem:**

Maximize

$$Z = 3x_1 + 5x_2$$

Subject to:

$$x_1 \leq 4$$

$$2x_2 < 12$$

$$3x_1 + 2x_2 \leq 18$$

$$x_1, x_2 \geq 0$$

The corresponding dual problem is:

Minimize

$$W = 4y_1 + 12y_2 + 18y_3$$

Subject to:

$$y_1 + 3y_3 \geq 3$$

$$2y_2 + 2y_3 \geq 5$$

$$y_1, y_2, y_3 \geq 0$$

**Relationship between primal and dual problems:**

1. The primal objective maximizes profit, while the dual objective minimizes the corresponding resource value.
2. Each primal constraint corresponds to a dual variable, and each primal variable corresponds to a dual constraint.
3. The coefficient matrix of the dual is the transpose of the primal coefficient matrix.
4. **Weak duality:** For feasible solutions of a maximization primal and its minimization dual, the primal objective value cannot exceed the dual objective value.
5. **Strong duality:** If both problems have feasible optimal solutions, their optimal objective values are equal.

| | | |
|---|---|---|
| F. | Explain polynomial-time reduction and its importance in proving that a problem is NP-hard.<br>**Polynomial-time reduction** is a technique used to transform one computational problem into another in polynomial time while preserving the answer to the original problem.<br>If problem $A$ can be reduced to problem $B$, it is represented as:<br>$$A \leq_p B$$<br>This means an instance of problem $A$ can be converted into an equivalent instance of problem $B$ in polynomial time.<br>**Steps involved:**<br>1. Select a known NP-hard or NP-complete problem $A$.<br>2. Construct a transformation from $A$ to the target problem $B$.<br>3. Show that the transformation takes polynomial time.<br>4. Prove that the original instance is a yes-instance if and only if the transformed instance is a yes-instance.<br>5. Conclude that if $A$ is NP-hard and $A \leq_p B$, then $B$ is also NP-hard.<br>**Example:**<br>The Boolean Satisfiability Problem (SAT) can be reduced in polynomial time to 3-SAT. Since SAT is NP-complete, this reduction establishes that 3-SAT is NP-hard. As 3-SAT also belongs to NP, it is NP-complete.<br>**Importance:**<br>• Helps establish the computational difficulty of problems.<br>• Is used to prove NP-hardness and NP-completeness.<br>• Helps classify problems according to their computational complexity.<br>• Indicates why an efficient exact algorithm may be difficult to find. | 05 |
| | | |
| Q.2 A | Solve the following recurrence relation using the Master Theorem:<br>$T(n) = 2T\left(\frac{n}{2}\right) + n$<br>**Step 1: Compare with the Master Theorem**<br>The standard form is<br>$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$<br>Comparing both equations, we get:<br>• $a = 2$<br>• $b = 2$<br>• $f(n) = n$<br>**Step 2: Calculate $n^{\log_b a}$**<br>$$n^{\log_b a} = n^{\log_2 2}$$<br>Since $\log_2 2 = 1$,<br>$$n^{\log_2 2} = n^1 = n$$ | 10 |

**Step 3: Compare $f(n)$ with $n^{\log_b a}$**

We have

$$f(n) = n$$

and

$$n^{\log_b a} = n$$

Therefore,

$$f(n) = \Theta(n^{\log_b a})$$

This satisfies **Case 2 of the Master Theorem**.

According to Case 2:

$$T(n) = \Theta(n^{\log_b a} \log n)$$

**Step 4: Substitute the values**

$$T(n) = \Theta(n^1 \log n)$$

Hence, the final answer is

$$\boxed{T(n) = \Theta(n \log n)}$$

**Time complexity:** $O(n \log n)$

**Example:** This recurrence commonly occurs in Merge Sort, where the problem is di
subproblems of size $n/2$, and merging requires linear time.

OR

| | | |
|---|---|---|
| Q.2 B. | Solve T(n)=3T(n-1) +2 using the substitution method and determine its asymptotic bound. | 10 |

Assume the base condition is $T(0) = 1$.

The objective is to solve the recurrence using the **substitution method** and determine its asymptotic bound.

**Step 1: Expand the recurrence relation (2 Marks)**

Given,

$$T(n) = 3T(n-1) + 2$$

Substitute $T(n-1) = 3T(n-2) + 2$:

$$T(n) = 3[3T(n-2) + 2] + 2$$

$$T(n) = 3^2T(n-2) + 3(2) + 2$$

$$T(n) = 3^2T(n-2) + 2(3+1)$$

Now, substitute $T(n-2) = 3T(n-3) + 2$:

$$T(n) = 3^2[3T(n-3) + 2] + 2(3+1)$$

$$T(n) = 3^3T(n-3) + 2(3^2 + 3 + 1)$$

Therefore, after repeated substitution,

$$T(n) = 3^kT(n-k) + 2(3^{k-1} + \cdots + 3 + 1)$$

**Step 2: Obtain the general expression (2 Marks)**

After $k$ substitutions,

$$T(n) = 3^kT(n-k) + 2\sum_{i=0}^{k-1} 3^i$$

Using the geometric series formula,

$$\sum_{i=0}^{k-1} 3^i = \frac{3^k - 1}{3 - 1}$$

Therefore,

$$T(n) = 3^kT(n-k) + 2\left(\frac{3^k - 1}{2}\right)$$

Simplifying,

$$\boxed{T(n) = 3^kT(n-k) + 3^k - 1}$$

**Step 3: Apply the base condition (2 Marks)**

Let $k = n$. Then $T(n-k) = T(0)$.

Given $T(0) = 1$,

$$T(n) = 3^nT(0) + 3^n - 1$$

Substituting $T(0) = 1$,

$$T(n) = 3^n(1) + 3^n - 1$$

Hence,

$$\boxed{T(n) = 2(3^n) - 1}$$

**Step 4: Determine the asymptotic bound (2 Marks)**

The exact solution is

$$T(n) = 2(3^n) - 1$$

For large values of $n$, the constant factor 2 and the constant term $-1$ can be ignored when determining the asymptotic growth rate.

Therefore,

$$T(n) = \Theta(3^n)$$

This means the recurrence grows exponentially with input size $n$.

**Step 5: Final answer (2 Marks)**

- **Exact solution:** $\boxed{T(n) = 2(3^n) - 1}$, assuming $T(0) = 1$.
- **Big-O bound:** $\boxed{O(3^n)}$.
- **Big-Omega bound:** $\boxed{\Omega(3^n)}$.
- **Tight asymptotic bound:** $\boxed{\Theta(3^n)}$.

| | | |
|---|---|---|
| Q.3 A | Consider the hiring problem in which candidates are interviewed in random order. Derive the expected number of hires using indicator random variables. | 10 |

**1. Introduction**

In the hiring problem, $n$ candidates are interviewed in a random order. The first candidate is always hired. After that, a candidate is hired only if they are better than all previously interviewed candidates.

We use **indicator random variables** to derive the expected number of hires.

**2. Define the indicator random variable**

Let $X$ be the total number of candidates hired.

Define an indicator random variable $X_i$ for each candidate $i$:

$$X_i = \begin{cases} 1, & \text{if candidate } i \text{ is hired} \\ 0, & \text{otherwise} \end{cases}$$

The total number of hires is

$$X = \sum_{i=1}^{n} X_i$$

Taking expectations on both sides,

$$E[X] = E\left[\sum_{i=1}^{n} X_i\right]$$

By the linearity of expectation,

$$\boxed{E[X] = \sum_{i=1}^{n} E[X_i]}$$

**3. Calculate the expected value of each indicator**

Candidate $i$ is hired if they are the best candidate among the first $i$ candidates interviewed.

Since the candidates are interviewed in random order, each of the first $i$ candidates has an equal probability of being the best among those $i$ candidates.

Therefore,

$$P(X_i = 1) = \frac{1}{i}$$

For an indicator random variable,

$$E[X_i] = 1 \cdot P(X_i = 1) + 0 \cdot P(X_i = 0)$$

Thus,

$$\boxed{E[X_i] = \frac{1}{i}}$$

**4. Derive the expected number of hires**

Substituting $E[X_i] = 1/i$ into the expectation equation,

$$E[X] = \sum_{i=1}^{n} \frac{1}{i}$$

Therefore,

$$E[X] = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

This is the $n$-th **harmonic number**, denoted by $H_n$.

| | | |
|---|---|---|
| | Hence,<br>$$\boxed{E[X] = H_n = \sum_{i=1}^{n} \frac{1}{i}}$$<br>For large $n$, the harmonic number can be approximated by<br>$$H_n \approx \ln n + \gamma$$<br>where $\gamma \approx 0.5772$ is the Euler–Mascheroni constant.<br>Therefore, the expected number of hires is<br>$$\boxed{E[X] = \Theta(\log n)}$$<br>**5. Example**<br>Suppose $n = 4$ candidates are interviewed in random order.<br>The expected number of hires is<br>$$E[X] = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4}$$<br>$$E[X] = 1 + 0.5 + 0.333 + 0.25$$<br>$$\boxed{E[X] \approx 2.083}$$<br>Thus, for four candidates, the expected number of hires is approximately **2.083**. | |
| | OR | |
| Q.3 B | Apply randomized Quick Sort to the array [12, 5, 8, 1, 10, 3, 15, 7], showing the partitioning steps for a given sequence of pivot selections. Analyze its expected and worst-case time complexities.<br>Array: [12, 5, 8, 1, 10, 3, 15, 7]. One valid run, choosing the last element of each subarray as pivot for illustration:<br>Pivot 7 partitions into [5,1,3], 7, [12,8,10,15]. Recursively sort left: pivot 3 gives [1],3,[5], so left becomes [1,3,5]. Sort right: pivot 15 gives [12,8,10],15,[]; pivot 10 gives [8],10,[12], so this part becomes [8,10,12,15].<br>Final sorted array: [1, 3, 5, 7, 8, 10, 12, 15].<br>Expected time is $\Theta(n \log n)$ when randomized pivots yield balanced partitions in expectation; worst-case time is $\Theta(n^2)$ if partitions repeatedly split as 0 and $n-1$.<br>Expected auxiliary recursion space is O(log n), worst-case O(n). | 10 |
| | | |
| Q.4 A | Solve the Travelling Salesman Problem using dynamic programming for a weighted graph of four cities. Construct the required state table, determine the minimum-cost tour, and analyze the time complexity. | 10 |

assume the following symmetric graph (costs):

| City pair | Cost |
|---|---|
| A–B | 10 |
| A–C | 15 |
| A–D | 20 |
| B–C | 35 |
| B–D | 25 |
| C–D | 30 |

Use Held–Karp DP: dp[S][j] is the minimum cost to start at A, visit subset S of other cities, and finish at j. Base dp[{j}][j]=cost(A,j).

| Visited subset S | Finish B | Finish C | Finish D |
|---|---|---|---|
| {B} | 10 | — | — |
| {C} | — | 15 | — |
| {D} | — | — | 20 |
| {B,C} | 15+35=50 | 10+35=45 | — |
| {B,D} | 20+25=45 | — | 10+25=35 |
| {C,D} | — | 20+30=50 | 15+30=45 |

For all three destinations, minimum path costs ending at each city are:

dp[{B,C,D}][B]=min(dp[{C,D}][C]+CB, dp[{C,D}][D]+DB)=min(50+35,45+25)=70.

dp[{B,C,D}][C]=min(dp[{B,D}][B]+BC, dp[{B,D}][D]+DC)=min(45+35,35+30)=65.

dp[{B,C,D}][D]=min(dp[{B,C}][B]+BD, dp[{B,C}][C]+CD)=min(50+25,45+30)=75.

Close the tour by returning to A: via B=70+10=80; via C=65+15=80; via D=75+20=95. Minimum tour cost optimal tours include A→B→D→C→A (10+25+30+15=80) and A→C→D→B→A (15+30+25+10=80).

Time complexity $O(n^2 2^n)$; space complexity $O(n2^n)$.

OR

Q.4 B — 10

Solve the Matrix Chain Multiplication problem for matrices with dimensions 5 ×10, 10 ×15 ,15×20 and 20×25. Construct the cost table, determine the optimal parenthesization, and calculate the minimum number of scalar multiplications.

Matrices: $A_1$=5×10, $A_2$=10×15, $A_3$=15×20, $A_4$=20×25. Dimension array p=[5,10,15,20,25].

Length 2 costs: $m_{12}$=5×10×15=750; $m_{23}$=10×15×20=3000; $m_{34}$=15×20×25=7500.

Length 3: $m_{13}$=min(0+3000+5×10×20=4000, 750+0+5×15×20=2250)=2250, split at k=2. $m_{24}$=min(0+7500+10×15×25=11250, 3000+0+10×20×25=8000)=8000, split at k=3.

Length 4: split k=1 gives 0+8000+5×10×25=9250; k=2 gives 750+7500+5×15×25=10125; k=3 gives 2250+0+5×20×25=4750.

Cost table:

| i\j | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 750 | 2250 | 4750 |
| 2 | — | 0 | 3000 | 8000 |
| 3 | — | — | 0 | 7500 |
| 4 | — | — | — | 0 |

Optimal parenthesization: $((A_1(A_2A_3))A_4)$, with minimum 4,750 scalar multiplications. Time $O(n^3)$, space

Q.5 A — 10

Formulate the dual of the following primal linear programming problem and explain the relationship between their optimal objective values.

- Maximize: $Z = 4x_1 + 3x_2$
- Subject to: $2x_1 + x_2 \leq 8$, $x_1 + 2x_2 \leq 6$
- Non-negativity: $x_1, x_2 \geq 0$

OR

Q.5 B — 10

Apply the Ray Crossing algorithm to determine whether Q (3,2) lies inside or outside the polygon with vertices A (1,1) B (5,1), C (5,4) and D (1,4). Show the edge-

intersection calculations and justify the result.

Vertices: A(1,1), B(5,1), C(5,4), D(1,4). The polygon is an axis-aligned rectangle. Point Q=(3,2) satisfies 1<3<5 and 1<2<4.

Ray-crossing test: cast a horizontal ray to the right from Q. It crosses the right vertical edge BC at x=5, y=2. The bottom edge AB (y=1) and top edge CD (y=4) do not intersect the ray at y=2; the left edge AD is to the left of Q.

Number of crossings = 1 (odd), so Q lies inside the polygon. Equivalently, the point's x- and y-coordinates are strictly between the rectangle bounds.

Q.6 A — Differentiate between the Ford–Fulkerson and Edmonds–Karp algorithms. (5)

| **Feature** | **Ford–Fulkerson** | **Edmonds–Karp** |
|---|---|---|
| Path selection | Any augmenting path in residual graph. | Uses BFS to find a shortest (fewest-edge) augmenting path. |
| Complexity | For integer capacities, $O(E \cdot \lvert f^* \rvert)$ in the common bound; may not terminate with arbitrary irrational capacities. | $O(VE^2)$. |
| Implementation | DFS or another path-search method can be used. | BFS is required. |
| Guarantee | Depends on path selection/capacity assumptions for the standard bound. | Polynomial-time bound for general capacity values. |

Both repeatedly augment flow along residual paths and update forward/reverse residual capacities until no augmenting path remains.

Differentiate between an exact algorithm and an approximation algorithm with a suitable example. (5)

| **Feature** | **Exact algorithm** | **Approximation algorithm** |
|---|---|---|
| Output | Optimal solution. | Feasible solution with a provable quality guarantee. |
| Purpose | Finds the true optimum. | Finds a near-optimal result efficiently, especially for hard optimization problems. |
| Example | Dynamic programming for 0/1 knapsack (pseudo-polynomial). | A polynomial-time approximation scheme or greedy approximation for a hard optimization variant. |

For a minimization problem, a 2-approximation guarantees a solution cost at most twice the optimum, if that guarantee is established for the algorithm.

OR

Q.6 B — Explain the Ford–Fulkerson method for finding maximum flow in a network. (5)

Initialize flow on every edge to zero.

| | | |
|---|---|---|
| | 1. Build the residual graph, containing unused forward capacity and reverse capacity that can cancel existing flow.<br>2. Find an augmenting path from source s to sink t.<br>3. Find the bottleneck capacity (minimum residual capacity) on that path and augment the flow by that amount.<br>4. Update residual capacities and repeat until no augmenting path exists. The current flow is maximum by the Max-Flow Min-Cut Theorem.<br><br>For integer capacities, the basic method terminates and has a common running-time bound $O(E \cdot \lvert f^* \rvert)$, where $\lvert f^* \rvert$ is the maximum-flow value. | |
| | Explain how merge sort can be implemented as a multithreaded algorithm. Discuss the role of parallel recursive calls.<br>Merge sort divides the array into two halves, sorts each half recursively, then merges the sorted halves. In a multithreaded implementation, the two recursive sort calls can run concurrently on separate threads/tasks.<br>• Split the input into left and right halves.<br>• Spawn a task/thread to sort the left half and another to sort the right half.<br>• Wait for both tasks to finish (join/synchronization).<br>• Merge the two sorted halves sequentially or with a parallel merge implementation.<br>• Use a size threshold to switch to sequential sorting for small subarrays, avoiding excessive thread-creation overhead.<br><br>Work complexity remains $O(n \log n)$. With sufficient processors, idealized parallel depth can be $O(\log^2 n)$ for ordinary recursive merge sort with sequential merging, while practical speedup depends on processor count, memory bandwidth, synchronization and task overhead. Auxiliary space is $O(n)$. | 5 |
| | ********* **All the Best**********| |