VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

REG.NO.: 0434

SLOT: F1+TF1

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
CONTINUOUS ASSESSMENT TEST - I
WINTER SEMESTER 2025-2026

Programme Name & Branch	:	**BTech**
Course Code and Course Name	:	**BCSE204 L - Design and Analysis of Algorithms**
Faculty Name(s)	:	DEBI PRASANNA ACHARJYA ,UMADEVI K S ,GAYATHRI P,SRIVANI A,SHARIEF BASHA S,ILANTHENRAL K P S K,SATYAJIT DAS ,JOSHVA DEVADAS T,ATHIRA K,DURGESH KUMAR,IYAPPAN P,MALINI S,KARTHIK G M,VASANTHI P,ANKAREDDY RAJESH ,THIRUNAVUKKARASAN M,PADMAVATHY T,DINESH P,RANJITH R,DIVIYA M,SHARMILA C,MUKHTAR AHMAD SOFI,VANI RAJASEKAR,HARAPRIYA KAR,SYED RIYAZUL HAQ,MONESH S,EVELIN NISSY THOMAS,NAVEEN KUMAR S
Class Number(s)	:	**ALL Batches**
Date of Examination	:	**1-02-2026 (FN)**
Exam Duration	:	**90 minutes** **Maximum Marks: 50**

General instruction(s):
Answer All Questions – **Provide the necessary steps wherever required**
M - Max mark; CO – Course Outcome; BL – Bloom's Taxonomy Level (1 – Remember, 2 – Understand, 3 – Apply, 4 – Analyze, 5 – Evaluate, 6 – Create)

Course Outcomes

1. Apply the mathematical tools to analyse and derive the running time of the algorithms
2. Demonstrate the major algorithm design paradigms.

Q. No	Question	Module	Marks	CO	BL

1.	**a)** Consider the following recursive procedure to solve the Tower of Hanoi problem for *n disks*:
	The rules of Tower of Hanoi are:
	1. Only one disk can be moved at a time.
	2. A disk can only be placed on an empty peg or on top of a larger disk.
	3. All disks must be moved from the source peg to the destination peg.
	`Procedure TOH (n, A, B, C):`
	`  if n == 1:`
	`  Move disk from A to C`
	`  else:`
	`  TOH (n-1, A, C, B)`
	`  Move disk from A to C`
	`  TOH (n-1, B, A, C)`
	Define a suitable recursion invariant for this algorithm and use it to prove that the algorithm correctly moves all *n disks* from the source peg to the destination peg while following the rules. *(4 Marks)*
	b) Consider the recurrence relation:
	$T(n) = 2T(n-2) + n, \quad T(2) = \Theta(1)$
	Explain why the Master Theorem cannot be applied to this recurrence, and solve it using the back-substitution (substitution) method to obtain a tight Θ-bound. *(6 marks)*
		1	4+6	1	3

VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

REG.NO.:

SLOT: F1+TF1

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
CONTINUOUS ASSESSMENT TEST - I
WINTER SEMESTER 2025-2026

2.	A data file uses symbols from an alphabet Σ, where one symbol occurs in more than 50% of the total frequency. The symbols and their frequencies (in hundreds) are: **Symbol** p q r s t v w **Frequencies** 16 5 25 12 9 45 13 Construct an optimal Huffman code for the above symbols and draw the Huffman tree. Determine the percentage of saving in file size compared to fixed-length encoding. Explain why a perfectly balanced Huffman tree cannot be constructed for this frequency distribution, and discuss how the skewness of symbol frequencies affects the codeword lengths of other symbols and the overall compression efficiency.	1	10	1	3
3.	Consider the array ***A= [13, −3, −25, 20, −3, −16, −23, 18, 20, −7,12, −5, −22,15,]*** Find the maximum subarray using the divide-and-conquer method, clearly showing the left subarray sum, right subarray sum, and crossing subarray sum at each recursive step. Briefly explain why the overall time complexity of this algorithm is $O(n\log n)$.	1	10	1	3
4.	A salesman must visit 4 cities (A, B, C, D) exactly once and return to the starting city. The distances (in km) between the cities are given in the table below: **A** 0 10 15 20 **B** 10 0 35 25 **C** 15 35 0 30 **D** 20 25 30 0 Using the Dynamic Programming method, determine the minimum cost tour starting and ending at city A. Show the DP state representation and state transitions, and explain why storing only the minimum cost for each state preserves optimality. Suppose the distance between cities B and C is incorrectly recorded as 5 instead of 35; explain how this change affects the DP computation and the resulting optimal tour.	2	10	2	3
5.	A monitoring system needs to place 4 sensors on a 4 × 4 grid, one sensor per row, such that the following constraints are satisfied: • No two sensors are placed in the same row. • No two sensors are placed in the same column. • No two sensors are placed at grid positions (i, j) and $(i+2, j+1)$or $(i+2, j-1)$. . Here (i-row, j-column) i. Model this problem as a backtracking problem by clearly defining the state representation, decision variables, and constraints. (4 Marks). ii. Using the backtracking approach, determine one valid placement of the sensors. Clearly indicate the points where backtracking occurs (4 Marks) iii. Analyse the time complexity of this approach and briefly explain how pruning reduces the number of explored configurations compared to brute-force placement. (2 Marks)	2	10	2	3

****

Q-1

Q1	1 [4][1.5]	Recusion Invariant	Base Case	Inductive Hypothesis
		1	1	2
	2[6][2.5]	Why MT Not Applicable?	Back Substitution - Series	Time Complexity Analysis
		2	2	2

Invariant:
"A call to TOH (n, S, D, A) moves n disks from S to D such that a larger disk is **never** placed on a smaller disk, and all other disks on the pegs (larger than these *n)* remain undisturbed."

We prove this by induction on the number of disks, ***n***.
Base Case (*n* =1)
The code simply moves the disk from S to D. Since it's only one disk, no "stacking" rules can be broken. The goal is achieved.
Assumption (Inductive Hypothesis)
Assume that TOH (k, S, D, A) works perfectly for any number of disks *k* less than *n*
The Proof for *n* disks
We break the procedure into the three steps defined in the pseudocode:
1. **Phase 1: TOH (n-1, S, A, D**
    - o By our assumption, this correctly moves the top ***n***-1disks to the Auxiliary peg.
    - o **Rule Check:** Disk ***n*** (the largest) stays on the Source peg. Since it is larger than all ***n*** -1 disks, those disks can move "above" it without violating Rule 2.
2. **Phase 2: Move disk *n,* from S to D**
    - o Now the Source peg has only disk ***n***, and the Destination peg is empty (or only has disks larger than ***n***).
    - o **Rule Check:** Moving disk ***n*** to an empty peg (or onto a larger disk) is perfectly legal.
3. **Phase 3: TOH (n-1, A, D, S)**
    - o By our assumption, this correctly moves the ***n-1*** disks from the Auxiliary peg to the Destination peg.
    - o **Rule Check:** Disk ***n*** is already on the Destination peg. Because disk ***n*** is larger than all disks in the ***n* -1**stack, they can be safely piled on top of it.

Why the Master Theorem Cannot Be Applied?
The Master Theorem is designed for recurrences of the form T(n) = aT(n/b) + f(n), where the subproblem size is a **fraction** of n (divide-and-conquer).
In this case, T(n) = 2T(n-2) + n:
- **Subproblem Reduction:** The subproblem size is n-2. This is a **subtraction-based reduction** (linear decrease) rather than a division-based reduction.
- **Form Mismatch:** Because the input size decreases by a constant amount rather than a constant factor, the Master Theorem does not apply.

T(n) = 2T(n-2) + n
T(n) = 2[2T(n-4) + (n-2)] + n
T(n) = 2^2 T(n-4) + 2(n-2) + n
T(n) = 2^2 [2T(n-6) + (n-4)] + 2(n-2) + n
T(n) = 2^3 T(n-6) + 2^2(n-4) + 2^1(n-2) + 2^0(n)
After *i* substitutions, the formula is:

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

Reach the Base Case We reach the base case when $n - 2i = 2$, which means i=n-2/2

$$T(n) = 2^{\frac{n-2}{2}} T(2) + \sum_{j=0}^{\frac{n-2}{2}-1} 2^j (n-2j)$$

3. Obtaining the Tight Θ-bound

To find the bound, we evaluate the dominant terms:

1. **The Recursive Part:** $2^{\frac{n-2}{2}} T(2) = \frac{1}{2} \cdot 2^{n/2} \cdot \Theta(1) = \mathbf{\Theta(2^{n/2})}$.
2. **The Summation Part:** The sum $\sum_{j=0}^{\frac{n-2}{2}-1} 2^j (n-2j)$ is an Arithmetico-Geometric series. The largest term in the sum occurs when j is near its upper limit. For example, the last few terms are proportional to $2^{n/2}$.
   - The sum evaluates to $6(2^{n/2-1})$ n 4.
   - The dominant term of this sum is also $\mathbf{2^{n/2}}$.

Combining these, we get: $T(n)$ $c_1 2^{n/2} + c_2 2^{n/2}$ n 4

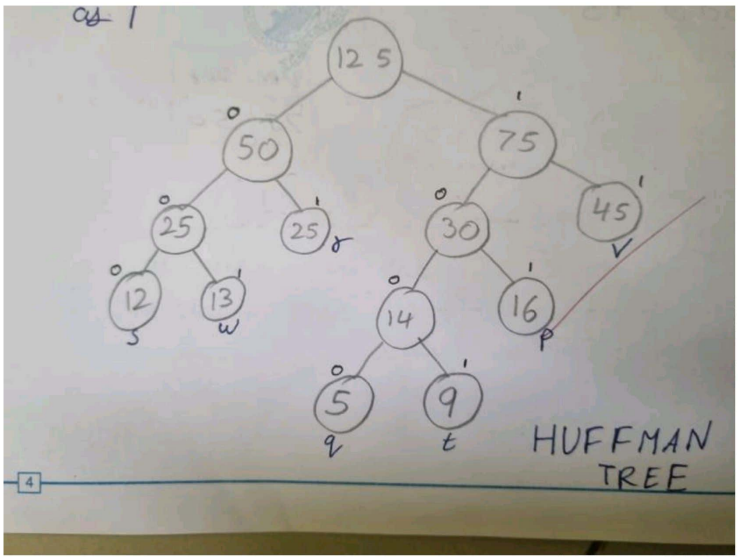
Since the $2^{n/2}$ term grows much faster than the linear n term:

$$T(n) = \Theta(2^{n/2})$$

(Note: This can also be written as $\Theta((\sqrt{2})^n)$, which is an exponential growth rate).

Q 2

Q2	[10][4]	H-Tree	H-Codes	Saving	Why Not Balanced	Effect of skewness
		4	3	1	1	1



Symbol	Code	Length
r	00	2
s	010	3
w	011	3
q	1000	4
t	1001	4
p	101	3
v	11	2

Average Code Length

$$L_{avg} = \frac{\sum f_i l_i}{\sum f_i}$$

$$= \frac{25 \times 2 + 12 \times 3 + 13 \times 3 + 5 \times 4 + 9 \times 4 + 16 \times 3 + 45 \times 2}{125}$$

$$= \frac{50 + 36 + 39 + 20 + 36 + 48 + 90}{125} = \frac{319}{125} \approx 2.55 \text{ bits/symbol}$$

Fixed-Length Encoding

er of symbols = 7

$$\lceil \log_2 7 \rceil = 3 \text{ bits/symbol}$$

Percentage Saving

$$\text{Saving} = \frac{3 - 2.55}{3} \times 100 \approx 15\%$$

Why Perfectly Balanced Huffman Tree Is Not Possible

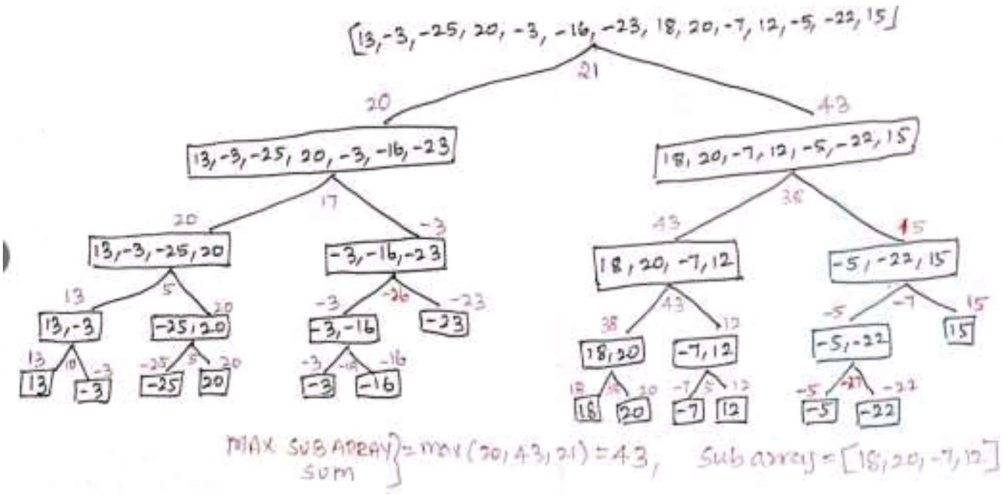
A perfectly balanced tree requires all symbols to have **equal or nearly equal frequencies**. Here, symbo **(45)** has much higher frequency than others, forcing it to be closer to the root, making the tree **unbalanc**

Effect of Skewness on Compression

- Highly frequent symbols get **shorter codewords** (e.g., v = 11).
- Less frequent symbols get **longer codewords** (e.g., q = 1000).
- This **reduces the average code length**, improving compression efficiency.

Q 3

Q3	[10][4]	DAC - Divide	DAC - Conquer	TCA
		4	4	2



43 **[18 20 -7 12]**

L	20
R	43
C	20

L	20
R	43
C	20

L	20
R	-3
C	17

L	43
R	`5
C	38

L	13
R	20
C	5

L	38
R	15
C	43

3. Time Complexity: $\Theta(n \log n)$

The algorithm follows the recurrence relation:

$$T(n) = 2T(n/2) + \Theta(n)$$

- $2T(n/2)$: The array is divided into two equal halves, and we solve each recursively.
- $\Theta(n)$: Finding the "Crossing Subarray" requires a linear scan from the midpoint to the ends of the current segment.

By **Case 2 of the Master Theorem** (where $a = 2, b = 2, f(n) = n$), since $n^{\log_2 2} = n^1$, the complexity is:

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

Q 4

Q4	[10][4]	DP State representation	Step1-4	Preserving Optimality	Scenario
		2	6	1	1

In the Dynamic Programming approach to TSP, we use the following state:
• **State:** g(i, S) represents the minimum cost to reach city i having visited all cities in the set S exactly
once, starting from city A
• **Base Case:** g(i, Ø) = d(i, A), which is the distance to return to the start from city i.
• **Transition:** To find g(i, S), we consider all cities ***j Ɛ S*** as the previous stop:

$$g(i, S) = \min_{j \in S} \{d(i, j) + g(j, S \setminus \{j\})\}$$

Given the distances: d(A, B) =10, d (A, C) =15, d (A, D) =20, d (B, C) =35, d (B, D) =25, d (C, D) =30.

Step 1: Sets of size 1 (Return to A)
• g (B, Ø) = 10
• g (C, Ø) = 15
• g (D, Ø) = 20

Step 2: Sets of size 1 (From Start through one city)

- g (B, {C}) = d(B, C) + g(C, Ø) = 35 + 15 = 50
- g (B, {D}) = d(B, D) + g (D, Ø) = 25 + 20 = 45
- g (C, {B}) = d(C, B) + g(B, Ø) = 35 + 10 = 45
- g (C, {D}) = d(C, D) + g(D, Ø) = 30 + 20 = 50
- g (D, {B}) = d(D, B) + g(B, Ø) = 25 + 10 = 35
- g (D, {C}) = d(D, C) + g(C, Ø) = 30 + 15 = 45

Step 3: Sets of size 2

- g (B, {C, D}) = min {d (B, C) + g (C, {D}), d (B, D) + g (D, {C})} = min {35+50, 25+45} = 70
- g (C, {B, D}) = min {d (C, B) + g (B, {D}), d (C, D) + g (D, {B})} = min {35+45, 30+35} = 65
- g (D, {B, C}) = min {d (D, B) + g (B, {C}), d (D, C) + g (C, {B})} = min {25+50, 30+45} = 75

Step 4: Final Tour (Starting from A)

- ***f = min {d (A, B) + g (B, {C, D}), d (A, C) + g (C, {B, D}), d (A, D) + g (D, {B, C})}***
- ***f =min {10 + 70, 15 + 65, 20 + 75} =min {80, 80, 95}***

Minimum Cost: 80 km

Optimal Tour: Either ***A → B → D → C → A*** *or* ***A → C → D → B → A.***

Storing only the minimum cost works because of the **Principle of Optimality**, which states that an optimal full tour must consist of optimal sub-paths. Since the cost to complete the journey depends only on your current city and the remaining unvisited
cities, the specific order of how you reached your current state is irrelevant—only the cheapest route to get there matters.

If the distance between B and C drops to 5:

1. **Computation Change:** All DP states involving the (B, C) edge will decrease significantly.
In Step 2, g (B, {C}) becomes 5 + 15 = 20 (was 50), and g (C, {B}) becomes 5 + 10 = 15 (was 45).
2. **New Optimal Tour:**
   - g (B, {C, D}) = min {5+50, 25+45} = 55
   - g (C, {B, D}) = min {5+45, 30+35} = 50
   - g (D, {B, C}) = min {25+20, 30+15} = 45
   - **Final: min {10+55, 15+50, 20+45} = 65**
3. **Result:** The cost drops from **80 to 65.**

The new optimal tour would be **A → B → C → D → A** (or its reverse), as it now prioritizes the very cheap B-C connection.

Q5

Q5	[10][4]	Model	Tracing	TCA	Pruning
		4	4	1	1

State Representation: A state is represented by an array: x[1..4]
where x[i] = j means a sensor is placed in row i, column j.

Decision Variables : For each row i, choose column x[i] ∈ {1,2,3,4}.

Constraints

1. **Row constraint:** One sensor per row → ensured by design.
2. **Column constraint:** $x[i] \neq x[k]$ **for** $i \neq k$
3. Special constraint

1. Zero-Based Indexing

Used when rows and columns are numbered **0, 1, 2, 3**.

The Decision Variable:
Let $C[i]$ be the column index for row i.

The Constraints:

1. **Column Constraint:** $C[i] \neq C[k]$ for all $i \neq k$ (No two sensors in the same column).
2. **Special Positional Constraint:** For $i \in \{0, 1\}$:

$$|C[i] - C[i+2]| \neq 1$$

This means:

- The column for Row 2 cannot be $C[0] + 1$ or $C[0] - 1$.
- The column for Row 3 cannot be $C[1] + 1$ or $C[1] - 1$.

---

2. One-Based Indexing

Used when rows and columns are numbered **1, 2, 3, 4**.

The Decision Variable:
Let $C[i]$ be the column index for row i.

The Constraints:

1. **Column Constraint:** $C[i] \neq C[k]$ for all $i \neq k$.
2. **Special Positional Constraint:** For $i \in \{1, 2\}$:

$$|C[i] - C[i+2]| \neq 1$$

This means:

- The column for Row 3 cannot be $C[1] + 1$ or $C[1] - 1$.
- The column for Row 4 cannot be $C[2] + 1$ or $C[2] - 1$.

Tracing for any one solution

#	Solution Array [C0,C1,C2,C3]
1	[0, 1, 2, 3]
2	[0, 3, 2, 1]
3	[1, 0, 3, 2]
4	[1, 2, 3, 0]
5	[2, 1, 0, 3]

6	[2, 3, 0, 1]
7	[3, 0, 1, 2]
8	[3, 2, 1, 0]

Meaning of Each Constraint
1. No two sensors in the same row
→ Each row must contain exactly one sensor.
2. No two sensors in the same column
→ No two sensors can share a column.
3. No two sensors at (i, j)***and*** $(i + 2, j + 1)$***or*** $(i + 2, j - 1)$
→ Sensors must not be placed in positions that form a knight's move in chess (2 rows down and 1 column left/right).

Time Complexity & Pruning
• **Worst-case time complexity of backtracking:**
$O(4!) = O(24)$
(One column choice per row, no repetitions)
• **Pruning effect:**
Invalid placements (same column or violating the (i+2, j±1) rule) are rejected early, so many branches of the search tree are never explored, **making backtracking much faster than brute-force**
checking of all $44 = 256$ **placemes.**