



Fourth Semester B.E./B.Tech. Degree Examination, June/July 2025

Analysis and Design of Algorithms

Time: 3 hrs.

Max. Marks: 100

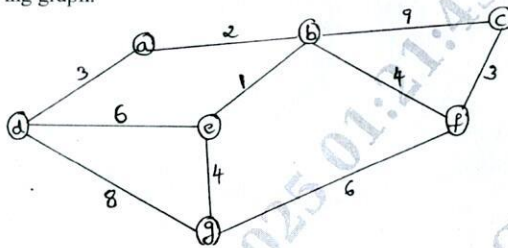
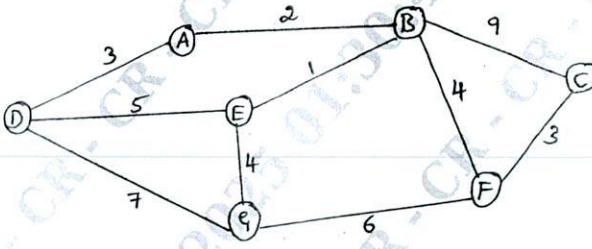
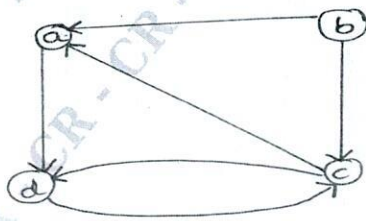
Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.

2. M: Marks, L: Bloom's level, C: Course outcomes.

Module – 1				M	L	C
Q.1	a.	Define algorithm Explain asymptotic notations Big oh, Big omega and Big theta notations.	08	L2	CO1	
	b.	Explain the general plan for analyzing the efficiency of a recursive algorithm. Suggest a recursive algorithm to find factorial of number. Derive its efficiency.	08	L3	CO1	
	c.	If $t_1(n) \in O(g_1(n))$ and $t_2(n) \in O(g_2(n))$ then show that $t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$	04	L2	CO1	
OR						
Q.2	a.	With a neat diagram explain different steps in designing and analyzing algorithm.	08	L2	CO1	
	b.	Write an algorithm to find the max element in an array of n elements. Give the mathematical analysis of this non- recursive algorithm.	08	L3	CO1	
	c.	With the algorithm derive the worst case efficiency for selection sort.	04	L3	CO1	
Module – 2						
Q.3	a.	Explain the concept of divide and conquer. Design an algorithm for merge sort and derive its time complexity.	10	L3	CO2	
	b.	Design an algorithm for insertion algorithm and obtain its time complexity. Apply insertion sort on these elements. 89, 45, 68, 90, 29, 34, 17	10	L3	CO2	
OR						
Q.4	a.	Design an algorithm for Quick sort. Apply quick sort on these elements. 5, 3, 1, 9, 8, 2, 4, 7.	10	L3	CO2	
	b.	Explain Strassen's Matrix multiplication and derive its time complexity.	10	L2	CO2	
Module – 3						
Q.5	a.	Define AVL trees. Explain its four rotation types.	10	L2	CO3	
	b.	Design an algorithm for Heap sort. Construct bottom – up heap for the list 15, 19, 10, 7, 17, 16.	10	L3	CO4	
OR						
Q.6	a.	Design Horspool's Algorithm for string matching Apply Horspool algorithm to find pattern BARBER in the test: JIM_SAW_ME_IN_A_BARBERSHOP.	10	L3	CO4	
	b.	Define heap. Explain the properties of heap along with its representation.	10	L2	CO3	

BCS401

BCS401

Module – 4																		
Q.7	a.	Construct minimum cost spanning tree using Kruskal's algorithm for the following graph.	10	L3	CO4													
																		
		Fig. 7(a)																
	b.	What are Huffman trees? Construct the Huffman tree for the following data	10	L3	CO4													
		<table border="1"><thead><tr><th>Character</th><th>A</th><th>B</th><th>C</th><th>D</th><th>-</th></tr></thead><tbody><tr><td>Probability</td><td>0.4</td><td>0.1</td><td>0.2</td><td>0.15</td><td>0.15</td></tr></tbody></table> <p>i) Encode the text ABAC ABAD ii) Decode the code 100010111001010</p>	Character	A	B	C	D	-	Probability	0.4	0.1	0.2	0.15	0.15				
Character	A	B	C	D	-													
Probability	0.4	0.1	0.2	0.15	0.15													
OR																		
Q.8	a.	Apply Dijkstra's algorithm to find single source shortest path for the given graph by considering A as the source vertex.	10	L3	CO4													
																		
		Fig. 8 (a)																
	b.	Define transitive closure of a graph. Apply Warshall's algorithm to compute transitive closure of a directed graph.	10	L3	CO4													
																		
		Fig. 8 (b)																

BCS401

Module – 5

Q.9	a.	Explain the following with examples. i) P problem ii) NP problem ii) NP-Complete problem iv) NP – Hard problem	10	L2	CO5															
	b.	What is backtracking? Apply backtracking to solve the below instance of sum of subset problem. $S = \{ 1, 2, 5, 6, 8 \}$ and $d = 9$.	10	L3	CO6															
OR																				
Q.10	a.	Illustrate N Queen's problem using backtracking to solve 4 – Queens problem.	10	L2	CO6															
	b.	Using Branch and Bound method solve the below instance of Knapsack Problem. <table><tr><th>Item</th><th>Weight</th><th>Value</th></tr><tr><td>1</td><td>4</td><td>40</td></tr><tr><td>2</td><td>7</td><td>42</td></tr><tr><td>3</td><td>5</td><td>25</td></tr><tr><td>4</td><td>3</td><td>12</td></tr></table> Capacity = 10	Item	Weight	Value	1	4	40	2	7	42	3	5	25	4	3	12	10	L3	CO6
Item	Weight	Value																		
1	4	40																		
2	7	42																		
3	5	25																		
4	3	12																		

CMRIT LIBRARY
BANGALORE - 560 037

CMRIT LIBRARY
BANGALORE - 560 037

Q1. a. Define algorithm. Explain asymptotic notations Big Oh, Big Omega and Big Theta notations.

An algorithm is a finite set of well-defined instructions for solving a problem or performing a computation.

Asymptotic notations describe the running time of an algorithm as a function of the input size:

- Big O (O): Upper bound. Worst-case time complexity.
- Big Omega (Ω): Lower bound. Best-case time complexity.
- Big Theta (Θ): Tight bound. Describes both upper and lower bounds.

These notations are essential for analysing algorithm efficiency.

Q1. b. Explain the general plan for analysing the efficiency of a recursive algorithm. Suggest a recursive algorithm to find factorial of number. Derive its efficiency.

To analyse a recursive algorithm:

1. Identify the base case and recursive case.
2. Write the recurrence relation.
3. Solve the recurrence.

Example: Factorial(n)

Factorial(n):

if $n == 0$ or $n == 1$:

return 1

else:

return $n * \text{Factorial}(n-1)$

Time Complexity: $T(n) = T(n-1) + O(1) \rightarrow O(n)$

Q1. c. If $t_1(n) \in \Theta(g_1(n))$ and $t_2(n) \in O(g_2(n))$, then show that $t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$

Let $t_1(n) \in \Theta(g_1(n)) \Rightarrow \exists c_1, c_2, n_0$ such that $c_1 \cdot g_1(n) \leq t_1(n) \leq c_2 \cdot g_1(n)$ for all $n \geq n_0$

Let $t_2(n) \in O(g_2(n)) \Rightarrow \exists c_3, n_1$ such that $t_2(n) \leq c_3 \cdot g_2(n)$ for all $n \geq n_1$

Let $n_2 = \max(n_0, n_1)$

Then for $n \geq n_2$: $t_1(n) + t_2(n) \leq c_2 \cdot g_1(n) + c_3 \cdot g_2(n) \leq (c_2 + c_3) \cdot \max\{g_1(n), g_2(n)\}$

Hence, $t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$

Q2. a. With a neat diagram explains different steps in designing and analysing algorithm.

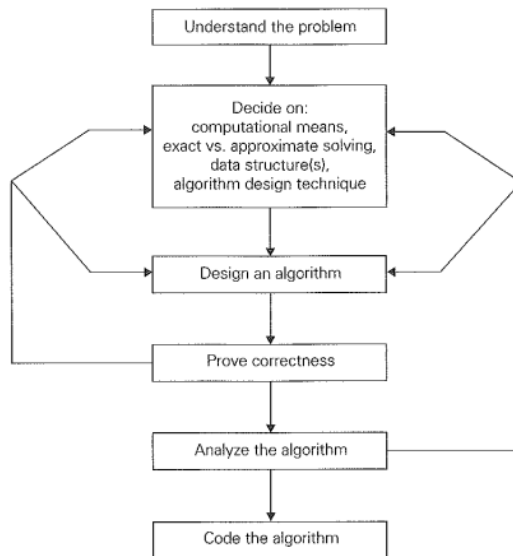


FIGURE 1.2 Algorithm design and analysis process

Steps in designing and analysing algorithms:

1. Problem Definition
2. Algorithm Design
3. Algorithm Specification
4. Algorithm Analysis (Time and Space Complexity)
5. Coding
6. Testing and Debugging

Diagram: A flowchart showing these steps sequentially from problem to analysis and coding.

Q2. b. Write an algorithm to find the max element in an array of n elements. Give the mathematical analysis of this non-recursive algorithm.

Algorithm:

MaxElement($A[1..n]$):

max $\leftarrow A[1]$

for $i = 2$ to n do:

if $A[i] > \text{max}$ then

max $\leftarrow A[i]$

return max

$$C(n) = \sum_{i=1}^{n-1} 1.$$

This is an easy sum to compute because it is nothing else but 1 repeated $n - 1$ times. Thus,

$$C(n) = \sum_{i=1}^{n-1} 1 = n - 1 \in \Theta(n).$$



Time Complexity: $O(n)$

Space Complexity: $O(1)$

Q2. c. With the algorithm derive the worst case efficiency for selection sort.

SelectionSort($A[1..n]$):

for $i = 1$ to $n-1$:

$\text{minIndex} \leftarrow i$

 for $j = i+1$ to n :

 if $A[j] < A[\text{minIndex}]$:

$\text{minIndex} \leftarrow j$

 Swap $A[i]$ with $A[\text{minIndex}]$

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i).$$

We have already encountered the last sum in analyzing the algorithm of Example 2 in Section 2.3 (so you should be able to compute it now on your own). Whether you compute this sum by distributing the summation symbol or by immediately getting the sum of decreasing integers, the answer, of course, must be the same:

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

Time Complexity: $O(n^2)$

Space Complexity: $O(1)$

Q3. a. Explain the concept of divide and conquer. Design an algorithm for merge sort and derive its time complexity.

Divide and Conquer involves breaking a problem into sub-problems, solving them recursively, and combining the results.

MergeSort(A):

if $\text{len}(A) > 1$:

$\text{mid} \leftarrow \text{len}(A)//2$

$L \leftarrow A[:\text{mid}]$; $R \leftarrow A[\text{mid}:]$

 MergeSort(L); MergeSort(R)

 Merge L and R into A

$$C_{\text{worst}}(n) = 2C_{\text{worst}}(n/2) + n - 1 \quad \text{for } n > 1, \quad C_{\text{worst}}(1) = 0.$$

Hence, according to the Master Theorem, $C_{\text{worst}}(n) \in \Theta(n \log n)$ (why?). In fact, it is easy to find the exact solution to the worst-case recurrence for $n = 2^k$:

$$C_{\text{worst}}(n) = n \log_2 n - n + 1.$$

Time Complexity: $O(n \log n)$

Q3. b. Design an algorithm for insertion sort and obtain its time complexity.


```

InsertionSort(A[1..n]):
for i = 2 to n:
    key ← A[i]
    j ← i - 1
    while j > 0 and A[j] > key:
        A[j+1] ← A[j]
        j ← j - 1
    A[j+1] ← key

```

Time Complexity: $O(n^2)$ in worst case, $O(n)$ in best case

Q.4	a.	Design an algorithm for Quick sort. Apply quick sort on these elements. 5, 3, 1, 9, 8, 2, 4, 7.	10	L3	CO2
-----	----	--	----	----	-----

Solution

Algorithm: - [5 Marks]

Analysis step by step: - [5 Marks]

Algorithm

QUICKSORT(arr, low, high):

1. If low < high:
 - a. pivotIndex = PARTITION(arr, low, high)
 - b. QUICKSORT(arr, low, pivotIndex - 1)
 - c. QUICKSORT(arr, pivotIndex + 1, high)

PARTITION(arr, low, high):

1. Set pivot = arr[high]
2. i = low - 1
3. For j from low to high - 1:
 - if arr[j] <= pivot:
 - i = i + 1
 - swap arr[i] and arr[j]
4. swap arr[i+1] and arr[high]
5. return i + 1

Apply Quick Sort on:

Input: 5, 2, 1, 9, 8, 2, 4, 7

Let's sort it using the Quick Sort process step-by-step.

Initial array:

[5, 2, 1, 9, 8, 2, 4, 7]

Apply quicksort on low=0, high=7 \rightarrow pivot = 7

Partitioning:

- Elements \leq 7: [5, 2, 1, 2, 4]
- Elements $>$ 7: [9, 8]
- Pivot 7 goes to correct position \rightarrow index 5

Array becomes:

[5, 2, 1, 2, 4, 7, 8, 9]

Now recursively quicksort on [5, 2, 1, 2, 4] and [8, 9]

Continue Sorting [5, 2, 1, 2, 4] (low=0, high=4)

- Pivot = 4

Partition:

- Elements \leq 4: [2, 1, 2]
- Pivot 4 at correct position index 3

Array becomes:

[2, 1, 2, 4, 5, 7, 8, 9]

Now sort [2, 1, 2] and [5]

Sort [2, 1, 2] (low=0, high=2)

- Pivot = 2

Partition:

- Elements \leq 2: [2, 1, 2] (in-place adjustments)

After partitioning:

[1, 2, 2, 4, 5, 7, 8, 9]

	b.	Explain Strassen's Matrix multiplication and derive its time complexity.	10	L2	CO2
--	----	--	----	----	-----

Solution

Algorithm: - [5 Marks]

Analysis step by step: - [5 Marks]

Strassen's algorithm is an efficient divide-and-conquer algorithm for matrix multiplication. It was proposed by Volker Strassen in 1969 and reduces the number of multiplications compared to the standard method.

Traditional Matrix Multiplication

Given two $n \times n$ matrices A and B, the standard algorithm computes matrix $C = A \times B$ using:

$$C[i][j] = \sum A[i][k] \times B[k][j] \text{ (for } k = 1 \text{ to } n)$$

Number of scalar multiplications: n^3

Time complexity: $O(n^3)$

Idea Behind Strassen's Algorithm

Instead of computing 8 multiplications of submatrices (as in standard divide-and-conquer), Strassen reduces this to 7 multiplications and a few additions/subtractions.

Divide Step

Let A and B be $n \times n$ matrices, where n is a power of 2. Split them into four $n/2 \times n/2$ submatrices:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Strassen's 7 Recursive Multiplications

- $M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$
- $M_2 = (A_{21} + A_{22})B_{11}$
- $M_3 = A_{11}(B_{12} - B_{22})$
- $M_4 = A_{22}(B_{21} - B_{11})$
- $M_5 = (A_{11} + A_{12})B_{22}$
- $M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$
- $M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$

Result Computation

- $C_{11} = M_1 + M_4 - M_5 + M_7$
- $C_{12} = M_3 + M_5$
- $C_{21} = M_2 + M_4$
- $C_{22} = M_1 - M_2 + M_3 + M_6$

Time Complexity Derivation

Let $T(n)$ be the time to multiply two $n \times n$ matrices.

Strassen performs:

- 7 recursive multiplications of size $n/2 \times n/2$: $7T(n/2)$

- 18 matrix additions/subtractions, each costing $O(n^2)$

So: $T(n) = 7T(n/2) + O(n^2)$

Using the Master Theorem:

- $a = 7$, $b = 2$, $f(n) = O(n^2)$

- $\log_2 7 \approx 2.807$

$\Rightarrow T(n) = O(n^{2.807})$

Q.5	a.	Define AVL trees. Explain its four rotation types.	10	L2	CO3
-----	----	--	----	----	-----

Solution

Definition - [2 Marks]

Explanation: - [4Marks]

Example: - [1 Marks]

Definition of AVL Tree

An AVL tree (named after inventors Adelson-Velsky and Landis) is a type of self-balancing binary search tree (BST). In an AVL tree, the heights of the two child subtrees of any node differ by at most one. If at any time they differ by more than one, rebalancing is done using tree rotations. This guarantees that AVL trees maintain $O(\log n)$ time complexity for insertion, deletion, and search operations.

Types of Rotations in AVL Trees

Rotations are used to restore balance when the AVL tree becomes unbalanced after insertions or deletions. There are four types of rotations:

1. Left-Left (LL) Rotation

This rotation is used when a node is inserted into the left subtree of the left child of the unbalanced node. It involves a single right rotation.

Example: Insert into left of left child

2. Right-Right (RR) Rotation

This rotation is used when a node is inserted into the right subtree of the right child of the unbalanced node. It involves a single left rotation.

Example: Insert into right of right child

3. Left-Right (LR) Rotation

This is a double rotation used when a node is inserted into the right subtree of the left child. It involves a left rotation followed by a right rotation.

Example: Insert into right of left child

4. Right-Left (RL) Rotation

This is a double rotation used when a node is inserted into the left subtree of the right child. It involves a right rotation followed by a left rotation.

Example: Insert into left of right child

b.	Design an algorithm for Heap sort. Construct bottom – up heap for the list 15, 19, 10, 7, 17, 16.	10	L3	CO4
----	---	----	----	-----

Solution

Algorithm: - [5 Marks]

Analysis step by step: - [5 Marks]

Heap Sort Algorithm

Heap Sort is a comparison-based sorting algorithm that uses a binary heap data structure. It works by first building a max heap from the input list and then repeatedly removing the maximum element and adjusting the heap.

1. Algorithm Steps:

HEAPSORT(arr):

1. BUILD_MAX_HEAP(arr)

2. for $i = n-1$ down to 1:

 swap arr[0] and arr[i]

 heap_size = heap_size - 1

 MAX_HEAPIFY(arr, 0)

BUILD_MAX_HEAP(arr):

1. for $i = \text{floor}(n/2)$ down to 0:

 MAX_HEAPIFY(arr, i)

MAX_HEAPIFY(arr, i):

1. left = $2i + 1$

2. right = $2i + 2$

3. largest = i

4. if left < heap_size and arr[left] > arr[largest]:

 largest = left

5. if right < heap_size and arr[right] > arr[largest]:

largest = right

6. if largest \neq i:

swap arr[i] and arr[largest]

MAX_HEAPIFY(arr, largest)

Bottom-Up Heap Construction

Given List: [15, 19, 10, 7, 17, 16]

Step 1: Represent as Binary Tree (Index-based)

Index: 0 1 2 3 4 5

Value: 15 19 10 7 17 16

Binary Tree:

```
      15
     /  \
    19   10
   / \  / \
  7 17 16
```

Step 2: Apply Bottom-Up Heap Construction (BUILD_MAX_HEAP)

Start from last non-leaf node: index = floor(n/2) - 1 = 2

i = 2 \rightarrow node = 10

Children: 16

$\rightarrow 16 > 10 \Rightarrow$ swap \rightarrow [15, 19, 16, 7, 17, 10]

i = 1 \rightarrow node = 19

Children: 7, 17

\rightarrow 19 is already largest \Rightarrow no change

i = 0 \rightarrow node = 15

Children: 19, 16

$\rightarrow 19 > 15 \Rightarrow$ swap \rightarrow [19, 15, 16, 7, 17, 10]

Now MAX_HEAPIFY(1): Children: 7, 17

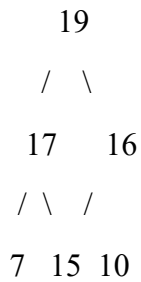
$\rightarrow 17 > 15 \Rightarrow$ swap \rightarrow [19, 17, 16, 7, 15, 10]

Final Max Heap

Index: 0 1 2 3 4 5

Value: 19 17 16 7 15 10

Binary Tree:



OR					
Q.6	a.	Design Horspool's Algorithm for string matching Apply Horspool algorithm to find pattern BARBER in the test: JIM_SAW_ME_IN_A_BARBERSHOP.	10	L3	CO4

Solution

Algorithm: - [5 Marks]

Analysis step by step: - [5 Marks]

Horspool's Algorithm for String Matching

Horspool's algorithm is an efficient algorithm for exact string matching. It is a simplified version of the Boyer-Moore algorithm that uses only the **bad character heuristic** to shift the pattern efficiently when mismatches occur.

Algorithm Design (Horspool's Algorithm)

Input:

- Text TT of length nn
- Pattern PP of length mm

Output:

- All positions where pattern PP occurs in text TT

🔑 Steps of the Algorithm

1. Preprocessing: Create Shift Table

- For each character in the alphabet:

$\text{Shift}[c] = m(\text{default value})$ $\text{Shift}[c] = m \quad \text{default value}$

- For each character in the pattern except the last:

$\text{Shift}[P[i]] = m - i - 1$ $\text{Shift}[P[i]] = m - i - 1$

2. Matching: Slide pattern over text

- Start comparing from the end of the pattern.

- If a mismatch occurs, shift the pattern by the value in the shift table for the mismatched character.
- If a match is found, report the position and shift.

Apply Horspool's Algorithm to Find BARBER in the Text

Pattern (P): "BARBER" (length = 6)

Text (T): "JIM_SAW_ME_IN_A_BARBERSHOP." (length = 28)

Step 1: Build the Shift Table

Pattern: B A R B E R

(only first 5 characters used for shift table)

Character Position Shift = $m - i - 1$

B	0	5
A	1	4
R	2	3
B	3	2 (overwrite previous B)
E	4	1

Final Shift Table:

- B → 2
- A → 4
- R → 3
- E → 1
- All other characters → 6 (pattern length)

Step 2: Matching Phase

We compare pattern with substrings in the text, aligning from right to left.

- Start with text index = 0
- Pattern aligns from index 0 to 5 in text, then slide

Let's go through important alignment positions:

Alignment at index 17:

Text substring = "BARBER"

Compare from right → left:

- R == R ✓
- E == E ✓

- $B == B \checkmark$
- $R == R \checkmark$
- $A == A \checkmark$
- $B == B \checkmark$

Pattern found at index 17

Next alignment would go beyond the length of text, so algorithm stops.

Output:

Pattern BARBER found at position 17 in the text.

	b.	Define heap. Explain the properties of heap along with its representation.	10	L2	CO3
--	----	--	----	----	-----

Solution

Definition: - [2 Marks]

Explanation & Example: - [4+4 Marks]

Definition of Heap

A Heap is a specialized tree-based data structure that satisfies the heap property. It is a complete binary tree, meaning all levels are fully filled except possibly the last, which is filled from left to right.

There are two types of heaps:

- Max-Heap: The value of each node is greater than or equal to its children.
- Min-Heap: The value of each node is less than or equal to its children.

Properties of Heap

1. Shape Property:
 - Heap is always a complete binary tree ensuring efficient storage and minimal height.
2. Heap Order Property:
 - In Max-Heap, the parent node has a value greater than or equal to its children.
 - In Min-Heap, the parent node has a value less than or equal to its children.
3. Efficient Operations:
 - Insertion, deletion, and access to the root element (max or min) are done in $O(\log n)$ time.

Representation of Heap

Heaps are commonly implemented using arrays. The relationships between parent and child nodes can be defined as:

- $\text{Parent}(i) = (i - 1) // 2$
- $\text{Left Child}(i) = 2 * i + 1$
- $\text{Right Child}(i) = 2 * i + 2$

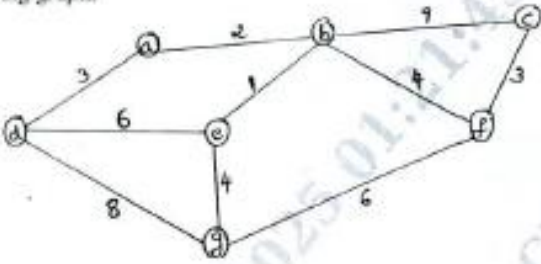
Example (Max-Heap)

Array Representation: [50, 30, 40, 10, 5, 20, 30]

Tree Representation:

```
      50
     /  \
    30   40
   / \  / \
  10  5 20 30
```

Q7

Q.7	a.	Construct minimum cost spanning tree using Kruskal's algorithm for the following graph.	10	1.3	CO4
 <p>Fig. 7(a)</p>					

(a). Construct the Minimum Cost Spanning Tree using Kruskal's algorithm

Given Graph:

Vertices: A, B, C, D, E, F, G, H

Edges (with weights):

Edge	Weight
------	--------

A-B	3
-----	---

A-D	5
-----	---

B-C	2
-----	---

B-E	6
-----	---

C-F	7
-----	---

D-E	7
-----	---

E-F	3
-----	---

E-G	8
-----	---

F-H 4

G-H 9

Kruskal's Algorithm Steps:

Sort edges by increasing weight.

Add edge if it doesn't form a cycle.

Stop when $n-1$ edges are included (where n = number of vertices = 8, so we need 7 edges).

✔ Step-by-step Execution:

Sorted Edges:

Step	Edge	Weight	Added?	Reason
1	B-C	2	✔	No cycle
2	A-B	3	✔	No cycle
3	E-F	3	✔	No cycle
4	F-H	4	✔	No cycle
5	A-D	5	✔	No cycle
6	B-E	6	✔	No cycle
7	C-F	7	✗	Forms a cycle
8	D-E	7	✗	Forms a cycle
9	G-H	9	✔	No cycle

✔ Minimum Cost Spanning Tree (Selected Edges):

B-C (2)

A-B (3)

E-F (3)

F-H (4)

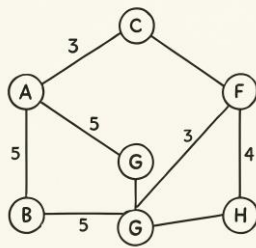
A-D (5)

B-E (6)

G-H (9)

Total Cost = $2 + 3 + 3 + 4 + 5 + 6 + 9 = 32$

**Minimum Cost Spanning
Tree using Kruskal's algorithm**



7.b – Huffman Tree Construction and Encoding/Decoding

Given:

Character A B C D

Probability 0.4 0.1 0.2 0.15

Step 1: Build Huffman Tree

1. **List all probabilities (ascending):**
 - B (0.1), D (0.15), C (0.2), A (0.4)
2. **Combine lowest two nodes:**
 - B (0.1) + D (0.15) → Node1 (0.25)
3. **New list:**
 - Node1 (0.25), C (0.2), A (0.4)
4. **Combine lowest two:**
 - C (0.2) + Node1 (0.25) → Node2 (0.45)
5. **New list:**
 - Node2 (0.45), A (0.4)
6. **Combine remaining:**
 - A (0.4) + Node2 (0.45) → Root (0.85)

Step 2: Assign Binary Codes

- **Root**
 - Left: A → 0
 - Right: Node2

- Left: C \rightarrow 10
- Right: Node1
 - Left: B \rightarrow 110
 - Right: D \rightarrow 111

Final Codes:

Character Huffman Code

A	0
B	110
C	10
D	111

Step 3: Encode ABAC ABAD

Text: ABAC ABAD

Encoding:

- A \rightarrow 0
- B \rightarrow 110
- A \rightarrow 0
- C \rightarrow 10

(space is ignored or handled separately)

- A \rightarrow 0
- B \rightarrow 110
- A \rightarrow 0
- D \rightarrow 111

Encoded String:

0110010010011001111

Step 4: Decode 1000101111001010

Encoded: 1000101111001010

Using tree:

- $1 \rightarrow \text{next is } 0 \rightarrow 10 = C$
- $0 = A$
- $0 = A$
- $1 \rightarrow \text{next } 0 \rightarrow 10 = C$
- $1 \rightarrow \text{next } 1 \rightarrow \text{next } 1 = 111 = D$
- $0 = A$
- $1 \rightarrow \text{next } 1 \rightarrow 0 = 110 = B$

Decoded String: CAACDAB

Q.8	a.	Apply Dijkstra's algorithm to find single source shortest path for the given graph by considering A as the source vertex.	10	L3	CO4
-----	----	---	----	----	-----

Fig.8 (a)

Solution :

Graph Vertices: A, B, C, D, E, F

Edges and Weights: $A \rightarrow B=2$, $A \rightarrow D=3$, $A \rightarrow E=5$, $B \rightarrow C=9$, $B \rightarrow E=1$, $C \rightarrow F=3$, $D \rightarrow F=7$, $E \rightarrow F=4$

Initialize distances from A: $A=0$, $B=\infty$, $C=\infty$, $D=\infty$, $E=\infty$, $F=\infty$; Visited = \emptyset

Iteration 1 (Current = A): Update $B=2$, $D=3$, $E=5$

Iteration 2 (Current = B): Update $C=11$, $E=3$ (better path via B)

Iteration 3 (Current = D): Update $F=10$

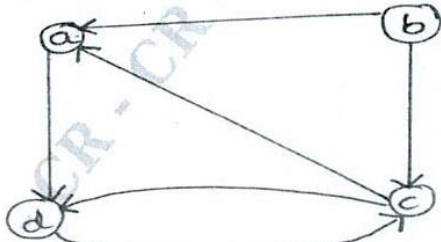
Iteration 4 (Current = E): Update $F=7$ (better path via E)

Iteration 5 (Current = F): Update $C=10$ (better path via F)

Iteration 6 (Current = C): All neighbours visited. Done.

Final Shortest Paths from A:

Vertex	Distance	Path
A	0	A
B	2	$A \rightarrow B$
C	10	$A \rightarrow B \rightarrow E \rightarrow F \rightarrow C$
D	3	$A \rightarrow D$
E	3	$A \rightarrow B \rightarrow E$
F	7	$A \rightarrow B \rightarrow E \rightarrow F$

	<p>b. Define transitive closure of a graph. Apply Warshall's algorithm to compute transitive closure of a directed graph.</p>  <p>Fig.8 (b)</p>	10	L3	CO4
--	---	----	----	-----

Solution :

Definition: The transitive closure of a directed graph $G = (V, E)$ is a graph $G = (V, E)$ such that for every pair of vertices (u, v) , there is an edge (u, v) in E if and only if there is a path from u to v in G .

Warshall's Algorithm:

For an adjacency matrix A of size $n \times n$:

for k from 1 to n :

for i from 1 to n :

for j from 1 to n :

$A[i][j] = A[i][j] \text{ OR } (A[i][k] \text{ AND } A[k][j])$

Initial Adjacency Matrix:

	a	b	c	d
a	0	1	1	0
b	0	0	1	0
c	0	0	0	1
d	0	0	1	0

Final Transitive Closure Matrix:

	a	b	c	d
a	0	1	1	1
b	0	0	1	1
c	0	0	1	1
d	0	0	1	1

Module – 5				
Q.9	a.	Explain the following with examples. i) P problem ii) NP problem ii) NP-Complete problem iv) NP – Hard problem	10	L2 CO5

Solution :**i) P Problem (Polynomial-Time Problems)**

A problem is in P if it can be solved by a deterministic Turing machine in polynomial time.

Examples: Sorting a list using merge sort → Time: $O(n \log n)$

Finding the shortest path using Dijkstra's algorithm → Time: $O(V^2)$

Matrix multiplication

ii) NP Problem (Nondeterministic Polynomial Time)

A problem is in NP if a proposed solution can be verified in polynomial time.

Examples: Sudoku puzzle: Hard to solve, easy to verify.

Hamiltonian Path: Is there a path that visits each vertex exactly once?

iii) NP-Complete Problem

A problem is NP-Complete if: 1. It is in NP. 2. Every problem in NP can be reduced to it in polynomial time.

Examples: Travelling Salesman Problem (Decision version)

3-SAT Problem, Subset Sum Problem

iv) NP-Hard Problem

A problem is NP-Hard if every problem in NP can be reduced to it in polynomial time, but it may not be in NP.

Examples: Halting Problem (undecidable)

Optimization version of Travelling Salesman Problem

	b.	What is backtracking? Apply backtracking to solve the below instance of sum of subset problem. $S = \{1, 2, 5, 6, 8\}$ and $d = 9$.	10	L3	CO6
--	----	---	----	----	-----

Solution :

Backtracking is a problem-solving technique where we **build a solution incrementally**, and if we find that the current path cannot lead to a solution, we **backtrack** and try a different path.

We will explore each element (Include or Exclude it):

Step	Current Subset	Sum	Decision
1	{}	0	Start
2	{1}	1	Include 1
3	{1, 2}	3	Include 2
4	{1, 2, 5}	8	Include 5
5	{1, 2, 5, 6}	14	$> 9 \rightarrow$ Backtrack
6	{1, 2, 5, 8}	16	$> 9 \rightarrow$ Backtrack
7	{1, 2, 6}	9	Valid Subset
8	{1, 5}	6	Continue
9	{1, 5, 6}	12	$> 9 \rightarrow$ Backtrack
10	{1, 8}	9	Valid Subset
11	{2}	2	Try this path
12	{2, 5}	7	Try more
13	{2, 5, 6}	13	$> 9 \rightarrow$ Backtrack
14	{2, 5, 8}	15	$> 9 \rightarrow$ Backtrack
15	{2, 6}	8	Continue
16	{2, 6, 8}	16	$> 9 \rightarrow$ Backtrack
17	{2, 8} 10	> 9	\rightarrow Backtrack
18	{5, 6} 11	> 9	\rightarrow Backtrack
19	{5, 8} 13	> 9	\rightarrow Backtrack
20	{6, 8} 14	> 9	\rightarrow Backtrack

Valid Subsets Found:

{1, 2, 6}

{1, 8}

Q.10	a.	Illustrate N Queen's problem using backtracking to solve 4 – Queens problem.	10	L2	CO6
------	----	--	----	----	-----

Solution :

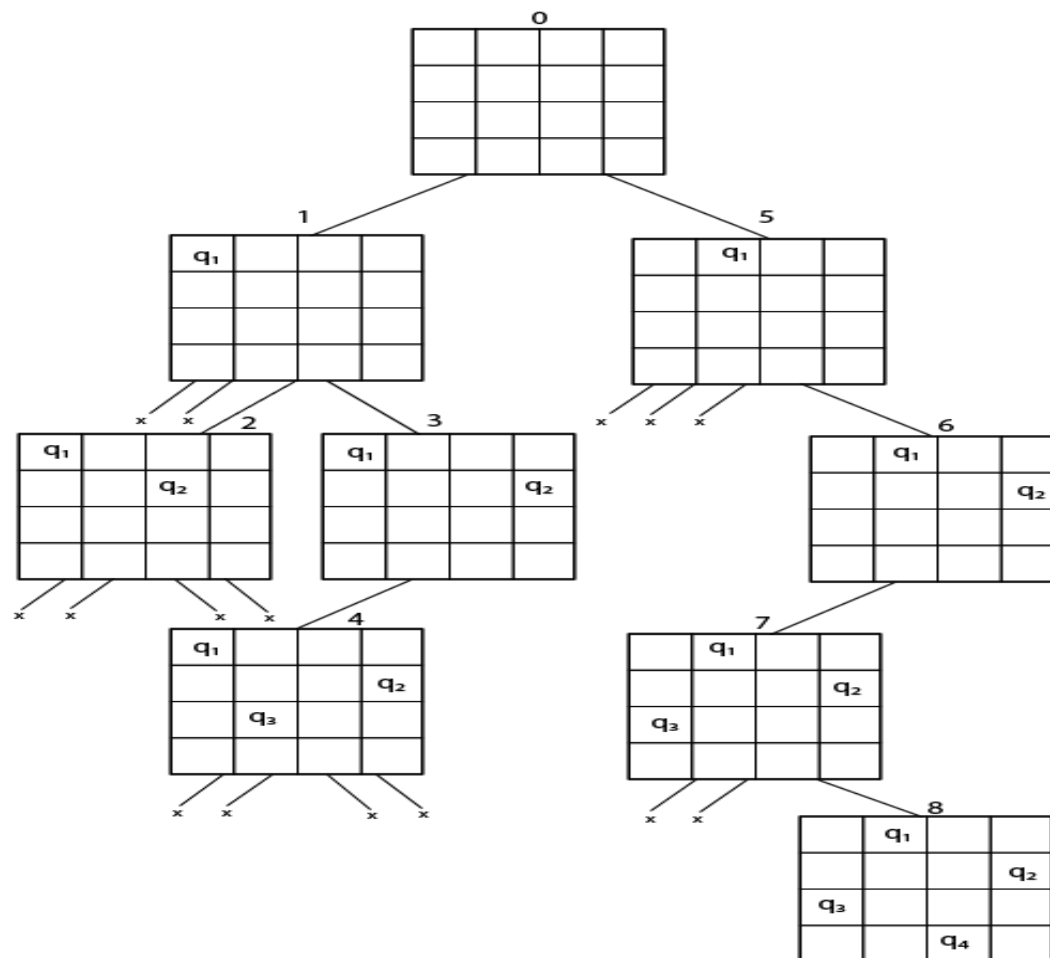
N - Queens problem is to place n - queens in such a manner on an n x n chessboard that no queens attack each other by being in the same row, column or diagonal.

It can be seen that for $n=1$, the problem has a trivial solution, and no solution exists for $n=2$ and $n=3$. So first we will consider the 4 queens problem and then generate it to n - queens problem.

Given a 4 x 4 chessboard and number the rows and column of the chessboard 1 through 4

	1	2	3	4
1			q_1	
2	q_2			
3				q_3
4		q_4		

The implicit tree for 4 - queen problem for a solution (2, 4, 1, 3) is as follows:



b.	Using Branch and Bound method solve the below instance of Knapsack Problem.	10	L3	CO6															
	<table><tr><th>Item</th><th>Weight</th><th>Value</th></tr><tr><td>1</td><td>4</td><td>40</td></tr><tr><td>2</td><td>7</td><td>42</td></tr><tr><td>3</td><td>5</td><td>25</td></tr><tr><td>4</td><td>3</td><td>12</td></tr></table>	Item	Weight	Value	1	4	40	2	7	42	3	5	25	4	3	12			
Item	Weight	Value																	
1	4	40																	
2	7	42																	
3	5	25																	
4	3	12																	
	Capacity = 10																		

Solution :

Here's a step-by-step solution for the 0/1 Knapsack Problem using the Branch and Bound method based on the given instance:

Problem Statement

Item	Weight	Value
1	4	40
2	7	42
3	5	25
4	3	12

Capacity of Knapsack = 10

Step 1: Calculate Value/Weight Ratio

To prioritize items in bounding, sort them by value-to-weight ratio (v/w):

Item	Weight	Value	v/w (Value/Weight)
1	4	40	10.00
2	7	42	6.00
3	5	25	5.00
4	3	12	4.00

So we reorder as: Item 1, Item 2, Item 3, Item 4

Step 2: Branch and Bound (Best-First Search Approach)

Each node includes:

Level (item number)

Current profit

Current weight

Bound (maximum possible profit from that node)

Bounding Function:

Used to compute upper bound of maximum profit in subtree rooted at current node:

Include items greedily until capacity is full or fractionally if needed.

Step 3: Execution Table

Node	Level	Profit	Weight	Bound	Action
------	-------	--------	--------	-------	--------

N0	-1	0	0	94	Root Node
N1	0	40	4	85	Include I1
N2	0	0	0	85	Exclude I1
N3	1	40	4	85	From N1
N4	1	82	11	-	Invalid (wt>10)
N5	1	40	4	85	Exclude I2
N6	2	65	9	77	Include I3 Best Feasible
N7	2	40	4	64	Exclude I3
N8	3	52	7	-	Include I4
N9	3	40	4	-	Exclude I4

Optimal Solution:

Maximum profit = **65**

Selected Items:

- Item 1 (weight 4, value 40)
- Item 3 (weight 5, value 25)

Total weight = 9, which is within capacity.