LESSON PLAN 2025-26(WINTER)						
NAME OF THE TEACHER : DEEPAK KUMAR BARDA, LECT.(STAGE-II,CSE)						
Subject: **ALGORITHMS(Course Code: CSEPC 209/TH5)** Program: Diploma in Computer Science and Engineering Semester: 3rd Total Contact Hours: 45 Total Marks: 100 Assessment: Internal Assessment – 30, End Term – 70						
After completion of the course, the students will be able to: CO1- Define Algorithm with its characteristics. CO2-Write algorithms with pseudocode. CO3-Implement algorithms for sorting and searching using appropriate data structures. CO4-Analyze the time and space complexity of algorithms CO5-Design solutions using advanced data structures for real-world applications, such as shortest path problems or flow-based algorithms.						
Day	**Unit**	**Topic/Sub-Topic**	**Learning Objective**	**Activities**	**Homework**	**COURSE OBJECTIVE**
Unit I: Introduction to Algorithms (4 Lectures)						
1	I	Introduction to Algorithms: Definition and Criteria	Define an algorithm and its essential characteristics.	Discuss real-world algorithms (e.g., a recipe, a set of directions). Work through a simple example of a sorting algorithm.	Write down a simple algorithm for making a sandwich, listing all the steps.	CO1
2	I	Writing an Algorithm with Pseudocode	Write an algorithm using pseudocode.	Introduce the conventions of pseudocode. Practice writing a simple algorithm for finding the largest number in a list.	Write a pseudocode algorithm for checking if a number is prime.	CO1
3	I	Algorithms vs. Programs	Differentiate between an algorithm and a program.	Discuss how an algorithm is a conceptual solution, while a program is its implementation in a specific language.	Explain the difference between an algorithm and a program in your own words.	CO1
4	I	Review and Mini-Quiz	Consolidate knowledge of Unit I.	Quiz on definitions, characteristics, and pseudocode. Solve a simple pseudocode problem.	N/A	CO1
Unit-2:Algorithmic Complexity(8 Periods)						
5	II	Algorithmic Complexity: Introduction	Understand the concept of algorithmic complexity.	Introduce the idea of measuring an algorithm's efficiency. Discuss why complexity is important.	Research and write a short paragraph on why a computer scientist needs to understand algorithmic complexity.	CO2

6	II	Space Complexity	Analyze the memory usage of an algorithm.	Walk through examples to calculate the space complexity of simple algorithms.	Find the space complexity of an algorithm for reversing an array.	CO2
7	II	Time Complexity	Analyze the time an algorithm takes to run.	Work through examples to calculate the time complexity of a loop and nested loops.	Find the time complexity of an algorithm that checks if an array contains duplicate elements.	CO2
8	II	Worst-Case, Average-Case, and Best-Case Analysis	Distinguish between the three types of analysis.	Compare the performance of a linear search algorithm in its best, average, and worst-case scenarios.	Write a short paragraph on the difference between worst-case and best-case time complexity.	CO2
9	II	Big-O Notation (1/2)	Understand the concept of Big-O notation.	Introduce Big-O as the formal notation for expressing worst-case complexity. Practice finding the Big-O for simple algorithms.	Determine the Big-O notation for an algorithm that finds the sum of all elements in an array.	CO2
10	II	Big-O Notation (2/2)	Practice finding the Big-O notation for more complex algorithms.	Work on examples involving multiple loops, conditional statements, and function calls.	Analyze a provided code snippet and determine its Big-O notation.	CO2
11	II	Finding the Complexity of an Algorithm	Apply all concepts to analyze a full algorithm.	Walk through a step-by-step analysis of a complete algorithm, breaking it down into operations.	Find the time and space complexity of an algorithm that sorts an array using a simple sorting method.	CO2
12	II	Unit II Review & Quiz	Consolidate knowledge of algorithmic complexity.	Quiz on Big-O notation, space and time complexity, and the different analysis cases.	N/A	CO2
Unit-3: Recursive algorithms(6 Periods)						
13	III	Recursive Algorithms: Concept of Recursion and Iteration	Differentiate between recursive and iterative algorithms.	Discuss the concept of recursion with a clear base case. Compare a recursive function to an iterative loop for the same task.	Write both a recursive and an iterative algorithm for a simple problem like summing numbers up to N.	CO3
14	III	Examples of Recursive Algorithms: Factorial and Fibonacci	Work through classic examples of recursive algorithms.	Walk through the logic and call stack of the factorial and Fibonacci functions.	Implement a recursive function for the factorial of a number.	CO3

15	III	The Tower of Hanoi Problem	Solve a classic recursive problem.	Use a physical model or an online simulator to demonstrate the Tower of Hanoi. Walk through the recursive solution.	Write the pseudocode for the Tower of Hanoi problem.	CO3
16	III	Complexities of Recursive Algorithms	Analyze the time and space complexity of recursive functions.	Introduce recurrence relations to analyze the complexity of recursive algorithms.	Find the time complexity of a recursive Fibonacci algorithm.	CO3
17	III	Conversion of Recursive to Iterative Algorithm	Convert a recursive solution to an iterative one.	Show how to transform a recursive factorial function into an iterative one. Discuss the pros and cons of each.	Convert the recursive Fibonacci function into an iterative one.	CO3
18	III	Unit III Review & Quiz	Consolidate knowledge of recursion.	Solve practice problems and answer conceptual questions on recursion.	N/A	CO3
Unit-4 : Algorithm Paradigms(07 Periods)						
19	IV	Algorithm Paradigms: Greedy	Understand the Greedy approach to problem-solving.	Discuss the concept of making locally optimal choices. Work through a classic example like the coin change problem.	Explain why the Greedy approach doesn't always work for every problem.	CO4
20	IV	Algorithm Paradigms: Divide and Conquer	Understand the Divide and Conquer approach.	Explain the three steps: divide, conquer, and combine. Discuss famous examples like Merge Sort.	Write a short explanation of the Divide and Conquer approach.	CO4
21	IV	Algorithm Paradigms: Dynamic Programming (1/2)	Understand the concept of Dynamic Programming.	Introduce the idea of storing solutions to subproblems to avoid re-computation. Work through the Fibonacci sequence example with memoization.	Explain the difference between Dynamic Programming and Divide and Conquer.	CO4
22	IV	Algorithm Paradigms: Dynamic Programming (2/2)	Solve a more complex Dynamic Programming problem.	Work through a problem like the knapsack problem or finding the shortest path with costs.	Solve a simple version of the knapsack problem on paper.	CO4

23	IV	Algorithm Paradigms: Backtracking	Understand the Backtracking approach.	Explain the concept of exploring all possibilities and backtracking when a path fails. Work through the N-Queens problem or a Sudoku solver.	Draw a simple search tree for a backtracking problem like finding a path in a maze.	CO4
24	IV	Algorithm Paradigms: Branch and Bound	Understand the Branch and Bound approach.	Compare Branch and Bound with Backtracking, emphasizing the use of bounds to prune the search space.	Research a real-world application of the Branch and Bound algorithm.	CO4
25	IV	Unit IV Review & Quiz	Consolidate knowledge of algorithm paradigms.	Solve conceptual problems and trace the execution of an algorithm using each paradigm.	N/A	CO4
Unit-5: Sorting(09 Periods)						
26	V	Sorting: Bubble Sort, Selection Sort, Insertion Sort	Implement and analyze simple sorting algorithms.	Live coding session for each sorting algorithm. Draw diagrams to show how elements are moved.	Manually sort a list of 5 numbers using each of the three algorithms.	CO5
27	V	Sorting: Merge Sort	Understand and implement Merge Sort using the Divide and Conquer approach.	Draw the recursion tree for Merge Sort. Code the algorithm.	Write the pseudocode for the merge() function in Merge Sort.	CO5
28	V	Sorting: Quicksort	Understand and implement Quicksort.	Discuss the choice of pivot. Walk through the partitioning process with a whiteboard example.	Manually sort a list of numbers using Quicksort.	CO5
29	V	Sorting: Heap Sort	Understand and implement Heap Sort.	Introduce the concept of a heap data structure. Explain the heapify process.	Draw the heap representation of a given array.	CO5
30	V	Sorting: Radix Sort	Understand a non-comparison-based sorting algorithm.	Explain how Radix Sort works by sorting based on digits. Discuss its limitations.	Manually sort a list of 3-digit numbers using Radix Sort.	CO5

31	V	Searching: Symbol Tables and Binary Search Trees	Introduce the concept of searching and Symbol Tables.	Explain how data is stored and retrieved. Introduce Binary Search Trees as an efficient data structure for searching.	Draw a Binary Search Tree created from a given sequence of numbers.	CO5
32	V	Searching: Balanced Search Trees	Understand the need for balanced trees.	Discuss how an unbalanced tree can degrade to a linked list. Introduce the concept of AVL trees or Red-Black trees.	Explain why a balanced tree is important for search efficiency.	CO5
33	V	Hashing and Hash Tables	Understand the concept of hashing and Hash Tables.	Explain how hashing maps keys to indices. Discuss collision resolution techniques.	Draw a simple hash table and show how a few key-value pairs are stored.	CO5
34	V	Unit V Review & Quiz	Consolidate knowledge of sorting and searching.	Solve practice problems on different sorting algorithms and discuss the complexities of each.	N/A	CO5
Unit 6:Graphs(11 Periods)						
35	VI	Graphs: Definition and Terminologies	Define key terms related to graphs.	Draw a graph and label the vertices, edges, paths, and cycles. Discuss directed vs. undirected graphs.	Find a real-world example of a graph and identify its vertices and edges.	CO5
36	VI	Graph Traversal: BFS and DFS	Implement Breadth-First Search and Depth-First Search.	Use a sample graph to trace the path of both BFS and DFS. Discuss their applications.	Given a graph, write down the order of nodes visited by both BFS and DFS.	CO5
37	VI	Topological Sorting	Understand the concept and applications of topological sorting.	Walk through a dependency graph (e.g., course prerequisites) and perform a topological sort.	Write a short paragraph on a real-world use case for topological sorting.	CO5
38	VI	Minimum Spanning Tree Algorithms: Prim's Algorithm	Understand and implement Prim's algorithm.	Trace the algorithm on a weighted graph, step by step, to find the MST.	Solve a new MST problem on a weighted graph using Prim's algorithm.	CO5
39	VI	Minimum Spanning Tree Algorithms: Kruskal's Algorithm	Understand and implement Kruskal's algorithm.	Compare Prim's and Kruskal's algorithms. Trace Kruskal's on the same weighted graph.	Explain the key difference between Prim's and Kruskal's algorithms.	CO5

40	VI	Shortest Path Algorithms: Dijkstra's Algorithm (1/2)	Understand and implement Dijkstra's algorithm.	Walk through a step-by-step example of finding the shortest path in a weighted graph.	Given a new graph, manually find the shortest path using Dijkstra's algorithm.	CO5
41	VI	Shortest Path Algorithms: Dijkstra's Algorithm (2/2)	Continue practicing Dijkstra's algorithm.	Work on more complex graphs and discuss its limitations (e.g., negative weights).	Find the shortest path from a given source to all other vertices in a weighted graph.	CO5
42	VI	Flow-Based Algorithms	Introduce the concept of network flow.	Discuss the maximum flow problem and introduce the Ford-Fulkerson algorithm.	Research and explain a real-world application of flow-based algorithms (e.g., airline scheduling).	CO5
43	VI	Unit VI Review & Quiz	Consolidate knowledge of graphs and related algorithms.	Solve a mix of graph traversal and shortest path problems.	N/A	CO5
44	General	Comprehensive Course Review	Review all key topics from the entire course.	Q&A session covering all units. Work through a challenging problem that requires integrating multiple concepts.	N/A	CO5
45	Final	Final Assessment	Final practical exam or comprehensive test.	N/A	N/A	CO5

Deepak Kumar Barah 11.07.2025
Signature of Teacher

Deepak Kumar Bal 11.07.2025
Signature of HOD