Analysis and Design of Algorithms

COURSE OUTLINE

1.0 COURSE DETAILS			
Course Title	**Analysis and Design of Algorithms**		
Major	**B.Tech. CS & AI, B.Tech. CS & DS**		
Credits	**4**	**Domain with Course Code**	**CSA311**
Mode	**Regular Classroom**	**Period**	**Odd Semester 2025**
Academic Year and Semester	**2025-26, Sem-3**	**Category (Core/Audit)**	**Core**
Instructor		**Office Hours**	**One hour / week**
LTP (Lecture-Tutorial-Practical)	**2-0-4**		

2.0 SUBJECT OVERVIEW The course description should convey to students the intellectual goals of the course -- e.g., the rationale for the course, the guiding questions for the course, the general content of the course.	
Facilitator's Vision:	**1. Students are familiar with advanced data structures and problem solving techniques.** **2. Students will develop a structured approach to analyze the performance and trade-offs of different algorithms.** **3. Students have Intermediate level of problem solving skills**
Relevance for Students:	Students become better problem solver and are prepared for software engineering interviews
Uniqueness of the program	The proposed program emphasizes practical problem-solving over theoretical coverage, integrates modern topics like Monotonic Stacks, and offers early, in-depth exposure to dynamic programming and graph algorithms.
Pre-Requisites:	**1. Understanding and proficiency in at least 1 programing language** **2. Familiarity with data structures like Arrays, Trees** **3. Understanding of recursion**
Intellectual Goals & Guiding Questions:	**Intellectual Goals:** 1. Developing right mental models for problem solving 2. Given any problem, developing the ability to come up with at least one solutions 3. Ability to compare different solutions

	Guiding Questions: 1. Have students improved in problem solving skills? 2. What are the main pillars of problem solving? 3. How we can evaluate an algorithm 4. To solve a problem can we (and how do we) go through all the possible solution space?.

3.0 EDUCATIONAL OBJECTIVES	
Objectives	1. Understand and apply advanced data structures such as Trees, Graphs, Heaps in solving complex computational problems. 2. Demonstrate proficiency in intermediate to advanced problem-solving techniques, including greedy algorithms, dynamic programming, and backtracking. 3. Devise multiple algorithmic approaches to a given problem and evaluate their feasibility. 4. Analyze and compare solutions based on time and space complexity and edge case handling. 5. Develop the ability to optimize solutions iteratively by incorporating insights from dry runs, test cases, and complexity analysis.

4.0 LEARNING OUTCOMES

Outcomes

Identify the learning outcomes based on Skills, Knowledge, Attitudes and Values that will be worked upon during the course.

Skills

Problem Solving and Algorithmic Thinking

1. Break down complex problems and learn how to approach problems .
2. Ability to come up with at least one solution of DSA problems
3. Select and apply appropriate algorithmic paradigms such as divide and conquer, greedy, dynamic programming, and backtracking.
4. Use recursion and backtracking strategies effectively.

Implementation and Coding Proficiency

1. Implement data structures (e.g., BST, heaps, graphs) and algorithms in Python
2. Utilize built-in libraries and modules

Analytical Skills

1. Analyze time and space complexity using Big O notation.
2. Compare different algorithms based on efficiency and scalability.

Debugging and Testing

1. Ability to dry run the solution to validate algorithm correctness.
2. Ability to debug the code

Knowledge

Advanced Data Structures

1. Understand the structure, operations, and use-cases of Binary Search Trees, Heaps, and Graphs.
2. Grasp abstract data types such as Priority Queues and how they are implemented using heaps.

Algorithmic Paradigms

		1. Learn divide and conquer (e.g., Quick Sort), greedy methods, backtracking, and dynamic programming. 2. Understand graph traversal techniques (DFS, BFS) and their applications. 3. Grasp advanced concepts such as topological sort, shortest path algorithms (e.g., Dijkstra's, Bellman-Ford), and Minimum Spanning Trees. **Advanced Topics** 1. Understand optimization techniques within dynamic programming.
	Attitudes/Values	1. Embrace the iterative nature of problem-solving; debug and refine solutions through trial and error. 2. Learn to communicate algorithmic solutions clearly in both code and written form.

5.0 PROGRAM OUTLINE
Map the learnings of each week of the course

WEEK	CONTENT	ACTIVITY/OUTCLASS	ASSIGNMENTS/ASSESSMENTS
1	**Problem Solving Foundations** 1. Introduction to problem-solving approach **Complexity Analysis** 1. Time and Space Complexity 2. Big-O Notation 3. Best, Average, Worst-case analysis		*Code-based Quiz* *Coding Assignment*
2-3	**Advanced Sorting** 1. Quick Sort: Algorithm, Implementation, and Complexity **Binary Search Trees (BST)** 1. BST: Structure, Operations (Insert, Delete, Search) 2. Problem-solving using BSTs		*Code-based Quiz* *Coding Assignment*
3-5	**Heaps** 1. Problem solving using Heaps,		*Code-based Quiz* *Coding Assignment*

	2. Min/Max Heap **Backtracking** 1. Introduction to Backtracking 2. Problem solving using Backtracking		*Code-based Quiz* *Coding Assignment*
5-6	**Graphs-I** 1. Introduction to Graphs, Representation (Adjacency List/Matrix) 2. Depth-First Search (DFS) 3. Breadth-First Search (BFS)		*Code-based Quiz* *Coding Assignment*
7	**Dynamic Programming-I** 1. Introduction to DP, Memoization 2. Classic problems: Climbing Stairs, House Robber		*Code-based Quiz* *Coding Assignment*
8	**Greedy Algorithms** ● Introduction to Greedy Algorithms ● Problem solving using greedy approach		*Code-based Quiz* *Coding Assignment*

9-10	**Dynamic Programming-II** 1. Memoization, Tabulation and Space Optimization		*Code-based Quiz* *Coding Assignment*
11-12	**Graphs-II** 1. Shortest Path Algorithms 2. Topological Sorting 3. Minimum Spanning Tree **Monotonic Stack** 1. Implementation and use cases 2. Problem solving using Monotonic Stack		*Code-based Quiz* *Coding Assignment*

6.0 EVALUATION CRITERIA

Map the percentage weightage to each category.

Function	% Weightage	Notes
Attendance/Class Participation	**5%**	**Participation in class**
Assignment & contests	**35%**	**weekly contest, practice problems**
In-Class and Lab Assignments	**5%**	**Time bound quizzes and assignments in the labs and classes**
viva	**5%**	**Conceptual questions & and hands-on coding**
Mid Semester	**15%**	**Mid-term Exam**
End Semester	**35%**	**End-term Exam**

7.0 REQUIRED READINGS AND REFERENCES

List of required and suggested material. Can mention week-wise or topic-wise. Can also mention how to access readings.

Readings	Introduction to Algorithms (3rd Edition) CLRS
Videos	
Others	