

Course Code	Course Title	L	T	P	C
BACSE105	Data Structures and Algorithms	3	0	2	4
Pre-requisite	NIL	Syllabus Version			
		1			
Course Objectives					
1. Analyze algorithm efficiency using asymptotic notation to compare the complexities in various problem-solving scenarios.					
2. Understand and implement core data structures and their applications in solving computational problems					
3. Design and apply appropriate algorithmic strategies and data structures to solve real-world problems using suitable techniques					
Course Outcomes					
1. Design and analyze efficient algorithms using complexity analysis and formal development techniques					
2. Implement and utilize linear and non-linear data structures for effective data management and algorithmic problem-solving					
3. Apply various design strategies to solve real-world problems efficiently					
4. Analyze and solve complex computational problems using advanced algorithmic techniques and optimization methods					
5. Demonstrate the ability to implement and apply fundamental and advanced algorithms using various data structures to solve computational problems					
Module:1	Algorithm Analysis	8 hours			
Importance of algorithms and data structures - Fundamentals of algorithm analysis: Space and time complexity of an algorithm, Types of asymptotic notations and orders of growth - Algorithm efficiency – best case, worst case, average case - Analysis of non-recursive and recursive algorithms - Asymptotic analysis for recurrence relation: Iteration Method, Substitution Method, Master Method and Recursive Tree Method, Proof of Correctness of the algorithm.					
Module:2	Data Structures Linear and Non Linear	14 hours			
Arrays: 1D and 2D array- Stack, Queue - Types of Queue: Circular Queue, Double Ended Queue (deQueue), List: Singly linked lists, Doubly linked lists, Circular linked lists. Binary Tree: Definition and Properties - Tree Traversals- Binary Search Trees, Application, Balanced Binary Trees-AVL Tree, Graphs: representation, Traversal: BFS & DFS, Hashing					
Module:3	Algorithm Strategies Divide Conquer Back tracking Branch and Bound	7 hours			
Divide and Conquer: Quick sort, Merge Sort, Karatsuba faster integer multiplication algorithm					

Backtracking: N-Queens problem, Graph Coloring, 0/1 knapsack		
Branch & Bound: LIFO-BB and FIFO BB methods: Job Selection problem, Subset Sum		
Module:4	Algorithm Strategies Greedy and Dynamic Programming	7 hours
Greedy Technique: Huffman Coding, Minimum Spanning Tree: Prim's & Kruskal's, Shortest Path : Dijkstra's, Dynamic programming: Matrix Chain Multiplication, Longest Common Subsequence		
Module:5	Classes of Complexity and Approximation Algorithms	7 hours
The Class P - The Class NP - Reducibility and NP-completeness – SAT (Problem Definition and statement), 3SAT, Independent Set, Clique, Approximation Algorithm – Vertex Cover, Set Cover and Travelling salesman		
Total Lecture Hours:		43 hours
Text Book(s)		
Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C., " Introduction to algorithms ", MIT press, 4 th Edition, 2022		
Reference Books		
Kleinberg, J., & Tardos, E, " Algorithm design ", Pearson Education India, 1 st Edition, 2006 Mark A. Weiss, " Data Structures & Algorithm Analysis in C++ ", Pearson Education, 4 th Edition, 2013 Horowitz, Sahni and S. Anderson-Freed, " Fundamentals of Data Structures in C ", Universities Press, 2 nd Edition, 2008 Karumanchi, N, " Data Structures and Algorithms Made Easy: Data Structures and Algorithmic Puzzles ", VISIONIAS, 5 th Edition, 2017		
Indicative Experiments		
1. <div style="border: 1px solid black; padding: 5px; display: inline-block;">Arrays: 1D,2D, functions</div> Tentative Scenario: <div style="border: 1px solid black; padding: 10px; margin-top: 10px;"> Assume that you are a game developer designing a multiplayer battle game called " Battle of Legends". This game involves five players in real time. Each player has the following score : Health score (starts from 1000), Magic score (starts from 50) and Status (A-Active, KO-Knocked Out, D-Disconnected). Write a program to to perform / monitor: Health score and Magic score of each player, Update the values during the game play, Display the current statistics. </div>		2 hours

<p>2.</p> <p>Pointers</p> <p>Tentative Scenario:</p> <p>A college auditorium is used for various events like workshops, seminars, and fests. For each event, the seating arrangement may vary in terms of Number of rows, Number of seats per row, Dynamic seat availability (booked vs. empty).</p> <p>The administration needs a flexible program to configure the seat layout dynamically and simulate real-time seat booking and cancellation. Use the pointers to design this dynamic seat management system.</p>	<p>2 hours</p>
<p>3.</p> <p>Stacks and Applications: a) Infix to postfix conversion, b) Expression Evaluation c) Tower of Hanoi</p> <p>Tentative Scenario:</p> <p>In programming languages, expressions and code blocks often use grouping symbols like (), {}, and []. Ensuring these symbols are properly nested and balanced is crucial for: Compiling code correctly, Preventing runtime or syntax errors, Assisting with code auto-completion and formatting in IDEs. This checking is performed during lexical and syntactic analysis in compilers and interpreters — a task that is ideally solved using stacks due to their LIFO (Last-In, First-Out) nature.</p> <p>Write a C program that checks whether all opening and closing brackets in an input expression are correctly matched and properly nested.</p>	<p>2 hours</p>
<p>4.</p> <p>Queue: Bounded Queue, Priority Queue</p> <p>Tentative Scenario:</p> <p>In a busy call center, incoming customer calls are handled in the exact order they arrive to ensure fairness and efficiency. When all agents are occupied, incoming calls wait in a queue until an agent becomes available. Since the call center has a limited number of agents and finite queue capacity, calls may have to wait or sometimes be rejected when the queue is full.</p> <p>Demonstrates the concept of a bounded queue where the queue has a maximum size to implement this scenario.</p>	<p>2 hours</p>

<p>5.</p> <p>Linked list: Polynomial Manipulation</p> <p>Tentative Scenario:</p> <p>In mathematical computing, polynomials are often represented as a series of terms, each consisting of a coefficient and exponent. The number of terms and their order may vary dynamically based on operations like addition, multiplication, or simplification.</p> <p>Create a program to represent two polynomials using singly linked lists and perform addition / multiplication.</p>	<p>4 hours</p>
<p>6.</p> <p>Trees: Binary Search tree: Expression trees, Finding Kth Min and Max element, Binary tree traversals</p> <p>Tentative Scenario:</p> <p>Design a program that parses and analyzes a mathematical expression using both Expression Trees and Binary Search Trees (BSTs). The goal is to build an expression tree from an arithmetic expression, evaluate it, and store all constants in a BST to perform analysis like finding the K-th smallest and largest constants.</p>	<p>2 hours</p>
<p>7.</p> <p>AVL trees – Rotations</p> <p>Tentative Scenario:</p> <p>Imagine you're building a real-time leaderboard system for a competitive coding platform like HackerRank or Codeforces. Thousands of participants submit solutions, and their scores keep changing dynamically. The leaderboard must: Insert new users quickly, Update scores efficiently, Maintain ranking in sorted order, Always provide balanced search performance.</p> <p>A regular binary search tree may become imbalanced after multiple insertions, leading to inefficient lookups ($O(n)$ time).</p>	<p>2 hours</p>

<div data-bbox="245 206 1059 280" data-label="Text"> <p>To prevent this, we use AVL Trees — a type of self-balancing binary search tree.</p> </div> <div data-bbox="245 392 1043 535" data-label="Text"> <p>Create a program that builds an AVL Tree for storing user scores. Each node stores a user ID and their score. After each insertion, apply necessary rotations to keep the tree height-balanced.</p> </div>	
<div data-bbox="193 562 354 600" data-label="Section-Header"> <h2>8. Graphs</h2> </div> <div data-bbox="202 640 421 672" data-label="Section-Header"> <h3>Tentative Scenario:</h3> </div> <div data-bbox="193 725 1128 833" data-label="Text"> <p>You are tasked with designing a water-pipeline system for a town. The town layout is represented by an undirected, weighted graph $G = (V, E)$, where:</p> </div> <div data-bbox="193 873 675 909" data-label="Text"> <p>$V = n$ represents pipeline junctions,</p> </div> <div data-bbox="193 947 746 983" data-label="Text"> <p>$E = m$ represents possible pipe segments,</p> </div> <div data-bbox="193 1021 1128 1090" data-label="Text"> <p>each edge $(u, v) \in E$ has a weight $w(u, v)$ indicating the pipe installation cost or distance.</p> </div> <div data-bbox="193 1131 287 1164" data-label="Text"> <p>Tasks:</p> </div> <div data-bbox="193 1205 488 1238" data-label="Text"> <p>Traverse the Network</p> </div> <div data-bbox="193 1276 1106 1388" data-label="Text"> <p>(a) Use Breadth-First Search (BFS) starting from a given junction s to determine if all junctions are reachable and record the order of discovery.</p> </div> <div data-bbox="193 1424 1114 1536" data-label="Text"> <p>(b) Use Depth-First Search (DFS) from the same source s to produce a traversal order. (Algorithms should run in $O(n + m)$ time.)</p> </div> <div data-bbox="193 1572 684 1608" data-label="Text"> <p>(c) Compute Shortest Pipeline Routes</p> </div> <div data-bbox="193 1644 1139 1830" data-label="Text"> <p>Apply Dijkstra's algorithm from source s to calculate the minimum-cost path from s to every other junction. Specify the resulting distance and predecessor arrays. (Assume non-negative edge weights; the time complexity is $O(m + n \log n)$.)</p> </div> <div data-bbox="193 1863 684 1901" data-label="Text"> <p>Design the Overall Pipeline Network</p> </div> <div data-bbox="193 1935 1023 2011" data-label="Text"> <p>Construct a Minimum Spanning Tree (MST) of the graph using Kruskal's or Prim's algorithm.</p> </div>	<div data-bbox="1268 1267 1394 1303" data-label="Text"> <p>6 hours</p> </div>

Provide the set of edges chosen and the total installation cost of the MST. (Highlight that MST ensures full connectivity with minimal total weight.)	
9. Use Dynamic Programming to solve Matrix Chain Multiplication, LCS, 0/1 Knapsack problems.	4 hours
10. Demonstrate the following sorting techniques: Quick Sort, Merge Sort, Heap Sort.	2 hours
11. Write a program to solve the subset sum problem using the branch & bound strategy.	2 hours
Total Laboratory Hours:	30 hours
Text Book(s)	
Horowitz, Sahni and S. Anderson-Freed, " Fundamentals of Data Structures in C ", Universities Press, 2 nd Edition, 2008	
Reference Books	
Mark A. Weiss, " Data Structures & Algorithm Analysis in C++ ", Pearson Education, 4 th Edition, 2013 Karumanchi, N, " Data Structures and Algorithms Made Easy: Data Structures and Algorithmic Puzzles ", VISIONIAS, 5 th Edition, 2017	
Mode of Evaluation :Continuous Assessment Test, Digital Assignment, Quiz, Final Assessment Test, Lab Continuous Assessment, Lab Final Assessment, Seminar, Presentaion	
Recommended by Board of Studies :	21-05-2025
Approved by Academic Council : No. 78	12-06-2025