Gyanmanjari Innovative University

Syllabus
Gyanmanjari Diploma Engineering College
Semester-3 (Diploma)

Subject: Optimized Computing with Data Structures - DET2CE13203

Type of course: Professional Core

Prerequisite: Programming fundamentals, logical reasoning, and problem-solving skills with basic understanding of control structures and functions.

Rationale:
Data Structures enable efficient data organization, management, and retrieval for solving complex problems. This course introduces students to fundamental and advanced data structures, starting from arrays and progressing to linked lists, stacks, queues, trees, graphs, and hashing, along with algorithmic techniques such as searching, sorting, and recursion. It emphasizes problem-solving, time and space optimization, and real-world application development, preparing students for technical interviews and advanced domains like software engineering, data science, and system design.

Teaching and Examination Scheme:

Teaching Scheme			Credits	Examination Marks		Total Marks
CI	T	P	C	SEE	CCE	
4	0	2	5	100	50	150

Legends: CI-Class Room Instructions; T – Tutorial; P - Practical; C – Credit; SEE - Semester End Evaluation; CCE-Continuous and Comprehensive Evaluation.

Course Content:

Sr. No	Course Content	Hrs.	% Weightage
1	**Fundamentals of Data Structures, Algorithm Analysis, Arrays and Searching** *Theory Topics:* Introduction to Data Structures, types of data structures (linear and non-linear), Abstract Data Types (ADT), basic problem-solving approach, introduction to algorithms, time and space complexity, asymptotic notations (Big-O, Big-Theta, Big-	18	20%

	Omega), basic analysis of simple statements and loops, introduction to arrays, array representation and indexing, array traversal, insertion and deletion, finding maximum and minimum elements, frequency counting, duplicate element detection, linear search, binary search, and comparison of searching techniques. ***Practical:*** 1. Data Structure Classification: Identify linear and non-linear data structures using suitable examples. 2. Basic Algorithm Development: Prepare step-by-step algorithms for simple problem-solving tasks. 3. Time Complexity Analysis: Analyse the time complexity of simple statements, single loops and nested loops. 4. Array Creation and Traversal: Create an array, accept elements and display them using traversal. 5. Array Insertion: Insert an element at the beginning, end and specified position in an array. 6. Array Deletion: Delete an element from the beginning, end and specified position in an array. 7. Maximum and Minimum Elements: Find the largest and smallest elements in an array. 8. Frequency and Duplicate Detection: Count the frequency of elements and identify duplicate values in an array. 9. Linear Search: Implement linear search and display the position and number of comparisons. 10. Binary Search: Implement binary search on a sorted array and display the position and number of comparisons. 11. Searching Technique Comparison: Compare linear search and binary search based on input condition, number of comparisons and time complexity.		

Examination Style:

Sr. No.	Evaluation Methods	SEE	CCE
1	**Array State Analysis and Operation Planning** Analyse a sequence of array operations, reconstruct the array state, identify boundary conditions, and determine operation complexity.	10	-
2	**Search Decision Case and Evidence Justification** Select a suitable searching technique, demonstrate comparisons, evaluate performance, and justify the decision using time complexity.	10	-
3	**Test-Case Design and Result Validation** Design and execute normal, boundary, invalid, and exceptional test cases to verify the correctness of an array or searching program.	-	10
	Total	20	10

Examination Style:

1. **Array State Reconstruction and Constraint Analysis (10 Marks)**:
   - This assessment will be conducted as a part of the Semester End Evaluation. Students will be provided with an initial array and a problem scenario involving a sequence of operations such as insertion, deletion, traversal, maximum or minimum identification, frequency counting, or duplicate detection. Students are required to determine the appropriate order of operations, represent the array after each step, explain the changes in indexes and element positions, identify possible boundary conditions, and determine the final array state. Students must also mention the time complexity of the selected operations. This assessment focuses on logical analysis and operation planning rather than direct implementation of the practical programs.

	2. **Searching Strategy Selection and Performance Evaluation (10 Marks)**: • This assessment will be conducted as a part of the Semester End Evaluation. Students will be provided with different searching scenarios involving sorted or unsorted arrays, varying input sizes, repeated search requirements, and different performance conditions. Students are required to select the most appropriate searching technique, verify whether the required conditions are satisfied, demonstrate the sequence of comparisons, calculate the number of comparisons, and justify the selected technique using time-complexity analysis. This assessment focuses on technique selection, comparison and justification rather than direct implementation of linear search or binary search programs. **3. Program Testing and Result Validation (10 Marks):** • This assessment will be conducted as a Continuous and Comprehensive Evaluation activity in the programming laboratory. Students will be provided with a functional program based on array operations or searching techniques. Students are required to design and execute normal, boundary, invalid and exceptional test cases in the relevant programming environment. They must record the input, expected output, actual output and pass or fail status of each test case. Students must also explain whether the program handles all specified conditions correctly. The activity will be performed directly in the programming environment and evaluated by the faculty during the scheduled CCE.		
2	**Advanced Arrays, Recursion, Sorting and Problem-Solving Techniques** ***Theory Topics:*** Advanced problem-solving using one-dimensional and two-dimensional arrays, array reversal, matrix addition, subarray generation, sum of subarrays, maximum subarray sum, prefix sum technique and range-sum queries; introduction to recursion, base case and recursive case, recursive problem-solving and problem-reduction approach; recursive implementation of array reversal, maximum element, factorial, Fibonacci series, binary search and Euclidean algorithm for GCD; Tower of Hanoi and	18	20%

	recursive call analysis; basic divide-and-conquer approach; comparison of iterative and recursive solutions based on time and space complexity; introduction to sorting algorithms, bubble sort, selection sort and insertion sort; comparison of sorting algorithms based on comparisons, swaps and efficiency; and structured problem-solving using arrays and recursion. ***Practical:*** 12. Reverse an array using recursion and recursive element swapping. 13. Find the maximum element in an array using recursion and the problem-reduction approach. 14. Implement binary search recursively on a sorted array. 15. Calculate factorial using appropriate base and recursive cases. 16. Generate the Fibonacci series using recursion and analyse repeated computations. 17. Add two two-dimensional matrices and display the resultant matrix. 18. Find the GCD of two numbers using the recursive Euclidean algorithm. 19. Implement the Tower of Hanoi using recursion and determine the total number of moves. 20. Implement bubble sort and observe the comparison and swapping process. 21. Implement selection sort and analyse the comparison and swapping process. 22. Implement bubble sort and display the array after each pass to understand the comparison and swapping process. 23. Merge two sorted arrays into a single sorted array while maintaining the correct ascending order. 24. Generate subarrays and compute the sum of all subarrays using a basic nested-loop approach. 25. Find the maximum subarray sum using a basic approach and analyse its time complexity. 26. Construct a prefix-sum array and use it to solve range-sum queries efficiently.		

Examination Style:

Sr. No.	Evaluation Methods	SEE	CCE
1	**Recursive Behaviour and Resource Interpretation** Interpret a recursive process, identify its base condition, trace recursive calls, determine output, and analyse time and auxiliary space requirements.	10	-
2	**Sorting Behaviour Identification and Evidence Analysis** Identify a sorting algorithm from execution evidence, reconstruct passes, calculate comparisons and swaps, and evaluate its suitability and efficiency.	10	-
3	**Iterative and Recursive Performance Experiment** Develop iterative and recursive solutions, test them with different input sizes, and compare their execution time, calls or iterations, and memory usage.	-	10
	Total	20	10

Examination Style:

1. **Recursive Behaviour and Resource Interpretation (10 Marks)**:
   - This assessment will be conducted as a part of the Semester End Evaluation. Students will be provided with a recursive process represented through a recurrence relation, call diagram, execution table, or partially completed call structure. Students are required to identify the base condition, determine the sequence and total number of recursive calls, estimate the maximum call-stack depth, calculate the expected output, and analyse the time and auxiliary space requirements. Students must also explain how changes in the input size affect the recursive behaviour. This assessment focuses on interpreting recursive execution and resource usage rather than writing or executing the recursive programs included in the practical list.

	2. **Sorting Behaviour Identification and Evidence Analysis (10 Marks)**: • This assessment will be conducted as a part of the Semester End Evaluation. Students will be provided with pass-wise array states, comparison records, swapping information, or execution evidence generated by an unknown sorting process. Students are required to identify the sorting algorithm, justify the identification using the observed behaviour, reconstruct the missing or next pass, calculate the number of comparisons and swaps, and evaluate the efficiency of the algorithm for the given input arrangement. Students must also determine whether another sorting technique would be more suitable for the given data condition. This assessment focuses on evidence interpretation and efficiency analysis rather than direct implementation of a sorting program. 3. **Iterative and Recursive Performance Experiment (10 Marks):** • This assessment will be conducted as a Continuous and Comprehensive Evaluation activity in the programming laboratory. Students will be provided with a faculty-assigned problem that can be solved using both iterative and recursive approaches and that is not directly included in the regular practical list. Students are required to develop both solutions in the relevant programming environment, execute them using different input sizes, and record execution evidence such as the number of iterations, number of recursive calls, execution time, and auxiliary memory requirements. Students must compare the observed results and conclude which approach is more suitable under the given conditions. The activity will be performed directly in the programming environment and evaluated by the faculty during the scheduled CCE.		
3	**Linked Lists and Dynamic Data Management** ***Theory Topics:*** Introduction to linked list, types of linked list (singly, doubly, circular), dynamic memory representation, node structure, operations on linked list (insertion, deletion, traversal), searching in linked list, reversal of linked list, advantages over arrays, and basic problem-solving using linked lists.	18	20%

	Practical: 27. Create and traverse a singly linked list to understand dynamic data storage. 28. Perform insertion at the beginning, end, and specified position in a singly linked list. 29. Perform deletion from the beginning, end, and specified position in a singly linked list. 30. Search for an element in a singly linked list using an iterative approach. 31. Count the total number of nodes in a singly linked list. 32. Reverse a singly linked list using an iterative approach. 33. Reverse a singly linked list using recursion. 34. Find the middle element of a singly linked list using an efficient approach. 35. Detect a loop in a singly linked list using a basic approach. 36. Merge two sorted singly linked lists into a single sorted linked list. 37. Implement a doubly linked list and perform forward and backward traversal. 38. Perform insertion and deletion operations in a doubly linked list. 39. Implement a circular linked list and demonstrate its traversal. 40. Develop a basic record management application using linked-list insertion, deletion, searching, and traversal operations.		

Examination Style:

Sr. No.	Evaluation Methods	SEE	CCE
1	**Linked List Structural Diagnosis and Code Repair** Diagnose and correct faulty or incomplete linked-list programs, repair node connections, handle boundary conditions, and produce the expected output.	20	-
2	**Active Learning Activity (ALA) – Smart Campus Resource Manager Using Linked Lists** Develop a linked-list-based record management application supporting record creation, searching, updating, deletion, and traversal, with documented testing and output evidence.	-	10
	Total	20	10

Examination Style:

1. **Linked List Structural Diagnosis and Code Repair (20 Marks):**
   - This assessment will be conducted as a part of the Semester End Examination. Students must perform four compulsory practical questions carrying 5 marks each in the relevant programming environment during the examination. Each question will contain a faulty or incomplete linked list program based on a small application case. Students are required to identify the structural or logical errors, correct the node connections and program statements, execute the corrected program, and generate the expected output. Each question will combine relevant linked list concepts and suitable boundary conditions. The completed work will be evaluated through practical performance in the relevant programming environment during the examination.

2. **Active Learning Activity (ALA) – Smart Dynamic Record Manager Using Linked Lists (10 Marks):**
   - In this activity, students will design a small dynamic record management application using a suitable linked list structure. The application must combine dynamic record creation, searching, updating, deletion, and

	traversal within a meaningful context. Students must select the appropriate linked list type and handle relevant boundary conditions. The PDF must include the problem statement, selected linked list type, node structure, application flow, important code sections, test cases, output evidence, and conclusion. The completed PDF must be submitted individually through the GMIU Web Portal.		
4	**Stacks, Queues and Their Applications** ***Theory Topics:*** Introduction to stack and queue data structures, LIFO and FIFO concepts, implementation of stack using array and linked list, implementation of queue, circular queue, and double-ended queue (deque), basic operations (push, pop, enqueue, dequeue, peek), applications of stack (expression evaluation, recursion support, balanced parentheses), applications of queue (scheduling, buffering), and problem-solving using stack and queue. ***Practical:*** 41. Implement a stack using an array and perform push, pop, and peek operations. 42. Implement a stack using a linked list for dynamic memory handling. 43. Check balanced parentheses using a stack for expression validation. 44. Convert an infix expression into postfix form using a stack. 45. Evaluate a postfix expression using a stack. 46. Implement stack-based undo and redo functionality. 47. Find the next greater element using a stack. 48. Implement a queue using an array and perform enqueue, dequeue, and peek operations. 49. Implement a queue using a linked list for dynamic memory handling. 50. Implement a circular queue and perform insertion and deletion operations. 51. Implement a deque and perform insertion and deletion from both ends. 52. Simulate a queue-based task scheduling system. 53. Find the first non-repeating element using a queue.	18	20%

Examination Style:

Sr. No.	Evaluation Methods	SEE	CCE
1	**Stack and Queue Application Integration** Select and implement suitable stack, queue, circular queue, or deque structures for application-based problems while handling operations and relevant boundary conditions.	20	-
2	**Active Learning Activity (ALA) – Smart Service Flow Simulator Using Stack and Queue** Develop a service-flow simulation using suitable stack and queue structures, with a defined workflow, boundary-condition handling, test cases, and documented output evidence.	-	10
	Total	20	10

Examination Style:

1. **Stack and Queue Application Integration (20 Marks):**
   - This assessment will be conducted as a part of the Semester End Examination. Students must perform four compulsory practical questions carrying 5 marks each in the relevant programming environment during the examination. Each question will present a small application-based case that requires the suitable use of stack, queue, circular queue, deque, or a combination of these structures. Students are required to understand the given requirement, select the appropriate data structure, implement the required operations, handle relevant boundary conditions, execute the program, and generate the correct output. The questions will combine multiple skills from the unit and will not directly repeat a single regular practical. The completed work will be evaluated through practical performance in the relevant programming environment during the examination.

	2. **Active Learning Activity (ALA) – Smart Service Flow Simulator Using Stack and Queue (10 Marks):** • In this activity, students will design a smart service flow simulator using stack and queue concepts. The system must represent a meaningful process such as customer service handling, task scheduling, request processing, or undo and redo operations. Students must select suitable stack and queue structures, define the workflow, implement the required operations, handle relevant boundary conditions, and test the system using appropriate cases. The PDF must include the problem statement, selected data structures, system workflow, important code sections, test cases, output evidence, and conclusion. The completed PDF must be submitted individually through the GMIU Web Portal.		
5	**Trees, Graphs and Traversal Techniques** ***Theory Topics:*** Introduction to trees, binary tree and binary search tree (BST), tree traversal techniques (inorder, preorder, postorder), properties of trees (height, depth, level), basic operations on BST (insertion, deletion, searching), introduction to graphs, graph representation (adjacency matrix and adjacency list), graph traversal algorithms (BFS, DFS), and basic shortest path concept (introduction). ***Practical:*** 54. Create and traverse a binary tree using recursive methods. 55. Perform inorder, preorder, and postorder traversal of a binary tree. 56. Find the height, depth, and level of nodes in a binary tree. 57. Count the total number of nodes and leaf nodes in a binary tree. 58. Implement level-order traversal of a binary tree using a queue. 59. Implement a Binary Search Tree with insertion and searching operations. 60. Find the minimum and maximum elements in a Binary Search Tree. 61. Perform deletion in a Binary Search Tree while maintaining its properties. 62. Represent a graph using an adjacency matrix and adjacency list. 63. Perform Breadth-First Search traversal on a graph.	18	20%

64. Perform Depth-First Search traversal on a graph.
65. Determine graph connectivity using BFS or DFS traversal.

Examination Style:

Sr. No.	Evaluation Methods	SEE	CCE
1	**Tree and Graph Modelling and Traversal Evaluation** Model application-based information using suitable tree or graph structures, perform required operations and traversals, and generate the correct output.	20	-
2	**Active Learning Activity – Smart Campus Navigation and Hierarchy System** Develop a campus information system using trees for hierarchical data and graphs for location connectivity, supported by testing and output evidence.	-	10
Total		20	10

Examination Style:

1. **Tree and Graph Modelling and Traversal Evaluation (10 Marks):**
   - This assessment will be conducted as a part of the Semester End Examination. Students must perform four compulsory practical questions carrying 5 marks each in the relevant programming environment during the examination. Each question will present an application-based case in which students are required to represent the given information using a suitable tree or graph structure, implement the required operations, execute the selected traversal or searching method, and generate the correct output. The questions may require students to construct a binary search tree, analyse tree properties, represent graph connections, perform BFS or DFS, or determine connectivity using the given data. Each question will combine relevant concepts from the unit instead of directly repeating one regular practical. The completed work will be evaluated through practical performance in the relevant programming environment during the examination.

	2. **Active Learning Activity – Smart Campus Navigation and Hierarchy System (10 Marks):** • In this activity, students will design a smart campus information system using tree and graph concepts. The tree structure must represent hierarchical information such as departments, laboratories, classrooms, or services, while the graph structure must represent connections between selected campus locations. Students must implement suitable tree operations, graph representation, traversal methods, and connectivity checking within the selected system. The PDF must include the problem statement, system structure, selected data representations, application flow, important code sections, test cases, output evidence, and conclusion. The completed PDF must be submitted individually through the GMIU Web Portal.		

Suggested Specification table with Marks (Theory):100

Distribution of Theory Marks (Revised Bloom's Taxonomy)						
Level	**Remembrance (R)**	**Understanding (U)**	**Application (A)**	**Analyze (N)**	**Evaluate (E)**	**Create (C)**
Weightage %	10%	15%	30%	15%	10%	20%

Course Outcome:

After learning the course, the students should be able to:	
CO1	Understand data structure basics and complexity analysis.
CO2	Apply arrays, recursion, sorting, and searching techniques.
CO3	Implement linked lists and basic dynamic memory operations.
CO4	Apply stacks and queues to solve practical problems.
CO5	Design solutions using trees and graphs.

Instructional Method:

The course delivery method will depend upon the requirement of content and need of students. The teacher in addition to conventional teaching method by black board, may also use any of tools such as demonstration, role play, Quiz, brainstorming, MOOCs etc.

From the content 10% topics are suggested for flipped mode instruction.

Students will use supplementary resources such as online videos, NPTEL/SWAYAM videos, e-courses, Virtual Laboratory.

The internal evaluation will be done on the basis of Active Learning Assignment.

Practical/Viva examination will be conducted at the end of semester for evaluation of performance of students in laboratory.

Reference Books:

[1] *Data Structures and Algorithms Made Easy, by Narasimha Karumanchi, CareerMonk Publications.*
[2] *Introduction to Algorithms, by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, MIT Press.*
[3] *Data Structures Using C, by Reema Thareja, Oxford University Press.*
[4] *Data Structures and Algorithm Analysis in C++, by Mark Allen Weiss, Pearson Education.*
[5] *Grokking Algorithms: An Illustrated Guide for Programmers and Other Curious People*, by Aditya Bhargava, Manning Publications.

Suggested Rubrics:

Suggested Assessment Guidelines

Module-1: Fundamentals of Data Structures, Algorithm Analysis, Arrays and Searching

Array State Analysis and Operation Planning (10 Marks)		
Criteria	**Description**	**Marks**
Operation Identification and Planning	Correct identification and logical sequencing of the required array operations according to the given problem scenario.	03
Array-State Reconstruction	Accurate representation of the array after each operation, including changes in indexes and element positions.	03
Boundary and Complexity Analysis	Correct identification of boundary conditions, determination of the final array state, and analysis of the time complexity of the selected operations.	04
	Total	10

Search Decision Case and Evidence Justification (10 Marks)		
Criteria	**Description**	**Marks**
Searching-Technique Selection	Selection of an appropriate searching technique based on array order, input size, search frequency, and given performance conditions.	03
Search Execution and Comparison Analysis	Accurate demonstration of the search sequence, element comparisons, and calculation of the total number of comparisons.	03
Performance Evaluation and Justification	Clear justification of the selected technique using input conditions, efficiency, and time-complexity analysis.	04
	Total	10

Test-Case Design and Result Validation (10 Marks)		
Criteria	**Description**	**Marks**
Test-Case Design	Preparation of suitable normal, boundary, invalid, and exceptional test cases for the given array or searching program.	03
Test Execution and Documentation	Correct execution of test cases with proper recording of input, expected output, actual output, and pass or fail status.	03
Result Analysis and Validation	Accurate evaluation of program behaviour and a clear conclusion regarding whether all specified conditions are handled correctly.	04
	Total	10

Module-2: Advanced Arrays, Recursion, Sorting and Problem-Solving Techniques

Recursive Behaviour and Resource Interpretation (10 Marks)		
Criteria	**Description**	**Marks**
Base Condition and Call Identification	Correct identification of the base condition, recursive condition, call sequence, and total number of recursive calls.	03
Execution and Output Interpretation	Accurate tracing of the recursive process, calculation of the expected output, and determination of maximum call-stack depth.	03
Resource and Input-Size Analysis	Correct analysis of time complexity, auxiliary space requirements, and the effect of increasing input size on recursive behaviour.	04
	Total	10

Sorting Behaviour Identification and Evidence Analysis (10 Marks)		
Criteria	**Description**	**Marks**
Algorithm Identification and Justification	Correct identification of the sorting algorithm from the provided execution evidence with suitable behavioural justification.	03
Pass Reconstruction and Operation Analysis	Accurate reconstruction of the missing or next pass and calculation of comparisons and swaps.	03
Efficiency and Suitability Evaluation	Appropriate evaluation of algorithm efficiency for the given input arrangement and justification of a more suitable alternative, where applicable	04
	Total	10

Iterative and Recursive Performance Experiment (10 Marks)		
Criteria	**Description**	**Marks**
Solution Development and Correctness	Correct development and execution of both iterative and recursive solutions for the faculty-assigned problem.	03
Performance Testing and Evidence Recording	Systematic testing using different input sizes and accurate recording of iterations, recursive calls, execution time, and auxiliary memory usage.	03
Comparative Analysis and Conclusion	Meaningful comparison of both approaches and a justified conclusion regarding their suitability under the given conditions.	04
	Total	10

Module-3: Linked Lists and Dynamic Data Management

Linked List Structural Diagnosis and Code Repair (20 Marks)		
Criteria	**Description**	**Marks**
Error Identification and Diagnosis	Correct identification of the base condition, recursive condition, call sequence, and total number of recursive calls.	05
Node Connection and Operation Repair	Accurate tracing of the recursive process, calculation of the expected output, and determination of maximum call-stack depth.	05
Boundary-Condition Handling	Appropriate handling of conditions such as an empty list, single-node list, invalid position, unavailable element, and beginning or end operations.	05
Program Execution and Output Validation	Successful execution of the corrected programs with accurate output and verification of the expected linked-list behaviour.	05
	Total	20

Module-4: Stacks, Queues and Their Applications

Stack and Queue Application Integration (20 Marks)		
Criteria	**Description**	**Marks**
Requirement Analysis and Data-Structure Selection	Correct identification of the sorting algorithm from the provided execution evidence with suitable behavioural justification.	05
Operation Implementation and Integration	Accurate reconstruction of the missing or next pass and calculation of comparisons and swaps.	05
Boundary-Condition Handling	Appropriate evaluation of algorithm efficiency for the given input arrangement and justification of a more suitable alternative, where applicable	05
Program Execution and Output Validation	Successful execution of the developed programs with correct output and verification of the expected application behaviour.	05
	Total	20

Module-5: Trees, Graphs and Traversal Techniques

Tree and Graph Modelling and Traversal Evaluation (20 Marks)		
Criteria	**Description**	**Marks**
Problem Analysis and Structure Selection	Correct interpretation of the application case and selection of a suitable binary tree, binary search tree, or graph representation.	05
Structure Construction and Operation Implementation	Accurate construction of the selected structure and implementation of required operations such as insertion, deletion, searching, or property calculation.	05
Traversal and Connectivity Evaluation	Correct execution of tree traversals, BFS, DFS, level-order traversal, or connectivity checking according to the given requirement.	05
Program Execution and Output Validation	Successful program execution, appropriate handling of relevant conditions, and generation and verification of the expected output.	05
	Total	20