National University of Engineering (UNI)
School of Artificial Intelligence
Syllabus 2027-I

1. COURSE

CS212. Analysis and Design of Algorithms (Mandatory)

2. GENERAL INFORMATION

2.1 Course	:	CS212. Analysis and Design of Algorithms
2.2 Semester	:	5th Semester.
2.3 Credits	:	4
2.4 Hours	:	2 HT; 4 HP;
2.5 Duration of the period	:	16 weeks
2.6 Type of course	:	Mandatory
2.7 Learning modality	:	Face to face
2.8 Prerrequisites	:	• CS210. Algorithms and Data Structures. (4th Sem) • CS211. Theory of Computation. (4th Sem)

3. PROFESSORS

Meetings after coordination with the professor

4. INTRODUCTION TO THE COURSE

Algorithms are the heart of computer science. This course provides the mathematical and analytical tools necessary to evaluate the efficiency of algorithms in terms of time and space. Students will explore fundamental data structures, complexity analysis, and various design paradigms—such as brute force, greedy, and dynamic programming. Understanding these techniques is essential for building software that is not only functional but also scalable and resource-efficient.

5. GOALS

- Master the use of asymptotic notation to analyze algorithm performance.
- Apply foundational data structures to solve computational problems.
- Use algorithmic strategies like Greedy, Divide and Conquer, and Dynamic Programming.
- Analyze the complexity of algorithms in best, average, and worst cases.
- Develop the ability to select the most appropriate algorithm for a given problem.

6. COMPETENCES

AG-C08) Problem Analysis: Identifies, formulates, and analyzes complex computing problems. (Usage)

AG-C12) Uses systemic approaches to select, develop, apply, integrate, and manage secure computing technologies to achieve user objectives. (Usage)

7. TOPICS

Unit 1: Foundational Data Structures (6 hours)	
Competences Expected: AG-C08,AG-C12	
Topics	**Learning Outcomes**
• Abstract Data Type (ADT) and operations on an ADT: – Dictionary operations (insert, delete, find) • Arrays: – Numeric vs non-numeric, character strings – Single (vector) vs multidimensional (matrix) • Records/Structs/Tuples and Objects • Linked lists (for historical reasons): – Single vs Double and Linear vs Circular • Stacks • Queues and deques: – Heap-based priority queue • Hash tables/maps: – Collision resolution and complexity (e.g., probing, chaining, rehash) • Trees: – Binary, n-ary, and search trees – Balanced (e.g., AVL, Red-Black, Heap) • Sets	• For each ADT/Data-Structure in this unit: – Explain its definition, properties, representation(s), and associated ADT operations. – Explain step-by-step how the ADT operations associated with the data structure transform it. [Explain] • Explain how collision avoidance and collision resolution is handled in hash tables [Explain] • Explain the heap property and the use of heaps as an implementation of a priority queue [Explain]
Readings : [**Cormen2022**], [**Kleinberg2005**]	

Unit 2: Foundational Algorithms (6 hours)	
Competences Expected: AG-C08,AG-C12	
Topics	**Learning Outcomes**
• Graphs (e.g., [un]directed, [a]cyclic, [un]connected, and [un]weighted): – Graph representation: adjacency list vs matrix • Search algorithms: – $O(n)$ complexity (e.g., linear/sequential array/list search) – $O(\log_2 n)$ complexity (e.g., binary search) – $O(\log_b n)$ complexity (e.g., uninformed depth/breadth-first tree search) • Sorting algorithms (e.g., stable, unstable): – $O(n^2)$ complexity (e.g., insertion, selection) – $O(n \log n)$ complexity (e.g., quicksort, merge, timsort) • Graph algorithms: – Shortest path (e.g., Dijkstra's, Floyd's) – Minimal spanning tree (e.g., Prim's, Kruskal's) • Sorting algorithms: – $O(n \log n)$ complexity heapsort – Pseudo $O(n)$ complexity (e.g., bucket, counting, radix) • Graph algorithms: – Transitive closure (e.g., Warshall's) – Topological sort • Matching: – Efficient string matching (e.g., Boyer-Moore, Knuth-Morris-Pratt) – Longest common subsequence matching – Regular expression matching	• For each algorithm in this unit explain step-by-step how the algorithm operates [Explain] • For each algorithmic approach (e.g., sorting) in this unit apply a prototypical example of the approach (e.g., merge sort) [Apply] • Given requirements for a problem, develop multiple solutions using various data structures and algorithms. Subsequently, evaluate the suitability, strengths, and weaknesses selecting an approach that best satisfies the requirements [Create] • Explain factors beyond computational efficiency that influence the choice of algorithms, such as programming time, maintainability, and the use of application-specific patterns in the input data [Explain] • For each of the algorithms and algorithmic approaches in the KA Core topics: – Explain a prototypical example of the algorithm, and – Explain step-by-step how the algorithm operates. [Explain]
Readings : [**Cormen2022**], [**Kleinberg2005**]	

Unit 3: Advanced Algorithms (4 hours)	
Competences Expected: AG-C08,AG-C12	
Topics	**Learning Outcomes**

• Cryptography algorithms (e.g., SHA-256)
• Parallel algorithms
• Consensus algorithms (e.g., Blockchain):
  – Proof of work vs proof of stake
• Quantum computing algorithms:
  – Oracle-based (e.g., Deutsch-Jozsa, Bernstein-Vazirani, Simon)
  – Superpolynomial speed-up via QFT (e.g., Shor's)
  – Polynomial speed-up via amplitude amplification (e.g., Grover's)
• Fast-Fourier Transform (FFT) algorithm
• Differential evolution algorithm
	• An appreciation of quantum computation and its application to certain problems [Explain]

Readings : [**Cormen2022**], [**Kleinberg2005**]	

Unit 4: Algorithmic Strategies (24 hours)	
Competences Expected: AG-C08,AG-C12	
Topics	**Learning Outcomes**
• Paradigms: – Brute-Force (e.g., linear search, selection sort, traveling salesperson, knapsack) – Decrease-and-Conquer: ∗ By a Constant (e.g., insertion sort, topological sort) ∗ By a Constant Factor (e.g., binary search) ∗ By a Variable Size (e.g., Euclid's) – Divide-and-Conquer (e.g., binary search, quicksort, mergesort, Strassen's) – Greedy (e.g., Dijkstra's, Kruskal's, Knapsack) – Transform-and-Conquer: ∗ Instance simplification (e.g., find duplicates via list presort) ∗ Representation change (e.g., heapsort) ∗ Problem reduction (e.g., least-common-multiple, linear programming) ∗ Dynamic programming (e.g., Floyd's, Marshall, Bellman-Ford) – Space vs time tradeoffs (e.g., hashing) • Handling exponential growth (e.g., heuristic A*, branch-and-bound, backtracking) • Iteration vs recursion (e.g., factorial, tree search) • Paradigms: – Approximation algorithms – Iterative improvement (e.g., Ford-Fulkerson, simplex) – Randomized/Stochastic algorithms (e.g., max-cut, balls and bins) • Quantum computing	• For each of the paradigms in this unit: – Explain its definitional characteristics – Explain an example that demonstrates the paradigm including how this example satisfies the paradigm's characteristics. [Explain] • For each of the algorithms in the Foundational Data Structures, Foundational Algorithms, Advanced Algorithms unit, explain the paradigm used by the algorithm and how it exemplifies this paradigm [Explain] • Given an algorithm, explain the paradigm used by the algorithm and how it exemplifies this paradigm [Explain] • Give a real-world problem, evaluate appropriate algorithmic paradigms and algorithms from these paradigms that address the problem including evaluating the tradeoffs among the paradigms and algorithms selected [Evaluate] • Give examples of iterative and recursive algorithms that solve the same problem, explain the benefits and disadvantages of each approach [Explain] • Evaluate whether a greedy approach leads to an optimal solution [Evaluate] • Explain various approaches for addressing computational problems whose algorithmic solutions are exponential [Explain]
Readings : [**Cormen2022**], [**Dasgupta2006**]	

Unit 5: Complexity Analysis Framework (6 hours)	
Competences Expected: AG-C08,AG-C12	
Topics	**Learning Outcomes**
• Complexity Analysis Framework: – Best, average, and worst-case performance of an algorithm – Empirical and relative (Order of Growth) measurements – Input size and primitive operations – Time and space efficiency • Empirical measurements of performance • Time and space tradeoffs in algorithms	• For each algorithm in the Foundational Data Structures, Foundational Algorithms, Advanced Algorithms unit, explain its runtime complexity class and why it belongs to this class [Explain] • Develop empirical studies to determine and validate hypotheses about the runtime complexity of various algorithms by running algorithms on input of various sizes and comparing actual performance to the theoretical analysis [Create] • Explain examples that illustrate time-space tradeoffs of algorithms [Explain] • Explain how tree balance affects the efficiency of binary search tree operations [Explain]
Readings : [**Cormen2022**], [**Sipser2012**]	

Unit 6: Asymptotic Notation and Complexity Classes (6 hours)	
Competences Expected: AG-C08,AG-C12	
Topics	**Learning Outcomes**
• Asymptotic complexity analysis (average and worst-case bounds): – Big-O, Big-Omega, and Big-Theta formal notations – Foundational Complexity Classes and Representative Examples/Problems: ∗ $O(1)$ Constant (e.g., array access) ∗ $O(\log_2 n)$ Logarithmic (e.g., binary search) ∗ $O(n)$ Linear (e.g., linear search) ∗ $O(n \log_2 n)$ Log Linear (e.g., mergesort) ∗ $O(n^2)$ Quadratic (e.g., selection sort) ∗ $O(n^c)$ Polynomial (e.g., $O(n^3)$ Gaussian elimination) ∗ $O(2^n)$ Exponential (e.g., Knapsack, Satisfiability (SAT), Traveling Sales-Person (TSP), all subsets) ∗ $O(n!)$ Factorial (e.g., Hamiltonian circuit, all permutations)	• For each foundational complexity class in this unit, explain an algorithm that demonstrates the associated runtime complexity [Explain] • Apply Big-O notation to give upper case bounds on time/space complexity of algorithms [Apply]
Readings : [**Cormen2022**], [**Sipser2012**]	

Unit 7: Complexity Analysis: Recursion, Amortization, and Tight Bounds (12 hours)	
Competences Expected: AG-C08,AG-C12	
Topics	**Learning Outcomes**
• Little-o, Little-Omega, and Little Theta notations • Formal recursive analysis • Amortized analysis	• Given a problem to program for which there may be several algorithmic approaches, evaluate them and determine which are feasible, and select one that is optimal in implementation and run-time behavior [Evaluate] • Use recurrence relations to evaluate the time complexity of recursively defined algorithms [Apply] • Apply elementary recurrence relations using a form of the Master Theorem [Apply]
Readings : [**Cormen2022**], [**Sipser2012**]	

Unit 8: Computational Complexity Theory (6 hours)	
Competences Expected:	
Topics	**Learning Outcomes**
• Tractability and intractability: – P, NP, and NP-Complete Complexity Classes – NP-Complete Problems (e.g., SAT, Knapsack, TSP) – Reductions	• Explain the significance of NP-Completeness [Explain] • Explain examples of NP-complete problems [Explain] • Explain the classes P and NP [Explain]
Readings : [**Sipser2012**], [**Cormen2022**]	

8. WORKPLAN

8.1 Methodology

Individual and team participation is encouraged to present their ideas, motivating them with additional points in the different stages of the course evaluation.

8.2 Theory Sessions

The theory sessions are held in master classes with activities including active learning and roleplay to allow students to internalize the concepts.

8.3 Practical Sessions

The practical sessions are held in class where a series of exercises and/or practical concepts are developed through problem solving, problem solving, specific exercises and/or in application contexts.

9. EVALUATION SYSTEM

********* EVALUATION MISSING ********

10. BASIC BIBLIOGRAPHY