ANNEX 2 – COURSE DESCRIPTION

Course Title	Algorithms and Complexity				
Course Code	ACSC401				
Course Type	Compulsory				
Level	BSc (Level 1)				
Year / Semester	4th (Fall)				
Teacher's Name	Dr Savvas Pericleous				
ECTS	6	Lectures / week	3	Laboratories/week	-
Course Purpose	The aim of the course is to familiarize students with the concepts and the principles of the design and analysis of algorithms and the evaluation of their performance. We discuss the importance of the underlying data structures towards developing correct and efficient algorithms for solving computational problems and compare various paradigms for doing so. Students will be introduced to basic notions of complexity theory that can be used to classify problems and study examples of approximation algorithms.				
Learning Outcomes	By the end of the course, students must be able to: 1. Introduce the notion of an algorithm for solving computational problems, noting the existence of unsolvable problems. 2. Define the notion of time and space complexity and classify functions by their growth rates. 3. Analyze the running time of various algorithms; employ in particular, the Master Theorem for solving recurrences. 4. Describe and use general techniques, such as the Divide and Conquer and the Dynamic Programming paradigms, for designing correct and effective algorithms 5. Develop, evaluate and reason about the correctness and performance, of sorting algorithms (Insertion Sort, Merge Sort, Heapsort and Quick Sort), write programs to implement these and prove lower bounds for sorting by comparison keys. 6. Analyze graph traversing algorithms (BFS/DFS), compare Kruskall's and Prim's method for finding minimal spanning trees. Introduce Dijkstra's Single Source Shortest-path algorithm and compare with Bellman-Ford algorithm. Understand Floyd–Warshall algorithm for finding all pairs shortest paths and discuss Ford-Fulkerson algorithm for solving the Maximum flow problem. 7. Explain the general notion of complexity classes, P and NP, completeness and hardness, and the relationships between classes by reduction. Compare a range of computational problems according to their classification. 8. Study approximation algorithms for solving hard problems.				

Prerequisites	AMAT181, ACSC191, ACSC288	Corequisites	None
Course Content	• **Computability issues**, need for axiomatic models of computations; unsolvable problems and computers limitation • **Analysing Algorithms and Problems**: Notion of an algorithm; Principles and Examples; Time and space complexity; Classifying functions by their growth rates. • **Algorithms design:** Brute-force**,** Greedy methods**,** Divide and Conquer paradigm, Dynamic Programming paradigm. • **Solving Recurrences**: Substitution and Recursion Tree methods; the Master Theorem for solving recurrences. • **Sorting**: Selection and Insertion Sort; Heapsort, Merge-sort, Quick-sort. Lower bounds for sorting by comparison keys. • **Searching Methods**: Sequential, Binary Search, Binary Search Trees (BST), Balanced BST (Red-Black, AVL), Hash tables. • **Graphs and Trees**: Terminology and graph representation. Graph traversing (Breath/Depth First Search). Kruskal's and Prim's algorithms for Minimal Spanning Trees. Dijkstra's and Bellman-Ford methods for solving the Single Source Shortest-path problem (SSSP). Floyd–Warshall algorithm for finding all pairs shortest paths (APSP). Ford-Fulkerson algorithm for Maximum flow problem. • **Complexity theory**: Classes P and NP, NP-Completeness and Reducibility. Approximation algorithms for finding near-optimal solutions in polynomial time for intractable problems.		
Teaching Methodology	Students are taught the course through lectures (3 hours per week). For the delivery of the class material, power point presentations are primarily used, along with the whiteboard. The lecture notes, consisting of slides presented in class, the course outline and additional material, are made available to the students through the University's e-learning platform. Students are also advised to use the subject's textbook or reference books for further reading and practice in solving related exercises. The theoretical part of each lecture is accompanied with detailed solved examples on which emphasis is given in the class. The solutions to these exercises, as well as specimen solutions for all tests and assignments, are discussed with students. Students are encouraged to make full use of the instructor's office hours (6 per week), where they can ask questions and further discuss lecture material on a one-to-one basis.		
Bibliography	**(a) <u>Textbooks:</u>** • T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms. (3ed Edition) The MIT Press, 2009 **(b) <u>References:</u>** • Levitin A, The Design and Analysis of Algorithms, Pearson International 3rd edition, 2012 • M. T. Goodrich and R. Tamassia, Algorithm Design and Applications, 1st edition Wiley, 2015		
Assessment	The Students are assessed via continuous assessment throughout the duration of the Semester, which forms the Coursework grade and the final written exam. The coursework and the final exam grades are weighted		

	40% and 60%, respectively, and compose the final grade of the course. The methods for the continuous assessment of the students, are primarily mid-term written tests, and assignments, The assessment weight, date and time of each type of continuous assessment is being set at the beginning of the semester via the course outline. An indicative weighted continuous assessment of the course coursework is shown below: • Assignments: 25% • Mid-Term written exams: 75% Students are prepared for the final exam, by revision on the taught material, problem solving and concept testing. The final assessment is designed to comply with the subject's expected learning outcomes.
Language	**English**