School of Computing
UNIVERSITY OF GEORGIA

Course Information Sheet
CSCI 2720
Data Structures

Brief Course Description (50-words or less)	The design, analysis, and implementation of data structures and their associated algorithms; Lists; Stacks; Queues and Priority Queues; Trees; Graphs and Dictionaries; Time and Space Complexity; Sorting and Searching; Advanced problem-solving, and Introduction Algorithm Design Strategies.
Extended Course Description / Comments	This course surveys fundamental and advanced data structures and explores the different implementations (array-based and linked representations) of these data structures. A strong emphasis is placed on the connection between data structures and their algorithms, including analyzing algorithms' running-time complexity and space requirements. The course introduces approaches to algorithm design, including divide and conquer, greedy algorithms, and dynamic programming and their applications through well-known algorithms including tree algorithms, sorting algorithms, and graph algorithms. Key notions of object-oriented programming with a view for efficiency, maintainability, and code-reuse, are emphasized. The Java programming language will be used for coding examples on the concepts discussed in the lecture, and the students will code the course assignments and projects in Java.
Pre-Requisites and/or Co-Requisites	Prerequisite: CSCI 1302 and [CSCI 2610 or CSCI 2611]
Required, Elective or Selected Elective	Required Course
Approved Textbook	The textbook used is up to the instructor's discretion and must conform to the extended course description.
Specific Learning Outcomes (Performance Indicators)	1. Analyze algorithms using the asymptotic analysis notations. 2. Design, analyze, and implement recursive solutions. 3. Design, analyze, and implement generic reusable abstract data types. 4. Articulate algorithm design methods and be able to use them in problem-solving. 5. Implement, and analyze the principal algorithms for sorting, searching, hashing, and graph algorithms and be able to select the appropriate algorithm and apply it to solve real-world problems. 6. Choose the appropriate data structure and algorithm to solve real-world problems and to defend the selection.
ABET Learning Outcomes	Graduates of the program will have an ability to:

A. Analyze a complex computing problem and to apply principles of computing and other relevant disciplines to identify solutions.
B. Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline.
C. Communicate effectively in a variety of professional contexts.
D. Recognize professional responsibilities and make informed judgments in computing practice based on legal and ethical principles.
E. Function effectively as a member or leader of a team engaged in activities appropriate to the program's discipline.
F. Apply computer science theory and software development fundamentals to produce computing-based solutions.

NOTE: In the construction of the student learning outcomes for this course, the instructors interpreted "computing requirements" in (B) as the functional requirements for a software solution and not as specific hardware requirements for the target platform; likewise, the phrase "apply computer science theory" in (F) was interpreted as using computer science principles.

Relationship Between Student Outcomes and Learning Outcomes

	ABET Learning Outcomes						
Specific Learning Outcomes		A	B	C	D	E	F
	1	●	●	●			●
	2	●	●	●			●
	3	●	●	●			●
	4	●	●	●			●
	5	●	●	●			●
	6	●	●	●			●

Major Topics Covered

1.Algorithm Analysis (Knowledge Level: Varies by topic)
a) Explain and compare the growth of functions: Logarithmic functions, Polynomials, Linearithmic, and exponential functions. (Familiarity)
b) Given a function, prove that it falls into a particular complexity class. (Usage)
c) Formally Define the Asymptotic Complexity notations (Big O, Big theta, and Big Omega). Distinguish between case analysis methods (best case, average case, and worst-case analysis). (Familiarity)
d) Calculate the running-time T(n) for a given function by calculating the number of processing steps. (Usage)
e) Find the asymptotic complexity of an algorithms as a function of problem size. (Usage)

2.Recursion (Knowledge Level: Varies by topic)
a) Identify the base case and the general case of a recursively defined problem. (Familiarity)
b) Write recursive functions to solve computational problems. (Usage)
c) Convert a recursive function to an iterative function and vice versa. (Usage)

d) Analyze the running-time for recursive functions by finding the depth of recursion, the tree method, and the master method of decreasing functions. (Usage)
e) Compare iterative and recursive solutions for a problem and select the best solution and be able to defend the selected solution based on non-functional attributes such as efficiency, simplicity, maintainability, code reuse, and others. (Assessment)

3. Abstract Data Types and Their Implementations (Knowledge Level: Varies by topic)

3.1 Lists, Queue, and Stack:

a) Explain the different representations (array and linked implementations) of the ADT and its manipulating operations. (Familiarity)
b) Analyze and assess the impact of the ADT representation on the running-time complexity of the ADT operations and space requirements. (Usage)
c) Implement (with templates) at least two representations (array, singly linked list, doubly linked list, circularly linked list) of at least one of the ADTs and use the implementation to solve a real-world problem. (Usage)

3.2 Search tress: Binary Search trees BST, AVL trees, Red-black trees, and B-Trees: (Knowledge Level: Varies by topic)

a) Explain the different tree implementations and associated operations. (Familiarity)
b) Analyze and assess the impact of the ADT implementation on the running-time complexity of the ADT operations and space requirements. (Usage)
c) Implement at least one search tree. (Usage)
d) Select a tree type that is best suited to solve a problem and be able to defend the selection. (Assessment)

3.3 Heaps and Priority Queues (Knowledge Level: Usage)

a) Implement the Heap ADT and its basic operations.
b) Analyze the basic heap operations.
c) Use the heap structure as an internal traversal method to develop efficient solutions to real world problems.

3.4 Design and implement the Priority Queue ADT using various data structures (Array List, Linked List, and Heap) and evaluate the different implementations with respect to space and running-time efficiency. (Knowledge Level: Usage)

3.5 Graphs and Graph Algorithms (Knowledge Level: Varies by topic)

a) Depict various representation (adjacency list and adjacency matrix) given a picture or a description of the graph. (Familiarity)
b) Compare the adjacency list and the adjacency matrix implementations of the Graph ADT according to the running-time complexity of the graph operations and space requirements. (Usage)
c) Trace basic graph algorithms, including depth first traversal, breadth first traversal, minimum spanning trees (Prim's and Kruskal's), topological sort, and single-source and all-pairs shortest path (Dijkstra's and Bellman Ford's) for a given graph. (Usage)
d) Choose the appropriate graph algorithms to solve a problem given a description of the problem. (Assessment)

3.6 Hash Tables (Knowledge level: Usage)

a) Design and implement two different versions of the hash table interface: Open addressing (Linear probing, Quadratic probing, Double hashing, Rehashing) and chaining.

b) Analyze and assess the running-time complexity of basic hash table operations (insert, delete, and search)

4. Algorithm Design Strategies (Knowledge Level: Varies by topic)
    a) Differentiate between the strategies (greedy, divide-and-conquer, and dynamic programming) through teaching well-known algorithms, and identify a practical example to which the design strategy would apply. (Familiarity)
    b) Use divide-and-conquer to solve an appropriate problem. (Usage)
5. Sorting Algorithms: (Knowledge Level: Varies by topic)
    a) Implement, and analyze sorting algorithms including quadratic sorts, linearithmic sorts, and non-comparison linear sorts. (Usage)
    b) Select a sorting algorithm to solve a given problem based on running-time and space efficiency and be able to defend the selection. (Assessment)
6. Implement and test projects that use data structures to solve real-world problems and satisfying a set of predefined functional requirements (pre/post conditions) and non-functional requirements (efficiency, robustness, code-reuse, maintainability, and others). (Assessment)

Knowledge Levels

The following is the ACM's categorization of different levels of mastery: Assessment, Usage, and Familiarity. Note that Assessment encompasses both Usage and Familiarity, and Usage encompasses Familiarity.

Familiarity: The student understands what a concept is or what it means. This level of mastery concerns a basic awareness of a concept as opposed to expecting real facility with its application. It provides an answer to the question "What do you know about this?"

Usage: The student is able to use or apply a concept in a concrete way. Using a concept may include, for example, appropriately using a specific concept in a program, using a particular proof technique, or performing a particular analysis. It provides an answer to the question "What do you know how to do?"

Assessment: The student is able to consider a concept from multiple viewpoints and/or justify the selection of a particular approach to solve a problem. This level of mastery implies more than using a concept; it involves the ability to select an appropriate approach from understood alternatives. It provides an answer to the question "Why would you do that?"

Course Master

Dr. Eman Saleh

Modified

11/6/2022 by Eman Saleh