Department of Computer Science
Franklin College of Arts and Sciences
UNIVERSITY OF GEORGIA

Course information Sheet

CSCI 2725

Data Structures for Data Science

Brief Course Description (50-words or less)	The design and implementation of data structures. Comparative analysis of their manipulating algorithms and their application to solving data science problems. Topics include: Recursion, Data structures: Lists, Stacks, Queues, and Priority Queues, and Trees, and structures for data science: Graphs, Dictionaries, Decision Trees, Disjoint set, Tensors, and Data Frames.
Extended Course Description / Comments	This course surveys fundamental data structures and explores their different implementations with a focus on their manipulating algorithms. The objective is to effectively use data structures and their manipulating algorithms to design and implement efficient solutions, and to develop problem-solving skills required in the data science paradigm. The course introduces approaches to algorithm design, including divide and conquer, greedy algorithms, and dynamic programming. The course will draw on applications from data science.
Pre-Requisites and/or Co-Requisites	Prerequisite: CSCI 1302
Required, Elective or Selected Elective	Required Course
~~Approved~~ Textbook	Author: Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser Title: Data Structures and Algorithms in Java Edition: 6th Edition. ISBN-13: 9781118771334

Specific Learning Outcomes (Performance Indicators)

1. Use asymptotic analysis notations to give upper bounds on time and space complexity of algorithms.
2. Design and implement recursive functions.
3. Describe, design, and implement generic, reusable abstract data types (ADTs), including sorted list, unsorted list, Tensors, stack, queue, priority queue, tree, graph, and dictionary.
4. Analyze and assess the impact of data structures and algorithm design on the performance of algorithms.
5. Understand algorithm design methods such as the greedy method, divide and conquer, and dynamic programming.
6. Choose the appropriate data structures and algorithm to solve a real world problem and to defend the selection.
7. Design, apply, and compare the complexity of principal algorithms for sorting, searching, and hashing.
8. Understand, and apply graph algorithms such as graph traversal algorithms, minimum spanning tree, topological sort, and single-source shortest path.
9. Write programs to satisfy the requirements of a real word problem, integrating course concepts (data structures and algorithms efficiency) and object-oriented design principles.

ABET Learning Outcomes

A. Graduates of the program will have an ability to: 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.

	ABET Learning Outcomes						
Specific Learning		A	b	C	d	E	f
	1	●	●				●

Relationship Between Student Outcomes and Learning Outcomes

Major Topics Covered

1. Algorithm Analysis [Knowledge Level: Usage]
   a) Explain and compare the growth of functions: Logarithmic functions, Polynomials, Linearithmic, and exponential functions.
   b) 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).
   c) Measure operations as a function of problem size and Compute the asymptotic complexity of simple algorithms/functions.
2. Fundamental Data Structures [Knowledge level: Assessment]
   a) Design, implement, analyze, and assess the different representations of the ADT List (singly linked list, doubly linked list, circularly linked list) and their impact on the run-time and space complexity of the ADT operations.
   b) Design, implement, analyze, and assess the different representations of the Queue ADT and the Stack ADTs and their impact on the run-time and space complexity of the ADT operations.
3. Recursion
   a) Understand, design, and implement recursive functions. [Usage]
4. Binary Search tress (BST), and Balanced Trees. [Knowledge Level: Assessment]
   a) Understand, analyze, and compare BST implementation strategies (array vs. linked representations, iterative vs. recursive algorithms).
   b) Implement BSTs using linked structures.
   c) Design and analyze balanced search trees (AVL trees, red black trees or 2-3 trees, B trees). Evaluate which search tree techniques are best suited to various application scenarios.
5. Heaps and Priority Queues
   a) Understand the implementation of the Heap data structure and basic heap operations. [Knowledge Level: Familiarity]
   b) Use the heap structure as an internal traversal method to develop efficient solutions to real world problems. [Knowledge Level: Usage]
6. Understand, Design, and implement the Priority Queue ADT using various data structures (Array List, Linked List, and Heap); evaluate the different implementations with respect to space and run-time efficiency. [Knowledge Level: Assessment]
7. Graphs and Graph Algorithms
   a) Understand and compare the adjacency list and the adjacency matrix

implementations of the ADT Graph. [Knowledge Level: Familiarity]
   b) Understand graph algorithms, including depth first traversal, breadth first traversal, minimum spanning trees, topological sort, and single-source and all-pairs shortest path. Be able to use graph algorithms to solve problems. [Knowledge Level: Usage]

8. Algorithm design strategies
   a) For each of the strategies (greedy, divide-and-conquer, and dynamic programming), identify a practical example to which it would apply. [Knowledge Level: Familiarity]
   b) Use a divide-and-conquer algorithm to solve an appropriate problem. [Knowledge Level: Usage]
9. Hash Tables [Knowledge level: Usage]
   a) Design and implement two different versions of the hash table interface: Open addressing and chaining. Analyze and assess the complexity of basic hash table operations (insert, delete, and search)

10. Implement, and assess sorting algorithms including quadratic sorts, linearithmic sorts, and non-comparison linear sorts. [Knowledge Level: Assessment]
11. Discuss factors other than computational efficiency that influence the choice of algorithms, such as programming time, maintainability, and the use of application-specific patterns in the input data. [Familiarity]
12. Understand two string-matching algorithms. [Knowledge Level: Usage]
13. Understand and use advanced data structures for data science, including Tensors, Disjoint Sets, Data frames, and Decision Trees. Be able to evaluate and use existing data science libraries that implement these data structures. [Knowledge Level: Usage]

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?"

Modified 9/3/2019 by Dr. Eman Saleh