Course Information Form (CIF)

Course Code: CSC 220

Course Title: Data Structures

Department: Business/Computer Science and Technologies
Effective Date: Summer 2026
PCS Code: 1.1 - Baccalaureate/Transfer
CIP Code: 11.0802
Repeatability: 0

Credit Hours

Catalog Notation: 2-2-3
Credit Hour Distribution:

Lecture: 2	Lab: 2	Clinical: 0	**Total: 3**

General Course Information

Catalog Description

Complex data structures and algorithms including lists, searching and sorting, stacks, queues, trees, graphs, and memory management with emphasis on algorithm analysis.

General Course Objectives

Students will learn how to measure the performance of algorithms. They will also learn about specific concepts including lists, binary trees, graphs, searching, and sorting.

Minimum Placement Levels

English	Reading	Math
None	None	None

Prerequisites

Credit in CSC 125 or CSC 240

Methods of Evaluation

Quizzes (8 or more), Programming Projects (8 or more), a Midterm Exam, a Comprehensive Final, and Lab Programming Assignments (16 or more). Instructors should use additional methods as they see fit.

Instructional Materials and Additional Supplies

Data Structures and Algorithms in Python, Author: Goodrich, Tamassia and Goldwasser. Publisher: Wiley.

Course Content

General Learning Outcomes (GLOs)

- Reasoning and Inquiry: Students will demonstrate the ability to solve problems using deductive reasoning and logic, quantitative reasoning, or the scientific method.

Course Segments and Student Learning Outcomes

Course Segment	Learning Outcomes	Lecture Hours	Lab Hours	Clinical Hours
Programming in Python	1. Demonstrate functions and classes in Python. 2. Explain Python syntax. 3. Demonstrate the ability to make Read - Evaluate - Print - Loop programs in Python.	3	3	0
Object Oriented Programming	1. Describe the goals of OOP. 2. Design and develop classes in Python. 3. Demonstrate the ability to use constructors, accessors, and mutators in accordance with accepted software engineering principles. 4. Design and develop concrete data types using the OOP abstraction, and use classes to encapsulate and modularize code. 5. Design and develop inheritance and abstract classes in Python.	2	2	0
Algorithm Analysis	1. Explain basic algorithm analysis. 2. Classify algorithms using Big O and related notations. 3. Analyze code to determine its algorithm complexity and categorize it into constant, logarithmic, linear, quadratic, and other common classifications.	2	2	0
Dynamic Arrays	1. Describe the structure of referential arrays. 2. Describe shallow copying and deep copying. 3. Describe how dynamic arrays amortize resizing costs. 4. Describe different amortization strategies.	2	2	0
Stacks, Queues, Deques, Singly- and Doubly-linked Lists	1. Utilize abstract data structures such as stacks, queues, and deques. 2. Describe implementations of stacks, queues, and deques based on arrays and linked-node data structures. 3. Analyze when and where one implementation is more efficient than another using Big O notation. 4. Compare singly-linked to doubly-linked lists. 5. Assess the complexity costs of all the sequential data structures. 6. Design and develop an efficient program using these abstract data structures.	3	3	0
General Trees	1. Explain non-linear, hierarchical trees. 2. Perform tree-traversal algorithms. 3. Design and develop an efficient program using tree traversal algorithms.	2	2	0
Priority Queues and Heaps	1. Describe priority queue implementation strategies using arrays, sorted arrays, and heaps. 2. Compare heaps to other trees using Big O notation. 3. Describe the mechanics of a heap. 4. Develop a sort using a heap and analyze the running time. 5. Analyze the complexity costs and compare the tradeoffs of all the priority queue and heap implementations.	2	2	0
Maps and Hash Tables	1. Describe hash functions. 2. Design and develop a program using hashing to implement a map data structure. 3. Determine the complexity costs and tradeoffs of different probing methods.	2	2	0

Course Segment	Learning Outcomes	Lecture Hours	Lab Hours	Clinical Hours
Binary Search Trees	1. Recognize binary trees and general trees. 2. Define how binary trees can make a mapping. 3. Design and develop a program using self-balanced trees to make a map and analyze the benefits of balancing. 4. Describe the different balancing strategies of AVL Trees, RB trees, and Splay trees.	3	3	0
Sorting	1. Describe multiple sorting algorithms including Quicksort. 2. Calculate and graph the runtime complexities of sorting algorithms. 3. Design and develop sorting applications on sequential data structures. 4. Describe how to use pivoting to calculate order statistics.	2	2	0
Graphs	1. Analyze different graph implementations. 2. Analyze the runtime complexities of graph algorithms based on the underlying implementation of the graph. 3. Design and develop code using a graph data structure with traversal algorithms. 4. Design and develop code using a graph data structure with minimum spanning tree algorithms. 5. Design and develop code using a graph data structure to determine the least weighted path between two points.	4	4	0
Dynamic Programming, Greedy Method, Tries	1. Define dynamic programming and demonstrate how it is used. 2. Define a greedy method heuristic and demonstrate how it is used. 3. Define the trie data structure, and analyze the runtime complexities.	3	3	0

Total Contact Hours

Lecture Hours	Lab Hours	Clinical Hours
30	30	0