Modern Course Design and CS Materials

Kalpathi Subramanian, Erik Saule
krs@uncc.edu, erik.saule@gmail.com

The University of North Carolina at Charlotte

BRIDGES Workshop, June 15-17, 2026

Table of Contents

High Level Picture

Structured in Module

Content is grouped themes

Dependencies between modules

- Catch up modules
- Mandatory Modules
- Optional Modules

Goals

- Topics
- Course Learning Outcomes
- Program Learning Outcomes
- Competencies
- Pedagogical Strategies

A Variety of Content

- Lectures
- Videos
- In-class activities
- Assignments
- Exam
- Projects

Accreditation

- SACS
- ABET
- Quality Matters
- *insert your accreditation body here*

What is Alignment?

Properties of how content flow in

- Program
- Course
- Module
- Materials

That could apply to

- Topics
- Outcomes
- Competencies

That could be in term of

- What they cover
- What they assume students know

Aligning Modules with Course Objectives

Courses usually have objectives that come from program descriptions and assessments.
How do we ensure that the content of the class actually serve these higher objective? We want to align the objective modules with the objective of the course.
Two main properties to check:

- Are all the course objectives covered appropriately by a module objective?
- Are there module objectives that serve no course objective?

Alignment within Module

Typical module structure

- Exposition to new concept (lecture, textbook, videos)
- Clarification of concept (discussion, hands-on activity)
- Reinforcement of concept (problem, programming assignment)

Properties you want

- The clarification should not introduce new concepts
- The reinforcement should strengthen the exposition and clarification topics
- The materials should cover the topics the module is meant to cover
- The materials should not wander too far from the module objectives

Assessment

Exam should **never** introduce new concepts

Plan for ITCS 6114: Algorithms and Data Structures

Overreaching Learning Outcomes

OLO1. Articulate that design, complexity, and correctness of algorithms and data structures matter in the real world

OLO2. Design correct and low complexity algorithms and data structures by employing standard techniques

OLO3. Analyze and implement given algorithms and data structures

OLO4. Recognize faulty algorithmic logic

Detailed Learning Outcomes

On Complexity

DLOC1. Interpret complexity notation and their implications on the performance/resource consumption of algorithms (OLO1, OLO2)

DLOC2. Articulate the real-world implication of the design of algorithms and data structures in term of performance (OLO1)

DLOC3. Derive the complexity of algorithms using various techniques (for instance, master theorem, amortized analysis, and average case analysis) (OLO1, OLO2, OLO3, OLO4)

DLOC4. Prove the NP-Completeness of classic problems (OLO2, OLO4)

DLOC5. Leverage the P != NP conjecture to recognize dubious algorithmic claims (OLO4)

On Correctness

DLOCo1. Recognize and prove the invariant of data structures and algorithms (OLO2, OLO4)

On Data Structures

DLOD1. Design, analyze, and implement tree-based indexes (OLO2, OLO3)

DLOD2. Design, analyze, and implement hash-based indexes (OLO2, OLO3)

DLOD3. Design, analyze, and implement classic algorithms on graphs (OLO2, OLO3)

On Algorithmic Techniques

DLOA1. Create, analyze, and implement divide and conquer algorithms (OLO2, OLO3)

DLOA2. Create, analyze, and implement greedy algorithms (OLO2, OLO3)

DLOA3. Create, analyze, and implement dynamic programming algorithms (OLO2, OLO3)

Week 1. Sep 8.

Lecture:

1. Introduction.
2. Complexity notations [DLOC1].

Activity:

1. Proving simple complexity notation properties [DLOC1].
2. Interpreting complexity notation in term of practical cost or feasibility [DLOC1, DLOC2].

Week 2. Sep 15.

Lecture:

1. Analyzing simple algorithms [DLOC3].
2. Invariant and correctness [DLOCo1].
3. Simple recursive complexity formulas [DLOC3].

Activity:

1. Given simple algorithms (binary search, insertion sort, simple nearest neighboor), prove their correctness and complexity [DLOCo1, DLOC3].
2. Implement and benchmark insertion sort and simple nearest neighboor [DLOC1, DLOC2].

Week 3. Sep 22.

Lecture:

1. Divide and Conquer [DLOA1].
2. Merge sort [DLOA1, DLOCo1].
3. Master Theorem [DLOC3].

Activity:

1. Solve some other problem using D&C [DLOA1].
2. Implement and benchmark Merge Sort [DLOC2].

Week 4. Sep 29.

Lecture:

1. Tree-based indexing [DLOD1].
2. Invariant of data structure [DLOCo1].
3. Using BST for associative array [DLOD1, DLOC2].

Activity:

1. Run BST manually on toy example [DLOD1].
2. Design, analyze, and implement a tree base index for nearest neighbor query [DLOD1, DLOC2].

Week 5. Oct 6.

Lecture:

Curriculum Guidelines

What are they?

Usually they are recommendation of what should/could be taught across a program.

Expressed in term of topics, learning outcome, and competencies. Not in term of courses.

Usually make recommendation on how much one should learn in a particular topic, sometimes specified in number of hours.

How can we use them?

Give us a reference of what we should/could be teaching.

Am I covering all that? Should I? Why not?

Give us a common language to communicate between instructors.

General Guidelines: ACM/IEEE CS 2013

Structured in

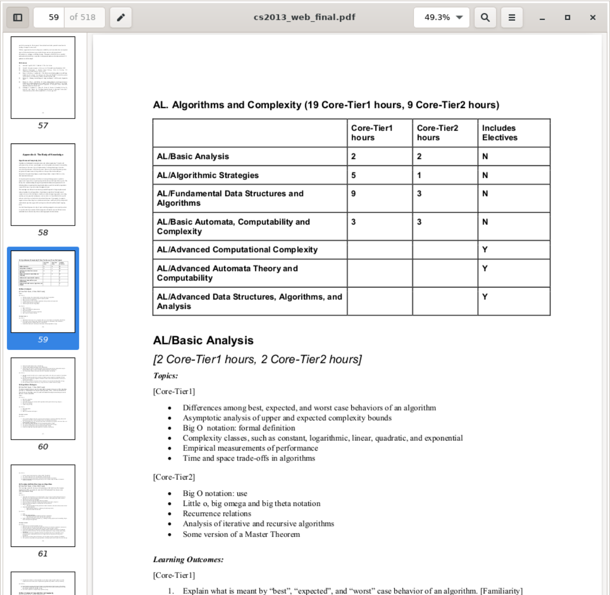
- Knowledge Area
- Knowledge Unit

Topics and Learning Outcomes are classified as

- Tier-1
- Tier-2
- Elective

Other general guidelines:

- Data Science
- Computer Engineering
- Upcoming revised CS



AL. Algorithms and Complexity (19 Core-Tier1 hours, 9 Core-Tier2 hours)

	Core-Tier1 hours	Core-Tier2 hours	Includes Electives
AL/Basic Analysis	2	2	N
AL/Algorithmic Strategies	5	1	N
AL/Fundamental Data Structures and Algorithms	9	3	N
AL/Basic Automata, Computability and Complexity	3	3	N
AL/Advanced Computational Complexity			Y
AL/Advanced Automata Theory and Computability			Y
AL/Advanced Data Structures, Algorithms, and Analysis			Y

AL/Basic Analysis

[2 Core-Tier1 hours, 2 Core-Tier2 hours]

Topics:

[Core-Tier1]

- Differences among best, expected, and worst case behaviors of an algorithm
- Asymptotic analysis of upper and expected complexity bounds
- Big O notation: formal definition
- Complexity classes, such as constant, logarithmic, linear, quadratic, and exponential
- Empirical measurements of performance
- Time and space trade-offs in algorithms

[Core-Tier2]

- Big O notation: use
- Little o, big omega and big theta notation
- Recurrence relations
- Analysis of iterative and recursive algorithms
- Some version of a Master Theorem

Learning Outcomes:

[Core-Tier1]

1. Explain what is meant by “best”, “expected”, and “worst” case behavior of an algorithm. [Familiarity]

Specific Guidelines: NSF/IEEE-TCPP PDC 2012

Structured in domains:

- Programming
- Algorithm
- Architecture

More descriptive.
Bloom levels.
Other specific guidelines: graphics, security

A Lingua Franca

CS Guidelines give us a fairly detailed description of what is in CS.
We can use them as ontologies to describe in a common language what a course of a class material is like.

What do you think is in a lecture entitled UNCC-ITCS-2214-Saule-Graphs?

- Depth- and breadth-first traversals
- Representations of graphs (e.g., adjacency list, adjacency matrix)
- Reflexivity, symmetry, transitivity
- Illustrate by example the basic terminology of graph theory, and some of the properties and special cases of each type of graph/tree.
- Undirected graphs
- Directed graphs
- Weighted graphs
- Iterative and recursive traversal of data structures

CS material pitch