TCSS 343 Master Syllabus
Version: April 2011
(Approved: 27 May 2011)

Catalog Description

Develops competencies associated with problem-solving, algorithms, and computational models. Explores algorithms analysis and design, and computational complexity. Includes efficient algorithms, models of computation, correctness, time and space complexity, NP-complete problems, and undecidable problems. Prerequisite: a minimum grade of 2.0 in TCSS 322; a minimum grade of 2.0 in TCSS 342; a minimum grade of 2.0 in either TQS 124 or MATH 124.

Preconditions

- Apply calculus and algebra skills (limits, derivatives, algebraic manipulation).
- Recognize and use mathematical formalisms (e.g., sets, logic, summations, proof).
- Translate problem descriptions into mathematical formalisms.
- Manipulate (procedural knowledge) and apply mathematical formalisms to solve problems.
- Use recurrence relations to determine the size of a problem space.
- Adapt or extend an implementation of a data structure.
- Analyze the worst- and average-case time and space complexity of the operations in common implementations of the data abstractions and structures.
- Analyze the worst-case time complexity of code that uses data abstractions in the canonical set.
- Compare and contrast time/space characteristics of different implementations of the same data abstraction.
- Select data abstractions, structures, and implementations that a developer would use in solving larger problems and defend the appropriateness of these choices.

Student Learning Goals (to be added to syllabus handed out to students)

By the end of the course, students should be able to:

- Recognize and apply different algorithm design techniques (including divide-and-conquer, decrease-and-conquer, transform-and-conquer, dynamic programming, and greedy approaches).
- Analyze the running times of algorithms.
- Design, implement, and document a medium-sized program that uses algorithms presented in class.

CSS Degree Student Learning Outcomes that this course contributes to (to be added to syllabus handed out to students)

a. an ability to apply knowledge of computing and mathematics appropriate to the discipline;

b. an ability to analyze a problem, identify and define the computing requirements appropriate to its solution;
c. an ability to design, implement and evaluate a computer-based system, process, component, or program to meet desired needs;
f. an ability to communicate effectively with a range of audiences;
i. an ability to use current techniques, skills, and tools necessary for computing practice.

UWT Student Learning Goals that this course contributes to (to be added to syllabus handed out to students)

Inquiry and Critical Thinking
Students will acquire skills and familiarity with modes of inquiry and examination from diverse disciplinary perspectives, enabling them to access, interpret, analyze, quantitatively reason, and synthesize information critically.

Communication/Self-Expression
Students will gain experience with oral, written, symbolic and artistic forms of communication and the ability to communicate with diverse audiences. They will also have the opportunity to increase their understanding of communication through collaboration with others to solve problems or advance knowledge.

Topics covered

1. Fundamentals of algorithm analysis
    a. RAM model
    b. Big oh notation, worst case and best case analysis
    c. proof techniques: proof by contradiction, proof by induction
    d. logarithms, exponents, summations, modular arithmetic
    e. pseudocode
2. Algorithmic design techniques (what problems discussed is at the instructor's discretion; representative algorithms within each category are listed below)
    a. Brute-force, exhaustive search
    b. Divide-and-conquer
        i. mergesort, quicksort, binary search
        ii. large integer multiplication, Strassen's matrix multiplication
    c. Decrease-and-conquer
        i. selection sort
        ii. depth-first search, breadth-first search, topological sort
        iii. interpolation search
    d. Transform-and-conquer
        i. presorting: finding the mode, element uniqueness
        ii. Horner's rule, exponentiation
        iii. string matching algorithms
    e. Dynamic programming
        i. optimal binary search trees
        ii. 0-1 knapsack

f. Greedy algorithms
    i. Prim's algorithm, Kruskal's algorithm
    ii. Dijkstra's algorithm
g. Lower bounds and limits of computation
    i. information-theoretic lower bound for comparison-based sorting
    ii. bucket sort, radix sort
    iii. P, NP, NP-completeness

Alternatively, the material in the course can be organized as a mixture of problem types and algorithmic techniques. For example,

- Searching and sorting algorithms
- Divide-and-conquer technique
- Greedy technique
- Dynamic programming technique
- String matching algorithms
- Numerical algorithms
- Graph algorithms
- Lower bounds, NP-completeness

3. Algorithm analysis
    a. Correctness
    b. Performance
    c. Recurrence relations

Additional Information

This is a fairly standard algorithms course. The two main themes of the course are mentioned in the course's title: the design and analysis of algorithms. By the end of the course, students should not only be able to read algorithms (expressed in pseudocode) and recognize algorithm design techniques, but also be able to design algorithms (expressed in pseudocode) using one or more design techniques. In addition, students should be able to perform appropriate correctness and performance analysis on algorithms. Towards the end of the course, discussions of lower bounds (e.g., the comparison-based lower bound for sorting and breaking the lower bound with an algorithm such as bucket or radix sort) and NP-completeness highlight the model-based nature of algorithmic analysis.

In a typical 10-week term, the first 2 weeks or so would cover the RAM model, big-Oh notation, pseudocode, and a review and strengthening of basic mathematical techniques (e.g., algebra, limits, derivatives, summations, proof by induction). The next 7 weeks or so would cover each design technique (brute-force, divide-and-conquer, decrease-and-conquer, etc.). In any time that remains, NP-completeness and intractable problems can be discussed. Programming assignments (usually two) can be used to reinforce concepts discussed in lecture and homework.