CS5310 – Algorithms – 3 credit hours – 2 hours lecture and 2 hours recitation every week

This course is a continuation of the study of data structures and algorithms, emphasizing methods useful in practice. It provides a theoretical foundation in designing algorithms as well as their efficient implementations. The focus is on the advanced analysis of algorithms and on how the selections of different data structures affect the performance of algorithms. Topics covered include: sorting; search trees, heaps, and hashing; divide-and-conquer; dynamic programming; backtracking; branch-and-bound; amortized analysis; graph algorithms; shortest paths; network flow; computational geometry; number-theoretic algorithms; polynomial and matrix calculations; and parallel computing.

Learning Outcomes

- Reinforce analytic development and problem solving abilities, and develop a foundation in computer science.
- Show progress with regard to understanding the analysis and performance of algorithms (for further use, e.g., in graduate level courses), including knowledge and use of terminology and how the theory connects with real-world applications, possibly in different and new areas.
- Apply the concepts covered in the course to written and practical problems, e.g., by combining problem solving with computer programming and the use of software tools as part of assignments.
- Students who earn a "C" or better in this course should have knowledge of
    - Sequential algorithms pertaining to the greedy, divide-and-conquer, dynamic programming, backtracking and branch-and-bound paradigms;
    - Parallel algorithms pertaining to SIMD, MIMD, shared memory and message passing systems;
    - Introductory databases and data management applications;
    - Analyzing iterative and recursive sequential and parallel algorithms;
    - Efficient data structures such as AVL trees, 2-3 trees, min-max heaps, B-trees.

Topics/modules with Bloom classifications are listed below:

K = Know the terminology

C = Comprehend so as to paraphrase/illustrate

A = Apply it in some way, in homework assignments or projects

N = Elective

Algorithms (CS 5310)

(Review and assignments on) Performance analysis: Iterative/recursive algorithms (A), Asymptotic analysis (A), Recurrence relations (C, A)

Algorithm design paradigms (review and assignments/implementations): Divide-and-conquer (quicksort (A), merge sort (A), K-th order selection (A), . . .), Greedy method (greedy knapsack (A), minimum cost spanning trees (A), shortest paths/Dijkstra's algorithm (A), Huffman trees/codes (C),. . .), Dynamic programming (all- pairs shortest paths/Floyd's algorithm (A), TSP (K), 0/1 knapsack (K)), Backtracking (N -queens (C), . . .), Branch-and-bound (($n^2 - 1$)-puzzle) (C)

Data structures: Embedded in the above topics, plus some more advanced such as AVL-trees (C), 2-3 trees (A), B-trees (A), min-max heaps (C)

Parallel algorithms: SIMD/MIMD (C), Some coverage/examples of shared memory (Pthreads, OpenMP) programs (K), and message passing (MPI) (K)

DB/data management applications.

Algorithms Recitation - Project/Assignment ideas

Implement and compare quicksort vs. mergesort. Consider variations of quicksort.
Compare sorting algorithms - linked list vs. array implementations.
Compare various priority queue implementations - can consider binary trees, Fib heaps etc.
Compare shortest path algorithms - Dijkstra vs modified Dijkstra (using heaps).
Compare min cost spanning tree solutions - Prim's vs. Kruskal's, etc.
Implement B-tree variations - 2-3 search trees - leaves at the same level versus others, B+ trees etc.
Implement dynamic programming - chain matrix multiplication, all pairs shortest paths etc.
K-select implementation - vary group-size and compare performance.
K-select - have multiple arrays and then k-select from combined (arrays could be sorted, unsorted)
Weighted selection.
Puzzle solving - n-Queens, Sudoku, chess etc.
Dictionary (with user-defined entries, multiple dictionaries, shared by multi-users, etc.)
Calendar, contact list, . . .
Document (email) organizer (by subject, keyword, etc.)

CS5541 – Computer Systems – 3 credit hours – 2 hours lecture and 2 hours recitation every week

This course offers an intensive study of computer system design, emphasizing modern operating systems and their impact on application programming. Topics covered include: processes and threads; CPU scheduling; process synchronization; deadlock, memory management; cache, main memory; virtual memory; virtual machine; shared-memory and message-passing based parallelism; clusters; database concepts; security and protection; authentication; and cloud computing.

Learning Outcomes

- Reinforce the essential concepts in computer systems, and develop a systems foundation in computer science.
- Apply the concepts covered in the course to written and practical problems.
- Students who earn a "C" or better in this course should have knowledge of
  - Processes management including processes and threads, CPU scheduling, process synchronization, deadlock;
  - Memory management including swapping, page replacement, segmentation, storage and I/O, file and directory, disk;
  - Memory Hierarchy including cache, main memory, virtual memory, virtual machine;
  - Parallelism including instruction-level parallelism, data-level parallelism, thread-level parallelism, shared-memory and message-passing, multiprocessors and clusters;
  - Database including data organization, indexing, RAID, concurrency control, transaction processing;
  - Security and Protection;
  - Cloud computing.

Topics/modules with Bloom classifications are listed below:

K = Know the terminology

C = Comprehend so as to paraphrase/illustrate

A = Apply it in some way, in homework assignments or projects

N = Elective

Computer Systems (CS 5541)

Process Management:	Processes and Threads (A), CPU scheduling (A), Process synchronization (A), Deadlock(C),
Memory Management:	Swapping (C), Page Replacement(C), Segmentation(C), Storage and I/O (C), File and Directory (A), Disk(C)
Memory Hierarchy:	Cache(C), Main memory(C), Virtual memory (C), Virtual Machine(C),
Parallelism:	Instruction-level (C), data-level (C), thread-level(C),

	Shared-memory and Message-Passing (A), Multiprocessors and Clusters(C)
Database:	Data Organization(C), Indexing, RAID(C), Concurrency Control(C), Transaction Processing (C)
Security:	Security and Authentication (K)
Cloud:	Cloud computing (K)

Systems Recitation - Project/Assignment ideas

- Implement a command line interpreter (that is, shell)
- Develop a real and working web server
- Build a user-level library that implements a good portion of a file system
- Build a version control system

Theory Foundations (CS 5800)

Catalog Description

The course covers the theory of computer science emphasizing automata, grammars and their applications in the specification of languages and computer systems, models of computation, and complexity. Analytic and problem solving abilities will be reinforced, and concepts covered in the course will be applied to real-world problems.

Learning outcomes

A. General learning outcomes:

- Reinforce analytic development and problem solving abilities, and develop a foundation in computer science.
- Show progress with regard to understanding the theory of computation (for use, e.g., in further graduate level courses), including knowledge and use of terminology and how the theory connects with real-world applications, possibly in different and new areas.
- Apply the concepts covered in the course to written and practical problems, e.g., by combining problem solving with computer programming and the use of software tools as part of assignments and a semester project. The latter may be done in teams or individually.

B. Content specific outcomes:

Students who earn a "C" or better will demonstrate

- an introductory to intermediate-level knowledge of definitions, theorems and algorithms for problem solving relating to automata, grammars and formal languages;
- knowledge of the theory of decidability, time & space complexity analysis and complexity classes, and models of computation;
- the ability to synthesize knowledge and use tools of automata, grammars and computational models to solve problems relating to language specification, compiling and machine computation.

Topics/modules with Bloom classifications are listed below:

K = Know the terminology

C = Comprehend so as to paraphrase/illustrate

A = Apply it in some way, in homework assignments or projects

N = Elective

Review of discrete mathematics and proof methods (A), cardinality/countability (C); *Formal languages, the Chomsky hierarchy:* languages, their grammars, and automata (regular (A), context-free (A), context-sensitive (K), recursively enumerable (C)), their grammars, and automata (finite state automata D/NFA (A), push-down automata (A),

linear-bounded automata (K), Turing machines (A)); Stochastic automata (Markov models)(C))
Undecidability: Church-Turing thesis (C); some undecidable problems (C); reductions (K); relation to recursively enumerable languages (C), recursive languages (C), Turing machines (C)
Intractable problems, complexity: Non-deterministic time and space complexity (C), Polynomial time and space, classes P and NP (C); some NP-Complete problems (K)
Applications to compiling: lexical analysis (C), parsing (LL(k), LR(k) (K)), ambiguity (C)
Models of computation: λ-calculus (K) (in relation to functional languages such as Lisp, Scheme, . . .); (partial-) recursive functions (K); Markov algorithms (C) (in relation to logic programming languages such as Prolog); cellular automata (K)

Theory Foundations Recitation - Project/Assignment ideas

Implement a Regular Expression parser: Take as input the regular expression; convert the input regular expression to an NFA; convert the NFA obtained to a DFA and regular grammar. Convert DFA to a minimum state DFA.

Implement a simulator for a vending machine, elevator, traffic light control.

Implement a yacc/bison description of a calculator.

Implement the CYK algorithm to decide membership in the language of a given a context-free grammar.

Example of a parser generator assignment: "Generate and implement a bison (yacc) grammar which incorporates a part of XML, that includes the syntax for the given example. Your grammar should adhere to xml as given at the url http://www.w3schools.com/xml. You can check the syntax of your xml code using the xml validator at http://www.w3schools.com/xml. Generate a lexical analyzer that inputs xml code and recognizes your terminals (tokens). You may use flex (lex). Generate and run your parser. Extend your parser to handle more features of the example."

Implement a Turing Machine simulator for a Turing machine transition table given on input. Apply to various examples. Supply a suitable user interface.

Implement a Markov model.