COMP 2080 – Analysis of Algorithms

Course Description

Calendar entry

Methods of analyzing the time and space requirements of algorithms. Average case and worst-case analysis. Models of computation. Prerequisites: MATH 1240, MATH 1241 or COMP 2130; and one of COMP 2140, or the former COMP 2061. STAT 1000 or STAT 1001 or STAT 1150 is strongly recommended.

General Course Description

This course introduces common algorithmic techniques used in the design of algorithms that solve a variety of problems, as well as techniques for analyzing the efficiency of algorithms in terms of their costs, such as running time and memory usage.

The material covered here provides the foundation upon which Computer Science depends. Regardless of the third- or fourth-year course, the design and analysis of algorithms is a part of everything we do.

Detailed Prerequisites

Before entering this course, a student should be able to:

- Design and implement iterative and recursive algorithms.
- Design and implement standard algorithms that manipulate arrays, linked lists, and binary trees.
- Implement abstract data types such as stacks, queues, dictionaries, and priority queues using data structures such as arrays, linked lists, and/or binary trees.
- Implement common iterative and recursive sorting algorithms.
- Apply common techniques in discrete mathematics, including logical equivalence, logical implication, quantifiers, evaluating finite summations, proofs, mathematical induction, introductory set theory, basic counting of permutations and combinations, introductory graph theory.

Course Goals

By the end of this course students will:

- Express the worst-case cost of iterative algorithms as a function of input size.
- Express the worst-case cost of a recursive algorithm using a recurrence relation.
- Solve certain types of recurrence relations, simplify the solution using asymptotic notation, and prove correctness using a proof by induction.

- Formally show whether a function f is Big Oh, Theta, Omega, little oh, or little omega of a function g.
- Apply divide-and-conquer, greedy, randomized (including Monte Carlo and Las Vegas algorithms), and dynamic programming algorithmic design techniques.
- Analyze the worst-case running time of typical divide-and-conquer, greedy, randomized, and dynamic programming algorithms.
- Prove whether common properties of greedy algorithms and dynamic programming hold.

Learning Outcomes

Algorithm Analysis

Students should be able to:

1. Compare the relative worst-case costs (e.g., worst-case running times) of two algorithms using more than a simple comparison of execution times on sample input.
2. Express the worst-case cost of a simple iterative algorithm as a function of its input size (under simplified assumptions for a unit of computation time, without formally defining a model of computation).
3. Explain the difference between best-case, worst-case, and average-case costs for a deterministic algorithm.
4. Explain the relative asymptotic rates of growth of common functions: constant, logarithmic, linear, quadratic, cubic, exponential, etc.
5. Explain why the largest-order term is responsible for the asymptotic growth, not constants nor lower-order terms.
6. Simplify an expression using asymptotic notation to identify the largest-order term.
7. Formally define Big Oh, Theta, Omega, little oh, and little omega notation.
8. For two polynomial or logarithmic functions f and g, formally show whether function f is Big Oh, Theta, Omega, little oh, or little omega of function g.
9. Appropriately use limits to show asymptotic relationships between two functions.

Recurrence Relations

Students should be able to:

1. Express the worst-case cost of a recursive algorithm using a recurrence relation.
2. Differentiate between the recursive expression for a recurrence relation and its closed-form expression.
3. Solve certain types of recurrence relations using the substitution method.
4. Simplify a recurrence relation's closed-form solution using asymptotic notation.
5. Prove the correctness of a recurrence relation's closed-form expression using a proof by induction.

6. Apply the Master Theorem to solve a recurrence relation.

Divide-and-Conquer Algorithms

Students should be able to:

1. Explain the structure of divide-and-conquer algorithms.
2. Apply divide-and-conquer as a technique in algorithm design.
3. Analyze the worst-case running time of typical divide-and-conquer algorithms using recurrence relations.

Greedy Algorithms

Students should be able to:

1. Explain the structure of greedy algorithms, including greedy choice and growing a solution incrementally.
2. Apply greedy choice as a technique in algorithm design.
3. Analyze the worst-case running time of typical greedy algorithms.
4. Prove whether the two properties of greedy algorithms hold: the optimal substructure property and the greedy-choice property.

Dynamic Programming Algorithms

Students should be able to:

1. Explain the structure of dynamic programming algorithms.
2. Apply dynamic programming as a technique in algorithm design.
3. Analyze the worst-case running time of typical dynamic programming algorithms.
4. Prove whether the two properties of dynamic programming algorithms hold: the optimal substructure property and the overlapping subproblems property.
5. Differentiate between memoization and tabulation.
6. Implement a dynamic programming algorithm that computes both the size of a solution and returns the solution.

Randomized Algorithms

Students should be able to:

1. Explain the structure of randomized algorithms.
2. Explain the differences between Las Vegas and Monte Carlo randomized algorithms.
3. Apply randomization as a technique in algorithm design.
4. Analyze the expected running time of typical randomized algorithms.
5. Differentiate between expected running time and worst-case time.
6. Analyze the probability that a randomized algorithm returns a correct solution.