Data structures and algorithms

Fall 2023

Course Information and Policies

Instructor: Reza Vaziri, Ph.D.

Email: Rvyazdi@gmail.com

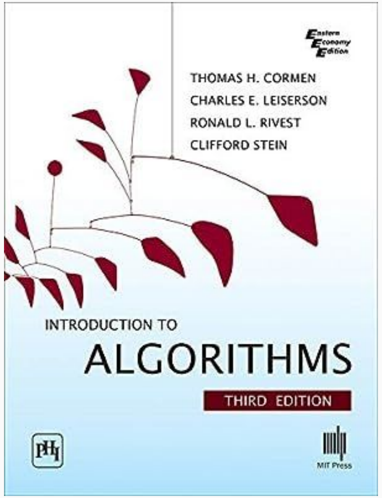
Course website: https://dsa-course.github.io/

Head TA: Arshia Gharooni

Email: arshiyagharoony@gmail.com

Course information and Policies

Textbook: **Introduction to algorithms** - by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein



Prerequisites:

- Advanced programming
- Probability and statistics

Course information and Policies

Grading Policy:

- Homework and assignments - 6
- Midterm - 4
- Final Exam - 10

What is algorithm?

- Definition: An algorithm is a step-by-step set of instructions designed to solve a specific problem or perform a task efficiently that can have input and output.
- Purpose: Algorithms play a crucial role in computer science and programming, enabling the development of software, data analysis, artificial intelligence, and much more.

Algorithm Analysis

- Process of evaluating algorithm efficiency.
- Factors influencing efficiency:
  - Size of input data.
  - Data structure used.
  - Algorithm implementation.
- Measures of efficiency:
  - Time complexity.
  - Space complexity.
- Importance: Choose best algorithm for tasks, optimize performance.

Time complexity - Definition

The time complexity of an algorithm refers to the amount of time it takes to run as a function of the size of the input data.

- Denoted by 'n,' representing the size of the input data.

Time complexity - Big O Notation

- Time complexity is expressed using Big O notation, a mathematical notation that describes the upper bound of an algorithm's running time in terms of 'n.'
- Notation: O(f(n)), where f(n) represents a function describing the upper bound of the algorithm's running time.
- It provides a simplified way to compare algorithms and understand their scalability.

Time complexity - Example

- Let's consider an algorithm with time complexity O(n^2).
- As 'n' increases, the running time grows quadratically, making it less efficient for large datasets.

- Lower time complexity (e.g., O(log n)) indicates better efficiency.
- Higher time complexity (e.g., O(n^2)) may become impractical for large datasets.

Space complexity

- Definition: Space complexity measures the amount of memory an algorithm requires as a function of the input data size (n).
- Notation: Typically expressed using big O notation (O(f(n))), where f(n) is a function representing the upper bound of memory required.

Space complexity - Examples

- O(1) - Constant space regardless of input size.
- O(n) - Linear space growth with input size.
- O(n^2) - Quadratic space growth with input size.

Computational Model

- Abstract representations of computations
- Used for studying algorithms and performance analysis
- Framework for reasoning about computer and algorithm capabilities and limitations

Computational Model - Different Types

Different Types of Computational Models:

1. Turing Machines
2. Register Machines
3. RAM Model
4. Comparison model(we’re going to concentrate on this model)

Algorithm Design Techniques

There are several algorithm design techniques that can be used to create efficient algorithms:

1. Brute-force algorithms
2. Divide-and-conquer algorithms
3. Dynamic programming algorithms

Brute-force algorithms

- Definition: Algorithms that systematically enumerate all possible solutions and select the satisfying one.
- Efficiency: Can be highly inefficient for large input sizes.
- Time Complexity: Often exponential or factorial.
- Caution: Consider alternatives for large input problems.

Divide-and-Conquer Algorithms

Divide-and-conquer algorithms solve problems by:

1. Breaking them down into smaller subproblems
2. Solving subproblems independently
3. Combining subproblem solutions for the original problem

Divide-and-Conquer Algorithms - Pros & Cons

Advantages:

- More efficient than brute-force algorithms
- Recurrence relations express time complexity
- Efficient when subproblems are well-designed

Disadvantages:

- High time complexity if subproblems are poorly designed

Advanced Topics in Algorithm Analysis(Optional)

Randomized Algorithms:

- Use randomization to solve problems efficiently.
- Efficiency measured in expected running time (not worst-case).
- Expected running time expressed using big O notation.

Advanced Topics in Algorithm Analysis(Optional)

Parallel Algorithms:

- Divide computation among multiple processors or cores.
- Efficient for independent subproblems.
- Time complexity expressed using parallel complexity measures.

Advanced Topics in Algorithm Analysis(Optional)

Approximation Algorithms:

- Provide approximate solutions within a guaranteed bound.
- Efficient for NP-hard or NP-complete problems.
- Quality expressed using approximation ratios.