Introduction

CCS 3102 Design and Analysis of Algorithms

What is an Algorithm?

- Algorithm
  - is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output.
  - is thus a sequence of computational steps that transform the input into the output.
  - is a tool for solving a well - specified computational problem.
  - Any special method of solving a certain kind of problem (Webster Dictionary)

Why do we study Algorithms?

- We study algorithms for both practical and theoretical reasons.
- From practical perspective you need to understand a standard set of algorithms and be able to analyse and design new ones.
- Algorithms are recognized as a cornerstone of computer science. Computer programs would not exist without algorithms.
- They help us to develop analytical skills. It is a problem-solving strategy

Characteristics of an algorithm

- Must take an input – zero or more inputs.
- Must give some output (yes/no, value etc.)
- Definiteness – each instruction is clear and unambiguous.
- Finiteness – algorithm terminates after a finite number of steps.
- Effectiveness – every instruction must be basic i.e. simple instruction

Deterministic Algorithms

- A deterministic algorithm
  - Returns the same answer no matter how many times it is called on the same data.
  - Always takes the same steps to complete the task when applied to the same data.

- The most familiar kind of algorithm !

Non-Deterministic Algorithms

- A non-deterministic algorithm
  - Can exhibit different behaviour, for the same input data, on different runs.
- Often used to obtain approximate solutions to given problem instances
  - When it is too costly to find exact solutions using a deterministic algorithm

Data structure

- A **data structure** can be defined as a particular scheme of organizing related data items.

- Linear data structures
- Graphs
- Trees
- Sets and dictionaries

Applications and Data Structures

Sl. No.	Applications	Data Structure
1	Checking of balanced parentheses, Conversion of Decimal number to binary.	Stack
2	Process of paying toll tax, banking transactions, scheduling of jobs in OS.	Queue
3	Simulation of different features of mobile phone.	Linked list
4	Visiting Web pages, Redo/Undo operation in an editor	Doubly linked list
5	Simulation of file system Unix OS, Decision table.	Trees

What is a problem?

- **A problem** is something we want to solve, or a question we want to answer.
- A computational problem is a general question to be answered.
- **General**: has inputs(parameters), whose values are left unspecified.
- To state a problem:
  - Define the inputs and the properties they must satisfy.
  - Define the relation between inputs and outputs, giving the properties a solution must satisfy

Examples of Problems

- **Example 1**: Sorting integers
  - **Input**: A sequence of N numbers $a_1 \ldots a_n$
  - **Output**: the permutation (reordering) of the input sequence such that $a_1 \leq a_2 \leq \ldots \leq a_n$.
- **Example 2**: searching a sequence of integers
  - **Input**: A sequence of integers, s

    Search value x (an integer)
  - **Output**: index I such that s[i] = x, if one exists, otherwise i = -1

Problem instances

- An instance is obtained by taking a problem and defining particular values for the inputs
- Example: sorting integers
  - Input s= {31, 41, 59, 26, 41, 58}
  - Output s' = {26; 31; 41; 41; 58; 59}.
- Such an input sequence is called an instance of the sorting problem.
- An instance of a problem consists of the input needed to compute a solution to the problem.

Types of Problems

- **Sorting** (arranging data in ascending or descending order)
- **Searching** (find an element in a data set)
- **Optimization**: Find the best X satisfying property Y
- **Decision**: Does X satisfy Y?
- **Adaptive**: Maintain property Y over time
- **String processing** (string matching)
- **Graph problems**: Graphs are used in modelling transportation, communication, social and economic networks, project scheduling, and games.

What is meant by Algorithm Design?

- Design: An algorithm design technique (or "strategy" or "paradigm") is a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing.
- Choosing the right data structure.
- Specifying the problem can include flow charts or pseudocode.

Algorithm Design Paradigms?

- What are Algorithm Design Paradigms?
  - Frameworks or strategies for designing algorithms
  - Influence how we approach problem-solving

- Importance:
  - Efficiency
  - Scalability
  - Clarity in code

Algorithm Design Paradigms

- Brute-Force
- Divide-and-Conquer
- Dynamic Programming
- Greedy Algorithms
- Randomization Approach
- Approximation Approach
- Backtracking
- Branch and Bound

Divide and Conquer

- Concept:
  - Break a problem into smaller subproblems
  - Solve each subproblem independently
  - Combine solutions for the final answer

- Examples:
  - Merge Sort
  - Quick Sort
  - Binary Search

Greedy Algorithms

- Concept:
  - Make the locally optimal choice at each stage
  - Aim for a global optimum
- Characteristics:
  - Efficient but not always optimal
- Examples:
  - Activity Selection
  - Huffman Coding
  - Minimum Spanning Tree (Prim’s and Kruskal’s algorithms)

Dynamic Programming

- Concept:
  - Solve problems by breaking them down into simpler subproblems
  - Use previously computed results to avoid redundant calculations
- Key Techniques:
  - Memoization (Storing results to avoid redundant calculations.)
  - Tabulation
- Examples:
  - Fibonacci Sequence
  - Knapsack Problem
  - Shortest Path (Dijkstra's Algorithm)

Randomization Approach

- Concept:
  - Utilize random numbers to influence the algorithm's behaviour.
  - Can lead to simpler and more efficient algorithms.
- Advantages:
  - Can provide good average-case performance
- Examples:
  - QuickSort (randomized version)
  - Randomized Algorithms for Primality Testing
  - Monte Carlo Methods

Approximation Approach

- Concept:
  - Used for NP-hard problems where finding an exact solution is impractical
  - Provides solutions that are close to the optimal solution.
- Characteristics:
  - Often involves heuristics or specific algorithms designed to yield a good enough solution.
- Examples:
  - Traveling Salesman Problem (TSP)
  - Vertex Cover Problem
  - Knapsack Problem

Applications and Algorithmic Techniques

Sl. No.	Applications	Algorithmic Technique
1	Sorting of files using merge sort in an OS	Divide and Conquer
2	Traversal of graph used to represent computer network topology and route scenario in transport system.	Decrease and Conquer
3	Finding minimum spanning tree and shortest path in computer network	Greedy Method
4	Horspool's algorithm used for string matching, calculation of permutations and combinations.	Dynamic Programming
5	Computer Games like Chess.	Branch and Bound, Backtracking

What is meant by Algorithm Analysis?

- Algorithm analysis is an important part of computational complexity theory, which provides theoretical estimation for the required resources of an algorithm to solve a specific computational problem.

- Analysis of algorithms is the determination of the amount of **time** and **space** resources required to execute it.

What do we analyze about them?

- Correctness
  - Does the input/output relation match algorithm requirement?
- Amount of work done (complexity)
  - Basic operations to do task
- Amount of space used:
  - Memory used
- Simplicity, clarity
  - Verification and implementation.
- Optimality
  - Is it impossible to do better?

Why Analysis of Algorithms is important?

- To predict the behaviour of an algorithm without implementing it on a specific computer.
- It is much more convenient to have simple measures for the efficiency of an algorithm than to implement the algorithm and test the efficiency every time a certain parameter in the underlying computer system changes.
- It is impossible to predict the exact behaviour of an algorithm. There are too many influencing factors.
- More importantly, by analysing different algorithms, we can compare them to determine the best one for our purpose.

Desired properties of algorithms

- **Correctness**: Whether the algorithm computes the correct solution for **all** instances

- **Efficiency** :Resources needed by the algorithm
  - Time efficiency: Number of steps
  - Space efficiency: Amount of memory used

Correctness

- An algorithm is correct with respect to a problem if
  - For every instance of the input (i.e. the specified properties of the inputs are satisfied)
    - The algorithm halts, and
    - The outputs produced satisfy the specified input/output relation
- Correctness must be proven.
- One counter-example is enough to disprove correctness, but, no amount of testing can prove correctness.

Correctness Analysis

- If two algorithms give correct solutions to some computational problem, which one do we use?
  - Available in library
  - Easy to code (or prove correct)
  - Resources needed for typical problem
  - Resources needed for worst-case problem
- Resources:
  - Computation time
  - Maximum memory use
  - Memory turnover
- Mostly we consider **time**.

Algorithm Efficiency

- Analyze algorithm efficiency
  - Running time ?
  - Memory space ?
- Time
  - How fast does an algorithm run?
- Space
  - Does an algorithm require additional memory?

Efficiency Analysis

- How fast does an algorithm run ?
  - Most algorithms run longer on larger inputs !

- How to relate running time to input size ?

- How to rank / compare algorithms ?
  - If there is more than one available…
- How to estimate running time for larger problem instances ?

Formal Analysis

- Understand algorithm behaviour
  - Count arithmetic operations / comparisons
  - Identify best, worst and average case situations.
- Iterative algorithms
  - Loops : how many iterations ?
  - Set a sum for the basic operation counts
- Recursive algorithms
  - How many recursive calls ?
  - Establish and solve appropriate recurrences

Analysing computation time

- One way to analyse computation time is to write a program, execute it and measure how long it takes.
- The answer will depend on
  - The algorithm used
  - The size of the input list
  - The elements in the input list
  - The programming language chosen
  - How the algorithm is implemented
  - The CPU clock speed.
  - The operating system
- Only the first three relate to the algorithm.

Applications of DAA

- **Software Development**: DAA is used to develop efficient algorithms for tasks like sorting, searching, and data manipulation.
- **Data Science and Machine Learning**: development of algorithms for data analysis, pattern recognition, and optimization problems in fields such as data science and machine learning.
- **Bioinformatics:** DAA is applied to tasks such as sequence alignment, gene sequencing, and protein structure prediction.
- **Finance**: DAA is applied in tasks like portfolio optimization, risk analysis, and algorithmic trading.

Applications of DAA

- **Telecommunications**: DAA is used in telecommunications for routing optimization, network design, and resource allocation.
- **Logistics and Supply Chain Management**: DAA contributes to solving logistical challenges in supply chain management, transportation optimization, and scheduling.
- **Gaming**: DAA is used in gaming for tasks like pathfinding, AI opponent behavior, and procedural content generation.
- **Cybersecurity**: DAA algorithms are applied in cybersecurity for tasks like encryption, decryption, and intrusion detection.