



Design and Analysis of Algorithms

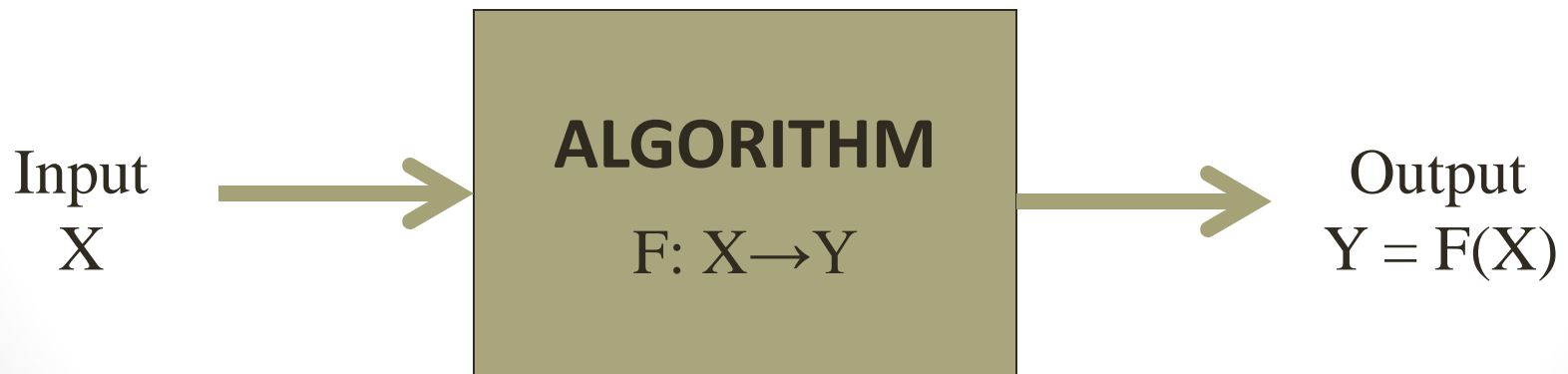
Chapter 1 - Introduction

Outline:

- Algorithm Definition
- Properties of Algorithms
- Brief History about Algorithm
- Why Algorithms are useful?
- Designing of Algorithm
- Techniques of Algorithm
- Analysis of Algorithm
- Complexity of an Algorithm

Algorithm Definition

- **An algorithm is a step-by-step procedure for solving a particular problem in a finite amount of time.**
- More generally, an **algorithm** is any well defined computational procedure that takes collection of elements as **input** and produces a collection of elements as **output**.

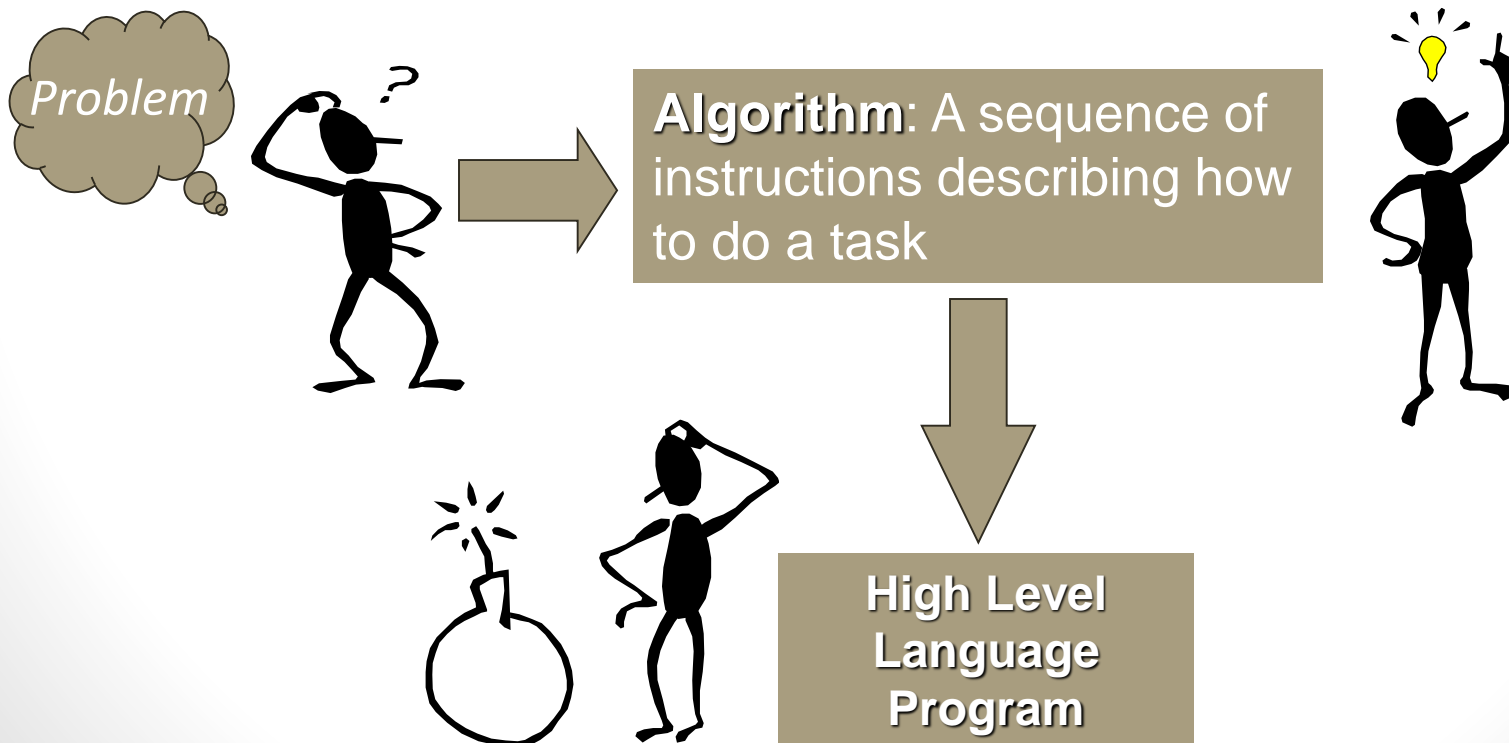


Algorithm - Examples

- Repairing a lamp
- A cooking recipe
- Calling a friend on the phone
- The rules of how to play a game
- Directions for driving from A to B
- A car repair manual
- Matrix Multiplication

Algorithm vs. Program

- **A computer program is an instance, or concrete representation, for an algorithm in some programming language**
- Set of instructions which the computer follows to solve a problem



Solving Problems (1)

When faced with a problem:

1. First clearly **define** the problem
2. Think of **possible solutions**
3. **Select** the one that seems **the best** under the **prevailing** circumstances
4. And then apply that solution
5. If the solution works as desired, fine; else **go back to step 2**

Solving Problems (2)

- It is **quite common** to first solve a problem for a **particular case**
- Then for another
- And, possibly another
- And watch for **patterns and trends that emerge**
- And to use the knowledge from these patterns and trends in coming up with a general solution
- **And this general solution is called “Algorithm”**

One Problem, Many Algorithms

Problem

- The statement of the problem specifies, in general terms, the desired **input/output relationship**.

Algorithm

- The algorithm describes a specific computational procedure for achieving **input/output relationship**.

Example

- Sorting a sequence of numbers into decreasing order.

Algorithms

- Various algorithms e.g. **merge sort, quick sort, heap sorts etc.**

Problem Instances

- An **input sequence** is called an **instance of a Problem**
- A problem has many particular *instances*
- An algorithm must **work correctly on all instances** of the problem it claims to solve
- Many interesting problems have infinitely many instances
 - Since computers are finite, we usually need to limit the number and/or size of possible instances in this case
 - This restriction doesn't prevent us from doing analysis in the abstract

Problem Solving Technique

The process of problem-solving using algorithms consists of the following steps.

- **Define the problem**
- **Analyze the problem**
- **Identification of possible solutions**
- **Choosing the optimal solution**
- **Implement**

Let's discuss some of the essential properties of Algorithms.

Properties of Algorithms

✓ **Completeness**

A search algorithm is said to be complete when it gives a solution or returns any solution for a given random input.

✓ **Optimality**

If a solution found is best (lowest path cost) among all the solutions identified, then that solution is said to be an optimal one.

Cont..

✓ Time Complexity

The time taken by an algorithm to complete its task is called time complexity. If the algorithm completes a task in a lesser amount of time, then it is an efficient one.

✓ Space Complexity

It is the maximum storage or memory taken by the algorithm at any time while searching.

These properties are also used to compare the efficiency of the different types of algorithms.

Characteristics of Algorithms

- It must be composed of an ordered sequence of precise steps.
- It must have finite number of well-defined instructions /steps.
- The execution sequence of instructions should not be ambiguous.
- It must be correct.
- It must terminate.

Syntax & Semantics

An algorithm is “correct” if its:

- Semantics are correct
- Syntax is correct

Semantics:

- The **concept** embedded in an algorithm (the **soul**!)

Syntax:

- The actual **representation** of an algorithm (the **body**!)

WARNINGS:

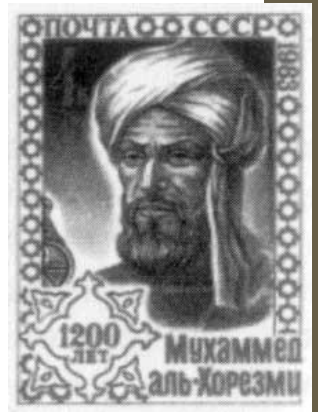
1. An algorithm can be syntactically correct, yet semantically incorrect – dangerous situation!
2. Syntactic correctness is easier to check as compared to semantic correctness

Algorithm Summary

- **Problem Statement**
 - Relationship b/w input and output
- **Algorithm**
 - Procedure to achieve the relationship
- **Definition**
 - A sequence of steps that transform the input to output
- **Instance**
 - The input needed to compute solution
- **Correct Algorithm**
 - for every input it halts with correct output

Brief History

- The study of algorithms **began with mathematicians** and was a significant area of work in the early years. The **goal of those early studies** was to find a single, general algorithm that could solve all problems of a **single type**.
- Named after 9th century Persian Muslim mathematician **Abu Abdullah Muhammad ibn Musa al-Khwarizmi** who lived in Baghdad and worked at the **Dar al-Hikma**
- Dar al-Hikma acquired & translated books on science & philosophy, particularly those in Greek, as well as publishing original research.
- The word *algorism* originally referred only to the **rules of performing arithmetic** using Hindu-Arabic numerals, but later evolved to include all definite procedures for solving problems.



Al-Khwarizmi's Golden Principle

All complex problems can be and must be solved using the following simple steps:

1. Break down the problem into small, simple **sub-problems**
2. Arrange the sub-problems in such an order that each of them can be solved without effecting any other
3. Solve them separately, in the correct order
4. **Combine** the solutions of the sub-problems to form the solution of the original problem

Why Algorithms are Useful?

- Once we find an algorithm for solving a problem, we do not need to re-discover it the next time we are faced with that problem
- Once an algorithm is known, the task of solving the problem reduces to following (almost blindly and without thinking) the instructions precisely
- All the knowledge required for solving the problem is present in the algorithm

Why Write an Algorithm Down?

- For your **own use in the future**, so that you don't have spend the time for **rethinking** it
- Written form is easier to modify and improve
- Makes it easy when **explaining** the process to others

Designing of Algorithms

Major Factors in Algorithms Design

- **Correctness:** An algorithm is said to be correct if for every input, it halts with correct output. An incorrect algorithm might not halt at all OR it might halt with an answer other than desired one.
Correct algorithm solves a computational problem
- **Algorithm Efficiency:** Measuring efficiency of an algorithm
 - do its analysis i.e. growth rate.
 - Compare efficiencies of different algorithms for the same problem.

Cont..

- Most basic and popular algorithms are searching and sorting algorithms
- Which algorithm is the best?
- Depends upon various factors, for example in case of **sorting**
 - The number of items to be sorted
 - The extent to which the items are already sorted
 - Possible restrictions on the item values
 - The kind of storage device to be used etc.

Algorithm Strategies and Techniques

- **Brute Force** – it goes through all possible choices until a solution is found.
- **Divide-and-Conquer** – Divide the main problem into smaller sub-problems.
 - e.g. merge sort
- **Greedy** – At each step of the journey, visit the nearest unvisited city.
 - e.g. travelling salesman problem
- **Dynamic programming** – Problem is first broken down into sub-problems, the results are saved, and then the sub-problems are optimized to find the overall solution, *reuse results*.

Algorithm Efficiency

- Several possible algorithms exist that can solve a particular problem
 - each algorithm has a given *efficiency*
 - compare efficiencies of different algorithms for the same problem
- The efficiency of an algorithm is a measure of the **amount of resources consumed** in solving a problem of size n
 - *Running time* (number of primitive steps that are executed)
 - *Memory/Space*
- Analysis in the **context of algorithms** is concerned with **predicting the required resources**
- Time is the resource of **most interest**
- By analyzing several candidate algorithms, the most efficient one(s) can be **identified**

Algorithm Efficiency

Two goals for best design practices:

1. To design an algorithm that is easy to understand, code, debug.
2. To design an algorithm that makes efficient use of the computer's resources.

How do we improve the time efficiency of a program?

The 90/10 Rule

- 90% of the execution time of a program is spent in executing 10% of the code. So, how do we locate the critical 10%?
 - software metrics tools (*SourceMonitor* or *VSCode Metrics*)
 - global counters to locate bottlenecks (loop executions, function calls)

Algorithm Efficiency

Time Efficiency improvement

- Good programming: Move code **out of loops** that does not belong there
- Remove any **unnecessary I/O** operations
- Replace an inefficient algorithm (best solution)

Moral - Choose the most appropriate algorithm(s)
BEFORE program implementation

Analysis of Algorithms

- Two essential approaches to measuring algorithm efficiency:
- **Empirical analysis:**
 - Program the algorithm and measure its running time on example instances
- **Theoretical analysis:**
 - Employ mathematical techniques to derive a *function* which relates the running time to the size of instance
- In this course our focus will be on Theoretical Analysis.

Analysis of Algorithms

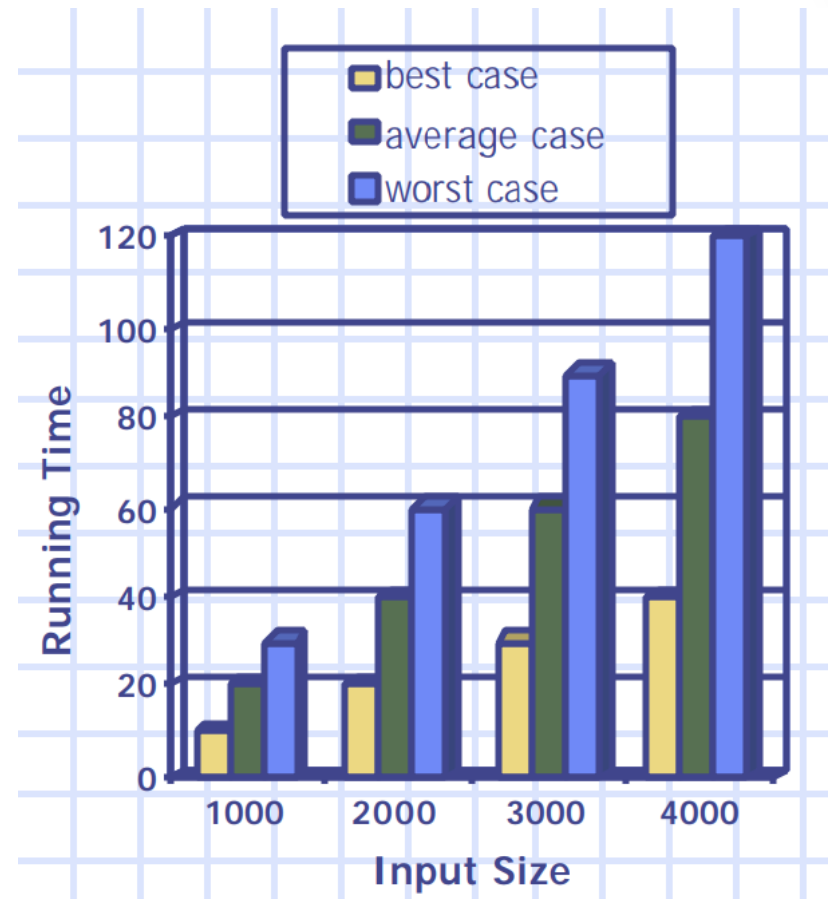
- Many criteria affect the running time of an algorithm, including
 - speed of CPU, RAM and etc
 - design time, programming time and debugging time
 - language used and coding efficiency of the programmer
 - quality of input (good, bad or average)
 - But
- Programs derived from two algorithms for solving the same problem should both be
 - Machine independent
 - Language independent
 - Realistic

Analysis of Algorithms

- The following three cases are investigated in algorithm analysis:
- **A) Worst case:** The worst outcome for any possible input
 - We often concentrate on this for our analysis as it provides a clear **upper bound** of resources
 - an absolute guarantee
- **B) Average case:** Measures performance over the entire set of possible instances
 - Very useful, but treat with care: what is “average”?
 - Random (equally likely) inputs vs. real-life inputs
- **C) Best Case:** The best outcome for any possible input
 - provides **lower bound** of resources

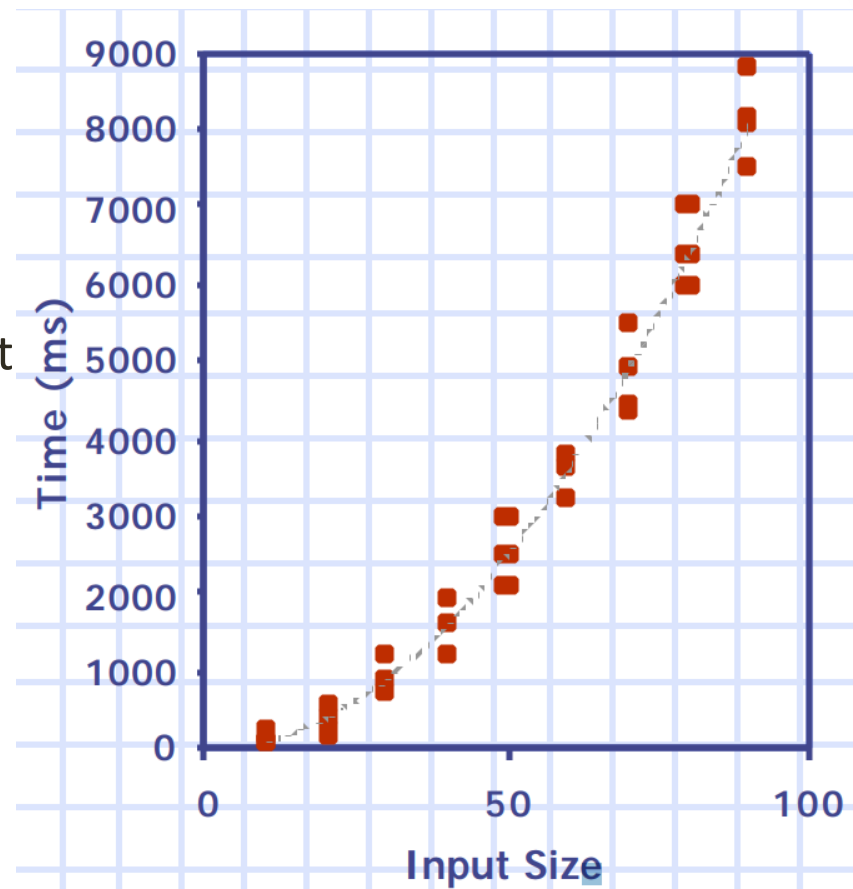
Empirical Analysis

- Most algorithms transform input objects into output objects
- The running time of an algorithm typically grows with the input size
- Average case time is often difficult to determine
- We focus on the worst case running time
 - Easier to analyze
 - Crucial to applications such as games, finance and robotics



Limitations of Empirical Analysis

- Implementation dependent
 - Execution time differ for different implementations of same program
- Platform dependent
 - Execution time differ on different architectures
- Data dependent
 - Execution time is sensitive to amount and type of data manipulated.
- Language dependent
 - Execution time differ for same code, coded in different languages



∴ absolute measure for an algorithm is not appropriate

Theoretical Analysis

- Data independent
 - Takes into account all possible inputs
- Platform independent
- Language independent
- Implementation independent
 - not dependent on skill of programmer
 - can save time of programming an inefficient solution
- Characterizes running time as a function of input size, n .
Easy to extrapolate without risk

Why Analysis of Algorithms?

- For real-time problems, we would like to prove that an algorithm terminates in a given time.
- Algorithmics may indicate which is the best and fastest solution to a problem without having to code up and test different solutions
- Many problems are in a complexity class for which no practical algorithms are known
 - better to know this before wasting a lot of time trying to develop a "perfect" solution: **verification**

Importance of Analyzing Algorithms

- Need to recognize limitations of various algorithms for solving a problem
- Need to understand relationship between problem size and running time
 - When is a running program not good enough?
- Need to learn how to analyze an algorithm's running time without coding it
- Need to learn techniques for writing more efficient code
- Need to recognize bottlenecks in code as well as which parts of code are easiest to optimize

What do we analyze about Algorithms?

- Algorithms are analyzed to understand their behavior and to improve them if possible
- **Correctness**
 - Does the input/output relation match algorithm requirement?
- **Amount of work done**
 - Basic operations to do task
- **Amount of space used**
 - Memory used
- **Simplicity, clarity**
 - Verification and implementation.
- **Optimality**
 - Is it impossible to do better?

Problem Solving Process

- Problem
- Strategy
- Algorithm
 - Input
 - Output
 - Steps
- Analysis
 - Correctness
 - Time & Space Complexity
- Implementation
- Verification

Computation Model for Analysis

- To analyze an algorithm is **to determine the amount of resources** necessary to execute it.
- These resources include **computational time**, **memory** and **communication bandwidth**.
- Analysis of the algorithm is performed with respect to a computational model called RAM (Random Access Machine).

Complexity of an Algorithm

- The complexity of an algorithm is the amount of work the algorithm performs to complete its task. It is the **level of difficulty** in solving mathematically posed problems as measured by:
 - Time (time complexity)
 - No. of steps or arithmetic operations (computational complexity)
 - Memory space required (space complexity)
- Complexity is a function $T(n)$ which yields the time (or space) required to execute the algorithm of a problem of size 'n'.

Conclusion

- At last it may said that, the so called Algorithm Design is one of the most enforceable solution program that can be used to solve any complicated, complex and hard programs.
- Under considering the above paragraph, the concerned authority included you (Students) should prefer this Course with great care.



Reading Assignment I

- **Link:** -

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6107538/pdf/rsta20170364.pdf>


The growing ubiquity of algorithms in society: implications, impacts and innovations

S. C. Olhede^{1,2,3} and P. J. Wolfe^{4,5,6}

¹Department of Statistical Science, ²Department of Computer Science, and ³Department of Mathematics, University College London, London, UK

⁴Department of Statistics, ⁵Department of Computer Science, and

⁶School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA

 SCO, 0000-0003-0061-227X

The growing ubiquity of algorithms in society raises a number of fundamental questions concerning governance of data, transparency of algorithms, legal and ethical frameworks for automated algorithmic decision-making and the societal impacts of algorithmic automation itself. This article, an introduction to the discussion meeting issue of the same title, gives an overview of current challenges and opportunities in these areas, through which accelerated technological progress leads to rapid and often unforeseen practical consequences. These consequences—ranging from the potential benefits to human health to unexpected impacts on civil society—are summarized here, and discussed in depth by other contributors to the discussion meeting issue.

This article is part of a discussion meeting issue ‘The growing ubiquity of algorithms in society: implications, impacts and innovations’.

THE END