# Chapter 1: Introduction

algorithmics - the study of algorithms
- Computer progams would not exist w/o algorithms
- studying algorithms improves analytical skills

## 1.1 What is an algorithm?

- There are two ways to find the greatest common factor of two nonnegative, nonzero integers: the Eucledian algorithm and the consecutive integer algorithm
  ↳ the middle schod procedure is not an algorithm due to ambigvous instructions.
  ↳ We can create a list of primes for a given integer greater than 1 using Sieve of Eratosthene's algorithm

## 1.2 Fundamentals of Algorithmic Problem Solving

1. Understand the problem
   - take full stock of what is being asked
   - instance: an input to an algorithm
2. Ascertaining the capabilities of the Computational Device
   - determine the capabilities of the computational device
   ↳ We'll usually be writing algorithms for the Neuman machine (based on Ram) which means we'll be creating sequential algorithms.
   ↳ computers that can execute operation concurrently can host parallel algorithms.

3. Choosing between approximate/exact problem solving
- approximate algorithms are typically used to solve problems wherein no exact answer can be computed or existing exact algorithms are too slow to solve a highly complex problem
- exact algorithms produce, well, exact results to a problem. Not always ideal but definitely important if you need the exact answer

4. Algorithm Design Techniques
Algorithm Design Technique: a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing

5. Designing an algorithm and data structure
- the type of data structure you choose can impact the speed of your algorithm

6. Methods of specifying an algorithm
- using natural language is appealing but natural language is susceptible to ambiguity, so you have to contend with that obstacle.
- pseudocode is a mix of technical programming language and natural language that aims to eliminate ambiguities.

## 6. Proving an algorithm's correctness

- correctness: the ability to prove that an algorithm outputs the required result for every legitimate input in a finite amount of time
  - ↳ algorithms lead naturally to mathematical induction to prove their correctness
  - ↳ you only need one incorrect instance to prove an algorithm's incorrectness

## 7. Analyzing Algorithms

- time efficiency: indicates how fast the algorithm runs
- space efficiency: indicates how much extra memory it uses
- simplicity: (in the eye of the beholder) makes the algorithm maintainable because most programmers should be able understand the src code.
- generality: ~~something~~ an algorithm that resolves the problem that can be moved around because of its lack of specifics
  - ↳ Two issues arise
    1. sometimes it's easier to solve a problem in general terms
    2. sometimes it is impractical/impossible to solve a problem in general terms

## 8. Coding an algorithm

- implementing an algorithm is only half the battle, there are ~~always~~ usually ways to improve its efficiency, ~~space~~ simplicity, and generality
  - ↳ sometimes it might even be better to try invisioning a new algorithm

- optimality: the minimum amount of effort any algorithm will need to exert to solve a problem

## 1.3. Important Problem Types

- the most important problem types:
  - ↳ sorting searching string processing
    graph problems combinatorial problems
    geometric problems numerical problems

### A. Sorting

- sorting problem: given a list we want to sort it in some manner (e.g for integers maybe in ascending/descending order, for strings maybe in alphabetical order, etc.)
  - ↳ key: a specific attribute within a record from which to base all sorting (e.g sorting students by name).
  - ↳ a sorting algorithm is stable <u>if</u> it preserves the relative order of any two equal elements in its input.
  - ↳ an sorting algorithm is in-place <u>if</u> it does not require extra memory

### B. Searching

- searching problem: locate a given value, called the search key, in a given set.
  - ↳ be way of additions/deletions from the data set of an item

C. String Processing

- string: a sequence of characters from an alphabet
  - ↳ text strings: consist of letters, numbers, and special characters
  - ↳ bit strings: consist of zeros and ones
  - ↳ gene sequences: modeled by strings of characters from the four character alphabet [A, C, G, T]
  - ↳ one particular problem of string processing is searching a word in a given text. String matching attempts to solve this problem

D. Graph Problems

- informally a graph is a collection of points (vertices) where some are connected together by line segements (edges).
  - ↳ famous graphing problems: the traveling salesmen problem (TSP) and the graph-coloring problem

E. Combinatorial Problem

- combinatorial problems: problems that ask explicitly/implicitly to find a combinatorial object, such as a permutation, a combination, or a subset that satisfies certain constraints.
  - ↳ there are no such algorithms more solving most combinatorial problems

## F. Geometric Problems

- geometric algorithms: algorithms that deal w/ geometric objects such as points, lines, and polygons
  - ↳ closest-pair problem: given n points in the plane, find the closest pair among them
  - ↳ convex hull problem: find the smallest convex polygon that would include all points of a given set

## G. Numerical Problems

- numerical problems: problems that involve mathematical objects of continuous nature: solving equations and systems of equations, computing definite integrals, evaluating functions, etc.
  - ↳ a majority of such problems can only be solved approximately