Introduction

What is an Algorithm?

An **algorithm** is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.

The Greatest Common Divisor (GCD) of two non-negative, not-both-zero integers m and n denoted $gcd(m, n)$, is defined as the largest integer that divides both m and n evenly, i.e., with a remainder of zero.

Euclid's algorithm is based on applying repeatedly the equality

$$\texttt{gcd}(m,\ n) = \texttt{gcd}(n,\ m \texttt{ mod } n)$$

where m `mod` n is the remainder of the division of m by n, until m `mod` n is equal to 0.

Euclid's algorithm for computing `gcd`(m, n)

Step 1. If $n = 0$, return the value of m as the answer and stop; otherwise proceed to Step 2.

Step 2. Divide m by n and assign the value of the remainder to r.

Step 3. Assign the value of n to m and the value of r to n. Go to Step 1

```
Euclid(m, n)
// Computes gcd(m, n) by Euclid's algorithm
// Input: Two non-negative, not-both-zero integers m and n
// Output: Greatest Common Divisor of m and n
while n ≠ 0 do
    r ← m mod n
    m ← n
    n ← r
return m
```

Here is an implementation in C/C++

```
int Euclid(int m, int n) {

// Computes gcd(m, n) by Euclid's algorithm
// Input: Two non-negative, not-both-zero integers m and n
// Output: Greatest Common Divisor of m and n

    while (n ≠ 0) {
        int r = m % n;
        m = n;
        n = r;
    }
    return m;

}
```

Here is an implementation in Java

```
static int Euclid(int m, int n) {

// Computes gcd(m, n) by Euclid's algorithm
// Input: Two non-negative, not-both-zero integers m and n
// Output: Greatest Common Divisor of m and n

    while (n ≠ 0) {
        int r = m % n;
        m = n;
        n = r;
    }
    return m;

}
```

Here is an implementation in Javascript

```
function Euclid(int m, int n) {

// Computes gcd(m, n) by Euclid's algorithm
// Input: Two non-negative, not-both-zero integers m and n
// Output: Greatest Common Divisor of m and n

    while (n ≠ 0) {
        int r = m % n;
        m = n;
        n = r;
    }
    return m;

}
```

Here is an implementation in Python3

```
def Euclid(m, n):

// Computes gcd(m, n) by Euclid's algorithm
// Input: Two non-negative, not-both-zero integers m and n
// Output: Greatest Common Divisor of m and n

    while (n ≠ 0):
        r = m % n
        m = n
        n = r
    return m;
```

```
                    Sieve($n$)
// Implements the sieve of Eratosthenes
// Input: A positive integer $n > 1$
// Output: Array $L$ of all prime numbers less than or equal to $n$

    for $p \leftarrow 2$ to $n$ do $A[p] \leftarrow p$
    for $p \leftarrow 2$ to $\lfloor\sqrt{n}\rfloor$ do
        // $p$ hasn't been eliminated on previous passes
        if $A[p] \neq 0$
            $j \leftarrow p * p$
            while $j \leq n$ do
                // mark element as eliminated
                $A[j] \leftarrow 0$
                $j \leftarrow j + p$
    // copy the remaining elements of $A$ to $L$ of the primes
    $i \leftarrow 0$
    for $p \leftarrow 2$ to $n$ do
        if $A[p] \neq 0$
            $L[i] \leftarrow A[p]$
            $i \leftarrow i + 1$
    return $L$
```

Important Problem Types

- Sorting
- Searching
- String Processing
- Combinatorial Problems
- Geometric Problems
- Numerical Problems

The **Sorting Problem** is to rearrange the items of a given list in non-decreasing order. A sorting algorithm is called **stable** if it preserves the relative order of any two equal elements in its input. If an input list contains two equal elements in positions i and j where $i < j$, then in the sorted list they have to be in the position i' and j', respectively, such that $i' < j'$.

The second notable feature of a sorting algorithm is said to be **in-place** if it does not require extra memory.

The **Searching Problem** deals with finding a given value, called a **search key**, in a given set (or a multiset, which allows several elements to have the same value).

Searches have to be considered in conjunction with two other operations: an addition to and deletion from the data set of an item.

String Matching Problem—Searching for a given word in a text.

Graph Problems—Using the properties of graphs to solve problems such as the Shortest-Path Problem, or finding a minimum spanning tree.

Combinatorial Problems—Problems that ask, explicitly or implicitly, to find a combinatorial object.

Geometric Problems: Geometric algorithms deal with geometric objects such as points, lines, and polygons. Two famous problems in this group are the **closest-pair problem** and **convex-hull problem**.

Numerical Problems: Numerical problems are problems that involve mathematical objects of continuous nature: solving equations and systems of equations, computing definite integrals, evaluating functions, and so on.

Fundamental Data Structures

Linear Data Structures

The two most important elementary data structures are:

- Array
- Linked List

An **array** is a sequence of n items of the same data type that are stored contiguously in computer memory and made accessible by specifying a value of the array's **index**. Each and every element of a array can be accessed in the same constant amount of time regardless of where in the array the element in question is located.

A **linked list** is a sequence of zero or more elements called **nodes**, each containing two kinds of information: some data and ore or more linkes called **pointers** to other nodes of the linked list.

Graphs

Graphs for computer algorithms are usually represented in one of two ways

- adjacency matrix
- adjacency lists

The **adjacency matrix** of a graph with n vertices is an $n \times n$ boolean matrix with one row and one column for each of the graph's vertices, in which the element in the ith row and jth column is equal to 1 if there is an edge from the ith vertex to the jth vertex, and equal to 0 if there is no edge from the ith vertex to the jth vertex.

The **adjacency lists** of a graph or a digraph is a collection of linked lists, one for each vertex, that contain all the vertices adjacent to the list's vertex.

Trees

A tree is a connected acyclic graph.

Rooted Trees A very important property of trees is the fact that for every two vertices in a tree, there always exists one simple path from one of these vertices to the other. This property makes it possible to select an arbitrary vertex in a tree and consider it as the **root** of the so-called **rooted tree**.

Ordered Trees—An ordered tree is a rooted tree in which all the children of each vertex are ordered. A **Binary Tree** can be defined as an ordered tree in which every vertex has no more than two children and each child is designated as either **left child** or a **right child** of its parent; a binary tree may also be empty. Note that a number assigned to each parental vertex is larger than all the numbers in its left subtree and smaller than all the number in its right subtree. Such trees are called **binary search trees**

Sets and Dictionaries

A **set** can be described as an unordered collection (possibly empty) of distinct items called **elements** of the set.

A **dictionary** allows the following operations on sets or multisets:

- Searching for a given item in the collection
- Adding a new item to the collection
- Deleting an item from the collection

Summary

- An algorithm is a sequence of non-ambiguous instructions for solving a problem in a finite amount of time. An input to an algorithm specifies an instance of the problem the algorithm solves.

- Algorithms can be specified in a natural language or pseudocode; they can also be implemented as computer programs.

- Among several ways to classify algorithms, the two principal alternatives are:
    - to group algorithms to types of problems they solve
    - to group algorithms according to underlying design techniques they are based on

- The important problem types are sorting, searching, string processing, graph problems, combinatorial problems, geometric problems, and numerical problems.

- Algorithm design techniques are general approaches to solving problems algorithmically, applicable to a variety of problems from different areas of computing.

- Although designing an algorithm is undoubtedly a creative activity, one can identify a sequence of interrelated actions involved in such a process.

- A good algorithm is usually the result of repeated efforts and rework.

- The same problem can often be solved by several algorithms.

- Algorithms operate on data. This makes the issue of data structing critical for efficient algorithmic problem solving. The most important elementary data structures are the array and the linked list. They are used for representing more abstract data structures such as the **list**, the **stack**, the **queue**, the **graph**, the **binary tree**, and the **set**.

- An abstract collection of objects with several operations that can be performed on them is called an **Abstract Data Type (ADT)**. The **list**, the **stack**, the **queue**, the **priority queue**, and the **dictionary** are important examples of abstract data types. Modern object-oriented languages support implementation of ADT's by means of classes.