INTRODUCTION

TO

COMPUTER SCIENCE

Programming is not Coding

How to build safe and secure software

Stefan Parvu

November 27, 2024

Table of Contents

Lesson 1

Computation. Algorithms. Computer Programs

In this lesson we will talk about the following concepts: computation, algorithms and programs.

Computer Science (CS)

But what's the difference between IT and CS?

Computer Science

Computer Science is the study of the principles of computing and how computer systems solve problems

Computer scientists design and build tools and software applications

Programming computers using mathematical algorithms, abstract concepts and models like, example the computational complexity theory

Information Technology

IT is the study of current tools and computing techniques that can be used for technological needs of a particular organization

IT professionals apply and use these tools or software applications

IT involves more practical aspects how to use and maintain software applications, to develop, maintain and improve business processes

Computation

What is a Computation?

A **computation** is any type of arithmetic or non-arithmetic calculation that is well-defined.[1][2] Common examples of computations are mathematical equations and computer algorithms.

Mechanical or electronic devices (or, historically, people) that perform computations are known as *computers*. The study of computation is the field of computability, itself a sub-field of computer science.

A **computation** is what a computing device does

We sometimes call the computation, a behaviour

Computing device

For example, a computer system, a tablet, a mobile phone or a basic calculator. But it can be as well, a non physical device, something more abstract, like an algorithm.

We describe a computing device by describing all its possible computations or its behaviours.

But what is an algorithm?

Sequence of computational steps that transform the input into the output

Example of computations

▶ well-defined mathematical statements
▶ solvable statements
▶ simple instructions

Note: There are however other problems difficult to solve or sometimes impossible to solve because there is no algorithm for them. Like the halting problem.

The halting problem

"The halting problem is the problem of determining, from a description of an arbitrary computer program and an input, whether the program will finish running, or continue to run forever. The halting problem is undecidable, meaning that no general algorithm exists that solves the halting problem for all possible program input pairs."

Computation as a Sequence of States

Computation as a sequence of states

INSTRUCTION 1	$x = 12$ $\quad$ State 1 $\quad$ $y = 18$	*Initial State*
INSTRUCTION 2		*State* 1
INSTRUCTION 3	$y - x$ $\quad$ State 2 $x = 12$ $y = 6$	*State* 2
.		.
.		.
INSTRUCTION N	$x - y$ $\quad$ State 3 $x = 6$ $y = 6$	*State N*

What is a computing state?

A state is an assignment of values to variables.

$$x = 12$$
$$y = 18$$

State 1

$$y - x$$
$$x = 12$$
$$y = 6$$

State 2

$$x - y$$
$$x = 6$$
$$y = 6$$

State 3

Standard Computational Model

▶ A program execution is represented by a computation or a behaviour

▶ A computation is a sequence of states

▶ A state is an assignment of values to variables

▶ A program is modeled as a set of computations

Computing devices

Computing devices are supposed to compute something. Like calculate and predict the weather, render to produce a movie, calculate first 1000 digits of π

Some well-defined computations:

▶ calculations carried by an electronic computer or calculator
▶ calculations performed on a analytical engine, Turing machine
▶ majority of mathematical statements and calculations
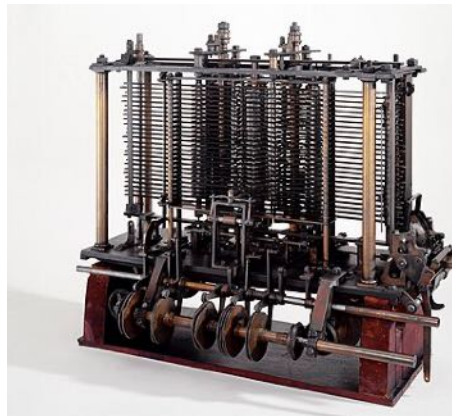
World's first computing device?

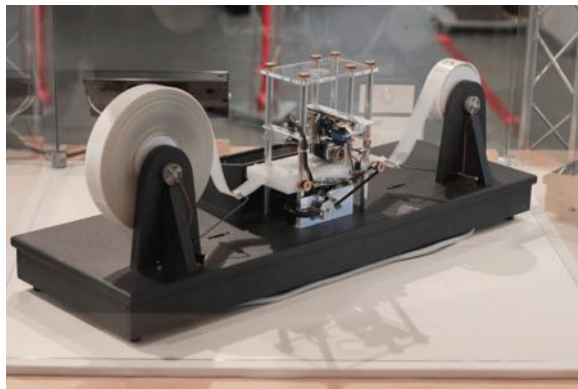The escapement clock, a man-made device, to keep track of time.
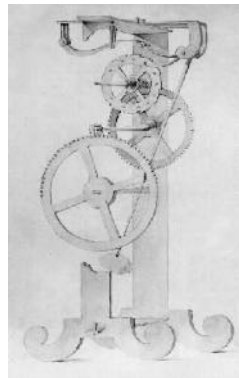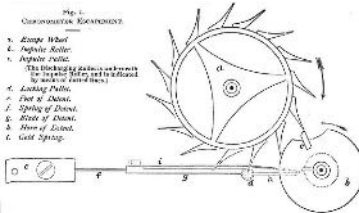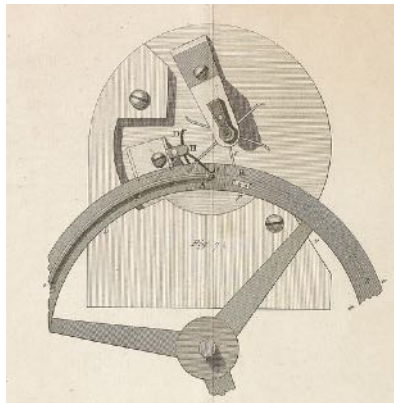
The Analytical Engine

- ▶ The very first mechanical general purpose computer system
- ▶ Designed in 1837 by the English mathematician and computer pioneer Charles Babbage
- ▶ Built as a programmable device to solve different things

Turing Machine

- ▶ Infinite tape, divided into cells with symbols
- ▶ A head can read/write symbols on the tape
- ▶ A register that stores the state of the machine

Modern calculator

▶ Execute a number of precise operations

▶ Designed to contain a set of such operations and instructions

▶ Includes even more complex operations, graphing charting

▶ Example: Texas Instruments
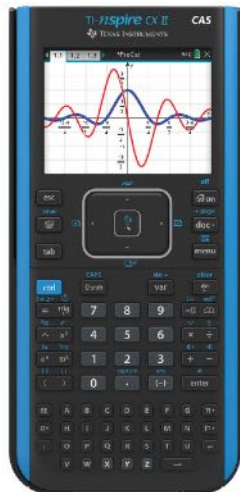
Single-Board Computers (SBC)

▶ A simple computing device

▶ CPU, GPU and Memory on a single chip

▶ Designed to run few applications

▶ Example: Raspberry PI

Datacenter Servers

- Runs 24x7
- Enterprise applications
- Suitable for large number of users and applications
- Available for data-centers

Computation can be seen as a **sequence of states** a computing device does.

▶ Contains one or many sequence of states
▶ Can be an infinite number of sequence of states
▶ Or it can terminate with a final state

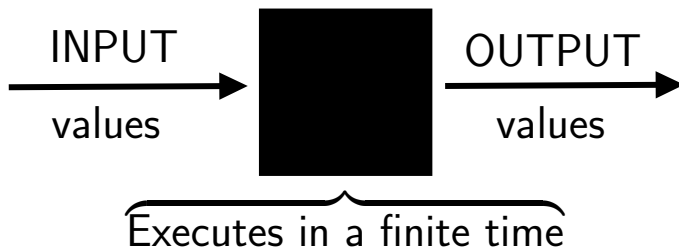# Algorithms

What is an algorithm?

In mathematics and computer science, an **algorithm** (/ˈælɡərɪðəm/ ◄ⁱ)) is a finite sequence of rigorous instructions, typically used to solve a class of specific problems or to perform a computation.[1] Algorithms are used as specifications for performing calculations and data processing.

- ▶ But can we call this a computer program?
- ▶ Or is it just a recipe, something higher than a sequence of code
- ▶ A higher level of abstraction of how to implement something we plan to develop. Example: how to find a name in a phone book

What is an algorithm?

INPUT values → [ ] → OUTPUT values

Executes in a finite time

Sequence of computational steps that transform the input into the output

Algorithms are everywhere

From your kitchen, in your microwave oven, your washing machine, to your phone or computer. When you browse Internet web sites your web browser is using different algorithms to decide how to display data to you.

Our society relies on algorithms to suggest sentences for convicted criminals. You even use algorithms to keep you alive: the control systems from your car, or in different medical devices.

But how can we describe them?

As an abstraction of something you plan to build or use, including all basic operations to achieve that. For example, think you plan to search in a phone book, a person phone number by the name:

▶ you can start page by page searching for that name

▶ or you can jump directly to certain letters and start following from there

▶ or you can apply a different strategy, by 'cutting' the book into half, checking the letter in which half belongs, and applying all over again the same principle until the name is found

How can we write one algorithm?

You can write it in plain English or in a more precise way using mathematics. Some others are using a form of pseudocode.

In computer science, **pseudocode** is a description of the steps in an algorithm using a mix of conventions of programming languages (like assignment operator, conditional operator, loop) with informal, usually self-explanatory, notation of actions and conditions.[1][2] Although pseudocode shares features with regular programming languages, it is intended for human reading rather than machine control. Pseudocode typically omits details that are essential for machine understanding of the algorithm. The programming language is augmented with natural language description details, where convenient, or with compact mathematical notation.

Example phonebook

1: **procedure** $\mathrm{PHONEBOOK}(N)$ ▷ Returns person name: N

2:      $N \leftarrow 1$

3:      Open page number N

4:      Look at the page N

5:      **if** Person is on page N **then**

6:          **Call person** N ▷ The person name is N

7:      **else**

8:          **Find next page.** $N \leftarrow N + 1$ Go back to 3

9:      **end if**

10: **end procedure**
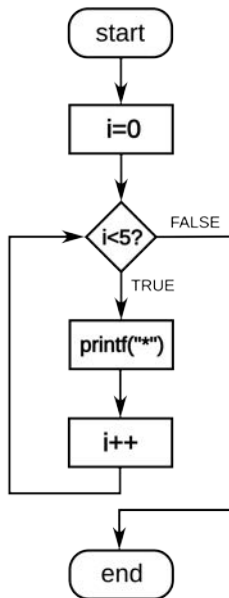
Can we write an algorithm as a diagram?

Flowcharts

Algorithms as flowcharts. What is a flowchart?

- ▶ is a type of technical diagram
- ▶ that represents a workflow or a process
- ▶ a sequence of states
- ▶ including their specific order
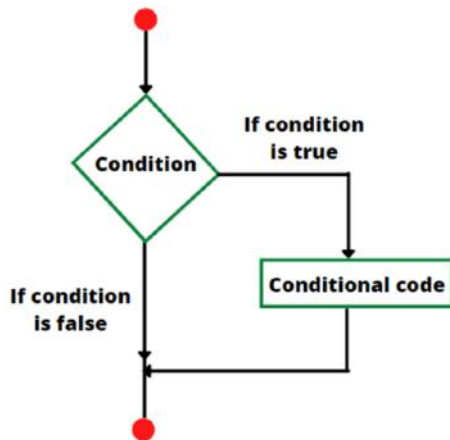
What is a flowchart?

A **flowchart** is a type of diagram that represents a workflow or process. A flowchart can also be defined as a diagrammatic representation of an algorithm, a step-by-step approach to solving a task.

The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. This diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.[1]

Overview [edit]

Flowcharts are used to design and document simple processes or programs. Like other types of diagrams, they help visualize the process. Two of the many benefits are flaws and bottlenecks may become apparent. Flowcharts typically use the following main symbols:

- A process step, usually called an *activity*, is denoted by a rectangular box.
- A decision is usually denoted by a diamond.
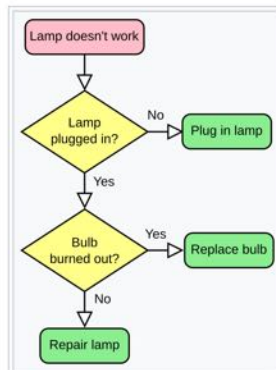
A simple flowchart representing a process for dealing with a non-functioning lamp.
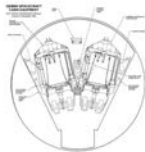
Flowchart diagram

▶ describes the sequence of movements or actions of people or things involved in a complex system or activity

▶ a graphical representation of a computer program in relation to its sequence of functions (distinct from the data it processes).

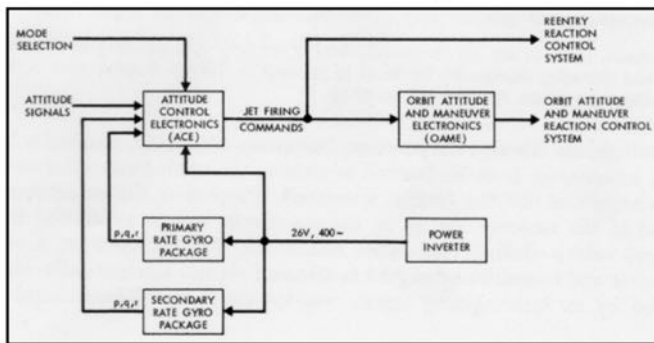**Yes, we can write and define an algorithm as a flowchart**

Functional block diagrams

The functional block diagram

Another type of diagram, a specialized high-level type of flow chart.

We can specify and describe the modules of a complex system
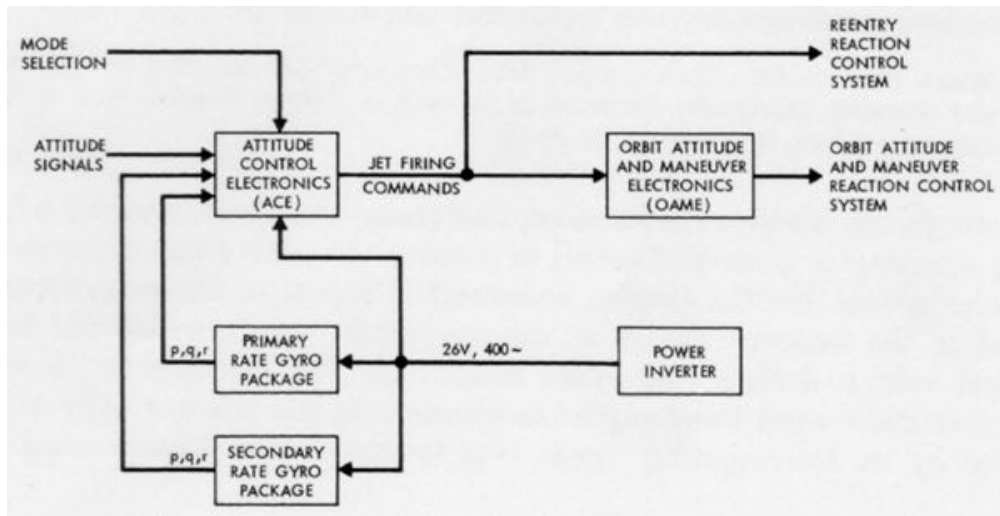
The functional block diagram

Functional block diagrams (FBD) describe system relationships along with their main functions. It is the basic, canonical document for any system or application before drawing any other more complex technical design plans. These diagrams are widely used in hardware engineering.

Functional block diagram

▶ describes the relationships of a system, by specifying their main functions, including the input and output of each main modules and components

▶ the input and output elements of a block representing by lines

▶ the relationships between the functions or modules

▶ the functional sequences and paths for the data flow

Lesson 1
Algorithm
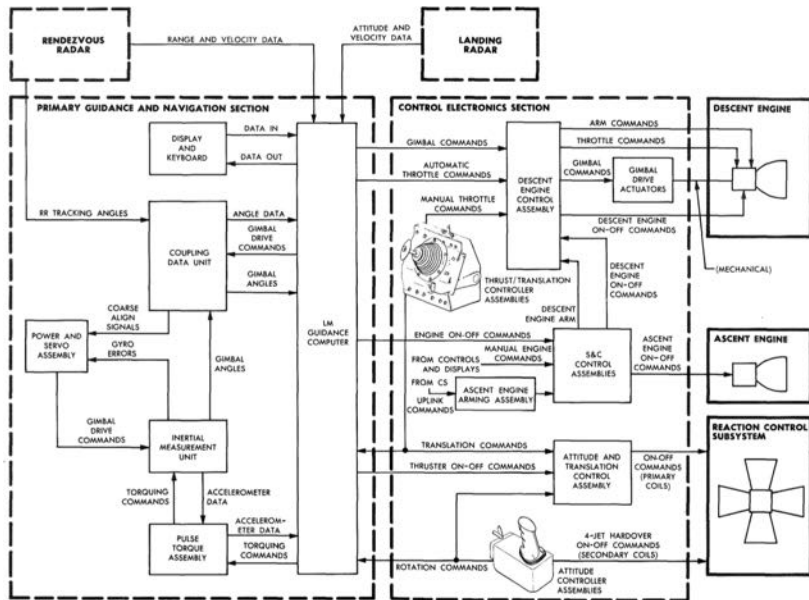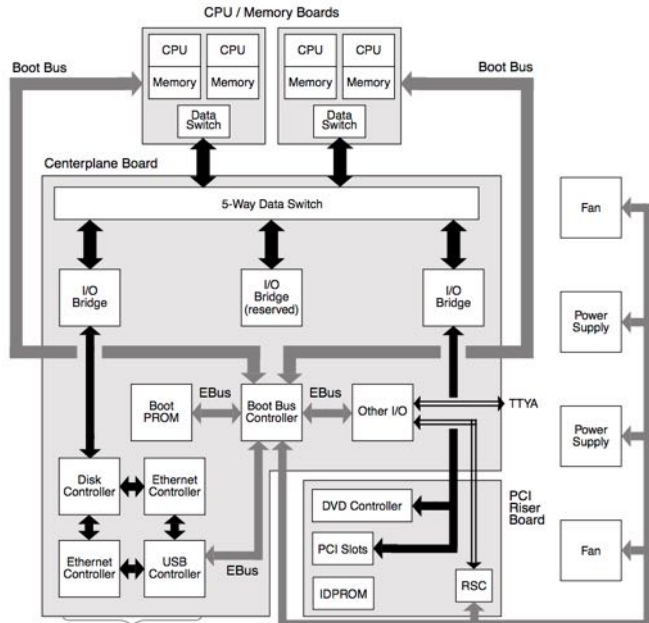

Figure 3-2.4. Primary Guidance Path - Simplified Block Diagram

Functional block diagrams are better to describe and document entire systems, their components and data flows, not algorithms. Use a flowchart for that!

But why not using a programming language?

We must think what we are planning to do, and how we plan to do it. And for that, we must not rely on a programming language: we will be restricted by the limits of the specific programming language not being abe to design and think freely about it.

A better approach is to think to write it as pseudocode or simple mathematics. This way we can have all flexibility and the power of precise mathematics.

Euclid's Algorithm or GCD

In mathematics, the **Euclidean algorithm**,[note 1] or **Euclid's algorithm**, is an efficient method for computing the greatest common divisor (GCD) of two integers (numbers), the largest number that divides them both without a remainder. It is named after the ancient Greek mathematician Euclid, who first described it in his *Elements* (c. 300 BC). It is an example of an *algorithm*, a step-by-step procedure for performing a calculation according to well-defined rules, and is one of the oldest algorithms in common use. It can be used to reduce fractions to their simplest form, and is a part of many other number-theoretic and cryptographic calculations.

Euclid's Algorithm

Greatest Common Divisor (GCD) of two numbers A and B is the largest number that divides both A and B. (Number here defined as an integer)

An integer is the number zero (0), a positive natural number (1, 2, 3, etc) or a negative integer with a minus sign (-1, -2, -3, etc) In mathematics we call this $\mathbb{Z}$ set of numbers.

Euclid's Algorithm

The very first version:

If A = 0 then GCD(A,B)=B, since the GCD(0,B)=B, and STOP
If B = 0 then GCD(A,B)=A, since the GCD(A,0)=A, and STOP
Write A in quotient remainder form (A = B * Q + R)
Compute then GCD(B,R) since GCD(A,B) = GCD(B,R)

Euclid's Algorithm, pseudocode

1: **procedure** $\mathrm{GCD}(A, B)$ $\triangleright$ The g.c.d. of A and B
2:      $R \leftarrow A \bmod B$
3:     **while** $R \neq 0$ **do** $\triangleright$ We have the answer if R is 0
4:         $A \leftarrow B$
5:         $B \leftarrow r$
6:         $R \leftarrow A \bmod B$
7:     **end while**
8:     **return** B $\triangleright$ The gcd is B
9: **end procedure**

Euclid's Algorithm, improved

1: **procedure** EUCLID(A, B) ▷ The g.c.d. of A and B
2:     **if** $B == 0$ **then**
3:         **return** A ▷ The gcd is A
4:     **else**
5:         **return** EUCLID($B, A \bmod B$)
6:     **end if**
7: **end procedure**

Euclid's Algorithm, using mathematics

──────────── MODULE *Euclid* ────────────

EXTENDS Integers

VARIABLES x, y

CONSTANTS a, b

$Init \triangleq (x = a) \wedge (y = b)$

$Next \triangleq (x > y$
$\wedge x' = x - y$
$\wedge y' = y)$

$\vee (y > x$
$\wedge y' = y - x$
$\wedge x' = x)$

But how do we know our algorithm does the right thing?

"An algorithm for a computational problem is CORRECT if, for every problem instance provided as input it HALTS, finishes its computing in finite time and outputs the correct solution to the problem instance"

A correct algorithm SOLVES the given computational problem

An incorrect algorithm might not HALT at all on some input instances, or it might halt with an incorrect answer

Classes of algorithms

- Divide and Conquer
- Sorting
- Searching
- Dynamic programming
- Greedy algorithms
- Graph algorithms
- Shortest path
- Maximum flow
- Parallel algorithms
- Matrix operations
- Online algorithms
- Machine learning
- Linear programming
- String matching

How about AI?

Still an algorithm

- ▶ Or more precisely many algorithms
- ▶ Complex algorithms
- ▶ But still algorithms nevertheless

Algorithms dont feel are not conscious, sentient beings!

We need safe and secure software

▶ predictable - which we can CONTROL
▶ not harmful - USEFUL to us
▶ being able to STOP it if we want

This is not about being nice, but rather being able to describe and proof that our algorithms do the right thing and do it efficiently based on mathematically tools

This is what we need.

Computer Programs

What is a computer program?

A **computer program** is a sequence or set of instructions in a programming language for a computer to execute. It is one component of software, which also includes documentation and other intangible components.[1]

How to think about programs

▶ We should think like engineers or computer scientists
▶ Scientific thinking is very successful
▶ Science makes mathematical models of reality
▶ Digital systems, processors, storage devices
▶ We model how computers are executing programs

Basic models

▶ Touring machines
▶ Functions
▶ Automata
▶ Sequence of states

Program. Computation. Sequence of States

▶ A program execution is a computation (behaviour)
▶ A computation is a sequence of states
▶ A state is an assignment of values to variables

A program is modeled as a set of computations, representing all possible program executions

Computers, digital systems are executing programs

From source code to executable

Source Code (Text Editor) → Compiler → Executable/ Machine Code → Operating System → Output
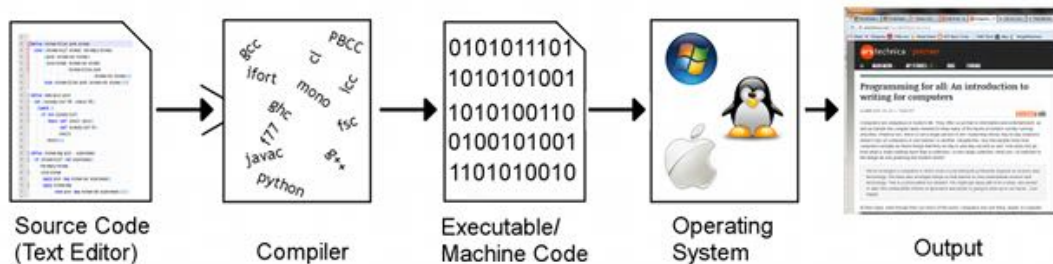
Program structure

A **computer program** is a sequence or set of instructions in a programming language for a computer to execute. It is one component of software, which also includes documentation and other intangible components.[1]

▶ Some programs run forever, some dont
▶ A program execution is defined by at least one computation
▶ A computation is a sequence of states
▶ And a state is an assignment of values to variables

Program types

- A program is modelled by a set of computations, representing all possible executions
- Remember an algorithm is just an abstract program
- Different programs: software applications and system software
- System software: operating systems

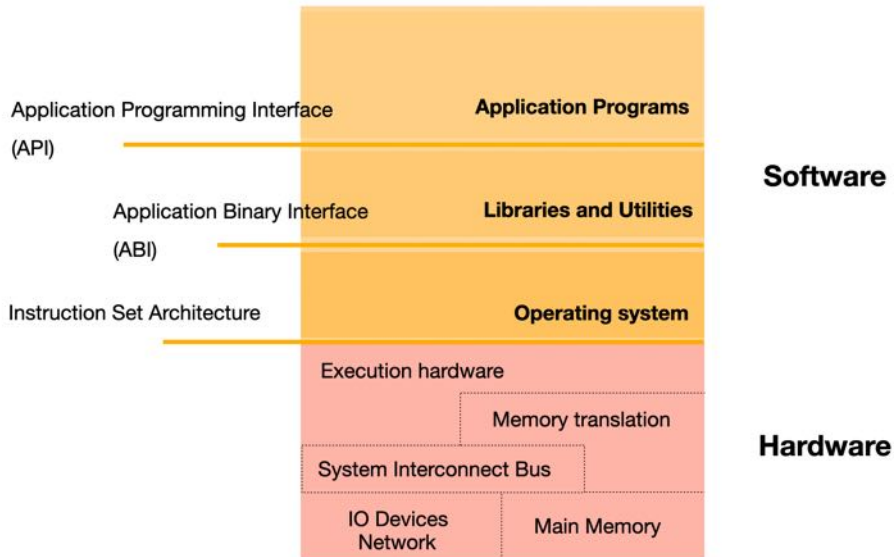**Application and System Programs**

▶ **Software applications**: enterprise resource planning, customer relationship management, supply chain management software, web, middleware, databases

▶ **Operating systems**: macOS, RedHat, FreeBSD, Windows

Lesson 2
Software speifications. Formal methods

Lesson 3
Data Structures and Algorithms

Lesson 4
Programming vs Coding

Lesson 5
Data Capturing, Transformation, Analysis

Lesson 6
Build Safe and Secure Software

Lesson 7
System Performance Analysis