CS2034B / DH2144B

Data Analytics: Principles and Tools

Western
UNIVERSITY · CANADA

Week 4

Computation and Algorithms

Introduction to Computation & Algorithms

Western Science

Assigned Reading

zyBook Chapter 6: Computation, Algorithms and Programming

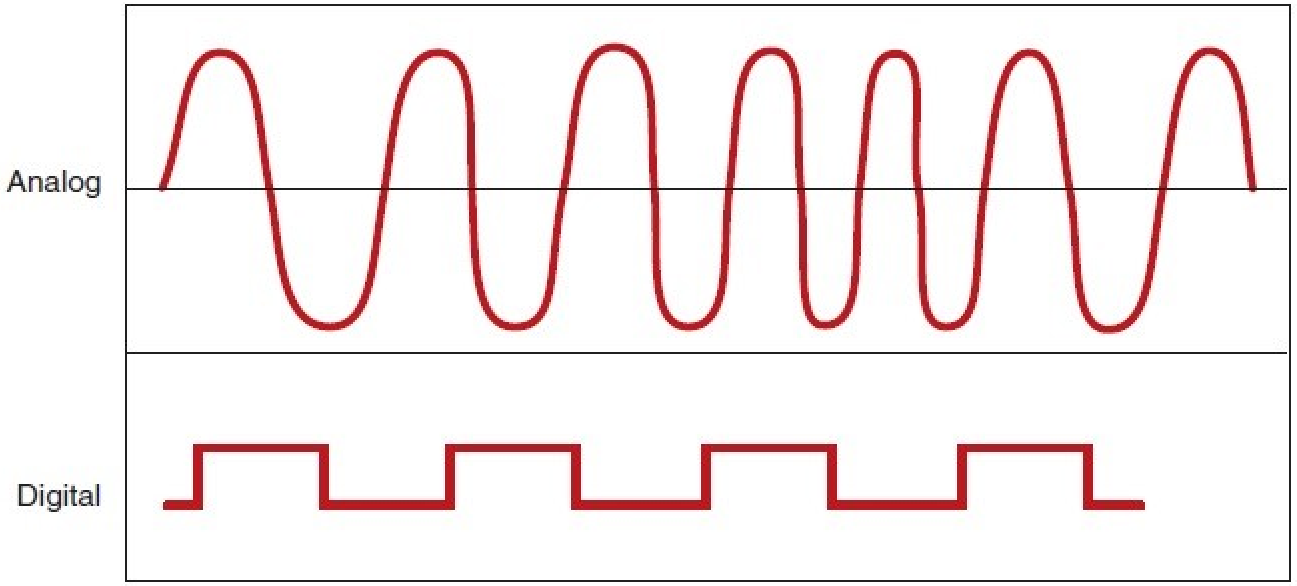
Home & Learn: Getting Started
https://www.homeandlearn.org/

What is Computation?

- Physical objects:
  - Addition by measuring and grouping.
  - Abacus, slide rules, etc.
- Human computation:
  - Mental or pencil and paper arithmetic.
- Natural computation:
  - Using ants, DNA, molecules. Optional reading: *The many facets of natural computing*
- Electronic computation

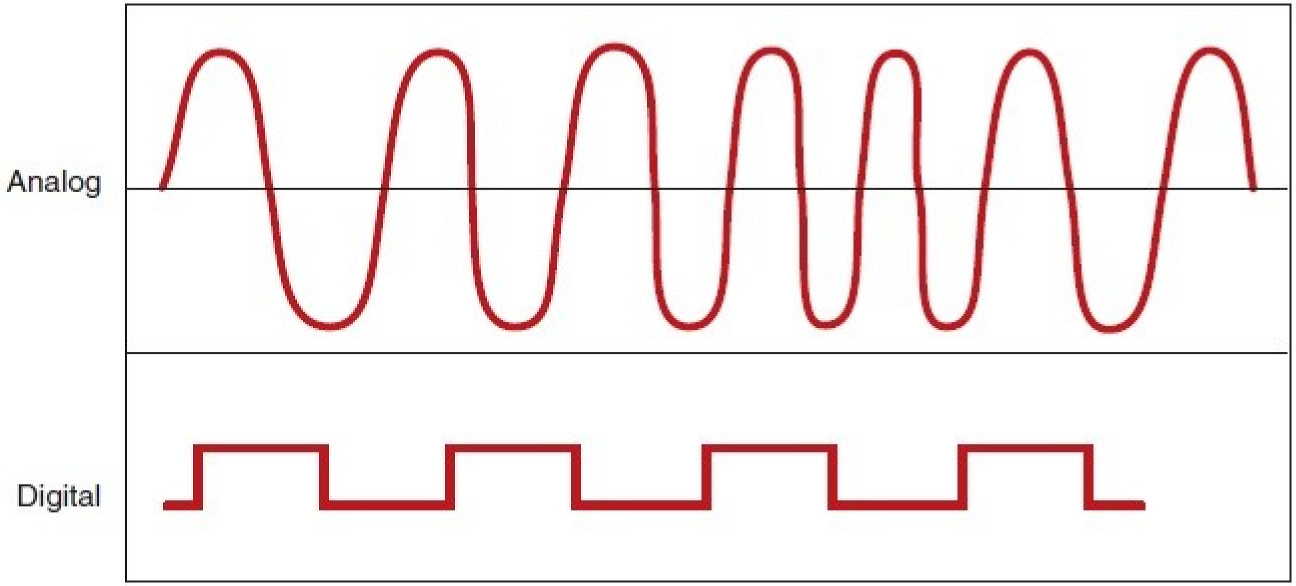
What is Computation?

- ***Analog*** electronic computing:
  - Use of circuits in which a voltage represents a numerical value, which is perhaps time dependent.
  - Additional parts of the circuit give voltages representing computed functions (e.g. addition, integration, …)



What is Computation?

- ***Digital*** electronic computing:
  - Use of circuits where a voltage threshold is met or not.
  - Represents 0 or 1.
  - Voltages on many lines can together represent numbers.

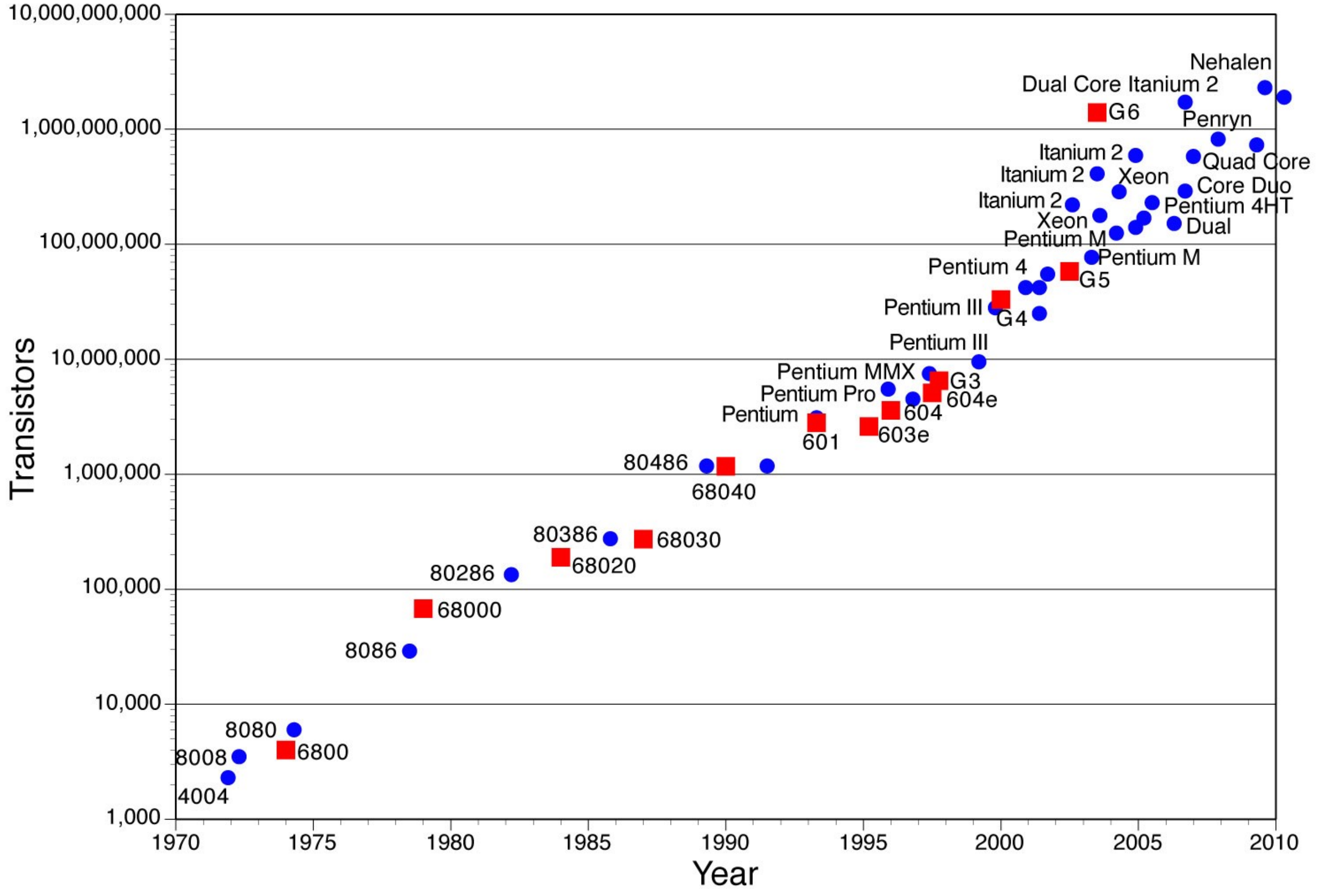


Moore’s Law

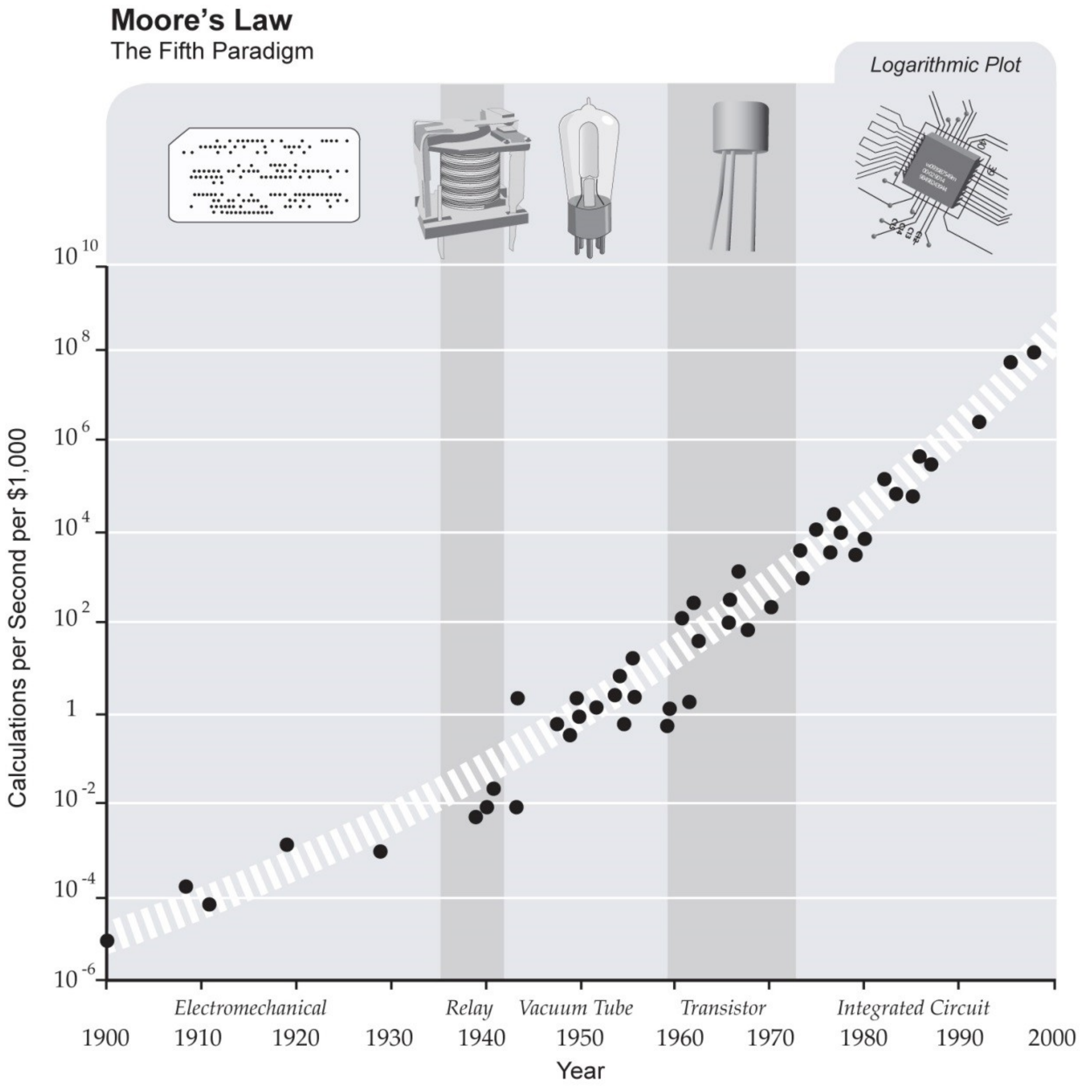
- Gordon E. Moore noticed in 1965 that integrated circuit speeds had been doubling every year for the past many years.
- This has remained essentially true to date: computer processor speed has doubled about once every year to 18 months since the 1950s.
- Has become self-fulfilling prophecy.

Moore’s Law

Circuit Complexity



Moore's Law



Moore's Law
The Fifth Paradigm
Logarithmic Plot
Calculations per Second per $1,000
10^{10}
10^{8}
10^{6}
10^{4}
10^{2}
1
10^{-2}
10^{-4}
10^{-6}
Electromechanical
Relay
Vacuum Tube
Transistor
Integrated Circuit
1900
1910
1920
1930
1940
1950
1960
1970
1980
1990
2000
Year

Limits of Computation

- Will Moore's law hold true forever?
- Unlikely, hard limits exist:
  - Speed of light
  - Planck length

- Other limits on computation?

Limits of Computation

"Imponderable" questions:

- What is the answer to life, the universe and everything?
- What does the color blue taste like?

Limits of Computation

"Soft" questions:

- Will a computer ever be able to compose an expressive piece of music?
- Will a computer ever be able to feel love?

Limits of Computation

"Exact" questions:

- Will a computer ever be able to simulate a financial system with one million participants?
- Will a computer ever be able to calculate the effects of global warming?
- Can a computer prove the Pythagorean Theorem?

Limits of Computation

Decidability

In logic, a set of questions is “decidable” if there is an algorithm that correctly returns true or false for all questions in the set.

Limits of Computation
Undecidable Problems

Proven there is no algorithm.

Examples:

- Given an arbitrary program and a finite input, decide whether the program finishes running or will run forever. (Turing, 1936).
- Decide whether a mathematical expression is identically zero.
- Given two grammars, determine whether they describe the same language.
- Determine whether a particular kind of equation has a solution or not. (Hilbert's 10th problem, 1900).

Undecidable Problems

These are *proven* to not be computable.

No computer, no matter how fast or how big can ever solve these problems, even if it uses all the matter in the universe and runs for billions of years.

Algorithms

An ***algorithm*** is a step by step description of how to solve a problem.

An ***algorithm*** describes a **sequence of steps** that is:

1. **Unambiguous**
   a. No "assumptions" are required to execute the algorithm
   b. The algorithm uses **precise** instructions
2. **Executable**
   a. The algorithm can be carried out in practice
3. **Terminating**
   a. The algorithm will eventually come to an end, **stop** or halt

An Example Problem

Count the number of people in the room:

```
let N = 0

for each person, x, in the room:
      point at x
      add 1 to N
      say N

say "There are N people in the room!"
```

An Example Problem

Count the number of people in the room:

```
let N = 0

for each person, x, in the room:
      point at x
      add 1 to N
      say N

say "There are N p
```

- Informal high-level description of the **algorithm**.
- Human readable **pseudocode**.
- Intended for humans and not machines.

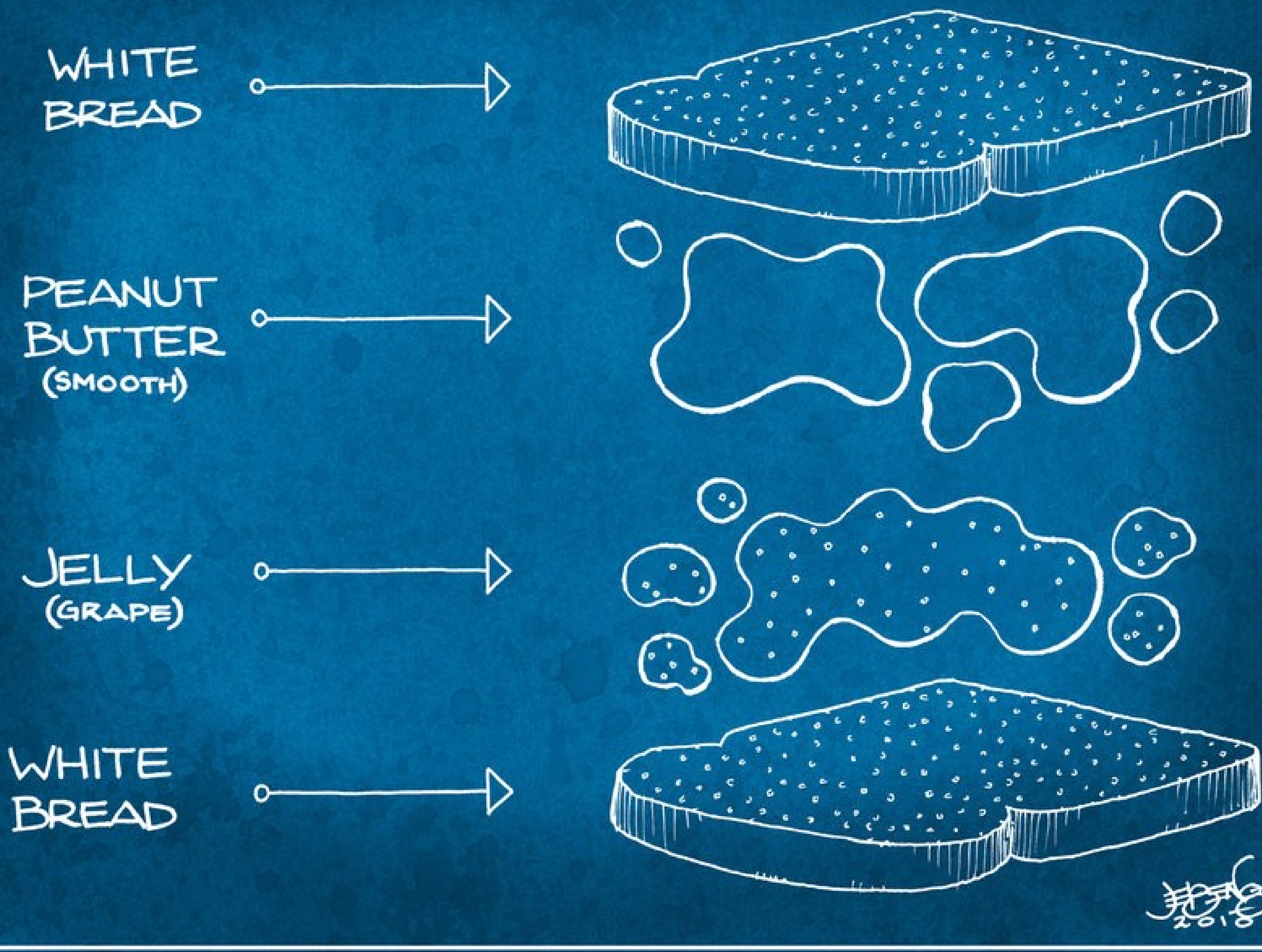
Write Your Own Algorithm!

Think about or write an **algorithm** to make a **peanut butter and jelly sandwich**.

Try to make your **algorithm**:

- **unambiguous** (precise)
- **executable** (possible)
- **terminating** (it eventually stops)

PEANUT BUTTER AND JELLY



Evaluating Algorithms

- How can we tell if an **algorithm** is "good"?
- How can we compare **algorithms** that have the same inputs and results?
- If the results are the the same are they equivalent regardless of the steps used?

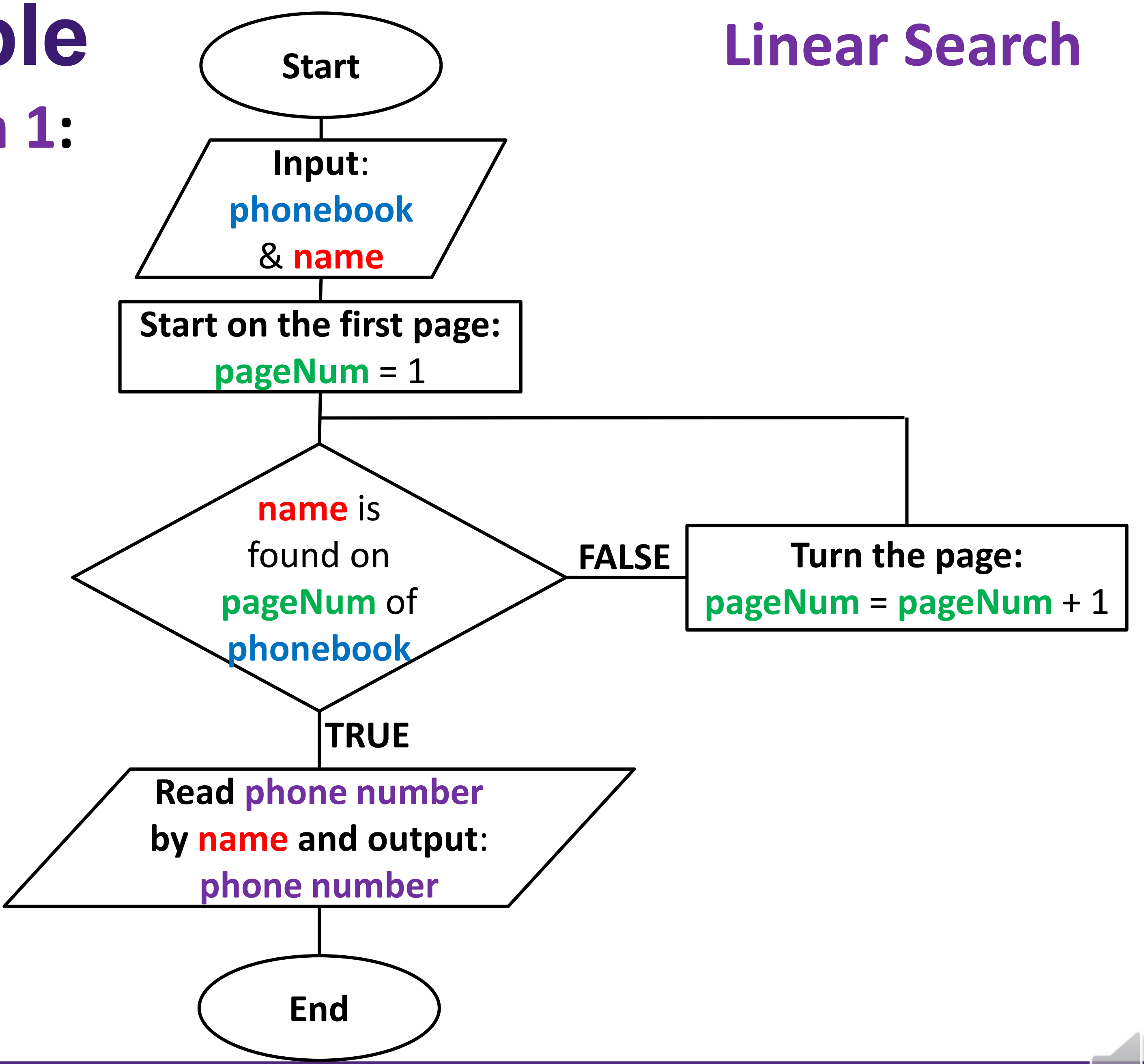
Example

Create an **algorithm** to look up a person's phone number in a phone book.

Example

Algorithm 1:

Linear Search

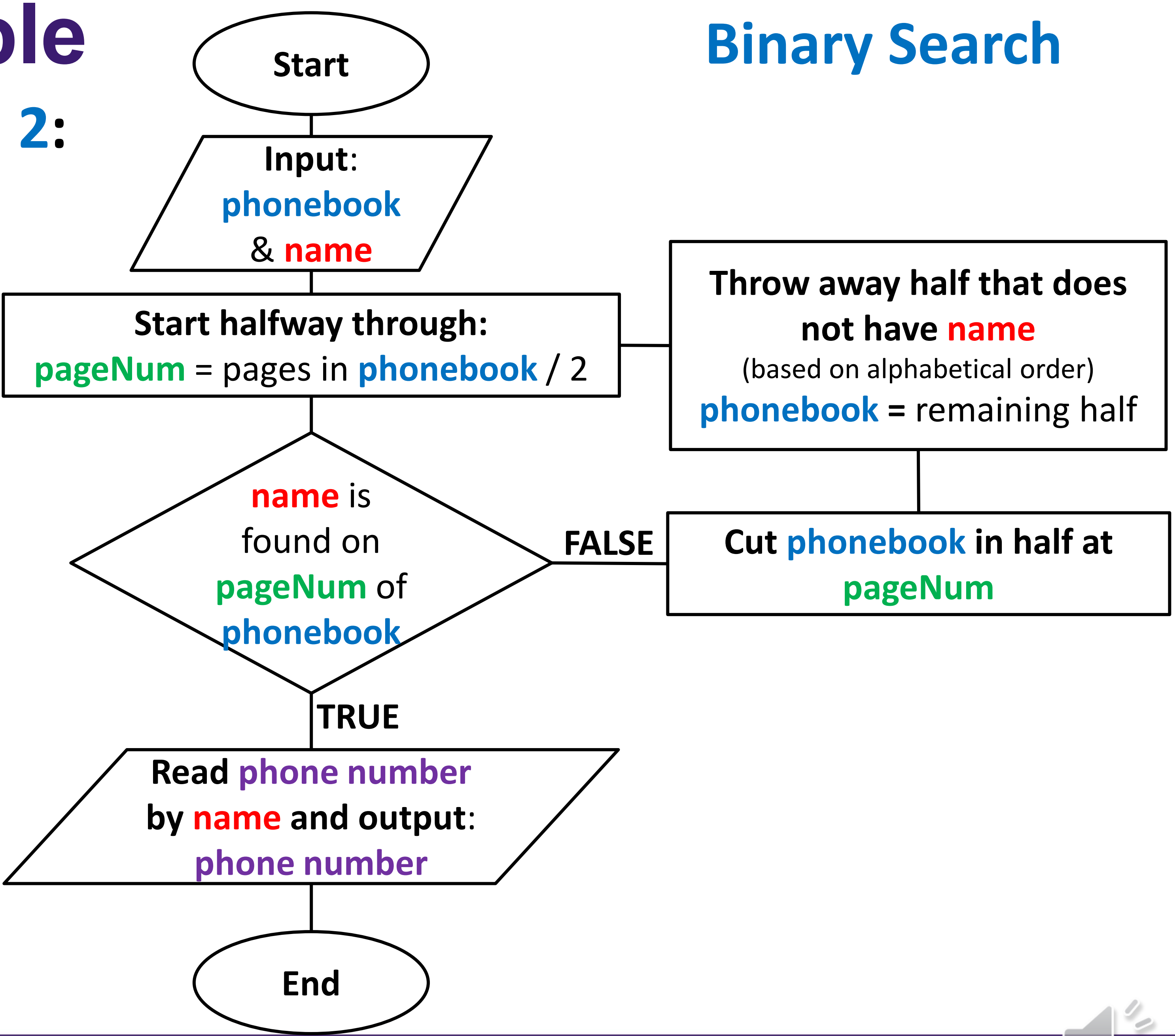


Start
Input: phonebook & name
Start on the first page: pageNum = 1
name is found on pageNum of phonebook
FALSE
Turn the page: pageNum = pageNum + 1
TRUE
Read phone number by name and output: phone number
End

Example

Algorithm 2:

Binary Search

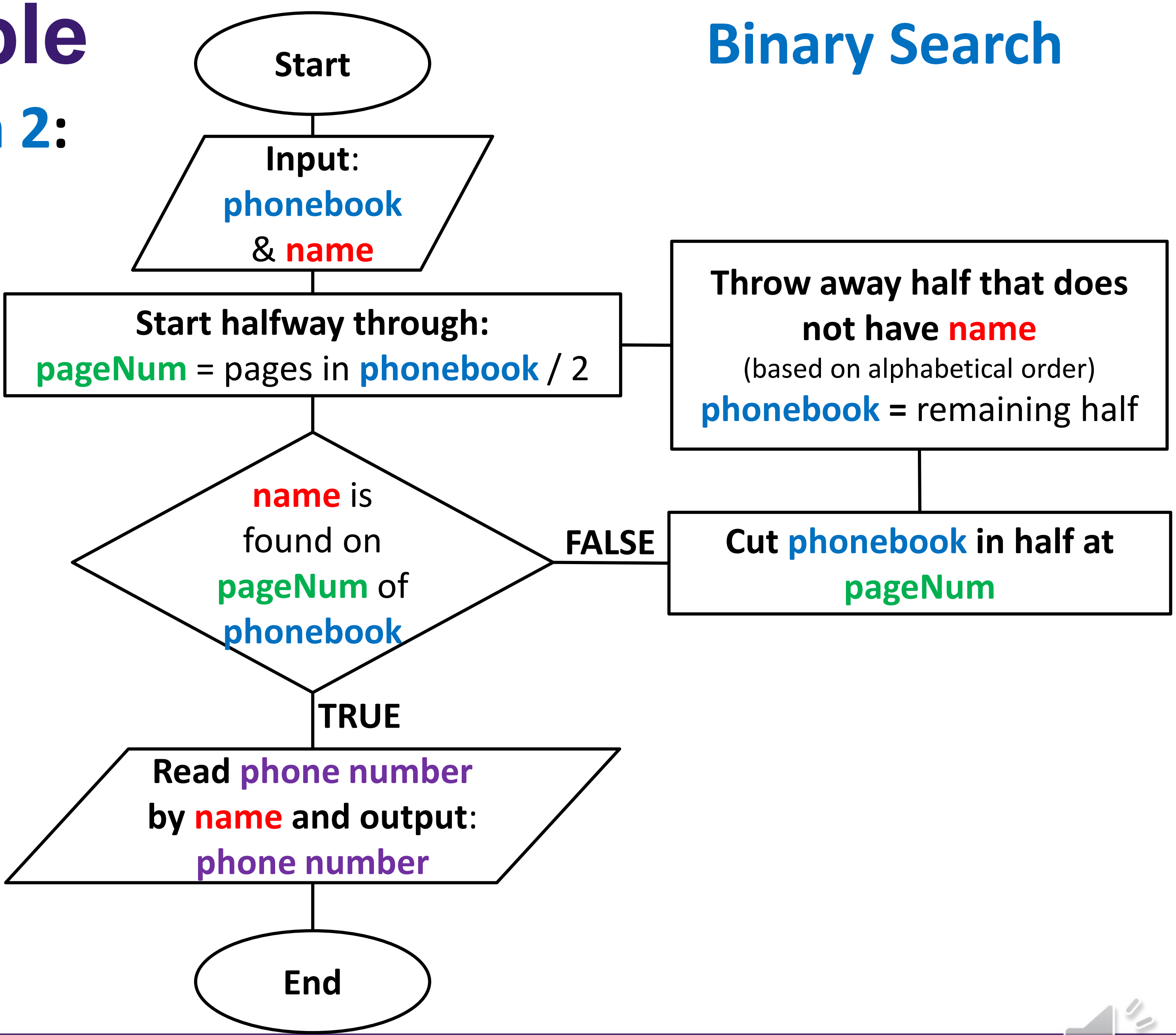


Start
Input: phonebook & name
Start halfway through: pageNum = pages in phonebook / 2
name is found on pageNum of phonebook
FALSE
Cut phonebook in half at pageNum
Throw away half that does not have name
(based on alphabetical order)
phonebook = remaining half
TRUE
Read phone number by name and output: phone number
End

Example

Algorithm 2:

Binary Search



Correctness

- Can the algorithm be executed correctly for all inputs (does not cause an error)?
- Dose the algorithm return the correct result in all cases?
- Is the algorithm guaranteed to terminate for all inputs?

These two algorithms are only correct if the person's name is guaranteed to be in the phonebook.

Correctness

- Can the algorithm be executed correctly for all inputs (does not cause an error)?
- Dose the algorithm return the correct result in all cases?
- Is the algorithm guaranteed to terminate for all inputs?

These two algorithms are only correct if the person's name is guaranteed to be in the phonebook.

Time Complexity

- How many **basic steps** or operations are performed by the **algorithm**?
  - In our example, a **basic step** might be turning to a page.
- Relative to the **input size**.
  - Phone book of 10 pages vs. 10,000.
- Normally, we care about the **worst-case**.
  - Looking for "Alice Aardvark" vs. "Zachary Zebra"

Time Complexity

Algorithm 1:

- We need **N** steps (page turns) in the worst case.

Algorithm 2:

- We need $\lfloor \log_2(N)+1 \rfloor$ steps (page turns) in the worst case.

$$N > \lfloor \log_2(N)+1 \rfloor$$

for N>2

Time Complexity

Algorithm 1:

- We need **N** steps (page turns) in the worst case.

Algorithm 2:

- We need $\lfloor \log_2(N)+1 \rfloor$ steps (page turns) in the worst case.

$$O(N) > O(\log(N))$$

Big O Notation

Programming

The process of creating an executable computer program to address a problem.

Also known as *coding or implementation*

It involves converting an algorithm into a set of precise instructions (i.e., a "program") that a computer can read.

Machine Instructions

Low-level commands such as:

Get the value stored at location X, add 12 to it and store it in location Y.

or

If location X contains an odd number, start executing instructions stored at location Z.

Here *X*, *Y* and *Z* are numbers saying which memory slots to use.

Machine Instructions

Machine Code

```
40052d:        55
40052e:        48 89 e5
400531:        bf e0 05 40 00
400536:        e8 d5 fe ff ff
40053b:        b8 00 00 00 00
400540:        5d
400541:        c3
400542:        66 2e 0f 1f 84 00 00
400549:        00 00 00
40054c:        0f 1f 40 00
```

Machine Instructions

Machine Code **Assembly**

```
40052d:      55                        push   %rbp
40052e:      48 89 e5                  mov    %rsp,%rbp
400531:      bf e0 05 40 00            mov    $0x4005e0,%edi
400536:      e8 d5 fe ff ff            callq  400410 <puts@plt>
40053b:      b8 00 00 00 00            mov    $0x0,%eax
400540:      5d                        pop    %rbp
400541:      c3                        retq
400542:      66 2e 0f 1f 84 00 00      nopw   %cs:0x0(%rax,%rax,1)
400549:      00 00 00
40054c:      0f 1f 40 00               nopl   0x0(%rax)
```

Programs

A program is a sequence of instructions like these stored in memory.

Programs may be hand written in “assembly code”, which gives a human-readable form of machine instructions.

Programs may also be written in higher-level languages that get translated into machine code.

Programming Languages

- Artificial languages for giving instructions that people can understand.
- **Examples:** JAVA, VBA, Python, C, C++, C#, FORTRAN, COBOL, R, PHP
- There are hundreds of them.

Programming Languages

More abstract ideas, e.g. lists, text, windows

More powerful tools, e.g. functions, objects

Ways we Use Programming Languages:

Compiled: Programs are written in high level languages and translated into machine code or other low level languages using a compiler (a type of program). Java and C are programming languages that are normally compiled.

Interpreted: An interpreter (a type of program) is used to read the program and do what the instructions in it say to do. Ruby and Python are languages that are normally interpreted .

Algorithms vs. Programs

- An ***algorithm*** is a general procedure to compute the solution to a problem.
  Given an input, it always produces an output.

- A ***program*** is a particular set of instructions in some language.