COMP 2310
Data Structures and Algorithms

Lecture 1

Introduction

Wisdom

Tell me and I forget.

Show me and I remember.

Let me do and I understand.

—Chinese Proverb

The Need for Algorithms and Data Structures

- Representing information internally is fundamental to computer science.
- The main goal of most computer programs is to store and retrieve information systematically and efficiently.
- For this reason, the study of data structures and algorithms are at the heart of computer science and data processing.

Course Objectives

The objective of this course is to help students understand how to structure information to support efficient data processing.

More specifically:

1. To study commonly used algorithms and data structures.
2. To introduce the idea that there are costs and benefits associated with every algorithm.
3. To teach how to determine the efficiency of using a data structure or algorithm.
4. To show how to implement common data structures and algorithms in programming languages.

Topics to be Covered

Topics:

- Commonly used algorithms
- Commonly used data structures
- Simple complexity analysis
- Implementation details in C++

Important:

C++ implementations aims to help learning the corresponding data structures and algorithms.

Teaching C++ is not the intended objective of this course!

Introduction

- What is a problem?
- What is an algorithm?
- What is a data structure?
- How to implement algorithms and data structures?

Problem

A problem is a mapping from an input to an output.

Examples:

- Sqrt: Value -> Value
- Sorting: Unsorted data -> Sorted data
- Solving an equation: Equation -> solution

Algorithms

- In computer science, algorithm refers to: ”A special method usable by a computer for solving a problem”.

- In general, we have the relationship:

  Algorithms + Data Structures = Programs

Algorithms

- An algorithm consists of a finite sequence of instructions.

Example:

How to find the minimum value in a list of integers?

Solution:

Scan the list from front to end and keep track of the minimum integer found so far.

Algorithms

- Commonly, we present algorithms using a simple language called pseudo-language.

Algorithms

Example: Finding minimum in an integer array

Solution: A function in pseudo code :

```
find_min ( n:integer, a:array of integer )
 {
    current_min = a[1]
    for i=2 to n do
      if a[i] < current_min then
         current_min = a[i]
     return current_min
 }
```

Algorithms: Desirable Properties

What are desirable properties of algorithms/Programs?

- correctness
- finite time
- modularity
- maintainability
- functionality
- user-friendliness
- extensibility

……………………

Efficiency is a common requirement for all the properties in the list.

Algorithms: Reliability and Robustness

Reliability

The probability of failure-free operation of an algorithm for a specified period of time in a specified environment.

- Robustness

The degree to which an algorithm can function correctly in the presence of invalid inputs or stressful environmental conditions.

Algorithms: Efficiency and Complexity

- Resource usage of an algorithm is important.
  - the time and space complexity for small inputs…
  - What will be the algorithm efficieny when input size grows unboundedly?
- This is the problem of algorithm complexity.

Efficieny and Complexity: Example

- Two algorithms A and B that solve the same class of problems.
- The time complexity of A is $5{,}000n$, the one for B is $\lceil 1.1^n \rceil$ for an input with n elements.
- For n = 10, A requires 50,000 steps, but B only 3, so B seems to be superior to A.
- For n = 1000, however, A requires 5,000,000 steps, while B requires $2.5 \cdot 10^{41}$ steps.

Why is Complexity Important?

- Comparison: Time for Algori*hm A : 5000n*

  Time for Algorithm B : $\lceil 1.1^n \rceil$

The number of steps

Input Size	Algorithm A	Algorithm B
n	5,000n	1.1^n
10	50,000	3
100	500,000	13,781
1,000	5,000,000	$2.5*10^{41}$
1,000,000	$5*10^9$	$4.8*10^{41392}$

Why is Complexity Important ?

- With powerful computers, why program efficiency/complexity is not becoming less important?

- But problem sizes and complexities are increasing too!

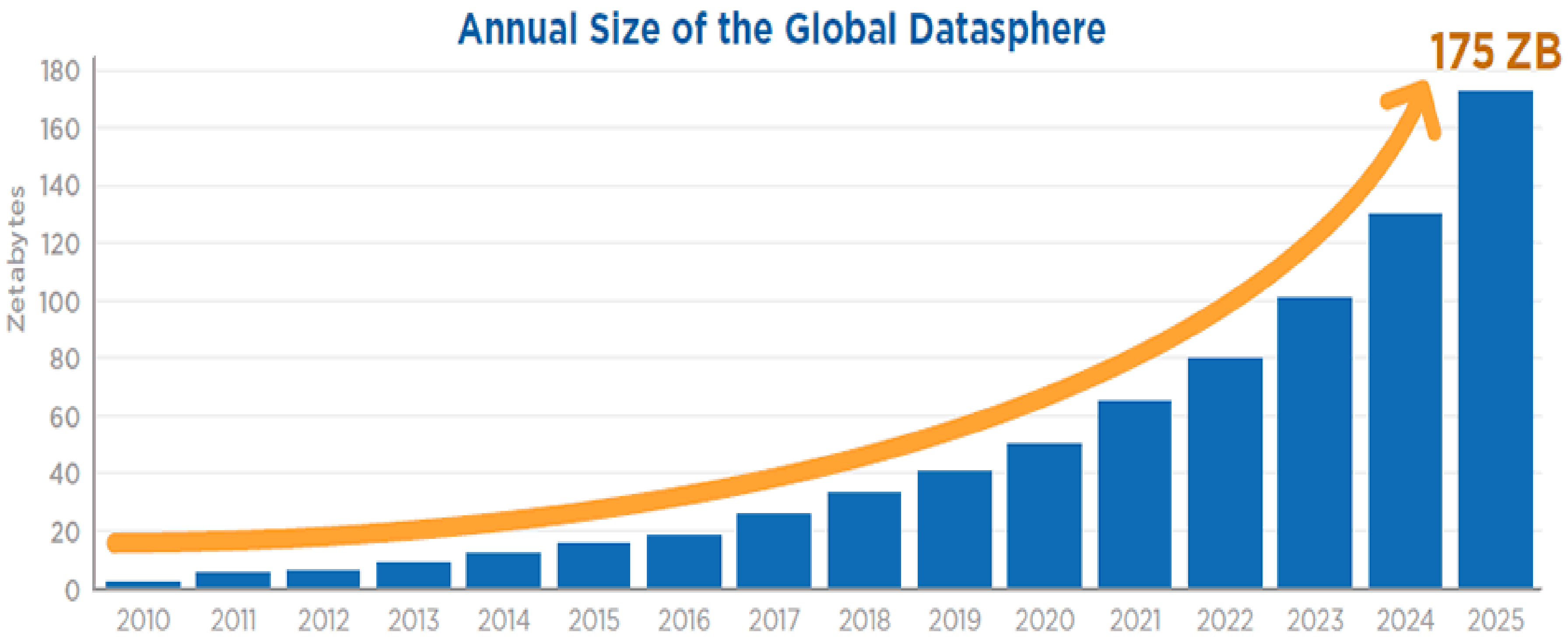
What is the Size of Big Data ?

Not possible to quantify:
Big data: the available computing and storage power on the market.

Example: The average commercial aircraft - like the Boeing 737 creates 20 terabytes of engine information per hour.

Remainder : Tera 10^{12}, Peta 10^{15} , Exa : 10^{18}, Zetta 10^{21}…..

Figure 1 - Annual Size of the Global Datasphere



Zettabyte(ZB) = 10^{21} Bytes.

Data Types

At the top level, there are two data types:

1. System-defined data types (also called *Primitive* data types)
2. User-defined data types

System-defined Data Types

- Data types that are defined by system are called primitive data types.
  - int, float, byte, char, double, bool,…

- The number of bits allocated for each primitive data type depends on the language, compiler and operating system.

User defined data types

- Formed by combining many system-defined data types into a single structure.
  - structures, unions, enum

- The system provides implementations and operations for the primitive data types.

Abstract Data Types

- An abstract data type (ADT) is composed of
  - A collection of data
  - A set of operations on that data

- Specifications of an ADT indicate: what the ADT operations do but not how to implement them.

- Implementation of an ADT requires choosing a particular data structure.

Data Structure

- A Data structure is a particular way of storing and organizing data in computer memory so that it can be used efficiently.

- A data structure is:
  - a collection of data values
  - the relationships among them
  - the functions that can be applied to the data.

Why do we Need Data Structures?

Suppose you wanted to write a program which uses a certain data set.

Some Important problems:

- How would you store the data in computer's memory?
- Would your method work for large data sets?
- Would your method allow for fast insert/delete operations?
- Would it allow for fast searching operations?
- How would you sort the data efficiently in a given order?

Data Structure Types

Data structures are classified into two types:

1. Linear data structures: Arrays, Linked Lists, Stacks and Queues.
2. Non – linear data structures: Trees and graphs.

Example: A simple algorithm

A is an array containing 8 numbers:

$$\mathbf{A} = \{7,\ 1,\ 3,\ 8,\ 0,\ 9,\ \ 6,\ 4\}$$

Problem : Are the values in the array sorted?

Informal algorithm :

Compare consecutive numbers until the end if there is an out of order pair return false

Pseudocode-1

```
procedure IsSorted( var $A$ : InputArray;  $N$ : integer );
    begin
     for $i$ : 1 to $N$ - 1 do
            if $A$[ $i$ ] > $A$[ $i$ + 1 ] then
            return false;
     return true;
   end;
```

Pseudocode-2

```
//Simplified notation
IsSorted( $A$, $N$ )
  for $i \leftarrow 1$ to $N - 1$
       if $A[ i ] > A[ i + 1 ]$
           then return false;
 return *true*;
```

Algorithm Complexity

- Algorithms differ in efficiency.
- The difficulty of algorithms for comparing their efficiencies is called computational complexity.
- Computational complexity indicates how much effort is required for an algorithm or how costly it is.
- An algorithm is hardly of any use if
  - very long time
  - large main memory