DATA STRUCTURES & ALGORITHMS

Chapter 1: Introduction

Le Van Vinh, PhD
Faculty of Information Technology
HCMC University of Technology and Education
Email: vinhlv@fit.hcmute.edu.vn

Outline

I. Why Data Structures and Algorithms?

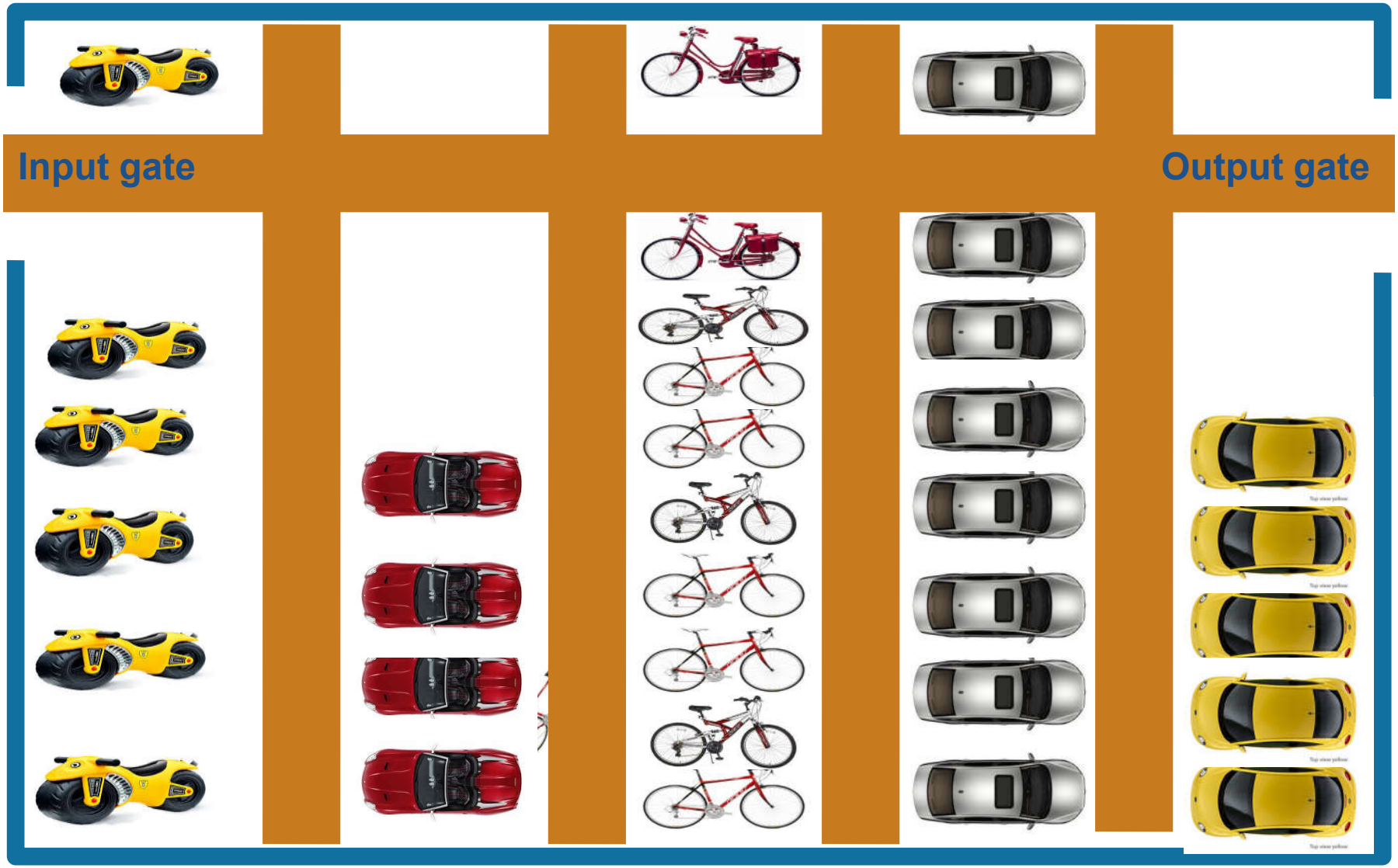
II. Concepts

III. Algorithm complexity

IV. Excercises

A garage

A garage



A garage vs. a program

A garage

- **Vehicles (cars, motobikes, bicycles)**
- **Structure to store vehicles**
- **Operations (Find a car, put a car in the garage)**

A program

- **Data**
- **Structure to store data**
- **Functions (Search(), insert())**

A good program

- **What is a good program?**

A good program

- **What is a good program**
  - Run correctly
  - Easy to understand source codes
  - Easy to debug
  - Easy to modify

A good program

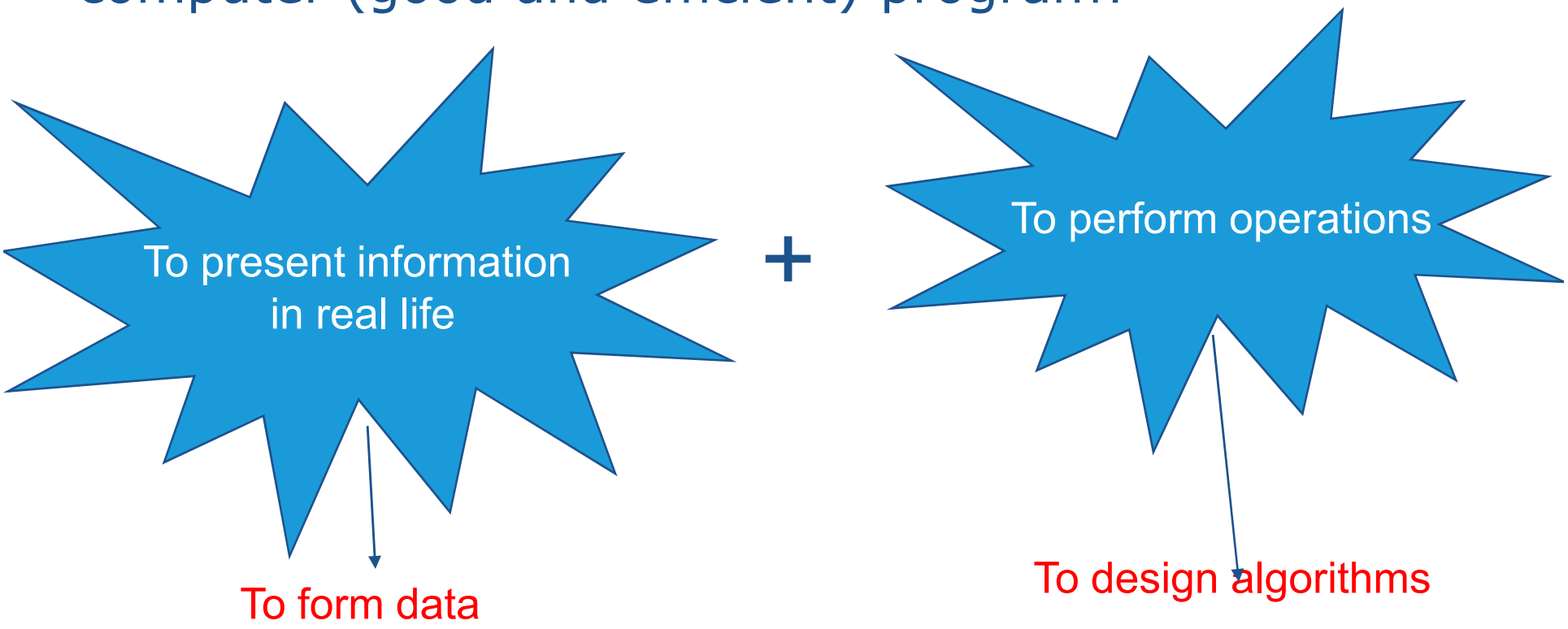
- **What is an efficient program (when it executes)?**

A good program

- **What is a good program**
  - Run correctly
  - Easy to understand source codes
  - Easy to debug
  - Easy to modify
- **What is a efficient program (when it executes)**
  - To be fast (minimum time)
  - Require minimum memory space

Why data structures and algorithms

- What do we have to consider to develop a computer (good and efficient) program?



What is data structures and algorithms

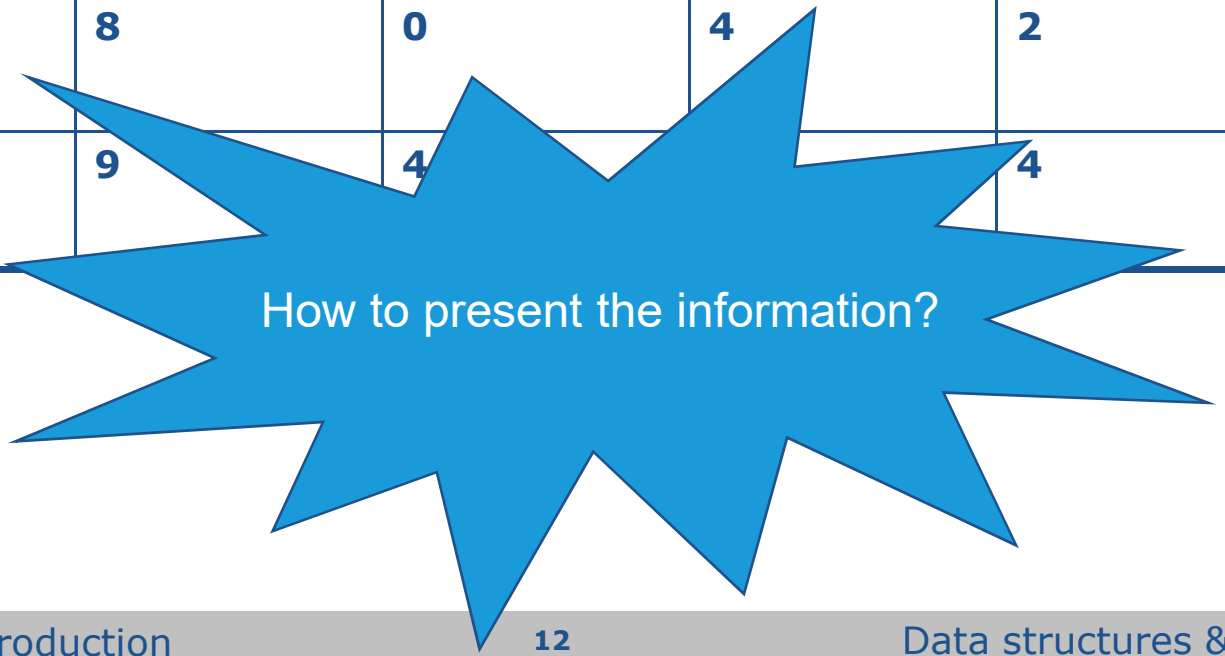
Data structures

- A *data structure* is basically a group of data elements that are put together under one name, and which defines a particular way of storing and organizing data in a computer so that it can be used efficiently

An example

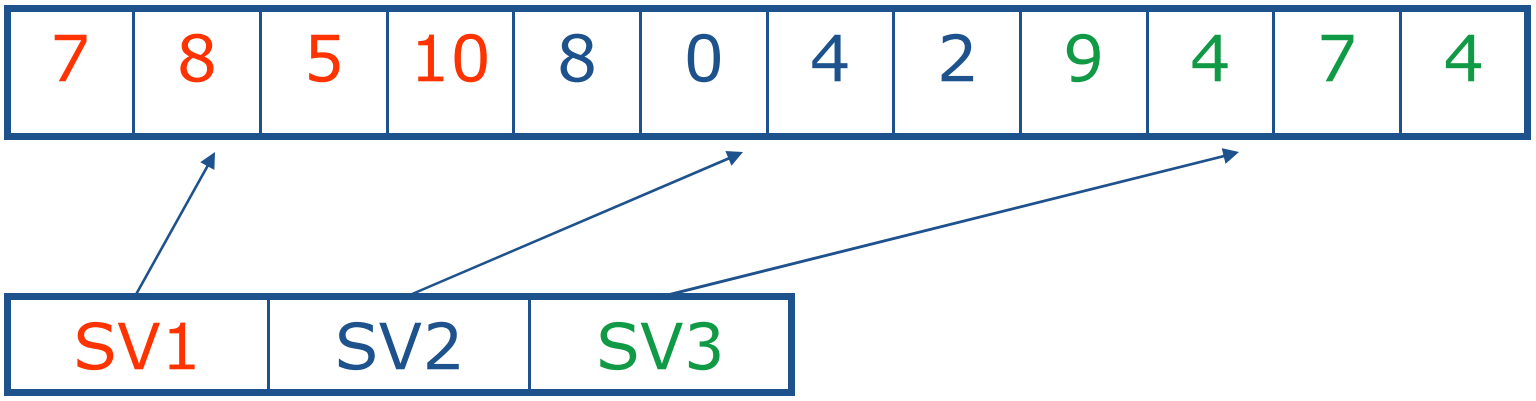
- School management program

Name	Data structures and algorithm	Graph theory	Computer graphic	Database
SV1	7	8	5	10
SV2	8	0	4	2
SV3	9	4		4



An example

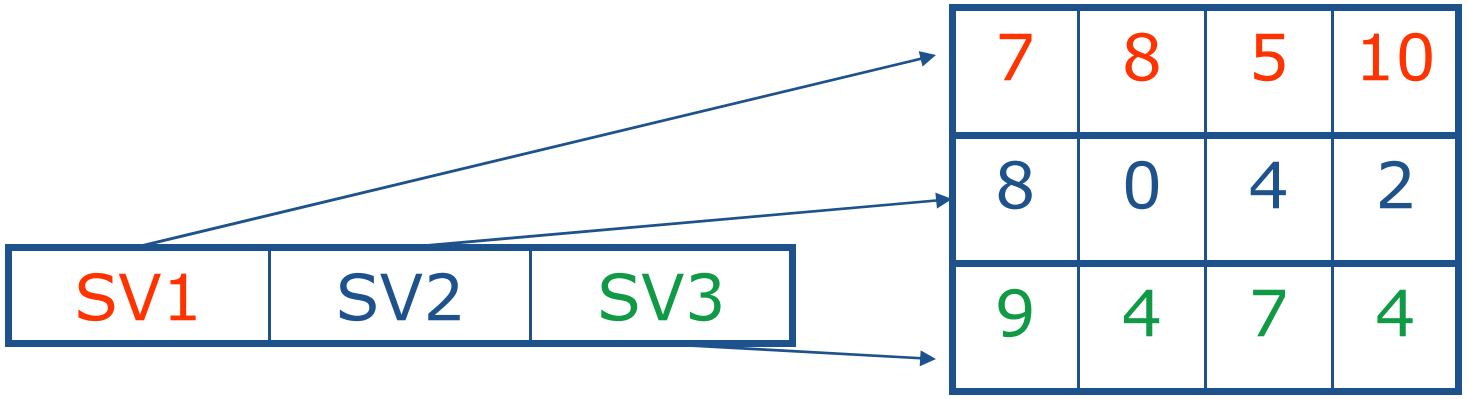
- School management program
  - Using an one-dimensional array:



float diem[12]= {7, 8, 5, 10, 8, 0, 4, 2,9, 4, 7, 4}
char* sv[3] = {“SV1”, “SV2”, “SV3”}

An example

- School management program
  - Using two-dimensional array:




```
float diem[3] [4]= {{7, 8, 5, 10},
                    {8, 0, 4, 2},
                    {9, 4, 7, 4}}
char* sv[3] = {“SV1”, “SV2”, “SV3”}
```

An example

- School management program
  - Using an array of objects (using structs)

Sinh viên
Họ tên
CTDL
LTDT
DHMT
CSDL

```
struct SinhVien
{
        char* HoTen;
        float diem[4];
}

SinhVien sv[3];
```

An example

- School management program
  - Using an array of objects (using class):


```
class SinhVien
{
        private:
                char* HoTen;
                float diem[4];
        public:
                void Nhap();
                void Xuat();
};
SinhVien sv[3];
```

An example

- How to perform the following operations on the previous data?
  - Add a student
  - Remove a student
  - Compute the GPA of a student
  - Sort students in descending of their GPA
  - ….

Outline

I. Why Data Structures and Algorithms?

II. Concepts

III. Algorithm complexity

IV. Excercises

II. Concepts

Data type

- Data type of a variable is the set of values that the variable can take

- For examples:
  - *int* in C language:
    - Value: -32768 to 32767.
    - Operations: + , - , * , / , % .
  - *bool* in Visual C++:
    - Value {TRUE, FALSE}
    - Operations: && (And), || (Or), ! (Not), ^ (Xor).

II. Concepts

Basic data types (primitive data type)

Abstract Data type	Pascal	C++	Java
Integer	interger, word,…	int, long,…	byte, short, int, long
Real	real	float, double	float, double
Boolean	boolean	Int	boolean
character	Char	char	Char

II. Concepts

- **Abstract data types (ADT)**
  - is a mathematical model for data types, as well as the functions that operate on the data
  - without regard to the implementation aspect of data types

II.Concepts

- **Abstract data types (ADT)**
  - For examples
    - Integer
      - A whole number
      - Operations: Addition, abstraction, multiplication, division, modulus
      - How is it implemented in C/Java/C#?

II.Concepts

- **Abstract data types (ADT)**
  - For examples
    - Stacks
      - A list of items with LIFO charecteristic
      - Operations: Pop, Push, Top
      - How is it implemented using a programming languages?

II. Concepts

- **List of data structures are studies in this course**
  - Lists
  - Stacks
  - Queues
  - (Binary) trees
  - Hash Tables

II.Concepts

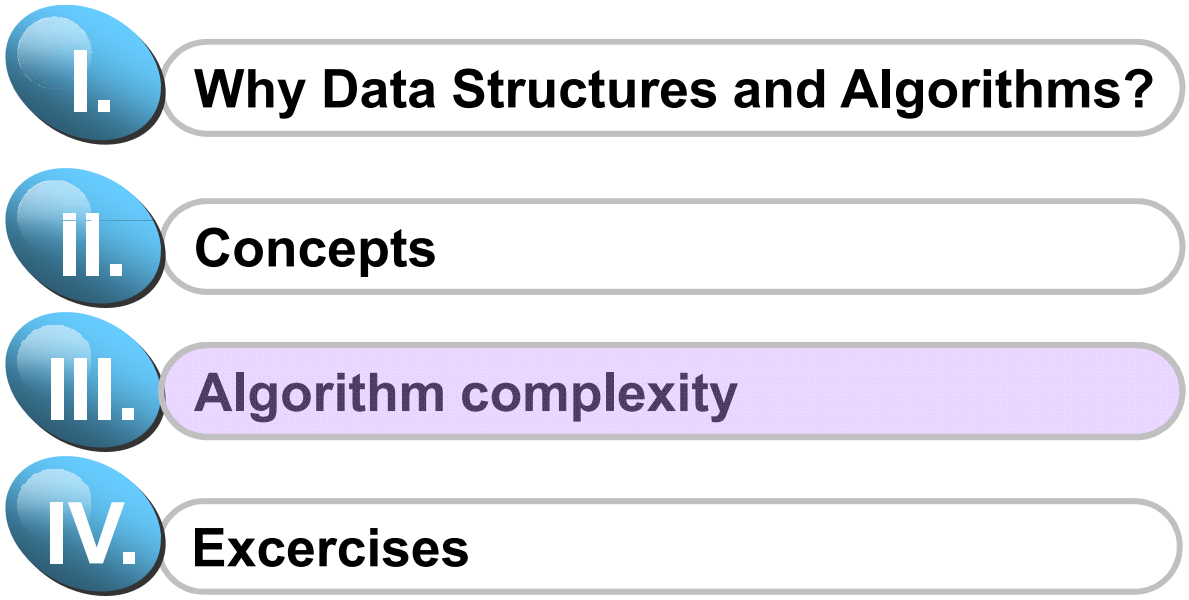
- **Operations on data structures**
  - Traversing
    - to access each data item exactly once
  - Searching
    - to find the location of one or more data items
  - Inserting
    - to add new data items to a collection/list

II.Concepts

Operations on data structures

- Deleting
  - To remove/delete a particular data item from the given collection
- Sorting
  - items can be arranged in some order (ascending / descending)

Outline


I.
Why Data Structures and Algorithms?
II.
Concepts
III.
Algorithm complexity
IV.
Excercises

III.Algorithm complexity

Algorithm

- is a formally defined procedure for performing some calculation

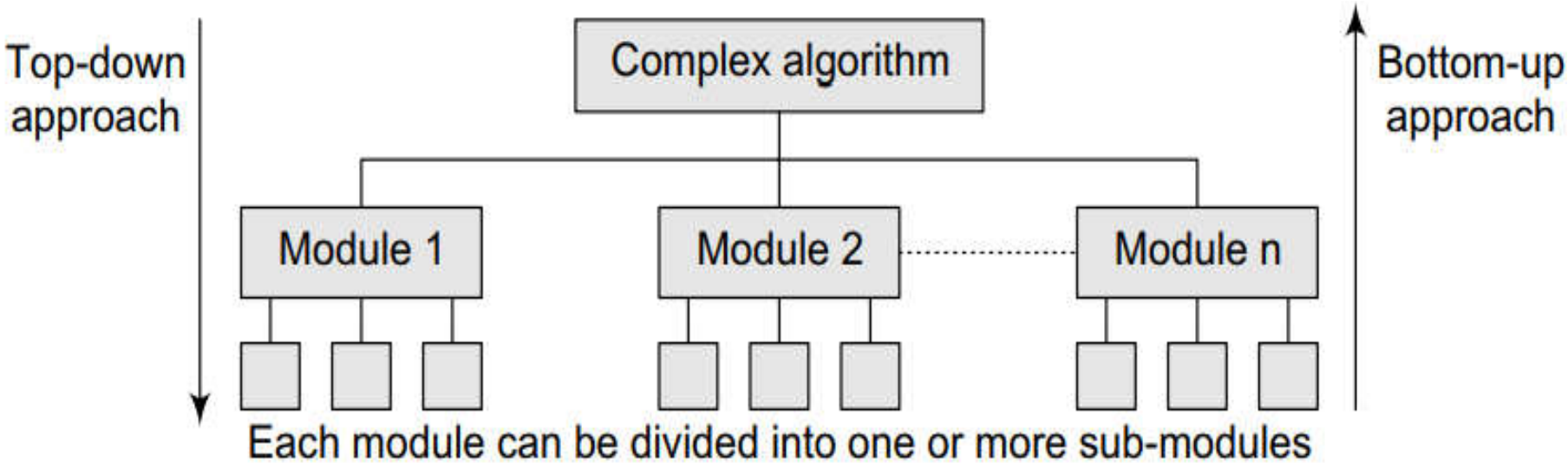


Figure 2.9 Different approaches of designing an algorithm

III.Algorithm complexity

Algorithm

- Top-down approach
  - Easy to write documents of the modules, test cases, implement code, debug
  - Sub-modules are analyzed without considering on their communication
- Bottom-up approach
  - identify what has to be encapsulated within a module
  - to define the module's boundaries as seen from the clients
- Should use both top-down and bottom-up approaches.

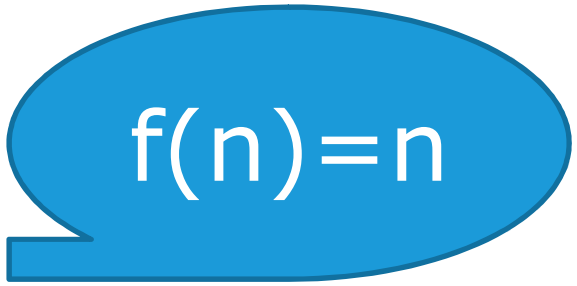
III.Algorithm complexity

- How to compare the performance of two algorithms?

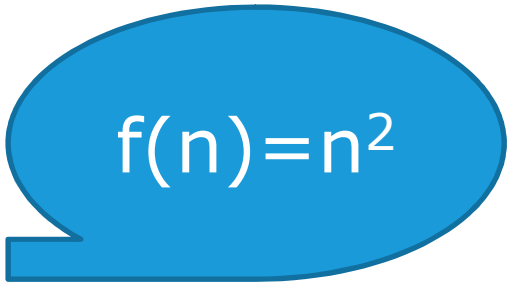
- Some methods
  - Calculate running time (practical aspect), depending on:
    - CPUs, main memory, storages, programming languages, complier/interpreter
  - Calculate the number of instructions (assignments/comparison) (theory aspect)

III.Algorithm complexity

- **The time complexity can be expressed using a function f(n).**
- **n is the input size**
- **For examples**
  - for(i=0;i<100;i++)
    statement block;



  - for(i=0;i<10;i++)
    for(j=0; j<10;j++)
    statement block;



III.Algorithm complexity

- **Categories of algorithms (Using Big O notation)**
  - Constant time: O(1).
  - Linear time: O(n)
  - Logarithmic time: O(log n)
  - Polynomial time: $O(n^k)$, $k>1$
  - Exponential time: 2^n, $n!$, n^n

We consider the worst-case running time of an algorithm which is an upper bound on the running time for any input.

III.Algorithm complexity

❖ Examples

Table 2.2 Number of operations for different functions of n

n	O(1)	O(log n)	O(n)	O(n log n)	$O(n^2)$	$O(n^3)$
1	1	1	1	1	1	1
2	1	1	2	2	4	8
4	1	2	4	8	16	64
8	1	3	8	24	64	512
16	1	4	16	64	256	4096

Outline

I. Why Data Structures and Algorithms?

II. Concepts

III. Algorithm complexity

IV. Excercises

IV.Excercises

1. Write a program to read an array of *n* number. Write **functions** to do the following tasks:

a) Find the minimum number of the array

b) Calculate the sum of the list

c) Find the first negative number in the array

2. Write a program to read and display the information of a student (first name, student ID, GPA) (**two functions**)

3. Write a program to read and display an array of n students (first name, student ID, GPA). Write **functions** to do the following tasks:

+ Find a student with the highest GPA (Grade point average) in the list

+ Sort the list in ascending order of GPA

- **4 bạn lên bảng làm bài**
  - 1: Câu 1a,b
  - 2: Câu 1c
  - 3: Câu 2
  - 4: Câu 3 (dùng lại 2 hàm của câu 2)