

# Graphs and Complexity for IoT

## Master IoT

**Mourad Hakem**

Contact : [mourad.hakem@univ-fcomte.fr](mailto:mourad.hakem@univ-fcomte.fr)  
IUT Belfort - Université de Franche-Comté

December 1, 2025

- ▶ Introduction to complexity
- ▶ Experimental vs. theoretical study
- ▶ Complexity computation
- ▶ Asymptotic complexity
- ▶ Landau notations ( $\mathcal{O}$ ,  $\Omega$ ,  $\Theta$ )
- ▶ Classes of complexity  $P$ ,  $NP$ ,  $NP$ -Completeness
- ▶ Famous hard problems
- ▶ Approximation algorithms
- ▶ Graph coloring, vertex and edge covering, domination in graphs, maximum clique, set covering, TSP problem

## Lecture I

- ▶  $P$  : a problem
- ▶  $M$ : a method to solve  $P$
- ▶ **Algorithm** : description of  $M$  in an algorithmic language
- ▶ Key questions:
  1. does the algorithm give an answer? → **termination**
  2. Is this the correct answer? → **correctness**
  3. answer in a reasonable time? → **complexity**

**Complexity:** evaluating the **efficiency** of algorithms independently of the environment (NB of **elementary operations** vs. **data size**)

## Data size:

- ▶ size of an array or list
- ▶ size of numbers
- ▶ the number of vertices or edges in a graph

## Elementary operations:

- ▶ assignment
- ▶ comparison
- ▶ read/write
- ▶ addition, multiplication, etc.
- ▶ Boolean test

Elementary operations cost one unit.

## Objectives:

- ▶ estimate the cost without running the algorithm
- ▶ classify problems according to their difficulty/efficiency
- ▶ compare algorithms solving a given problem to make an informed choice without having to implement them

## Notations :

$n$  : input size,

$T(n)$  : number of elementary operations

**Example 1** : search for a value in an array

- ▶  $n$  : input size  $\rightarrow$  array size (number of elements)
- ▶ elementary operations  $\rightarrow$  comparisons
- ▶  $T(n)$  :  $n$  comparisons

## which approach?

- ▶ best case scenario
- ▶ average
- ▶ worst case

→ we want to be sure that the algorithm will never take longer than estimated;

### Example 2: primality test:

- ▶ Algo-1 will search for a candidate from 2 to  $n - 1$ .
- ▶ Algo-2 will search for a candidate from 2 to  $n/2$ .  
divisors of  $n$  other than 1 and  $n$  lie in  $[2, n/2]$
- ▶ Algo-3 will search for a candidate between 2 and  $\sqrt{n}$   
if  $n$  is non-prime, then it has a divisor  $d \leq \sqrt{n}$ .

$$T_{A_1}(n) = (n - 1) - 2 + 1 = n - 2$$

$$T_{A_2}(n) = n/2 - 2 + 1 = \frac{n-2}{2}$$

$$T_{A_3}(n) = \sqrt{n} - 2 + 1 = \sqrt{n} - 1$$

## 1. Experimental study

- ▶ implement the algorithm in Java (or other)
- ▶ run the program with inputs of different sizes
- ▶ measure the execution time, using a method like  
`System.currentTimeMillis()`
- ▶ draw the graph of the obtained measures

## 1.1 Limitation of the experimental study

- ▶ we need to implement the algorithm  
we want to know the time complexity of an algorithm before implementing it, in order to save time
- ▶ to compare 2 different algorithms for the same problem, we need to use the same environment
- ▶ the results found are not representative of all inputs

## Experimental vs. theoretical analysis

- ▶ a fundamental operation in  $1\mu s$
- ▶ execution times for different  $n$  values

$T(n)$	$n = 10$	$n = 100$	$n = 1000$	$n = 10000$
$n$	$10\mu s$	$100\mu s$	$1ms$	$10ms$
$400n$	$4ms$	$40ms$	$0.4s$	$4s$
$2n^2$	$200\mu s$	$20ms$	$2s$	$3.3m$
$n^4$	$10ms$	$100s$	$11.5 \text{ days}$	$317 \text{ years}$
$2^n$	$1ms$	$4 \times 10^6 \text{ years}$	$3.4 \times 10^{287} \text{ years}$	.....

Size of data that can be processed in a given time?

$T(n)$	1second	1minute	1hour
$n$	$1 \times 10^6$	$6 \times 10^7$	$3.6 \times 10^9$
$400n$	2500	150000	$9 \times 10^6$
$2n^2$	707	5477	42426
$n^4$	31	88	244
$2^n$	19	25	31

## Machine power ?

What size of data can be processed when the machines are 100 and 1000 times faster?

### ► Example 1

– Today  $T(n) = n^2$

– Tomorrow:  $T'(n') = \frac{n'^2}{100} \longrightarrow n' = 10n$

### ► Example 2

– Today:  $T(n) = 2^n$

– Tomorrow :  $T'(n') = \frac{2^{n'}}{100} \longrightarrow n' = n + 6.67$

## 2. Theoretical analysis

- ▶ is based on the pseudo-code of the algorithm and not the implementation function of  $n$ , the size of the input
- ▶ considers all inputs
- ▶ independent of the environment

## How to calculate the complexity of an algorithm?

- ▶ Calculate the complexity of each part of the algorithm;
- ▶ Combine these complexities;
- ▶ Simplify the result (see below);

Sequences:

Treatment 1:  $T_1(n)$

Treatment 2:  $T_2(n)$

$$T(n) = T_1(n) + T_2(n)$$

Selections:

```
If (cond) Then  
    treatment 1: T1(n)  
Else  
    treatment 2: T2(n)  
EIF
```

$$T(n) = \max(T_1(n), T_2(n))$$

Loops:

```
While (cond) Do  
    treatment: Ti(n)  (cost of the i-th iteration)  
EndWhile
```

$$T(n) = \sum_{i=1}^k T_i(n)$$

## Example (factorial of n)

function factorial(n)	
fact = 1	initialisation : 1
i = 2	initialisation : 1
While (i <= n) Do	itérations : n-1
fact = fact * i	mult + assign : 2
i = i + 1	incr + assign : 2
EndWhile	
return fact	return : 1

Complexity :  $1 + 1 + (n - 1) \times 5 + 1 + 1 = 5n - 1$

## Example ( $X^n$ )

function Power(X, n)	
res = 1	initialisation : 1
p = 0	initialisation : 1
While (p <> n) Do	itérations : n
res = res * p	mult + assign : 2
p = p + 1	incr + assign : 2
EndWhile	
return res	return : 1

Complexity:  $1 + 1 + n \times (2 + 2 + 1) + 1 + 1 = 5n + 4$

## Reminder :

### ► Arithmetic sequences

$$\text{Sum} = \text{number of terms} \times \frac{\text{first term} + \text{last term}}{2}$$

Example: sequence with first term 2 and common difference 3:  
2 5 8 11 14 17 etc.

$$2 + 5 + 8 + 11 + 14 + 17 = 6 \times \frac{2+17}{2} = 57.$$

### ► Geometric sequences

$$\text{Sum} = \text{first term} \times \frac{\mathcal{R}^{\text{number of terms}} - 1}{\mathcal{R} - 1}, \mathcal{R} : \text{common ratio for the sequence}$$

Example: sequence with first term 2 and common ratio 3: 2 6 18 54 162  
etc.

$$2 + 6 + 18 + 54 + 162 = 2 \times \frac{3^5 - 1}{3 - 1} = 242$$

## Example (Data cleaning)

- ▶ version 1 (shifting):

- ▶ best case: no zero **complexity** =  $n$

- ▶ worst case: array filled entirely with zeros  
at each position  $i$ ,  $n - i - 1$  shifts are made

**complexity:**  $\sum_{i=0}^{n-1} n - i - 1 = (n-1) + (n-2) + \dots + 1 = \frac{n(n-1)}{2}$

- ▶ version 2 (without shifting): 1 traversal for best and worst case.

**complexity** =  $n$