# Algorithms Course
# Programming and Algorithms Workshop
# RUPP, 5-16 September 2022

# Références


- Cormen, Leiserson, Rivest, Stein, “Introduction to algorithms”.
- Aho, Hopcroft, Ullman,“Data structures and algorithms”.

# algorithms

An *algorithm* is a procedure described in terms of elementary steps, allowing to determine the solution of a problem starting from any of its possible data (said *instance of the problem*).

We say that the algorithm *solves* the problem.

For any given instance , an algorithm must always stop after a finite number of operations providing always the right solution/answer to the problem.

Algorithm $\neq$ Program

# Examples of problems, instances

- Given a number, determine if it is a power of 2.

  an instance: an integer number; expected answer: yes or no.

- Given an array of integers, compute the sum of its values.

  an instance: an array of integers; expected answer: an integer.

- Given an array of integers, compute the array obtained by rearranging its values in increasing order.

  an instance: an array of integers; expected answer: an array of integers.

- Given a text and a pattern, determine if the text contains the pattern.

  an instance: a couple of sequences of characters (strings);

  expected answer: a boolean.

# Correction, efficiency

There are two fundamental questions to ask :

- Verify/prove that the algorithm is correct.

  En general, difficult. There are some methods in simple cases (loops).

- Mesure the efficiency of the algorithm.

  In particular, compare the “performances” of two algorithms solving the same problem.

but how to mesure the performances of an algorithm?

# Notion of Complexity

It is reasonable to consider two aspects:

- the quantity of memory necessary for executing the algorithm (or rather the program that implements it) on a machine (space complexity).
- the time (of computation) necessary to obtain the answer/solution after entering the data (time complexity).

How to mesure time?

# How to mesure time

Use the second and its fractions ? $\longrightarrow$ bad idea !

As unity of mesure of time we choose the elementary operation.

## Example

- Assignments
- Arithmetic operations
- Comparaisons (of scalar type data )
- Inputs/outputs (of scalar type data )
- Functions calls
- Functions returns ...

# How to mesure space (memory)

Memory is classically mesured in bytes (1 byte = 8 bits).
Typically :

- A character is stored using 1 byte.
- An integer number is stored using 4 bytes.
- An array of integers of length $n$ hence needs $4n$ bytes of memory to be stored.
- A long integer number or a real (float) number in double precision is stored using 8 bytes.

# Complexity Function (for time complexity)

**Remark**

*the number of operations depends on the size of the data.*

(Imagine the computation of the sum of all the values of an array)

Note : sometimes this number may depend also on properties of the data other than the size

But if this number depends only on the size, then this number is the same for all the data having size $n$.

In this case we can define a function $f$ such that:

$f(n)$ = the number of elementary operations carried out by the algorithm on any data of size $n$.

(time complexity function of the algorithm)

# An example

**Example**

```
SUM (T[0..n-1]:array of integers):integer;

   i, res : integer;

   res = 0;


   for i from 0 to n-1 do

          res= res + T[i];

   endfor

   return(res);
```

# An example : Computation of the complexity function

**Example**

```
SUM (T[0..n-1]:array of integers):integer;      El. op.

   i, res : integer;

   res = 0;                                     1


   for i from 0 to n-1 do                       n times...

          res= res + T[i]:                          2 (1 add and 1 assign)

   endfor

   return(res);                                 1
```

= 2n+2 total

$f(n) = 2n + 2$ (linear complexity)

# An example : Computation of the complexity function

**Example**

```
SUM (T[0..n-1]:array of integers):integer;

   i, res : integer;

   res = 0;

   for i from 0 to n-1 do

          res= res + T[i]:

   endfor

   return(res);
```

Forgotten something ???

← maybe here ?

1 incrementation and 1 comparison for each iteration of the loop

$\longrightarrow f(n) = 4n + 2$ (remains linear)

# An example : space complexity

Rule : we never count the space used for data and result.

**Example**

```
SUM (T[0..n-1]:array of integers):integer;

   i, res : integer;

   res = 0;

   for i from 0 to n-1 do

          res= res + T[i]:

   endfor

   return(res);
```

1 integer variable (i)

4 bytes (regardless of the size n of the array)

Space complexity is constant ($g(n) = 4$)

# Another example

**Example**

```
MIN (T[0..n-1]:array of integers):integer;

   i, min : integer;

   min = T[0];
   for i from 1 to n-1 do

      if (T[i] < min) then
           min = T[i];
   endfor

   return(min);
```

# Another example, computation of time complexity

**Example**

```
Min (T[0..n-1]:array of int):int;

   i, min : integer;

   min = T[0];
   for i from 1 to n-1 do
      if (T[i] < min) then
           min = T[i];
   endfor
   return(min);
```

1 assignment
1 incr. and 1 comparison ($n-1$ times)
1 comparison ($n-1$ times)
← **this assignment is not always carried out!**

In this case the number of el. op. is not the same for all arrays of size $n$.

We analyse then the complexity in the best and in the worst case.

# Another example : worst case

Q: When does it occur?

R: When the array is decreasing (non increasing)

**Example**

```
Min (T[0..n-1]:array of int):int;

   i, min : integer;

   min = T[0];
   for i from 1 to n-1 do
      if (T[i] < min) then
           min = T[i];
   endfor
   return(min);
```

1 assignment
1 incr. and 1 comparison ($n-1$ times)
1 comparison ($n-1$ times)
**1 assignment ($n-1$ times)**

1 return

$1+4(n-1)+1=4n-2$ el. op.

# Another example : best case

Q: When does it occur?

R: When the min is in first position

**Example**

```
Min (T[0..n-1]:array of int):int;

   i, min : integer;

   min = T[0];
   for i from 1 to n-1 do
      if (T[i] < min) then
           min = T[i];
   endfor
   return(min);
```

1 assignment

1 incr. and 1 comparison ($n-1$ times)

1 comparison ($n-1$ times)

**0 affectations**

1 return

$1+3(n-1)+1 = 3n-1$ el. op.

# Another example : average (random) case ?

Best case : $3n - 1$

Worst case : $4n - 2$

For all other cases (of arrays of size $n$) the number of el. op. is between these two values.

Can we compute the complexity in the average case (average complexity)? $\longrightarrow$ difficult!

If there are $m$ possible cases, for each case $i$ you need to know its complexity $C_i$ and its probability to occur $p_i$ and compute:

$$\sum_{i=1}^{m} p_i C_i$$

For this algo, we know at least that the complexity is linear in all cases (because it is linear in the best and in the worst).

# Compare two functions

We have two algorithms solving the same problem. One has complexity function $f(n)$, the other $g(n)$.

How to compare the two functions f and g to determine which of the two algorithms is more efficient?

We need to define an order (dominance order) on functions.

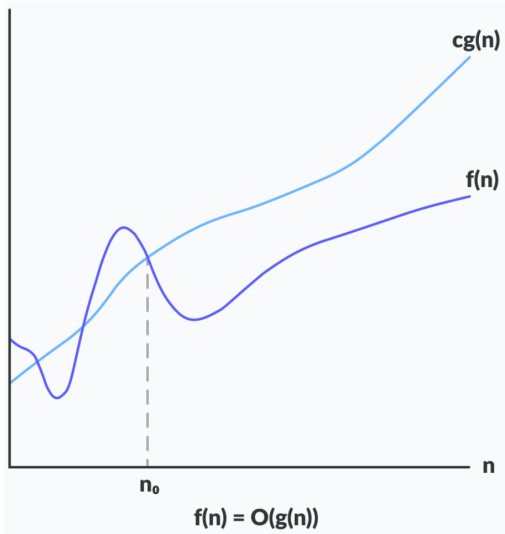
# the notation big-O

## Definition

*Let $f, g : \mathbb{N} \to \mathbb{R}^+$. We say that $f \in O(g)$ ("f is O of g") if there exist a positive integer $n_0$ and a positive (real) constant $c$ such that :*

$$f(n) \leq cg(n) \text{ for all } n \geq n_0$$

Maybe clearer with a picture:




the growth rate of $f$ is smaller than or equal to the growth rate of $g$, we say that $f$ is dominated by $g$ (or that $g$ dominates $f$).

$O(g)$ is the set of all functions dominated by $g$.

# the notation big-O, examples

**Example**

Let $f = n$ and $g = 2n + 4$.
Is $f \in O(g)$?
In this case, for all integers $n$ we have $n < 2n + 4$, so if we choose $n_0 = 0, c = 1$ we have:

$$f(n) \leq cg(n) \text{ for all } n \geq n_0.$$

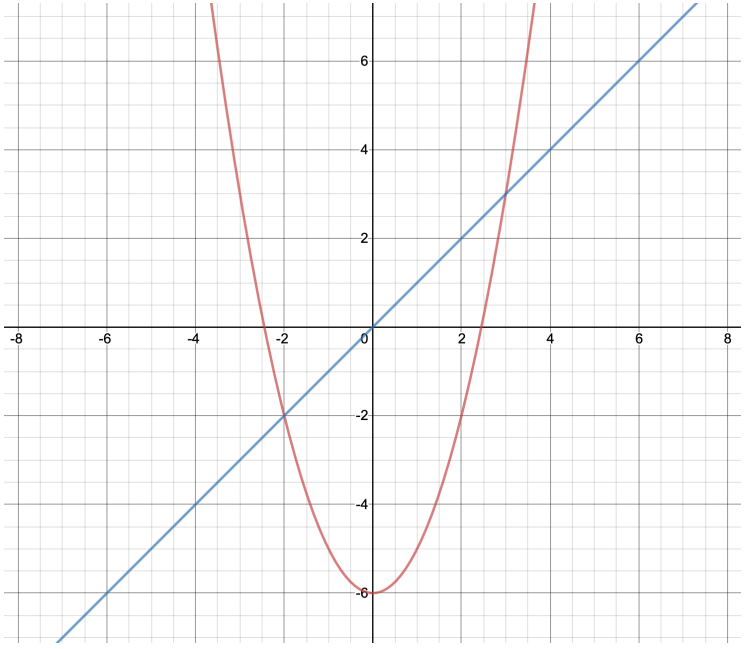
**Example**

Let $f = n$ and $g = n^2$.
Is $f \in O(g)$?
Since for all $n \geq 0$ we have $n \leq n^2$, if we choose $n_0 = 0, c = 1$ we have:

$$f(n) \leq cg(n) \text{ for all } n \geq n_0.$$

# the notation big-O, examples

Let $f = n$ and $g = n^2 - 6$.
Is $f \in O(g)$?




For computing $n_0$ we can compute the point of intersection having positive abscissa.
From the equation $n^2 - 6 = n$ we obtain $n = -2$ or $n = 3$.
Since for all $n \geq 3$ we have $f(n) \leq g(n)$, if we choose $n_0 = 3, c = 1$, we have that:

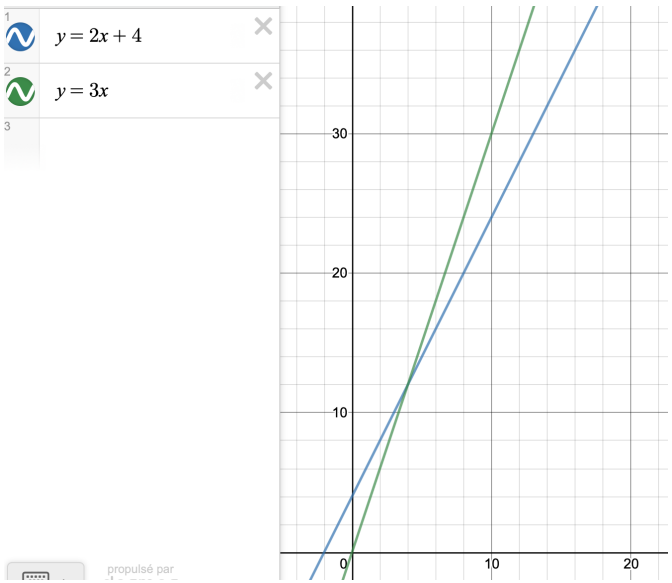
$$f(n) \leq g(n) \text{ for all } n \geq n_0.$$

# the notation big-O, examples

Let $f = n$ and $g = 2n + 4$.
Is $g \in O(f)$?

There exist $n_0 \in \mathbb{N}^+$ and $c \in \mathbb{R}^+$ such that $g(n) \leq cf(n)$ for all $n \geq n_0$?

If we take $c = 3$?




the two lines $y = 3x$ and $y = 2x + 4$ cross at the point of abscissa $n_0 = 4$ and on the right of this abscissa, $3n$ is always higher than $2n + 4$.

So $g(n) \leq 3f(n)$ for all $n \geq 4$, that is, $g \in O(f)$

# the notation Big-$\Theta$ (big-theta)

Two functions may dominate one another ($f \in O(g)$ and $g \in O(f)$).

**Definition**

*We say that $f \in \Theta(g)$ if $f \in O(g)$ and $g \in O(f)$.*

Formally : $f \in \Theta(g)$ if there exist a positive integer $n_0$ and two positive constants $c_1$ and $c_2$ such that :

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0$$

**Remark**

*$f \in \Theta(g)$ iff $g \in \Theta(f)$ (this is false in general for O).*

$\Theta(g) = \{$all functions having the same growth rate as $g\}$.

# Classes $\Theta$

We already saw that $2n + 4 \in \Theta(n)$, the class of *linear growth*, very good.

More generally, every polynomial of degree $k$ is in the class $\Theta(n^k)$. (polynomial growth, acceptable if $k$ small)

If two polynomials have different degrees, the one with larger degree always strictly dominates the other. This is also true for non integer exponents: if $\alpha > \beta \in \mathbb{R}^+$ then $n^\beta \in O(n^\alpha)$ (and $n^\alpha \notin O(n^\beta)$

All constants are in the class $\Theta(1)$.

$\log n$ is dominated (strictly) by any polynomial, and in fact is dominated (strictly) by any function $n^\alpha$ with $\alpha > 0$. (croissance logarithmic growth, excellent)

$2^n$ dominates all polynomials, and in fact dominates any function $n^\alpha$ with $\alpha > 0$. (exponential growth, horrible)

the class $\Theta(n \cdot \log(n))$ contains functions whose growth rate is between linear and quadratic.

# Les algorithmes efficaces… et les autres.

Avec un ordinateur exécutant $10^9$ instructions pas secondes…

| Fonc.\ n | 20 | 40 | 60 | 100 | 300 |
|---|---|---|---|---|---|
| $n^2$ | 1/2500 milliseconde | 1/625 milliseconde | 1/278 milliseconde | 1/100 milliseconde | 1/11 milliseconde |
| $n^5$ | 1/300 seconde | 1/10 seconde | 78/100 seconde | 10 secondes | 40,5 minutes |
| $2^n$ | 1/1000 seconde | 18,3 minutes | 36,5 année | $400.10^9$ siècles | (72c) siècles |
| $n^n$ | $3,3.10^9$ années | (46c) siècles | (89c) siècles | (182c) siècles | (725c) siècles |

On situe le big-bang à environ $13,8.10^9$ années !

Note : the notation (X) indicates a number made of X digits en base 10. So for instance “(72) siècles = (72) centuries" is not 72 centuries, but a number of centuries written with 72 digits.