



Data Structure & Algorithms

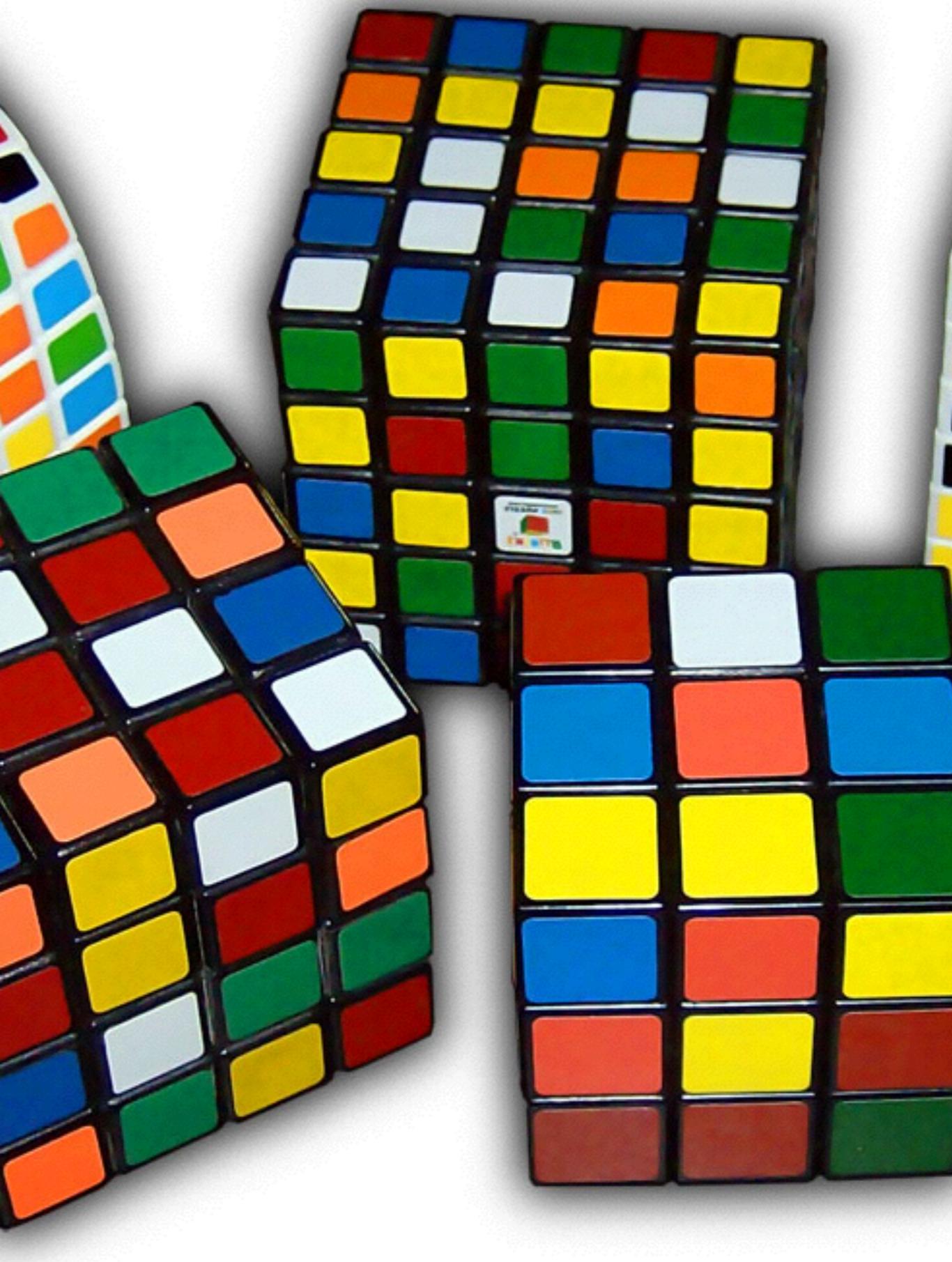


Slide: Matthieu Jimenez
Thème: Sébastien Mosser

Jimenez Matthieu
TD #2, 28.09.2015

Déroulement des TDs

- 21/09
- 28/09
- 12/10
- 26/10
- 9/11
- 23/11
- 7/12



A new foe has appeared!

WARNING
CHALLENGER
APPROACHING

**La Séance
Prochaine**



Interro de TD

Nintendo(c)

Programme du Jour:

Calcul de Complexité

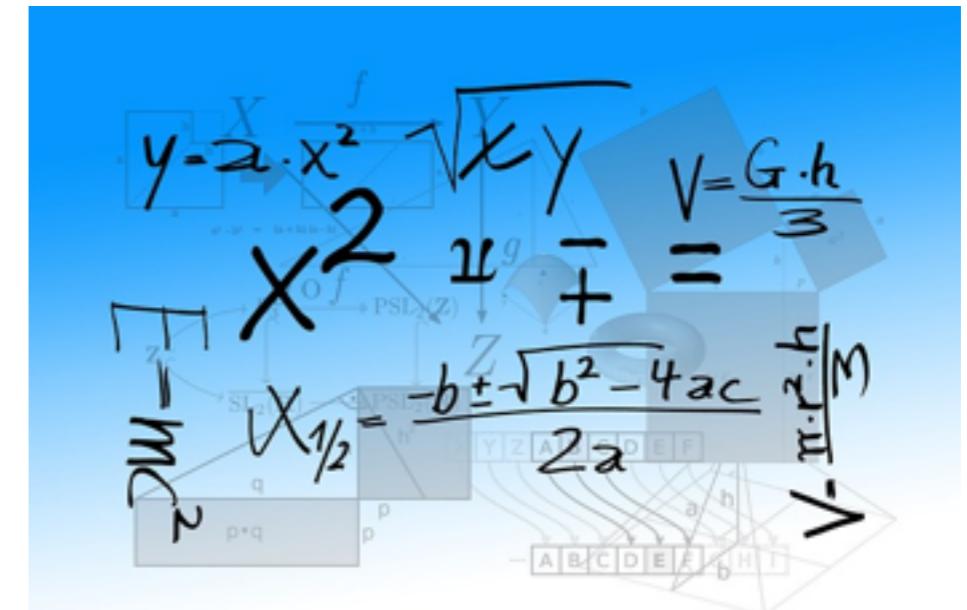
Souvenez Vous ...



On avait exprimé **la complexité** d'un **algorithme** sous forme de **fonction**
(e.g: $3cn + 10c$)

Le **problème** des **fonctions** ...

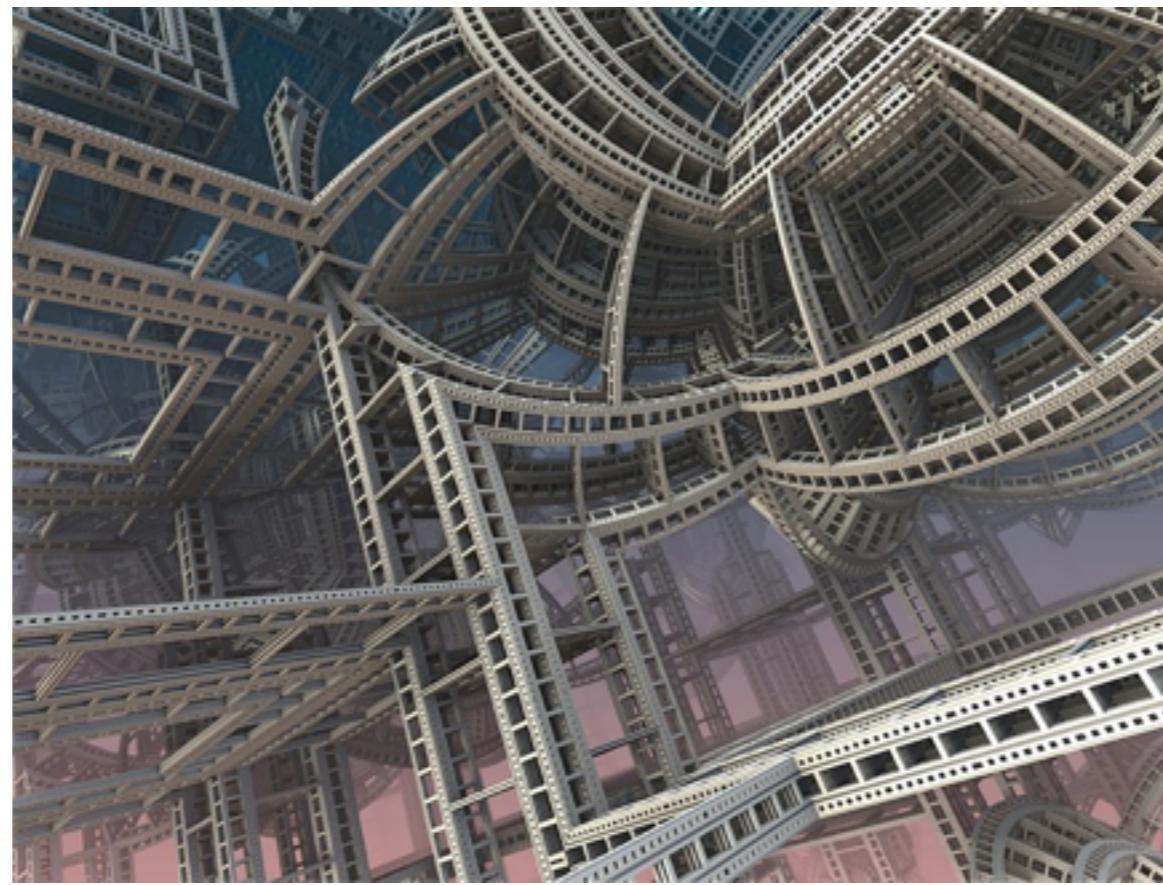
C'est qu'on ne se les **représente** pas
facilement !!!



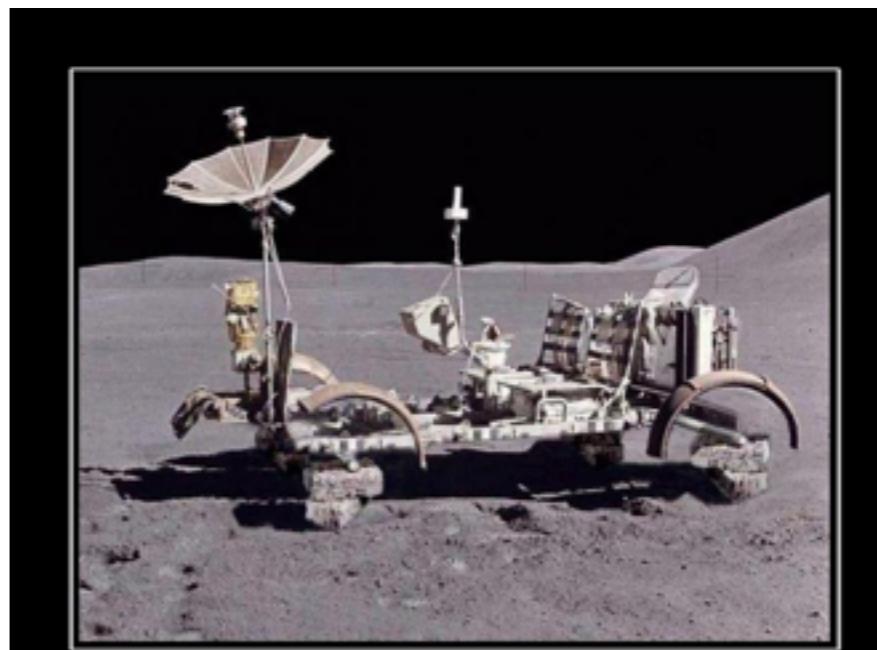
Exemple?

Si je vous dis qu'un **algorithme** est de
complexité $389n+10$ et un autre
 $n^2/389$...

Et que je vous **demande lequel** est le
plus complexe?



Réponse ...



Houston, we have a problem

$n^2/389$... évidemment



Mindblown(c)

Comment faire ?

On va chercher à **trouver** une **borne** à la
fonction de complexité

et ainsi exprimer la **complexité** sous
forme d'un **ordre de grandeur**

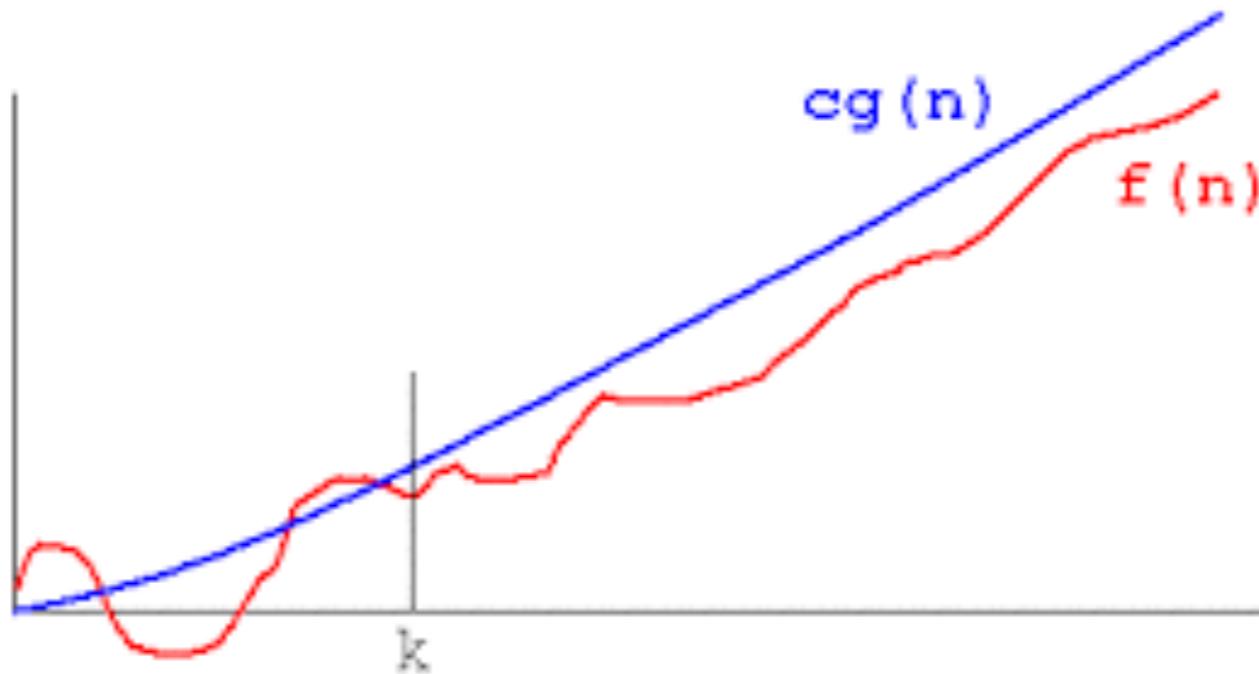
fonction de la **taille de l'input**



Data Structure & Algorithms TD#2

COMPLEXITÉ
Big-0

L'idée



Il s'agit juste de trouver une **fonction** qui **borne supérieurement** votre fonction !

...

Mais **uniquement à partir** d'un **certain point** !

(Pas la peine que ce soit le cas dès le début)

Et

Mathématiquement,

ça donne ...

Notation Big O

La **notation** est **$O(f(n)) = g(n)$**
et **se vérifie** avec:

$$O(f(n)) = \{g(n) : \exists n_0, c > 0 \text{ tels que } g(n) \leq cf(n) \text{ pour } n \geq n_0\}$$

Notation Big O

L'idée est donc de choisir un c cohérent puis trouver un n à partir duquel l'équation est vérifié

Trouver C

Trouver une **constante c** se fait **assez naturellement**, très souvent on choisit **la somme des coefficients** de $g(n)$

ex: pour démontrer $3n + 10 = O(n)$
on prendra $c = 13$

Trouver n

Trouver le **premier n** pour lequel l'équation
est désormais valide
nécessite un petit calcul

Trouver n

Dans l'ex: $3n + 10 = O(n)$
si on prend $c=13$

cela revient à résoudre
 $3n+10 \leq 13n$
 $10 \leq 10n$
soit $n \geq 1$



Pas **si compliqué** que ça
finalement non?

Exercice 1



Exercice 1

Calculer

$$15n+12=O(n)$$

$$17n^2+3n+4=O(n^2)$$

$$25n^3+20\log(n)=O(n^3)$$

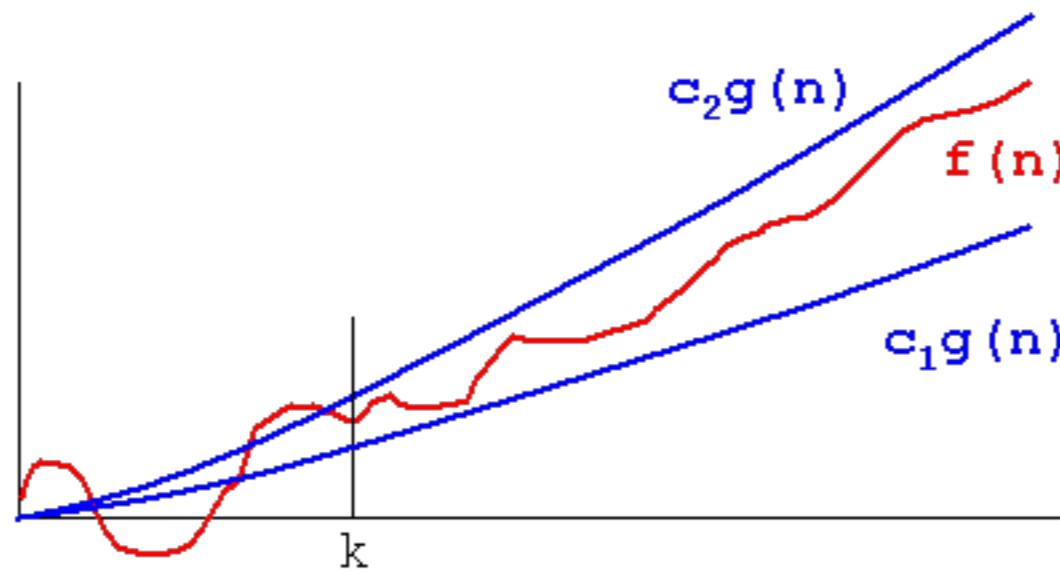


Matthieu Jimenez(c)

Data Structure & Algorithms TD#2

COMPLEXITÉ
GRAND OMEGA & GRAND THÉTA

L'idée: Grand Oméga



Il s'agit juste de trouver une **fonction** qui **borne inférieurement** (c_1) **votre fonction** !
Là où Big O bornait supérieurement (c_2)...

Notation Big Ω

La **notation** est $\Omega(f(n)) = g(n)$
et se **vérifie** avec:

$$\Omega(f(n)) = \{g(n) : \exists n_0, c > 0 \text{ tels que } g(n) \geq cf(n) \text{ pour } n \geq n_0\}$$

Notation Big Ω

L'idée est donc la même **choisir** un **c**
cohérent puis
trouver un **n**
à partir duquel l'équation est vérifié

Trouver C

Trouver une **constante c** se fait assez
naturellement,
essayez de **choisir** le **coefficients de la plus
haute puissance de g(n)**

ex: pour démontrer $3n + 10 = O(n)$
on prendra $c = 3$

**Et pour Grand Thêta,
ça donne ...**

Big Θ

Pour **prouver**
 $\Theta(f(n))=g(n)$, il suffit juste de **montrer** ...

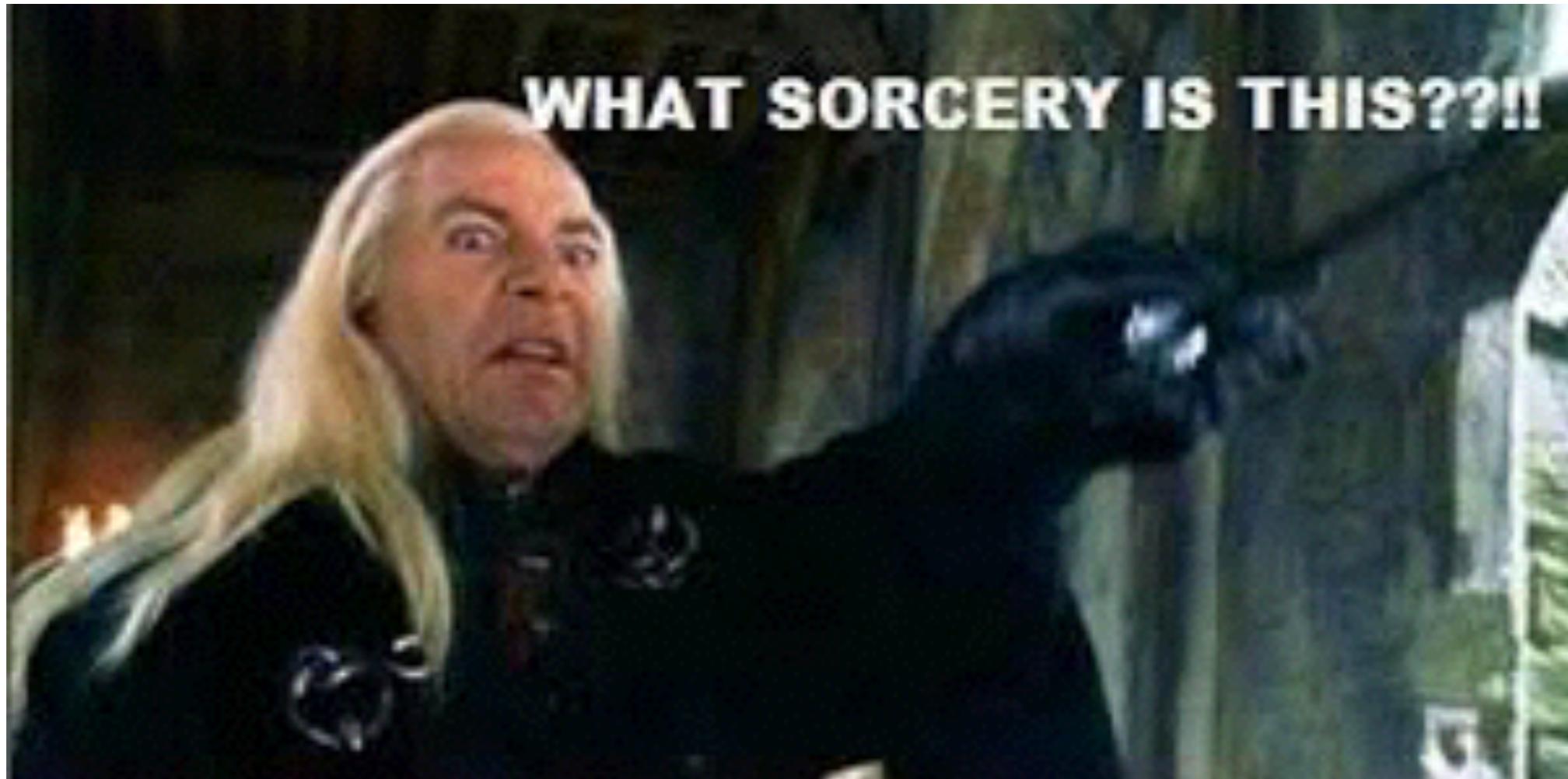
Big Θ

$$\Omega(f(n)) = g(n)$$

et ...

Big Θ

$$O(f(n)) = g(n)$$



Jusque là c'est pas **sorcier**, non?

Exercice 2



Exercice 2

Calculer

$$2n^3 + 10n + 17 = \Theta(n^3)$$

$$n/3 + 15 = \Theta(n)$$

Maintenant ...

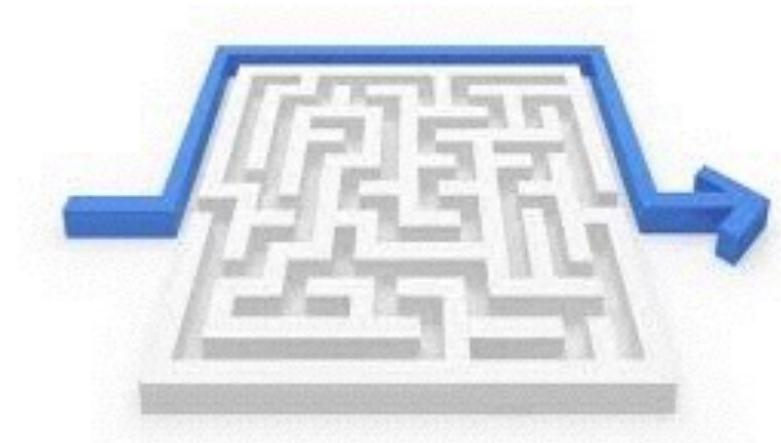


Et bien, **l'idée** est de **déterminer l'ordre de grandeur** de la **complexité** de vos **algorithmes...**



On va donc chercher le **Big Θ** des fonctions
que vous avez trouvez en étudiant vos
algorithmes

Par **simplification**, on notera ce **Big Θ**
comme **Big 0**



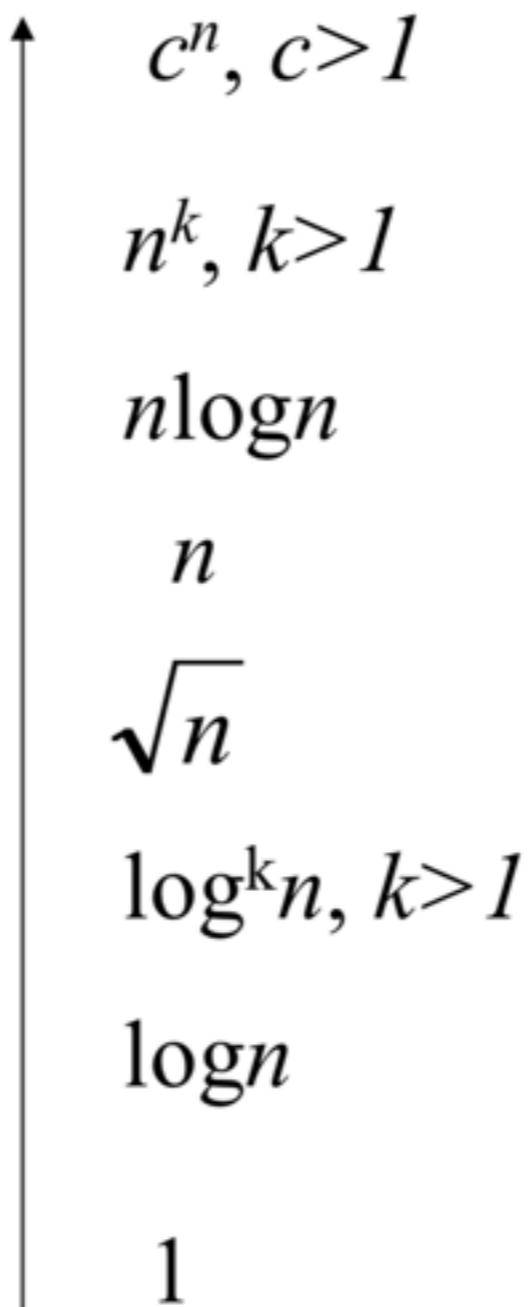
En effet, on prend le **Big 0 le plus petit**
possible donc équivalent à **Big Θ**

Quel $f(n)$ choisir?

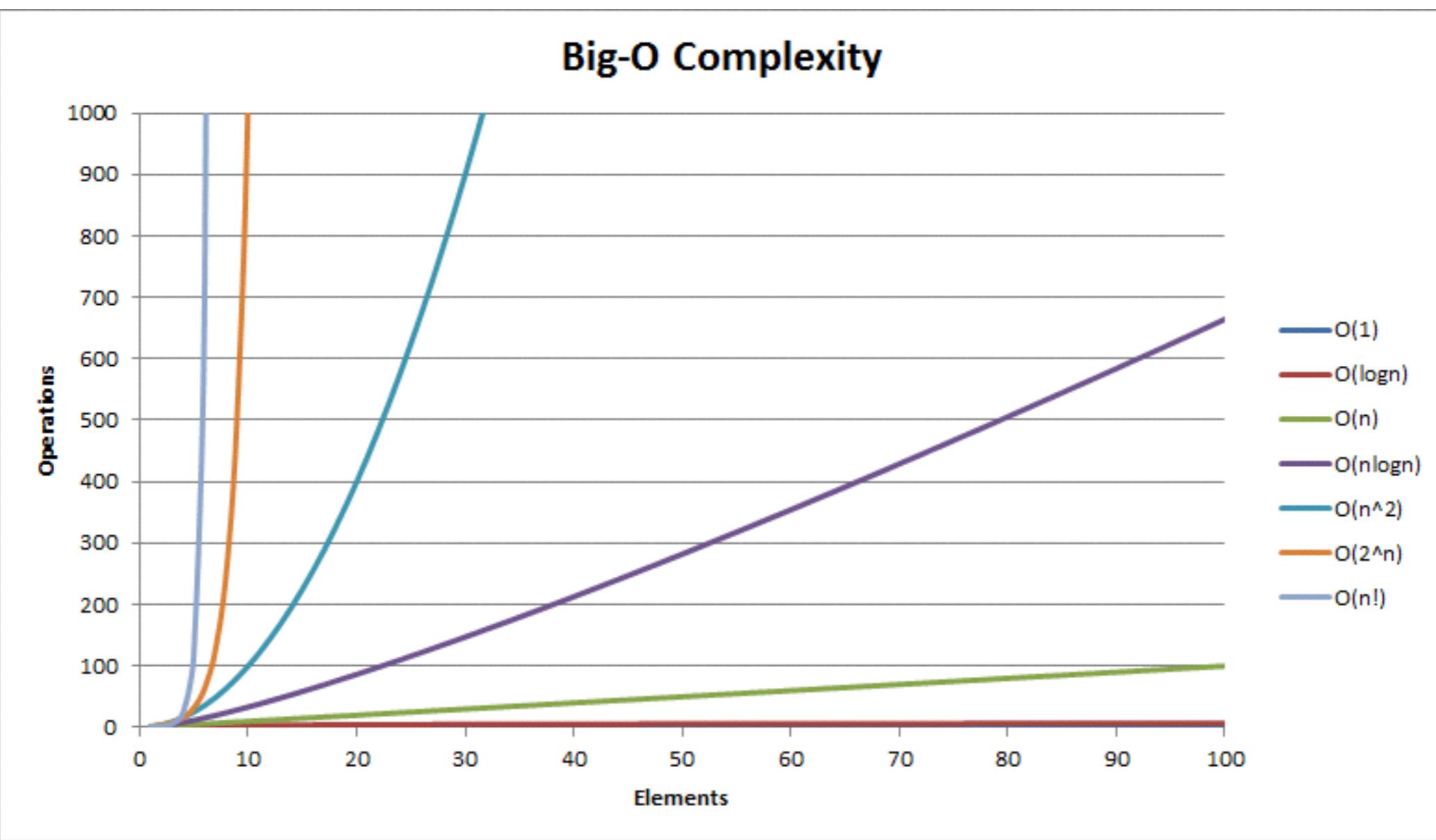


On cherche ici un ordre de grandeur donc hors de question d'avoir un coefficient ou plusieurs degré voir des constantes ...

On va donc appliquer des opérations mathématiques sur la taille de l'input ...



Et Après?



On peut dès lors se **représenter** comment
évoluera l'algorithme en **fonction de la taille**
de son input ...

Or c'était **notre problème**



Ainsi :

$$389n+10 = O(n)$$

$$n^2/389 = O(n^2)$$

Or:

$$n < n^2$$



On a donc bien

$n^2/389$

le **plus complexe !**



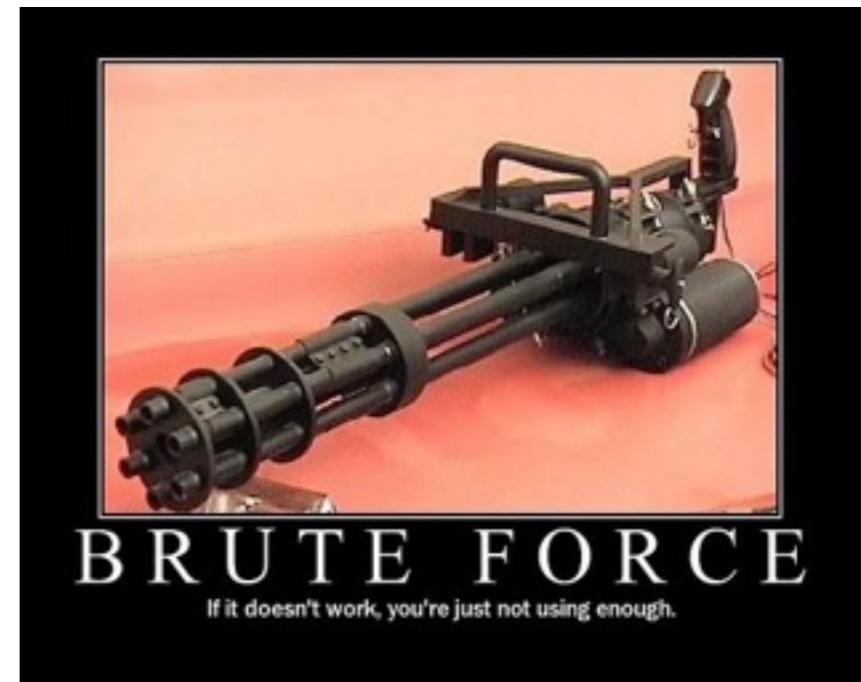
Exercice 3

Le côté obscur ...



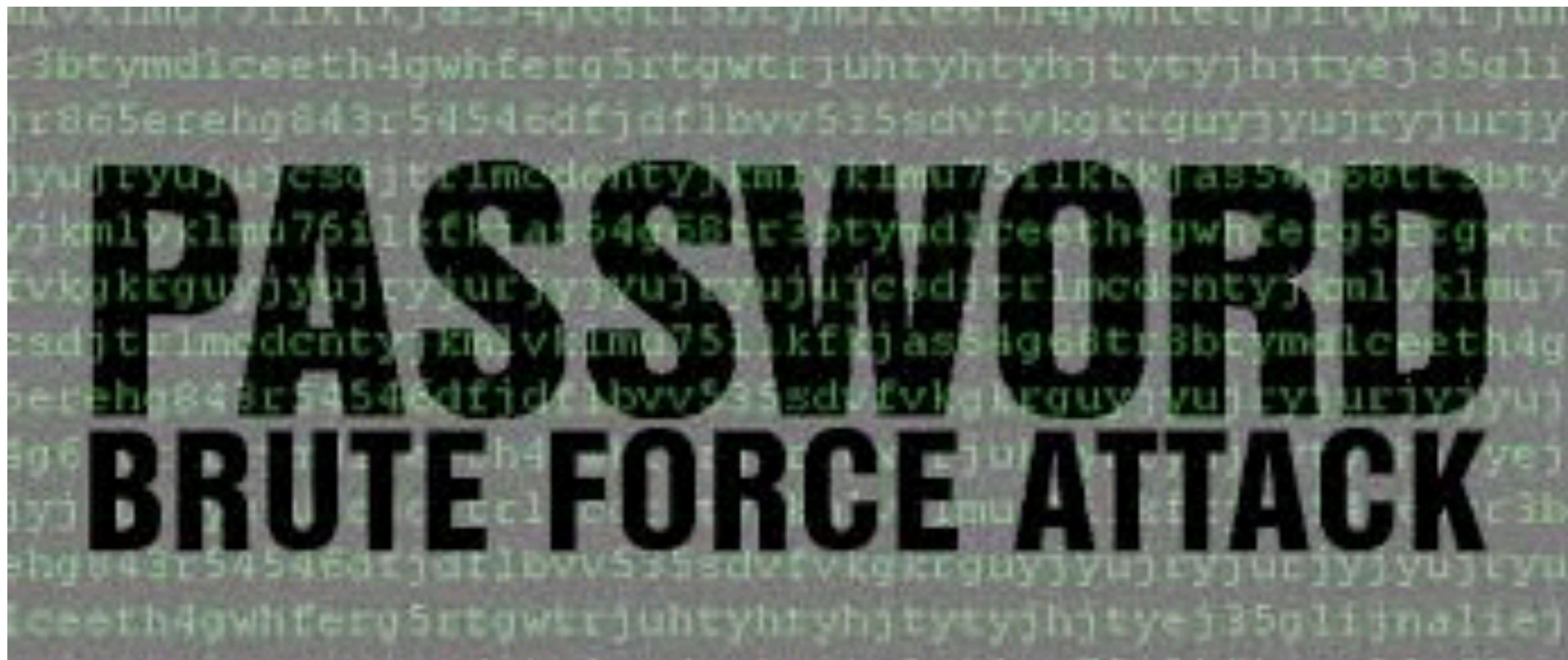
Exercice 3

Une **méthode simple** pour découvrir le **mot de passe** d'un compte est d'utiliser le **Brute Force**



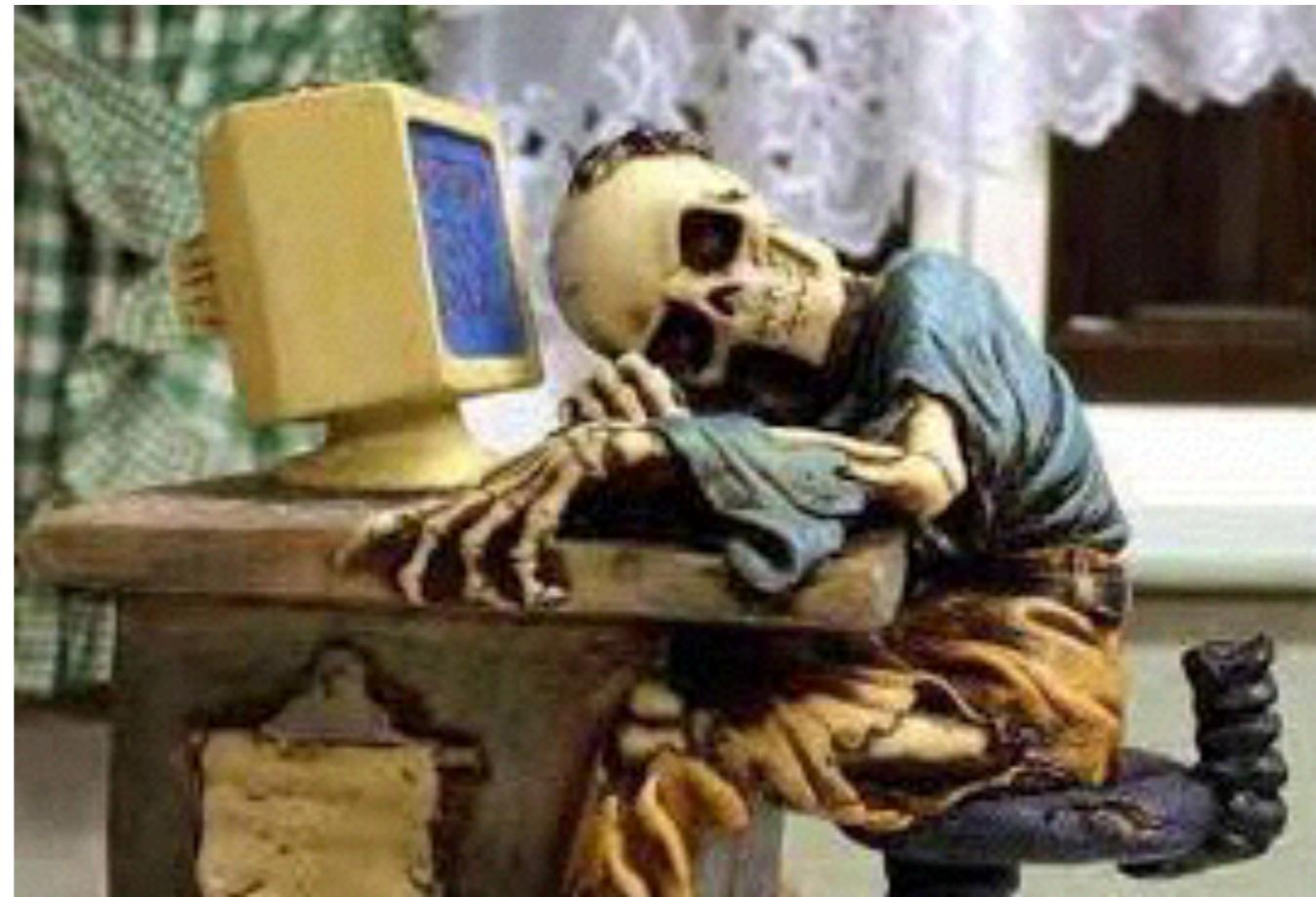
Exercice 3

C'est à dire de **tester toutes les possibilités** de
mots de passe existants



Exercice 3

En supposant que le mot de passe que vous tentez de découvrir par Brute Force est **composé de n caractères alphanumériques (A-Z,a-z,0-9)**, quel sera la **complexité** dans **le pire des cas** du code que vous utiliserez?



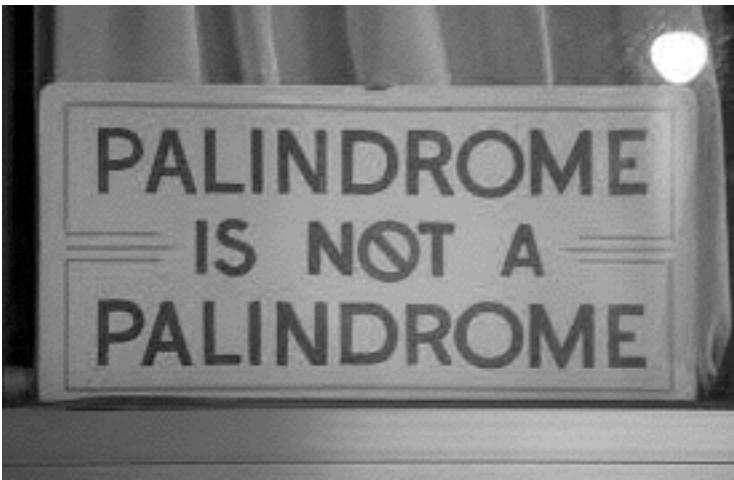
Exercice 4

Noyon est un ...



Exercice 4

On veut écrire **un algorithme** qui décide si un mot est un **palindrome ou non**. Un mot est un **palindrome** si sa **première lettre est identique à la dernière, sa deuxième à l'avant dernière etc...**



POP
SEVES
SELLES
GAG
REVER
ETE
ICI
KAYAK
ELLE
SOS
TOT
SES
SAGAS
SOLOS
ROTOR

Exercice 4

Exemples de mots palindromes : **ada, anna, bob, aviva, harrah, renner, salas, arora, eve, hannah, maham, otto**

Exemples de mots qui ne sont pas palindromes : **abca, aabcba, abcbd**

1. Proposez un pseudo-code

- input: Chaine de caractères
- output: boolean

2. Quelle est la complexité de votre algorithme?

Exercice 5



Exercice 5

Quel sera la **complexité** de cet **algorithme**?

Donner directement sous la forme Big O

```
int sum = 0;  
for (int i = 1; i <= N; i++)  
    for (int j = 1; j <= i*i; j++)  
        for (int k = 1; k <= j; k++)  
            sum++;
```