Rappel complexité

Le principe : on veut mesurer la consommation de ressources pour aboutir au résultat **: le temps** mis pour arriver au résultat **ou l'espace mémoire** nécessaire pour arriver au résultat (ou pourrait aussi s'interroger sur **l'énergie** dépensée pour arriver au résultat).

On parlera donc **de complexité en temps** ou en mémoire d'un algorithme**. Nous étudierons plus particulièrement la complexité en temps.**

1. Cette mesure doit être indépendante de la machine utilisée ainsi que du langage de programmation.

2. On peut considérer :

La complexité du meilleur cas (mais est-elle représentative ?) ; la complexité du cas moyen (comment le déterminer ?) ; <u>la complexité du pire cas (ne souffre pas de contestation…)</u>.

3. La complexité est une fonction de la taille des données, c'est à dire la taille d'un meilleur codage pour ces données.

En simplifiant :

- Un entier de petite taille, un caractère prends un espace constant.
- Un tableau de n entiers est de taille n …

4. Principe du raisonnement :

   a. On choisit une opération qui se fait au moins aussi souvent que toute autre opération.

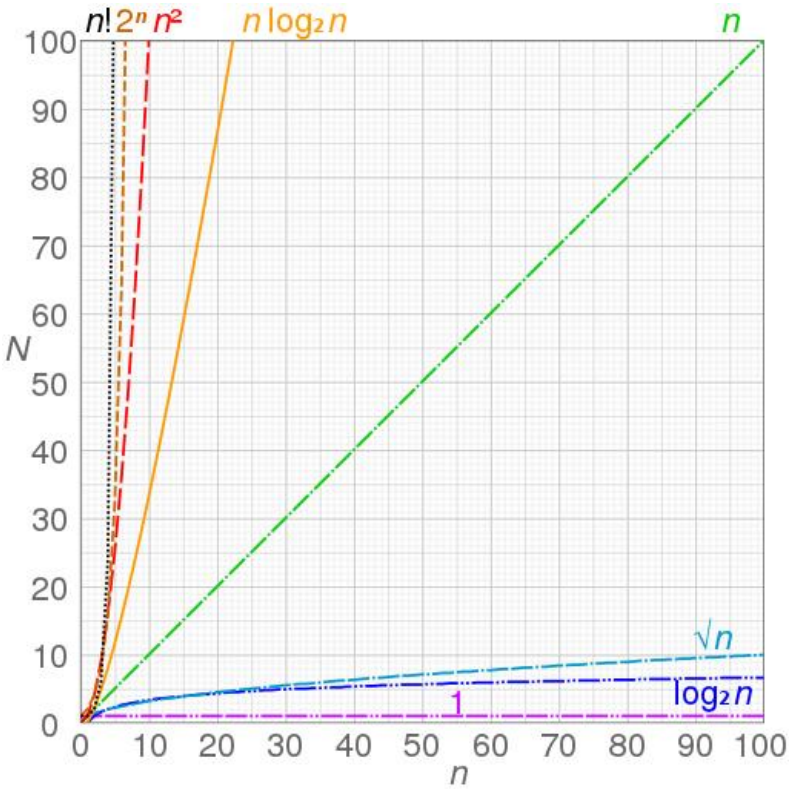
   b. On choisit une instance (la donnée à traiter) pour laquelle l'opération choisie aura lieu le plus souvent (pire cas). Appelons n sa taille.

   c. On comptabilise combien de fois cette opération choisie est effectuée en fonction de n. On obtient donc une fonction du type : $3n^2+4n+6$

5. On va simplifier les choses en regroupant les fonctions obtenues par classe :

- O(1) est la classe des fonctions constantes.
  - Par exemple, vérifier si un entier est égal à 0 admet un algorithme qui prend un temps constant. Sa complexité est dans O(1)
- O(n) est la classe des fonctions linéaires
  - Par exemple, trouver le max d'un tableau admet un algorithme qui prend un temps linéaire. Sa complexité est dans O(n)
  - Par exemple, trouver le nombre de « 0 » dans un tableau admet un algorithme qui prend un temps linéaire. Sa complexité est dans O(n)
  - 10 n $\in$O(n), 100 n$\in$O(n).
- Il n'est pas utile d'écrire $O(3n^2+5n+17)$. Parce que $O(3n^2+5n+17)=O(n^2)$
  Les tris par insertion et sélection sont en $O(n^2)$.
- Polynomiale : toute fonction en $O(n^k)$ où k est une constante. $O(n^3)$, $O(n^7)$.
- Exponentielle : toute fonction en $O(c^n)$: c constante : $O(2^n)$, $O(3^n)$ …
- Toute fonction exponentielle domine toute fonction polynomiale !

6) Représentations graphiques :



La recherche dichotomique d'un élément dans un tableau trié est en $O(\log_2(n))$

7) Pourquoi fuir l'exponentielle ?

n/f(n)	**n**	**n^2**	**n^3**	**2^n**	**3^n**	**n!**
10	10 µs	0,1 ms	1 ms	1 ms	59 ms	3,63 s
20	20 µs	0,4 ms	8 ms	1 s	58 min	77 094 ans
40	40 µs	1,6 ms	64 ms	12,73 J	385 253 ans	$2{,}58.\ 10^{34}$ ans
60	60 µs	3,6 ms	216 ms	36 533 ans	$1{,}34.\ 10^{15}$ ans	$2{,}63.\ 10^{68}$ ans

n : taille de l'instance à traiter

f(n) : complexité en temps (nombre d'opérations)

Une opération prend une microseconde (µs) = 10^{-6} s

Et si l'ordinateur va un million de fois plus vite ? Reprendre le calcul pour n = 60 et la classe $O(3^n)$:

Exercices : Pour chaque programme indiquer la complexité

Programme	Complexité

`n=10`
`a=3*n`
`print(a)`	

`n=20`
`for a in range(n):`
`  print(a)`	

`n=20`
`for a in range(n):`
`  for b in range(n):`
`  print(a,b)`	

`n=10`
`for a in range(2**n):`
`  print(a)`	

`n=200`
`while n>0:`
`  n=n//2`
`  print(n)`	