ALGO 2/5

studi

Auteur : Sébastien Inion

https://github.com/SebInfo/AlgoStudi2

Photo de emre keshavarz: https://www.pexels.com/fr-fr/photo/femme-table-jeune-jeu-7207270/

UN EXEMPLE COMPLET

- Pour un réel x et un entier $n \geq 1$, on veut calculer x^n
- Pour cela on va
  - proposer plusieurs **algorithmes** et les analyser :
  - démontrer leur **validité**,
  - estimer leur **complexité** (= **temps** et **espace mémoire** nécessaires au déroulement du programme)
  - voir une implémentation possible

UNE POSSIBILITÉ

- On peut faire des multiplications successives dans un tableau.
- Voici une proposition d'Algorithme

$\underline{\textsc{AlgoTableau}\ (x, n)}$

T un tableau de taille n;

$T[0] \longleftarrow x$;

pour tous les *i de 1 à* $n-1$ **faire**

$\quad T[i] \longleftarrow x * T[i-1]$;

retourner $T[n-1]$;

UNE POSSIBILITÉ

AlgoTableau (x, n)

T un tableau de taille n;
$T[0] \longleftarrow x$;
pour tous les i *de 1 à* $n-1$ **faire**
    $T[i] \longleftarrow x * T[i-1]$;
retourner $T[n-1]$;

SI ON EXÉCUTE CET ALGO AVEC LES VALEURS 3 ET 5

Un petit exemple :

On effectue **l'appel** AlgoTableau $(3, 5)$:

	$T=$					
- Initialisation de T :	$T=$	3				
- Étape $i=1$:	$T=$	3	9			
- Étape $i=2$:	$T=$	3	9	27		
- Étape $i=3$:	$T=$	3	9	27	81	
- Étape $i=4$:	$T=$	3	9	27	81	243

- **L'algo retourne 243**

IMPLEMENTATION EN PYTHON DE L'ALGO

$\underline{\text{ALGOTABLEAU } (x, n)}$

T un tableau de taille n;

$T[0] \longleftarrow x$;

pour tous les *i de 1 à $n-1$* **faire**

$\quad T[i] \longleftarrow x * T[i-1]$;

retourner $T[n-1]$;

```
def AlgoTableau(x,n):
    tab = []
    tab.append(x);
    for i in range(1,n):
        tab.append(x*tab[i-1])
    return tab[n-1]

print (AlgoTableau(2,3))
```

LE PROBLÈME DE LA TERMINAISON

$\textsc{AlgoTableau}\ (x, n)$

T un tableau de taille n;
$T[0] \longleftarrow x$;
pour tous les i *de 1 à* $n-1$ **faire**
  $T[i] \longleftarrow x * T[i-1]$;
retourner $T[n-1]$;

- Avant de passer à l'implémentation (passage au code pour nous Python) on doit **prouver** que l'algorithme s'arrête !
- On peut parfois avoir des boucles **non bornées** (tant que (while en Python) ou de la récursivité non bornée. Cela provoque des boucles infinies.
- La terminaison n'est pas synonyme de validité !
- Ici on a une boucle Pour donc la terminaison est évidente: i est automatiquement incrémenté de 1.
  Donc i va converger vers n-1. On rappelle que n >= 1.
  Dans le cas ou n=1 on a une boucle de 1 à 0 et donc on ne rentre pas dans le Pour -> l'algo retourne T[n-1] . En effet ici n indique le nombre de fois qu'on doit le faire donc si c'est 0 -> on ne fait pas.
- Quand on a une boucle Pour la preuve n'est pas nécessaire !
- C'est pour les boucle **while** qu'il faudra prouver qu'un moment la condition passe à FALSE.

LE PROBLÈME DE LA COMPLEXITÉ EN ESPACE

ALGOTABLEAU (x, n)

T un tableau de taille n;
$T[0] \longleftarrow x$;
pour tous les i *de 1 à* $n-1$ **faire**
  $T[i] \longleftarrow x * T[i-1]$;
retourner $T[n-1]$;

- La complexité peut se mesurer en terme d'espace : combien de variables ? quel place ? (en octets)
- Ici on simplifie ne parlant de case mémoire. ***La complexité sert à comparer les algorithmes entre eux et non pas à obtenir des informations précises. On veut donc un ordre de grandeur.***
  - Récupération des paramètres : x et n -> 2 cases mémoire
  - Déclaration de T (tableau) -> n cases mémoire
  - Déclaration de i -> 1 case mémoire
- On a donc n + 3 cases mémoires –> O(n) (de l'ordre de n)

LE PROBLÈME DE LA COMPLEXITÉ EN TEMPS

ALGOTABLEAU (x, n)

T un tableau de taille n;
$T[0] \longleftarrow x$;
pour tous les i *de 1 à* $n-1$ **faire**
  $T[i] \longleftarrow x * T[i-1]$;
retourner $T[n-1]$;

- On parle de complexité en temps mais comme les machines calculent avec des vitesses différentes on va s'intéresser au nombre d'opérations élémentaires.

- En dehors du Pour on a : 5 opérations
  - Récupération des paramètres -> 2 opérations
  - Déclaration de T -> 1 opération
  - Affectation T[0] 1 opération
  - retourner T[n-1] 1 opération

- Dans le Pour on a : 4 opérations mais fait n-1 fois donc (n-1)*4=4n-4 opérations
  - récupération de T[i-1] -> 1 opération
  - incrémentation de i (fait automatiquement par le Pour) -> 1 opération
  - multiplication -> 1 opération
  - Affectation à T[i] -> 1 opération

- Total 5 + 4n - 4 = 4 n + 1 -> O(n) (De l'ordre de n)

TRI A BULLES…

PRINCIPE

- Le tri à bulles est un algorithme qui consiste à comparer répétitivement les éléments consécutifs d'un tableau, et à les permuter lorsqu'ils sont mal triés.
- Il doit son nom au fait qu'il déplace les plus grands éléments en fin de tableau, comme des bulles d'air qui remonteraient rapidement à la surface d'un liquide.
- Voir demo : http://lwh.free.fr/pages/algo/tri/tri_bulle.html

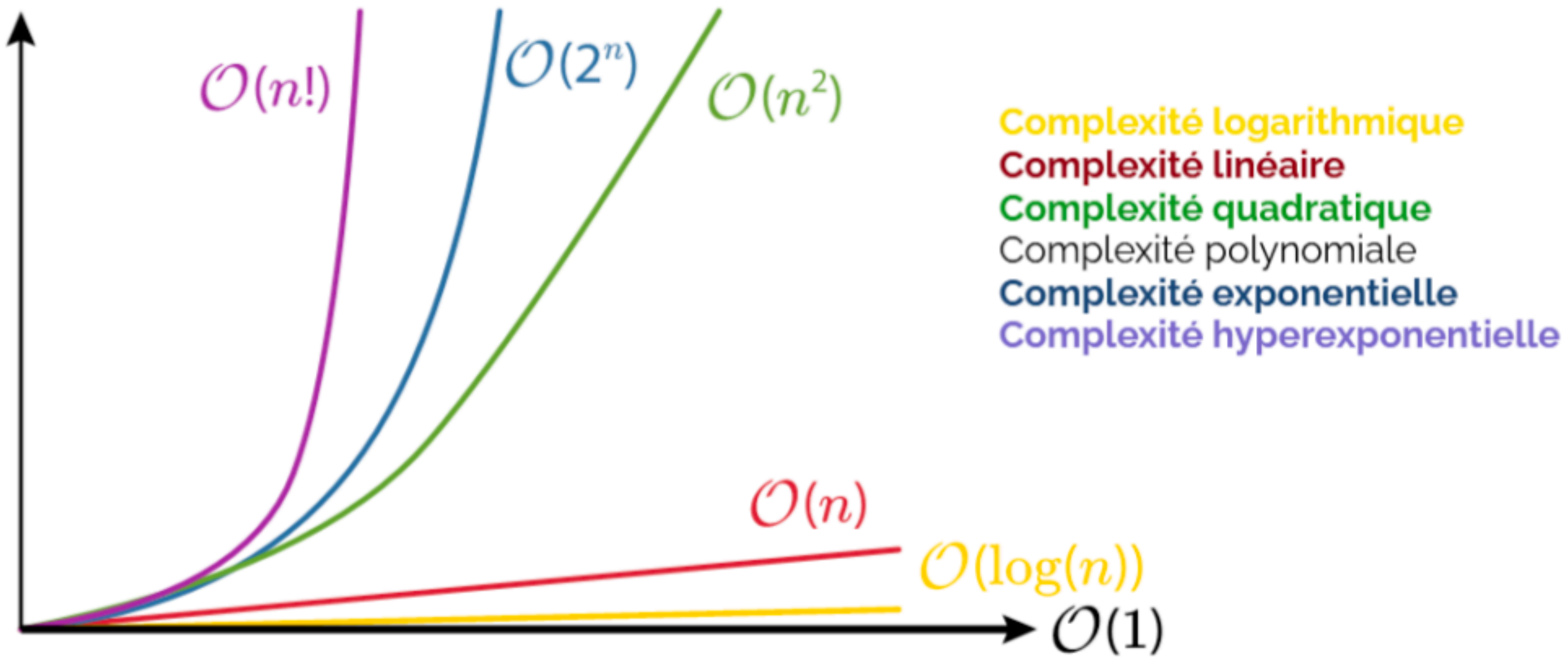
CODE PYTHON

```python
def tri_bulle(tableau):
    permutation = True
    passage = 0
    while permutation == True:
        permutation = False
        passage = passage + 1
        for en_cours in range(0, len(tableau) - passage):
            if tableau[en_cours] > tableau[en_cours + 1]:
                permutation = True
                # On echange les deux elements
                tableau[en_cours], tableau[en_cours + 1] = tableau[en_cours + 1],tableau[en_cours]
    return tableau
```

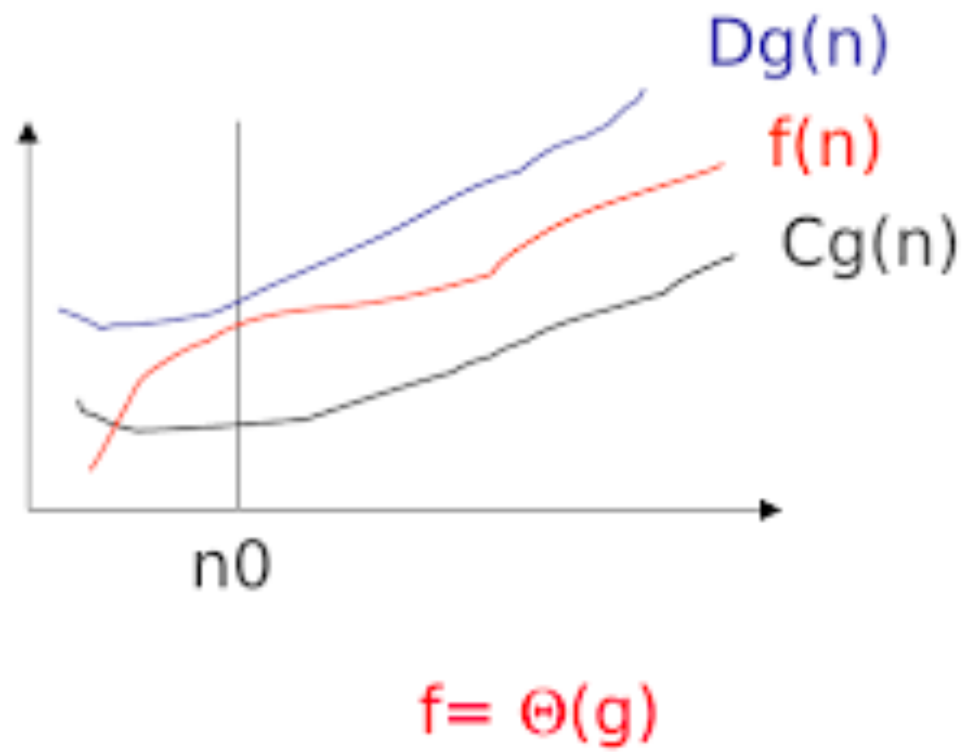
ETUDE DE LA COMPLEXITÉ

- Si l'on veut étudier la complexité de notre tri à bulles (on suppose ici en nbr d'opérations). On voit rapidement qu'on a deux boucles qui parcourent le tableau donc intuitivement n^2
- Dans le meilleur des cas l'algorithme fera n-1 comparaisons et aucune permutation. On a une complexité de O(n)
- Donc dans la pire des cas le tableau est trié dans le sens inverse on $\Theta(n^2)$
    - En effet deux boucle (n x n - n) / 2 = $(n^2\text{-n})/2$
- Dans un cas moyen on a $(n^2\text{-n})$ / 4 (on considère la moitié trié) on a aussi $\Theta(n^2)$.

ETUDE DE LA COMPLEXITÉ



ETUDE DE LA COMPLEXITÉ



Il s'agit d'une grandeur asymptotique.

EXERCICES

- Calculer la complexité des trois algorithmes de tri :
  - Tri à bulles
  - Tri par sélections
  - Tri par insertions
- Modifier le code des trois fonctions pour que ça retourne le nombre d'opérations (comparaisons et permutations)
- Vérifiez que la pratique rejoint la théorie ;)

MERCI POUR VOTRE ATTENTION