Parcours séquentiel d'un tableau – complexité d'un algorithme

La notion d'algorithme est souvent associée à l'informatique, pourtant le terme algorithme vient du nom du mathématicien perse du 9e siècle Muhammad Ibn Mūsā al-Khuwārizmī. La notion d'algorithme est donc très antérieure à la création du premier ordinateur.

Mais qu'est-ce qu'un algorithme ?

. Aglorithme

Lire la partie "Algorithme" du cours

La première chose à faire quand on étudie un algorithme, c'est de le "faire tourner à la main" : on "exécute" l'algorithme en utilisant uniquement une feuille et un crayon.

. Exercice 1

1. Soit l'algorithme suivant

```
1   VARIABLE
2   t : tableau d'entiers
3   x : nombre entier
4   tr : booléen (VRAI ou FAUX)
5   i : nombre entier
6   DEBUT
7   tr ← FAUX
8   i ← 0
9   tant que i<longueur(t) et que tr==FAUX:
10        si t[i]==x:
11            tr ← VRAI
12        i ← i+1
13  renvoyer la valeur de tr
14  FIN
```

Que fait cet algorithme ?

2. Répondre à l'activité en ligne suivante.
3. Faire tourner à la main cet algorithme pour **t = [5,8,15,23]** et **x = 15**

. Complexité d'un algorithme

Lire la partie "Complexité d'un algorithme" du cours

. Exercice 2

1. Résoudre l'activité en ligne
2. La courbe qui reste à la fin de l'activité montre 3 des fonctions quand n tend vers $+\infty$ (fiez vous au couleurs). En déduire un classement de ces 8 fonctions par ordre de croissance croissant

CC BY NC SA

. Exercice 3

1. Reprenons l'algorithme de l'exercice 1 et appliquons le à un tableau de longueur *n*.

```
1  tr ← FAUX
2  i ← 0
3  tant que i<longueur(t) et que tr==FAUX:
4          si t[i]==x:
5             tr ← VRAI
6          i ← i+1
7  renvoyer la valeur de tr
```

Lister les opérations élémentaires de cet algorithme

2. Reproduire le tableau suivant et compter le nombre de fois où chaque opération est effectuée pour les 2 premières lignes.

	Opération 1	Opération 2	Opération 3	Opération 4	
Ligne 1					
Ligne 2					
Ligne 3					
Ligne 4					
Ligne 5					
Ligne 6					
Ligne 7					
Total					

3. Combien de fois teste-t-on la condition du `tant que` ? Compléter la ligne 3
4. Si l'on fait x tests de la condition du `tant que`, alors combien de fois exécute-t-on l'intérieur du `tant-que` ? Compléter le ligne 4
5. Quel est le pire des cas pour la condition du `si` ? En déduire le nombre d'exécutions du corps du `si` dans ce cas. Remplir la ligne 5
6. Compléter les lignes 6 et 7.
7. Compléter la ligne Total en ajoutant les éléments de chaque colonne.
8. En déduire le nombre d'opérations élémentaires effectuées par cet algorithme dans le pire des cas en additionnant les éléments de la ligne Total.
9. Écrire T(n) en notation O (voir cours)

. Exercice 3

1. Écrivez un algorithme permettant de trouver le plus grand entier présent dans un tableau.
2. Faites "tourner à la main" votre algorithme en utilisant le tableau t = [3,5,1,8,4,2].
3. Déterminer la complexité de votre algorithme.

. Exercice 4

1. Écrivez un algorithme permettant de calculer la moyenne de tous les entiers présents dans un tableau.
2. Faites "tourner à la main" votre algorithme en utilisant le tableau t = [3,5,1,8,4,2].
3. Déterminer la complexité de votre algorithme.

cc BY NC SA

. Les paires de chaussettes

. Exercice 5

Une filature fabrique *n* chaussettes droite, toutes de tailles différentes (du 36 au 45) et *n* chaussettes gauches correspondantes, mais tout est mélangé ! Une association chaussette droite-chaussette gauche de même taille forment une paire.
L'objectif est d'apparier chaque chaussette droite avec la chaussette gauche qui lui correspond.

Le seul type d'opération autorisé consiste à plaquer une chaussette droite sur une chaussette gauche, ce qui peut amener trois réponses possibles :
 soit la chaussette droite est strictement plus grande que la chaussette gauche,
 soit elle est strictement moins grande,
 soit elles ont exactement le même taille.

On se propose d'implémenter des algorithmes en Python à partir de deux listes comportant les mêmes valeurs, mais dans un ordre différent.

1. Générer 2 listes aléatoires D0 et G0, de n éléments chacune, contenant les pointures des chaussettes.

- on affectera la valeur 10 à n pour commencer
- on pourra importer le module random et utiliser la méthode `random.shuffle(liste à mélanger)`

Pour apparier les chaussettes, il suffit de prendre une chaussette droite arbitrairement, de la tester avec toutes les chaussettes gauches.

2. **Écrire** un algorithme simple qui apparie chaque chaussette droite avec sa chaussette gauche. On écrira cet algorithme dans une fonction `paire_1(D, G)` qui attend 2 arguments de type *liste*, de mêmes tailles : la première contenant les **chaussettes droites**, la deuxième contenant les **chaussette gauches.** Cette fonction renvoie une liste des **paires** (des *tuples* (indice_chaussette droite, indice_chaussette gauche)).

Afin d'évaluer plus précisément le coût temporel de cette opération, on se propose d'utiliser le module Python `timeit`.

```
from timeit import timeit

duree_en_secondes = timeit("paire_1(D0,G0)",#Appel de la fonction à tester
                           globals=globals(),#Chargement de toutes les variables globales
                           number=10)           # Nombre d'essais
```

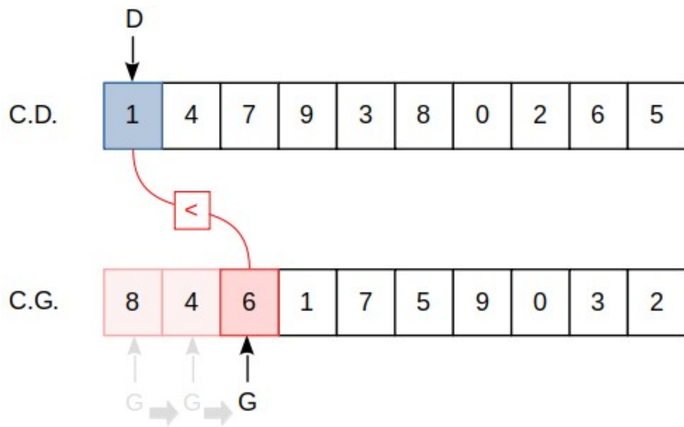
Et pour afficher les résultats, on réalisera plusieurs essais de l'algorithme, avec des tailles de liste différentes, et on affichera la durée d'exécution en fonction de la quantité de paires à constituer, à l'aide du module `matplotlib` :

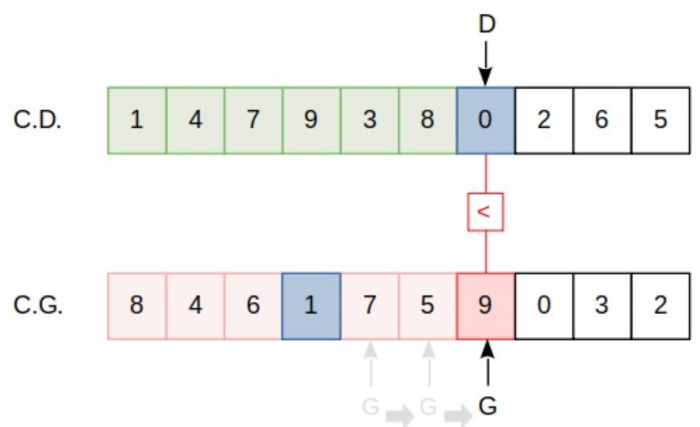
```
import matplotlib.pyplot as pl

pl.plot(liste_des_abscisses, liste_des_ordonnees, label = "un_nom_pour la légende")
pl.legend () # Ajout d'une légende
pl.show()    # Affichage du graphique
```

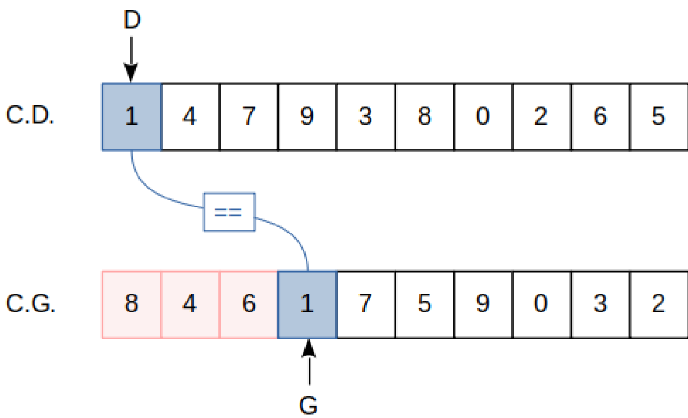
3. Faire afficher la courbe durée d'exécution en fonction de la quantité de paires à constituer (on choisit de faire varier la quantité de paires jusqu'à 1000, en 20 étapes).
4. Calculer l'ordre de complexité de cette fonction, et vérifier la cohérence avec la courbe obtenue.
5. Améliorer la fonction `paire_1()` (faire une fonction `paire_2()`) de sorte qu'une fois que la chaussette gauche compatible avec la chaussette droite a été trouvé, on cesse de chercher parmi les chaussette gauches.

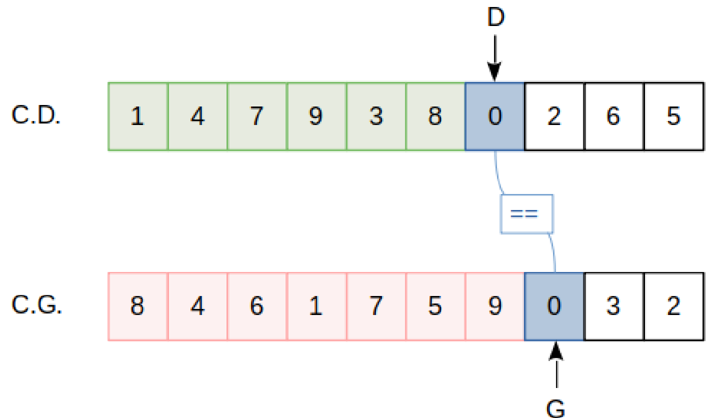
. (*POUR LES PLUS RAPIDES*)

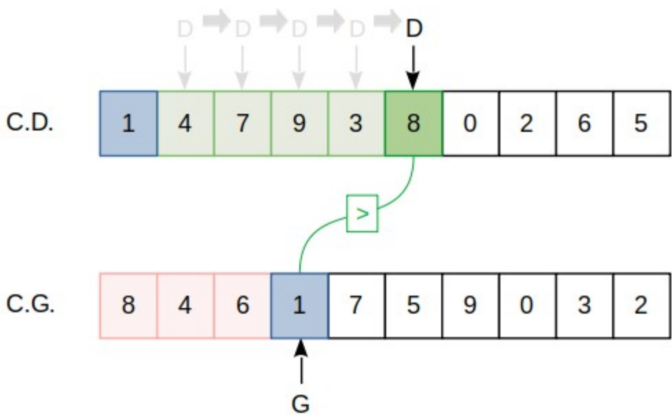
Supposons qu'au lieu de vouloir apparier toutes les chaussette gauche avec leur chaussette droite, on souhaite juste trouver <u>la plus petite chaussette droite et la chaussette gauche correspondante</u>.

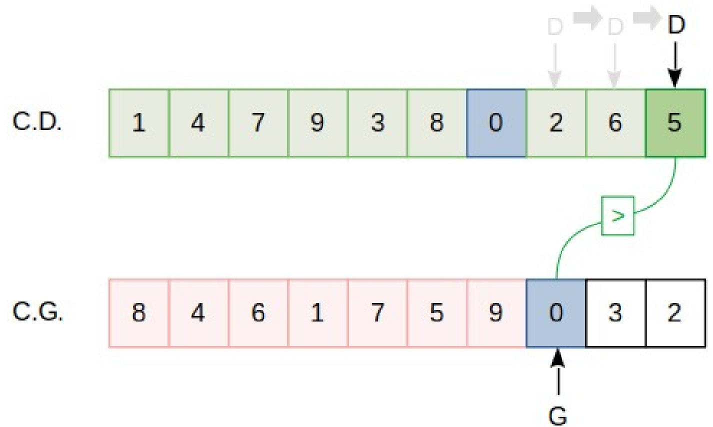












6. Montrer que ce problème peut être résolu en $2n-1$ essais, dans le pire des cas. Implémenter cet algorithme dans une fonction `chaussette_gauche_mini(D,G)` qui renvoie le tuple donnant les indices des chaussettes les plus petites

cc BY NC SA