LABORATOIRE DE MATHÉMATIQUES
LIFSE
CONGO BRAZZAVILLE
https://labomath-saintexbrazza.com

LYCÉE INTERNATIONAL FRANÇAIS
SAINT EXUPÉRY
BRAZZAVILLE CONGO

STAGE DE FORMATION

PARTIE III

Complexités

par

Valérien Eberlin

Année 2025 - 2026

Partie 1 : les automatismes

Question 1

a) Si liste désigne la liste [1, [2, 3], [4, 5], 6, 7], que vaut len(liste) ?
☐1 ☐3 ☐5 ☐7

b) Que vaut [2 * n for n in range(5)] ?
☐[0, 2, 4, 6, 8] ☐[0, 2, 4, 6, 8, 10] ☐[0, 2, 4] ☐Autre

c) Supposons que liste = [-5, 2, 3, -7, 42, 7]. Que vaut [n for n in liste if n > 0] ?
☐[-5, 2, 3, -7, 42, 7] ☐[2, 3, 42, 7] ☐[False, True, True, False, True, True]
☐Autre

Question 2

```
m = []
for i in range(5):
  n = []
  for j in range(3):
    n.append(i*j)
  m.append(n)
```

Quelle est la valeur de m à la fin de son exécution ?

Réponses :

☐[[0, 0, 0, 0, 0], [0, 1, 2, 3, 4], [0, 2, 4, 6, 8]]

☐[[0, 0, 0], [0, 1, 2], [0, 2, 4], [0, 3, 6], [0, 4, 8]]

☐[[1, 1, 1], [2, 4, 6], [3, 6, 9], [4, 8, 12], [5, 10, 15]]

☐[[1, 1, 1, 1, 1], [2, 4, 6, 8, 10], [3, 6, 9, 12, 15], [4, 8, 12, 16, 20], [5, 10, 15, 20, 25]]

Question 3

a) Soit le tableau semaine=['lundi','mardi','mercredi','jeudi','vendredi']. Proposer une solution avec la boucle for … in … pour afficher les jours de la semaine.

- -
- -
- -

b) Proposer un code avec une boucle for… in range(…) pour construire le tableau tab = [1, 2, 3, 4, 5, 6, 7, 8].

- -
- -
- -

c) Proposer un code avec une boucle for… in range(…) pour construire le tableau tab = [5, 10, 15, 20, 25, 30, 35, 40, 45, 50].

- -
- -
- -

Question 4

On donne : tab = [8,11]. Quelle instruction faut-il écrire pour :

a) ajouter en troisième élément " école" à tab ?

- -

b) changer le nombre 11 par le nombre 12 dans tab ?

- -

c) pour supprimer le premier élément de tab ?

- -

Question 5

Soit le dictionnaire groupe1 :

```
groupe1 = [{'nom': 'Dupuis', 'prenom': 'Jacque', 'age': 30},
           {'nom': 'Dupond', 'prenom': 'Paul', 'age': 28},
           {'nom': 'Boulanger', 'prenom': 'René', 'age': 28}]
```

1. Lors de la saisie de ce dictionnaire, l'informaticien s'est trompé sur le prénom de Monsieur Dupuis. Corriger l'erreur dans le prénom, la bonne valeur est “Jacques”.

- -

2. Faire une boucle pour afficher les noms de chaque personne.

\- -
\- -
\- -

Question 6

On considère la liste : Liste = [1, "bonjour", -15, 37, "algorithmes"].

Comment afficher la lettre j du mont bonjour ?- -

Partie 2 : Complexités

1. Coût

Lorsqu'un programme doit traiter deux listes de tailles différentes, le temps d'exécution du programme n'est pas nécessairement le même. Par exemple, si un programme doit traiter une liste de 1000 éléments puis une liste de 10 000 éléments. Est-ce que le temps d'exécution du programme est multiplié par 10 ? L'on peut donc s'interroger s'il existe un lien entre le temps d'exécution d'un programme et la taille d'une liste. En réalité, cela dépend de l'algorithme utilisé et de la liste. Pour une liste donnée, un programme peut être plus rapide qu'un autre, mais avec une autre liste, ce peut être le contraire. Le même programme peut être plus rapide avec la liste la plus longue.

De plus, pour traiter un même problème, non seulement nous pouvons disposer de plusieurs algorithmes mais un même algorithme peut avoir un temps d'exécution différent selon le langage de programmation utilisé et suivant la machine sur laquelle le programme est exécuté.

Analyse de l'algorithme de la tour d'Hanoï (vu au stage 2025-2026)

Le jeu consiste à déplacer n disques de diamètres différents d'une tour de « départ » à une tour d'« arrivée » en passant par une tour « intermédiaire » et ceci en un minimum de coups, tout en respectant les règles suivantes :

- on ne peut déplacer qu'un disque à la fois ;
- on ne peut placer un disque que sur un autre disque plus grand que lui ou sur une tour vide ;

- dans l'état initial, les n disques sont placés sur la tour « départ » ;
- dans l'état final, tous les disques se retrouvent placés dans le même ordre sur la tour « arrivée ».

Ainsi, la configuration de départ pour n = 3 est la suivante :

Nous avons vu lors d'un stage, l'algorithme ci-dessous, qui résout le problème de la tour d'Hanoï :

```
def hanoi(n,depart, arrivee, intermediaire):
    # s'il n'y a qu'un seule disque, on le déplace à la tour arrivee
    if n == 1 :
        print('deplacer', depart, '-->', arrivee)
    else:
        '''on déplace les n-1 disques vers la tour intermédiaire, en utilisant
        la tour arrivee comme tour auxilaire.'''
        hanoi(n-1,depart,intermediaire, arrivee)
        '''Il reste le plus grand disque sur la tour depart que l'on déplace
        sur la tour d'arrivee. Pour dire qu'on le déplace, on écrit simplement
        « print »'''
        print('deplacer', depart, '-->', arrivee)
        ''' Enfin, on déplace les n-1 disques de la tour intermediaire vers
        la tour d'arrivee en utilisant la tour depart comme tour intermédiaire.'''
        hanoi(n-1,intermediaire, arrivee, depart)
```

Tester progressivement : **hanoi(3,'T1', 'T3', 'T2'), hanoi(9,'T1', 'T3', 'T2') et hanoi(27,'T1', 'T3', 'T2')**

Que pouvez-vous dire sur le temps d'exécution ? Est-il proportionnel à la taille au nombre de disque ?

Définition : complexité linéaire

Soit n la taille d'une donnée. Si le nombre d'opérations à effectuer peut s'écrire sous la forme $\alpha n + \beta$, avec α et β réels, $\alpha > 0$, nous disons que l'algorithme a un ***coût linéaire*** ou une ***complexité linéaire***.

Définition : complexité quadratique

Soit n la taille d'une donnée. Si le nombre d'opérations à effectuer peut s'écrire sous la forme $\alpha n^2 + \beta n + \gamma$, avec α, β et γ réels, $\alpha > 0$, nous dirons que l'algorithme a un ***coût quadratique*** ou une ***complexité quadratique***.

Etude de la complexité de quelques algorithmes

Exemple 1

```
m = 0
p = 0
tant que m < a :
    m = m + 1
    p = p + 4
fin du tant que
```

Les passages dans la boucle ont lieu pour les valeurs de m égales à 0, 1, 2, . . . , a − 1 si la valeur de la variable a est un entier naturel a. Nous avons donc exactement a passages dans la boucle. Mais, à chaque passage, nous comptons deux additions ainsi que deux affectations. Nous pouvons donc dire que le nombre d'additions est 2a ou que le nombre d'opérations, au sens large en comptant les affectations, est 4a. Nous pouvons conclure que le coût est proportionnel à a c'est-à-dire qu'il est linéaire.

Exemple 2

```
somme = 0
for i in range(1, n+1):
    somme = somme + i
```

Il y a clairement n passages dans la boucle. A chaque passage nous avons une addition et une affectation. Donc un total de n additions et n affectations. Nous pouvons affirmer que le coût est de 2n et est linéaire.

Exemple 3

```
for i in range(n):
    ...
    for j in range(k):
        ...
```

Nous avons n passages dans la boucle externe. A chaque passage, nous avons un nombre fixe d'opérations q puis k passages dans la boucle interne. Dans la boucle interne, nous avons un nombre fixe d'opérations r. Donc pour chaque valeur de i, nous avons q + k × r = α opérations. Le nombre total d'opérations est donc αn et le coût est linéaire.

Exemple 4

```
for i in range(n):
    ...
    for j in range(n):
        ...
```

En raisonnant de même, nous obtenons ici, un coût de la forme $n(q + n \times r) = qn + n^2r$. Le coût est donc quadratique.

Exemple 5

```
for i in range(n):
    ...
    for j in range(i):
        ...
```

Nous avons n passages dans la boucle externe. A chaque passage, pour chaque valeur de i, nous avons un nombre fixe d'opérations r. Donc pour chaque valeur de i, nous avons q + i × r opérations. Les valeurs de i sont successivement 0, 1, 2, . . . , n −1. Le nombre total d'opérations est donc :

$$q + (q + 1 \times r) + (q + 2 \times r) + \ldots + (q + (n - 1) \times r) = nq + \frac{n(n-1)}{2} r$$
$$= \frac{r}{2} n^2 + \left(q - \frac{r}{2}\right) n$$

La complexité est de la forme $\alpha n^2 + \beta n$. Le coût est quadratique.

2. Parcours séquentiel

Dans un programme, lorsqu'une liste est parcouru élément par élément, on dit qu'on a un ***parcours séquentiel***.

a) Calcul de la moyenne

```
def moyenne(liste):
    n = len(liste)
    s = 0
    for u in liste:
        s = s + u
    return s / n
```

Le coût de ce programme est linéaire.

b) Recherche d'une occurrence

Le recherche d'une occurrence consiste à rechercher de manière séquentielle, la présence d'une valeur dans un tableau (liste, p-uplet, chaîne de caractères). L'algorithme s'arrête dès que l'indice de l'élément est trouvé ou si la fin du tableau est atteinte.

```
def recherche(x, t):
for i in range(len(t)):
    if t[i] == x:
        return i
```

Le coût de cet algorithme est linéaire.

c) Recherche d'un extremum

On construit l'algorithme de recherche d'un maximum dans un tableau non vide, de manière séquentielle, comme ceci : on suppose que le maximum est le premier élément, puis on parcourt la liste et chaque fois qu'on rencontre un élément plus grand que le maximum provisoire, on dit que c'est le nouveau maximum provisoire.

```
def maximum(liste):
    maxi = liste[0]
    for x in liste:
        if x > maxi:
            maxi = x
    return maxi
```

La recherche d'un maximum ou d'un minimum est basée sur le parcours séquentiel d'une liste avec un nombre fini d'opérations pour chaque élément. Le coût est linéaire.

3. Recherche dichotomique

Le principe de la recherche dichotomique consiste à déterminer si un entier `v` apparaît dans un tableau `t`, supposé déjà trié par ordre croissant. Plus précisément, on cherche à écrire une fonction qui renvoie un indice où la valeur `v` apparaît dans `t`.

Dans la vie courante, un exemple de recherche dichotomique consiste à rechercher un mot dans un dictionnaire. L'on ne parcourt pas toutes les pages mais l'on réduit progressivement l'intervalle de recherche jusqu'à trouver la page cherchée.

Pour mettre en œuvre la recherche dichotomique, on délimite la portion du tableau `t` à l'aide deux indices `g` et `d`.

Initialement ces deux indices délimitent l'intégralité du tableau : `g = 0, d =len(t)-1`.

Il faut maintenant examiner l'élément central pour prendre la décision.

On calcule sa position dans une variable `m`, en faisant la moyenne de `g` et de `d` : `m = (g+d)//2`.

Trois cas se présentent :

- Si `v > t[m]`, alors la recherche peut se restreindre à la moitié droite. On modifie donc la valeur de `g` en `g = m+1`.
- Si `v < t[m]`, alors la recherche peut se restreindre à la moitié gauche. On modifie donc la valeur de `d` en `d = m-1`.
- Si enfin les deux valeurs sont égales `v = t[m]`, c'est qu'on a trouvé une occurrence de la valeur de `v`. On renvoie alors l'indice `m` comme résultat.

On construit ainsi un algorithme de la recherche dichotomique de la manière suivante (à compléter) :

```
def recherche_dichotomique(tab, v):
    g = 0
    d = len(tab)-1
    while g <= d :
        m = (g+d)//2
        if tab[m] < ... :
            g = ...
        elif tab[m]> ... :
            d = ...
        else :
            return ...
```

La recherche dichotomique permet de rechercher une valeur dans un tableau trié. Le principe consiste à couper en deux à chaque fois la portion du tableau dans laquelle s'effectue la recherche.

Cet algorithme est très efficace. Par exemple, il ne faut que 7 étapes pour une taille de tableau ayant 100 éléments (car $2^6 \leq 100 < 2^7$) ; il ne faut que 10 étapes pour une taille de tableau ayant 1000 éléments (car $2^9 \leq 1000 < 2^{10}$) ; …

Le coût de la recherche dichotomique est de l'ordre du nombre de chiffres dans l'écriture binaire de n = len(tab), donc nettement inférieur à celui d'une recherche linéaire. Son coût est $\log_2(n)$.

Démonstration

Considérons un tableau de taille n. À chaque étape de l'algorithme, on compare l'élément central avec la valeur recherchée.

Si on ne l'a pas trouvé, on élimine la moitié du tableau. Le problème est donc réduit de moitié à chaque itération.

Dans le pire des cas, l'algorithme s'arrête lorsqu'il ne reste plus qu'un seul élément à vérifier. On cherche donc le nombre d'étapes k tel que la taille du segment soit égale à 1.

$$\frac{n}{2^k} = 1 \quad \text{D}'\text{où} \quad k = log_2(n)$$