

Fiche méthode : complexité

La complexité temporelle

1. De quoi parle-t-on ?

Deux programmes peuvent donner le même résultat, l'un en une seconde, l'autre en trois heures. Mesurer le temps au chronomètre ne suffit pas : il dépend de l'ordinateur, du langage, des autres programmes en cours.

● Méthode

La **complexité temporelle** d'un algorithme est le **nombre d'opérations élémentaires** qu'il effectue, exprimé en fonction de la **taille n des données** (longueur d'une liste, nombre de lignes d'une image, nombre de sommets d'un graphe...).

On ne cherche pas le nombre exact, mais **son ordre de grandeur quand n devient grand** : « si je double la taille des données, le temps est-il multiplié par 2 ? par 4 ? »

2. Les opérations élémentaires

Une opération est **élémentaire** si son coût ne dépend pas de la taille des données : on la compte pour 1.

affectation `x = ...` ; opérations arithmétiques `+` `-` `*` `/` `//` `%` `**` sur des nombres ;
comparaisons `<` `==` `!=` de nombres ; opérations logiques **and** **or** **not**
accès et modification d'un élément de liste `L[i]`, `L[i] = x` ; `len(L)` ; `L.append(x)` ;
`L.pop()` (dernier élément)
accès, ajout, test **cle** **in** **d** et suppression dans un dictionnaire

3. La notation O

● Méthode

On écrit que la complexité est en $O(f(n))$ quand le nombre d'opérations est, pour n grand, au plus de l'ordre de $f(n)$, à une constante multiplicative près. En pratique :

1. on garde le **terme qui croît le plus vite** ;
2. on **supprime les constantes multiplicatives**.

Exemples : $3n^2 + 5n + 2$ est en $O(n^2)$; $2n + 100$ est en $O(n)$; $\frac{n(n-1)}{2}$ est en $O(n^2)$; 50 est en $O(1)$.

Ordres de grandeur pour un ordinateur effectuant environ 10^8 opérations élémentaires par seconde :

Complexité	Nom	$n = 10^3$	$n = 10^6$	si n double...
$O(1)$	constante	instantané	instantané	inchangé
$O(\log n)$	logarithmique	instantané	instantané	+ 1 étape
$O(n)$	linéaire	instantané	0,01 s	$\times 2$
$O(n \log n)$	quasi linéaire	instantané	0,2 s	un peu plus que $\times 2$
$O(n^2)$	quadratique	0,01 s	3 heures	$\times 4$
$O(n^3)$	cubique	10 s	300 siècles	$\times 8$
$O(2^n)$	exponentielle	impossible	impossible	$\times 2^n$

4. Pire des cas, meilleur des cas

Le nombre d'opérations peut dépendre du *contenu* des données, pas seulement de leur taille. Pour chercher **v** dans une liste, on s'arrête au premier élément si **v** est en tête (**meilleur des cas**), on parcourt toute la liste si **v** est absent (**pire des cas**).

● Méthode

Sauf mention contraire, les sujets demandent la complexité **dans le pire des cas** : on suppose que les boucles **while** et les **return** anticipés font le maximum de tours.

5. Calculer une complexité : les règles

Structure	Règle
instructions successives	on additionne les coûts, puis on garde le plus grand : $O(n) + O(n^2) = O(n^2)$
if ... else ...	dans le pire des cas, on prend la branche la plus coûteuse (plus le coût du test)
boucle for	nombre de tours × coût d'un tour
boucles imbriquées indépendantes	on multiplie : n tours contenant m tours donnent $O(nm)$
boucle intérieure qui dépend de la boucle extérieure	on additionne les nombres de tours : $\sum_{i=0}^{n-1} (n-1-i) = \frac{n(n-1)}{2}$, soit $O(n^2)$
boucle while	on compte les tours grâce au variant ; si une quantité est divisée par 2 à chaque tour, il y a environ $\log_2 n$ tours
appel de fonction	on compte le coût de la fonction appelée, <i>pas</i> 1
fonction réursive	nombre d'appels × coût d'un appel (hors appels récurifs)

6. Les opérations qui cachent une boucle

Beaucoup d'opérations Python **ne sont pas élémentaires** : elles parcourent la liste. C'est la principale source d'erreurs.

Opération (n = longueur de L)	Coût	Pourquoi
x in L	$O(n)$	compare x à chaque élément
L.copy() , L[a:b] , list(d)	$O(n)$ ou $O(b-a)$	crée une nouvelle liste
L + M	$O(n+m)$	crée une nouvelle liste
L.pop(0) , L.insert(0, x) , del L[0]	$O(n)$	décale tous les éléments
sum(L) , max(L) , min(L)	$O(n)$	parcourent la liste
L.count(x) , L.index(x) , L.remove(x)	$O(n)$	parcourent la liste
L == M	$O(n)$	compare élément par élément
L.sort() , sorted(L)	$O(n \log n)$	tri
s + c sur des chaînes	$O(\text{len}(s))$	crée une nouvelle chaîne

● Attention !

- `res = res + [x]` dans une boucle de n tours coûte $O(n^2)$, alors que `res.append(x)` coûte $O(n)$ au total.
- `if x in L` : dans une boucle de n tours coûte $O(n^2)$; avec un dictionnaire, `if x in d` : coûte $O(n)$ au total.
- Une fonction récursive qui s'appelle sur `L[1:]` recopie la liste à chaque appel : $O(n^2)$.

7. Rédiger une réponse

● Méthode

Rédaction type : « La boucle ... effectue n tours. Chaque tour contient ... (des opérations élémentaires / une boucle de m tours / un appel à `f` de coût ...). La complexité est donc en $O(\dots)$. » Toujours préciser ce que représente n .

8. Exemples rédigés

```
def a_doublon(L: list) -> bool:
    '''Renvoie True si deux éléments
    de L sont égaux.'''
    n = len(L)
    for i in range(n):
        for j in range(i + 1, n):
            if L[i] == L[j]:
                return True
    return False
```

Pire des cas : aucun doublon, aucun `return` anticipé. Pour chaque `i`, la boucle intérieure fait $n-1-i$ tours, en $O(1)$ chacun. Total : $(n-1) + (n-2) + \dots + 0 = \frac{n(n-1)}{2}$ tours, soit $O(n^2)$ où n est la longueur de `L`.

```
def a_doublon_dico(L: list) -> bool:
    '''Même spécification.'''
    vus = {}
    for x in L:
        if x in vus:
            return True
        vus[x] = True
    return False
```

Au plus n tours ; chaque tour contient un test `in` et un ajout dans un dictionnaire, en $O(1)$. Complexité $O(n)$. Avec une liste `vus`, le test `x in vus` coûterait $O(n)$ et le total redeviendrait $O(n^2)$.

```
def nb_divisions(n: int) -> int:
    '''n >= 1. Renvoie le nombre de
    divisions par 2 pour atteindre 1.'''
    c = 0
    while n > 1:
        n = n // 2
        c = c + 1
    return c
```

`n` est divisé par 2 à chaque tour : après k tours, il vaut environ $\frac{n}{2^k}$, et la boucle s'arrête quand $2^k \approx n$, soit $k \approx \log_2 n$. Chaque tour est en $O(1)$: complexité $O(\log n)$.