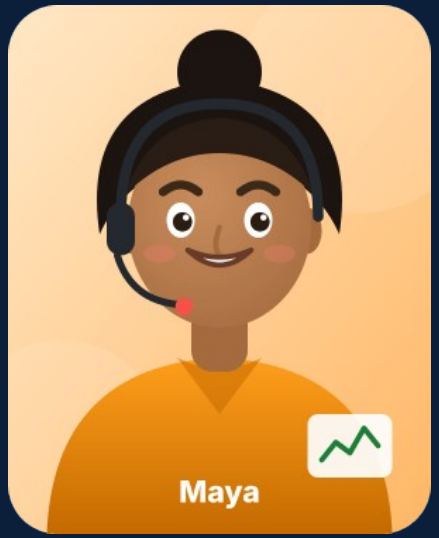
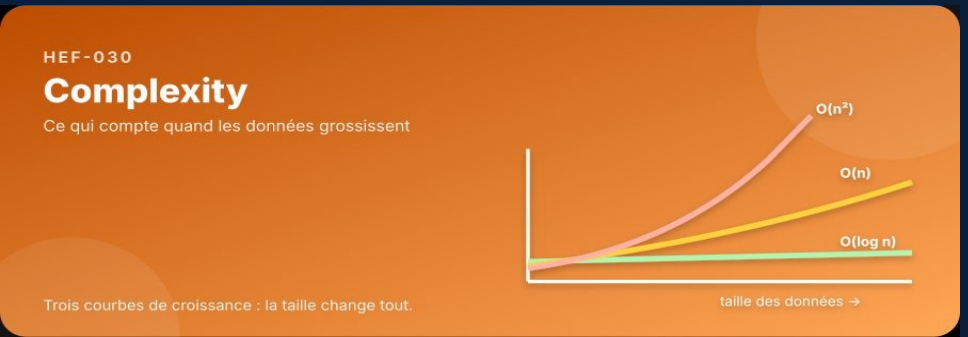
HEF-030 · MISSION 02 / 10

La notation Big O

Un langage commun pour dire comment un algorithme grandit quand les données grandissent.



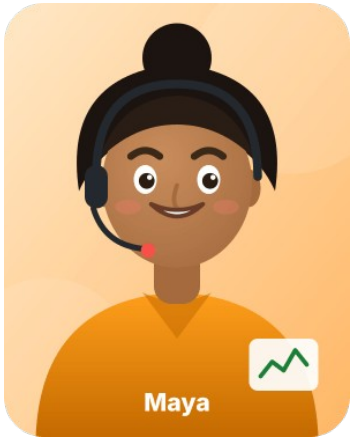


Badge : Lecteur de courbes

70 min · débutant

L'histoire

Kodi compare deux solutions : l'une fait `3n + 5` opérations, l'autre `n² / 100 + 2`. Pour n = 10, la seconde gagne. Pour n = 10 000, c'est l'inverse.



> *« On ne s'intéresse pas à n = 10, dit Sam. On s'intéresse à ce qui se passe quand n devient immense. C'est le rôle du Big O. »*

Tu apprends à lire et à écrire la notation qu'utilisent tous les ingénieurs pour parler de performance.

Ta mission

À la fin, je serai capable de…

1. expliquer ce que signifie O(n)
2. simplifier une formule de coût (ignorer constantes et termes mineurs)
3. reconnaître les classes O(1), O(log n), O(n), O(n log n), O(n²), O(2ⁿ)
4. ordonner ces classes par vitesse de croissance
5. comparer deux algorithmes avec leur Big O

Défi d'échauffement

Défi d'échauffement

Voici deux formules de coût : **A(n) = 100 n** et **B(n) = n²**. Calcule A et B pour n = 10, 100, 1 000, 10 000. À partir de quelle valeur de n, environ, B dépasse-t-il A ? Que se passe-t-il ensuite ?

Le concept

L'idée

Le **Big O** décrit comment le coût **croît** quand n devient grand, en ignorant :

1. les **constantes multiplicatives** (3n et n sont de la même famille) ;
2. les **termes mineurs** ($n^2 + 5n + 20$ se comporte comme n^2).

On écrit $O(n^2)$ et on lit « de l'ordre de n au carré ».

Règles de simplification

Formule de coût	Big O
5	O(1)
3n + 7	O(n)

Le concept

Règles de simplification (suite)

Formule de coût	Big O
$n^2 + 100n + 4$	$O(n^2)$
$2n + n \log n$	$O(n \log n)$
500	$O(1)$

Les grandes familles

Notation	Nom	Exemple typique
O(1)	constant	lire `liste[i]`, accéder à un dictionnaire
O(log n)	logarithmique	recherche binaire
O(n)	linéaire	parcourir une liste
O(n log n)	quasi-linéaire	bons tris (fusion)

Le concept

Les grandes familles (suite)

Notation	Nom	Exemple typique
$O(n^2)$	quadratique	double boucle
$O(2^n)$	exponentiel	tester tous les sous-ensembles

Combien d'opérations ?

n	log n	n log n	n^2	2^n
10	3	33	100	1 024
100	7	664	10 000	$\approx 10^{30}$
1 000	10	9 966	1 000 000	inimaginable
1 000 000	20	20 millions	10^{12}	:

Le concept

Combien d'opérations ? (suite)

Un ordinateur fait de l'ordre de un milliard d'opérations simples par seconde. Un algorithme en $O(n^2)$ sur 1 million d'éléments (10^{12} opérations) prend des minutes à des heures ; en $O(n \log n)$, moins d'une seconde.

Ce que Big O ne dit pas

- Il ne compare pas deux algorithmes de **même classe** (deux O(n) peuvent différer d'un facteur 10).
- Il décrit les **grandes** valeurs de n. Pour de petites données, un algorithme « moins bon » peut gagner.
- Il dépend de l'hypothèse (pire cas, cas moyen).

🧠 ***Astuce*** *: O(n) se lit « si les données sont multipliées par 10, le travail est multiplié par 10 ». O(n²) : « multiplié par 100 ».*

Exemple pas à pas

Exemple pas à pas

Problème : classer ces algorithmes du plus rapide au plus lent pour de grandes données.

1. Lire le premier élément d'une liste.
2. Chercher dans une liste non triée.
3. Chercher dans une liste triée par recherche binaire.
4. Trier avec le tri fusion.
5. Comparer toutes les paires.

Algorithme	Big O
1. Lire le premier	O(1)
3. Recherche binaire	O(log n)
2. Recherche linéaire	O(n)

Exemple pas à pas (suite)

Algorithme	Big O
4. Tri fusion	O(n log n)
5. Toutes les paires	O(n²)

Vérification par le calcul (le nombre d'opérations pour n = 1 024) :

```
import math

n = 1024
print("O(1)      :", 1)
print("O(log n)  :", int(math.log2(n)))
print("O(n)      :", n)
print("O(n log n):", int(n * math.log2(n)))
print("O(n²)     :", n * n)
```

Exemple pas à pas (suite)

Simplification d'une formule : `4n² + 20n + 100`. Quand n = 1 000, 4n² vaut 4 000 000, 20n vaut 20 000, 100 est négligeable. Le terme dominant décide : **O(n²)**.

À toi, avec aide

Donne le Big O de chaque formule :

1. $7n + 3$
2. $2n^2 + 50n$
3. 100
4. $n + n \log n$
5. $3 \times 2^n + n^3$

Puis classe les cinq du plus lent au plus rapide pour n très grand.

À toi, avec aide · réponse

Indices progressifs

Indice 1

On garde le terme qui grandit le plus vite et on retire la constante qui le multiplie.

Indice 2

n log n grandit plus vite que n, mais moins vite que n^2.

Indice 3

2^n finit toujours par dépasser n^3.

Réponse

1. $O(n)$ · 2. $O(n^2)$ · 3. $O(1)$ · 4. $O(n \log n)$ · 5. $O(2^n)$.

Du plus lent au plus rapide : $O(2^n) > O(n^2) > O(n \log n) > O(n) > O(1)$.

À toi, en solo

Écris un petit programme qui, pour n = 10, 100, 1 000, 10 000, 100 000, affiche un tableau avec $\log_2 n$, n, $n \log_2 n$, n^2 (utilise `math.log2`) et, pour chaque n, indique **le temps en secondes** si un ordinateur fait 10^9 opérations par seconde.

Puis réponds : pour n = 100 000, quel algorithme en $O(n^2)$ est-il encore utilisable ?

À toi, en solo · réponse

Pistes de correction

```
import math

print(f"{'n':>8} {'log n':>6} {'n log n (s)':>14} {'n² (s)':>14}")
for n in (10, 100, 1_000, 10_000, 100_000):
    nlogn = n * math.log2(n) / 1e9
    carre = n * n / 1e9
    print(f"{n:>8} {math.log2(n):>6.1f} {nlogn:>14.6f} {carre:>14.6f}")
```

Pour n = 100 000 : $n^2 = 10^{10}$ opérations, soit environ 10 secondes à 10^9 opérations par seconde. Utilisable pour un traitement ponctuel, pas pour une requête web. n log n vaut environ 1,7 million d'opérations, soit 2 millisecondes.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Garder les constantes	3n et n ne sont pas à distinguer en Big O	Retirer constantes et termes mineurs
Penser que O(n) = rapide	Deux O(n) peuvent différer de 100 fois	Mesurer quand l'écart compte
Ignorer les petits n	Un algorithme « lent » peut gagner sur 10 éléments	Choisir selon la taille réelle des données
Confondre log n et n	Erreur d'un facteur énorme	Se rappeler : $\log_2(1\,000\,000) \approx 20$

Mini-boss

Mini-boss

Pour chaque fonction ci-dessous, donne le Big O, justifie en une phrase, puis **vérifie en comptant** les opérations pour n = 100, 200, 400.

```
def A(n):
    c = 0
    for i in range(n):
        c += 1
    return c

def B(n):
    c = 0
    i = 1
    while i < n:
        c += 1
        i *= 2
    return c

def C(n):
```

Mini-boss

Mini-boss (suite)

```
    c = 0
    for i in range(n):
        for j in range(i):
            c += 1
    return c

def D(n):
    c = 0
    for i in range(n):
        j = 1
        while j < n:
            c += 1
            j *= 2
    return c
```

Mini-boss · réponse

Pistes de correction

- A : O(n) · B : O(log n) (i double à chaque tour) · C : O(n²) (n(n−1)/2) · D : O(n log n) (n tours × log n tours internes).

```
def A(n):
    c = 0
    for i in range(n): c += 1
    return c
def B(n):
    c, i = 0, 1
    while i < n: c += 1; i *= 2
    return c
def C(n):
    c = 0
    for i in range(n):
        for j in range(i): c += 1
    return c
def D(n):
    c = 0
```

Pistes de correction (suite)

```
    for i in range(n):
        j = 1
        while j < n: c += 1; j *= 2
    return c

for n in (100, 200, 400):
    print(n, A(n), B(n), C(n), D(n))
```

Quand n double : A double, B augmente de 1, C quadruple environ, D un peu plus que double.

Quiz éclair

Quiz éclair

1. O(n^2) signifie que si n est multiplié par 10, le travail est multiplié par :

A. 10
B. 1 000
C. environ 100

2. La classe la plus rapide parmi celles-ci est :

A. O(log n)
B. O(n)
C. O(n^2)

3. On simplifie $5n^2 + 30n$ en :

A. O(n)

Quiz éclair

Quiz éclair (suite)

B. O(n^2)
C. O(30n)

4. 🔁 *Révision mission 01* : Que devient le coût d'un algorithme linéaire si n double ?

A. Il quadruple
B. Il reste identique
C. Il double

5. 🔁 *Révision mission 01* : Le pire cas donne :

A. une garantie
B. un coût optimiste
C. un coût moyen exact

Quiz éclair · réponse

Réponses

1. **C**.
2. **A**.
3. **B**.
4. **C**.
5. **A**.

Je réfléchis

Je réfléchis

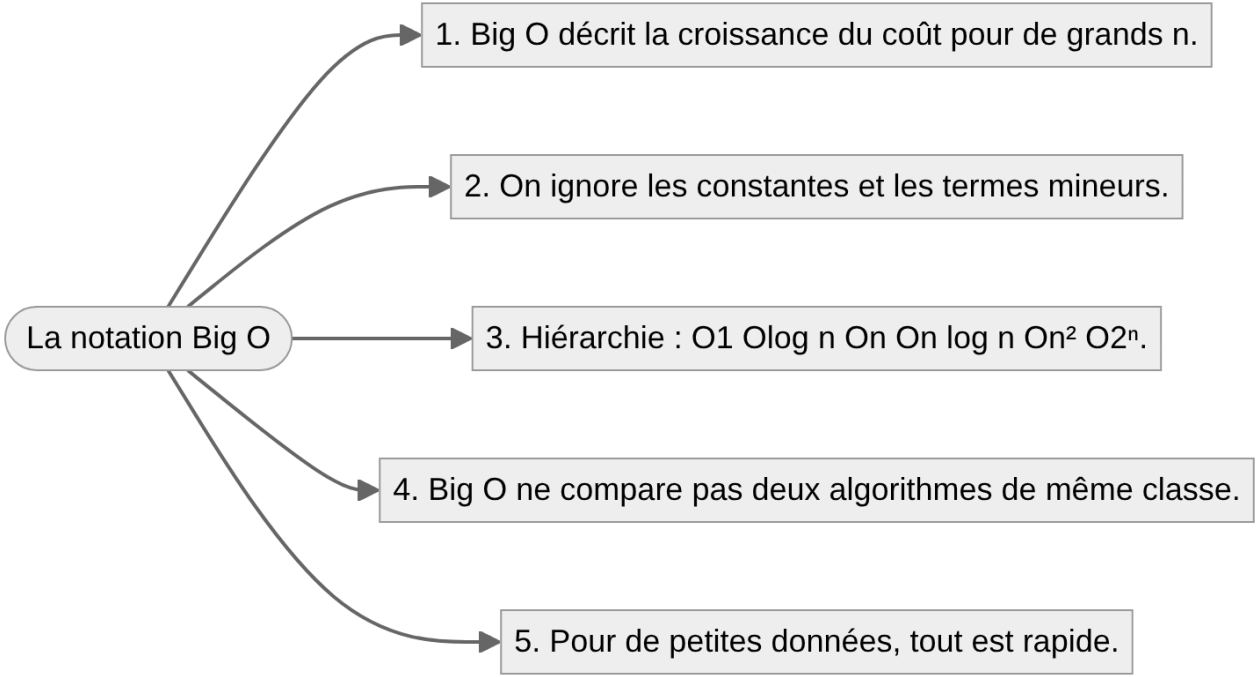
1. Dans ta vie, quelle tâche a un coût « quadratique » (chaque nouvel élément oblige à comparer avec tous les autres) ?
2. Pourquoi ignorer les constantes est-il à la fois utile et dangereux ?

À retenir

1. Big O décrit la croissance du coût pour de grands n.
2. On ignore les constantes et les termes mineurs.
3. Hiérarchie : $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(2^n)$.
4. Big O ne compare pas deux algorithmes de même classe.
5. Pour de petites données, tout est rapide.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Lecteur de courbes

Prochaine mission : Boucles et compositions

