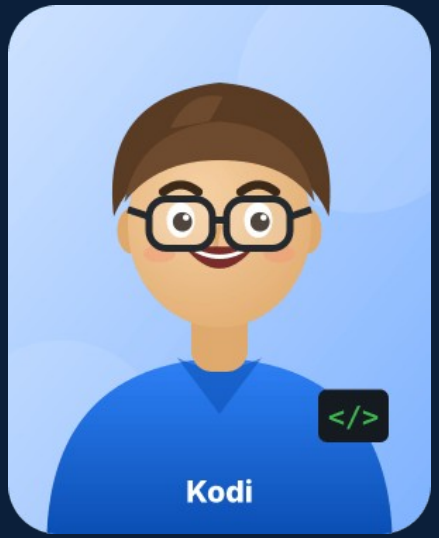
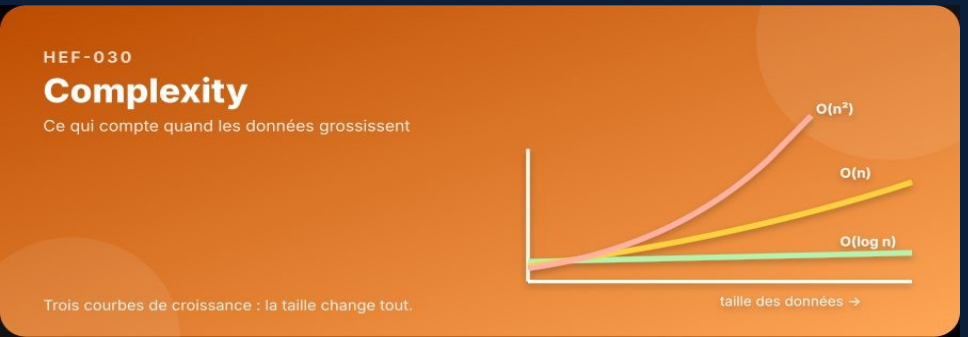
HEF-030 · MISSION 01 / 10

Compter ce que coûte un programme

Avant de dire « c'est lent », il faut savoir ce qu'on compte.





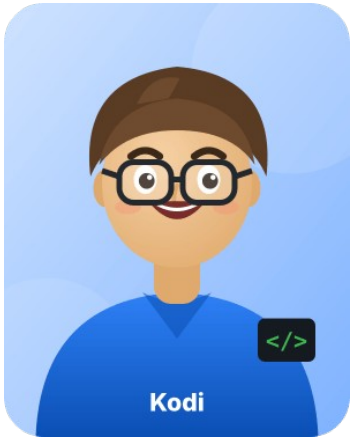
Badge : Comptable des opérations

65 min · débutant

L'histoire

Le tableau de bord de HashBox met 20 secondes à s'afficher pour les gros comptes. Maya veut savoir si « ça passera à l'échelle » quand HashBox aura dix fois plus de membres.

« Prédire ça sans tester, c'est possible, dit Sam. Mais il faut d'abord se mettre d'accord sur ce qu'on compte. »



Dans cette mission, tu apprends à mesurer le coût d'un algorithme indépendamment de l'ordinateur.

Ta mission

À la fin, je serai capable de…

1. expliquer pourquoi on ne compare pas des secondes mais des opérations
2. compter les opérations d'un petit algorithme
3. identifier la taille de l'entrée (n)
4. distinguer meilleur cas, pire cas et cas moyen
5. prédire comment le coût évolue quand n double

Défi d'échauffement

Défi d'échauffement

Tu cherches un nom dans un annuaire papier de **1 000 pages**. Combien de pages regardes-tu environ si tu les lis une par une ? Et si l'annuaire passe à **2 000 pages** ? À **1 000 000** ? Écris tes estimations. Le temps de lecture d'une page dépend-il de ton âge ou de ton humeur ? Pourquoi est-il préférable de compter les pages ?

Le concept

Pourquoi pas les secondes ?

Le temps dépend de la machine, du langage, de la charge du moment. Un algorithme rapide sur ton ordinateur peut sembler lent sur un autre. On préfère **compter les opérations élémentaires** (comparaison, addition, lecture d'une case…).

La taille de l'entrée : n

On exprime le coût en fonction de **n**, la taille de l'entrée : nombre d'éléments d'une liste, de lignes d'un fichier, de sommets d'un graphe. Choisir le bon n est la première décision.

Un exemple de comptage

Le concept

Un exemple de comptage (suite)

```
def somme(liste):
    total = 0              # 1 opération
    for x in liste:        # n tours
        total += x         # 1 opération par tour
    return total           # 1 opération
```

Coût ≈ n + 2 opérations. Pour n = 1 000 : environ 1 002. Pour n = 2 000 : environ 2 002. **Le coût double quand n double.**

```
def paires(liste):
    compteur = 0
    for a in liste:            # n tours
        for b in liste:        # n tours pour chaque a
            if a < b:
                compteur += 1  # 1 opération
    return compteur
```

Le concept

Un exemple de comptage (suite)

Ici, on fait $n \times n = n^2$ comparaisons. Pour n = 1 000 : 1 000 000. Pour n = 2 000 : 4 000 000. **Le coût quadruple quand n double.**

Meilleur, pire, moyen

Pour chercher un élément dans une liste :

Cas	Situation	Comparaisons
Meilleur	l'élément est en première position	1
Pire	l'élément est en dernière position ou absent	n
Moyen	position uniformément répartie	environ n / 2

On s'intéresse en général au **pire cas** : c'est la garantie qu'on peut donner.

Le concept

Compter avec du code

On peut instrumenter un algorithme avec un compteur pour vérifier ses prédictions :

```
def chercher(liste, cible):
    comparaisons = 0
    for x in liste:
        comparaisons += 1
        if x == cible:
            return comparaisons
    return comparaisons
```

🧠 ***Astuce*** *: la question utile n'est pas « combien de secondes ? » mais* ***« que devient le coût si n double ? »***

Exemple pas à pas

Problème : compter les opérations de `maximum` et prédire le coût pour n = 1 000 000.

```
def maximum(liste):
    comparaisons = 0
    meilleur = liste[0]
    for x in liste[1:]:
        comparaisons += 1
        if x > meilleur:
            meilleur = x
    return meilleur, comparaisons

for n in (10, 100, 1000):
    _, c = maximum(list(range(n)))
    print(n, c)
# 10 9
# 100 99
# 1000 999
```

Exemple pas à pas (suite)

Observation : le coût est n − 1 comparaisons, quelle que soit la liste.
Prédiction : pour n = 1 000 000, environ 999 999 comparaisons.
Comparaison avec un tri complet pour trouver le maximum : trier d'abord puis prendre le dernier coûte beaucoup plus que n comparaisons. Compter permet de juger sans chronomètre.

À toi, avec aide

Instrumente cette fonction avec un compteur, puis remplis le tableau pour n = 4, 8, 16 :

```
def a_doublon(liste):
    for i in range(len(liste)):
        for j in range(i + 1, len(liste)):
            if liste[i] == liste[j]:
                return True
    return False
```

Utilise une liste **sans doublon** (pire cas). Que remarques-tu quand n double ?

À toi, avec aide · réponse

Indices progressifs

Indice 1

Ajoute `comparaisons += 1` juste avant `if liste[i] == liste[j]`.

Indice 2

Pour n = 4, il y a 3 + 2 + 1 = 6 comparaisons.

Indice 3

La formule est n(n−1)/2.

À toi, avec aide · réponse

Réponse

```
def a_doublon(liste):
    comparaisons = 0
    for i in range(len(liste)):
        for j in range(i + 1, len(liste)):
            comparaisons += 1
            if liste[i] == liste[j]:
                return True, comparaisons
    return False, comparaisons

for n in (4, 8, 16):
    print(n, a_doublon(list(range(n)))[1])
# 4 6
# 8 28
# 16 120
```

Quand n double, le coût est multiplié par un peu moins de 4 : $n(n-1)/2$ est de l'ordre de n^2.

À toi, en solo

Voici trois fonctions. Pour chacune : (1) identifie n ; (2) compte les opérations dans le pire cas (formule) ; (3) vérifie avec un compteur pour n = 10, 20, 40 ; (4) prédis le coût pour n = 1 000.

```
def f(liste):              # liste de n éléments
    return liste[0] + liste[-1]

def g(liste):
    total = 0
    for x in liste:
        total += x
    for x in liste:
        total += 2 * x
    return total

def h(liste):
    total = 0
    for x in liste:
        for y in liste:
```

À toi, en solo (suite)

```
            total += x * y
    return total
```

À toi, en solo · réponse

Pistes de correction

- `f` : 3 opérations (deux lectures, une addition), indépendantes de n → **constant**.
- `g` : environ 2n additions (deux boucles de n tours) → **proportionnel à n** (le facteur 2 ne change pas la nature de la croissance).
- `h` : n × n = n² multiplications → **quadratique**.

Pour n = 1 000 : f ≈ 3, g ≈ 2 000, h ≈ 1 000 000.

```
def compter_g(n):
    return 2 * n

def compter_h(n):
    return n * n

for n in (10, 20, 40):
    print(n, compter_g(n), compter_h(n))
```

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Mesurer en secondes uniquement	Résultat dépendant de la machine	Compter les opérations et vérifier avec un chronomètre
Oublier de définir n	On ne sait pas de quoi dépend le coût	Dire explicitement : n = nombre de …
Ne regarder que le meilleur cas	Fausse impression de rapidité	Raisonner sur le pire cas
Confondre petit et grand n	Pour n = 10, tout est rapide	Se demander ce qui se passe pour n énorme

Mini-boss

Mini-boss

Voici trois méthodes pour savoir si une liste de n nombres contient **deux éléments dont la somme vaut S** :

- **A** : toutes les paires (double boucle) ;
- **B** : trier puis utiliser deux pointeurs ;
- **C** : un ensemble des valeurs déjà vues.

1. Écris les trois fonctions.
2. Instrumente-les avec un compteur d'opérations (comparaisons / accès à l'ensemble).
3. Mesure pour n = 100, 200, 400, 800 (liste sans solution : pire cas).
4. Remplis un tableau et dis comment chaque méthode réagit quand n double.

Mini-boss · réponse

Pistes de correction

```
def a(liste, s):
    ops = 0
    for i in range(len(liste)):
        for j in range(i + 1, len(liste)):
            ops += 1
            if liste[i] + liste[j] == s:
                return True, ops
    return False, ops

def c(liste, s):
    ops, vus = 0, set()
    for x in liste:
        ops += 1
        if s - x in vus:
            return True, ops
        vus.add(x)
    return False, ops

def b(liste, s):
```

Mini-boss · réponse

Pistes de correction (suite)

```
    liste = sorted(liste)
    ops, i, j = len(liste), 0, len(liste) - 1     # coût du tri compté à part
    while i < j:
        ops += 1
        t = liste[i] + liste[j]
        if t == s:
            return True, ops
        if t < s:
            i += 1
        else:
            j -= 1
    return False, ops

for n in (100, 200, 400, 800):
    donnees = [2 * k for k in range(n)]          # que des pairs : S impaire impos
   sible
    print(n, a(donnees, 1)[1], b(donnees, 1)[1], c(donnees, 1)[1])
```

Mini-boss · réponse

Pistes de correction (suite)

Attendu : A quadruple quand n double ; B et C doublent environ (B ajoute le coût du tri).

Quiz éclair

Quiz éclair

1. On compte les opérations plutôt que les secondes parce que :

A. les secondes dépendent de la machine
B. les opérations sont plus longues
C. c'est obligatoire

2. Dans `somme(liste)`, n représente :

A. la valeur de la somme
B. le nombre d'éléments de la liste
C. la durée

3. Si un coût passe de 1 000 à 4 000 quand n double, il est :

A. constant

Quiz éclair

Quiz éclair (suite)

B. linéaire
C. quadratique

4. 🔁 *Révision HEF-000, mission 03* : Une condition `A ET B` est vraie quand :

A. A et B sont vrais
B. l'un des deux est vrai
C. aucun n'est vrai

5. 🔁 *Révision HEF-010, mission 08* : Un cas limite typique est :

A. la liste moyenne
B. la liste vide
C. le cas heureux

Quiz éclair · réponse

Réponses

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

Je réfléchis

1. Où, dans ta vie, une tâche double-t-elle de durée quand la quantité double ? Où quadruple-t-elle ?
2. Pourquoi le pire cas est-il plus utile à connaître que le meilleur cas ?

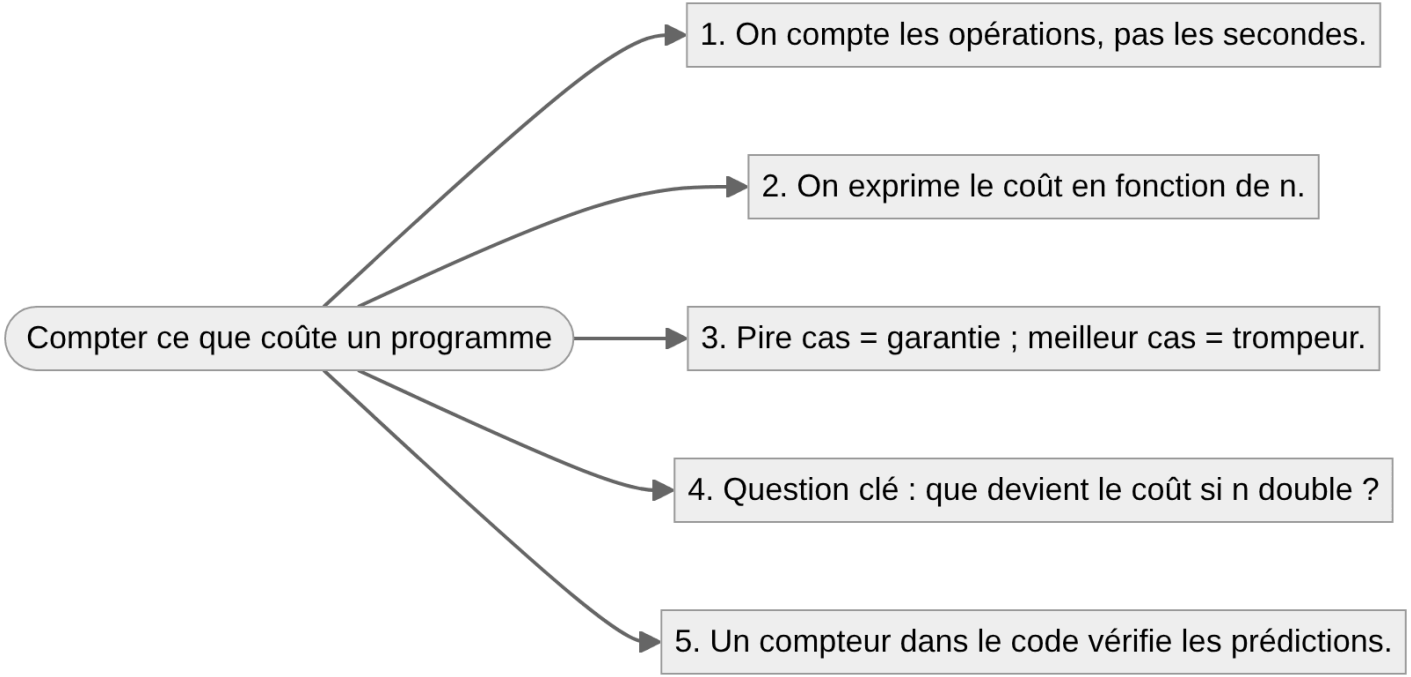
À retenir

À retenir

1. On compte les opérations, pas les secondes.
2. On exprime le coût en fonction de n.
3. Pire cas = garantie ; meilleur cas = trompeur.
4. Question clé : que devient le coût si n double ?
5. Un compteur dans le code vérifie les prédictions.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Comptable des opérations

Prochaine mission : La notation Big O

