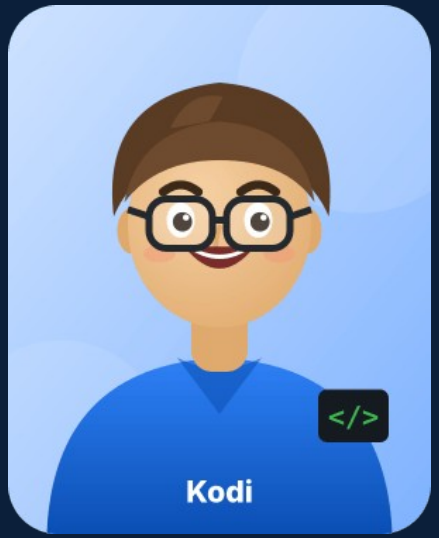
HEF-010 · MISSION 05 / 10

# Chercher efficacement

*Trouver une aiguille dans une botte de foin : méthode lente, ou méthode maligne ?*





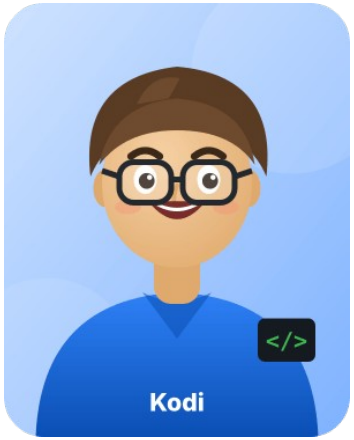


**Badge : Chasseur de trésors**

70 min · débutant

# L'histoire

La base HashBox compte maintenant 100 000 membres. Maya veut retrouver un membre par son numéro. Le programme de Kodi vérifie les membres un par un… et met plusieurs secondes.




> *« Si la liste est triée, tu n'as pas besoin de tout regarder, dit Sam. Joue au jeu du juste prix. »*

**Dans cette mission, tu apprends deux façons de chercher et quand utiliser chacune.**

Ta mission

# À la fin, je serai capable de…

1. écrire une recherche linéaire
2. expliquer la recherche binaire avec mes mots
3. déterminer quand la recherche binaire est possible
4. faire la trace d'une recherche binaire
5. compter les comparaisons effectuées

# Défi d'échauffement

Je pense à un nombre entre 1 et 100. Tu peux demander « est-ce plus grand que X ? » et je réponds oui ou non. **Quelle stratégie utilises-tu pour trouver mon nombre en un minimum de questions ?** Combien de questions faudrait-il au maximum, à ton avis ?

Le concept

# Recherche linéaire

On regarde les éléments **un par un** jusqu'à trouver. Ça marche toujours, même si la liste est dans le désordre.

```
def chercher(liste, cible):
    for i in range(len(liste)):
        if liste[i] == cible:
            return i
    return -1
```

Dans le pire cas, on regarde **tous** les éléments.

Le concept

# Recherche binaire

Si la liste est **triée**, on regarde le **milieu** :

- si c'est la cible, gagné ;
- si la cible est plus petite, on garde la moitié gauche ;
- sinon, la moitié droite.

À chaque étape, on **divise par deux** les possibilités. C'est exactement ta stratégie du jeu d'échauffement.

```
def chercher_binaire(liste, cible):
    bas, haut = 0, len(liste) - 1
    while bas <= haut:
        milieu = (bas + haut) // 2
```

Le concept

# Recherche binaire (suite)

```
        if liste[milieu] == cible:
            return milieu
        elif liste[milieu] < cible:
            bas = milieu + 1
        else:
            haut = milieu - 1
    return -1
```

## Combien de comparaisons ?

| Taille de la liste | Linéaire (pire cas) | Binaire (pire cas) |
|---|---|---|
| 100 | 100 | 7 |
| 1 000 | 1 000 | 10 |
| 1 000 000 | 1 000 000 | 20 |

Le concept

# Combien de comparaisons ? (suite)

Pour 1 million d'éléments, la binaire en fait environ 20.

⚠ ***Condition obligatoire*** *: la liste doit être triée. Sur une liste non triée, la recherche binaire donne des résultats faux.*

# Exemple pas à pas

**Problème** : chercher 23 dans `[3, 8, 11, 15, 19, 23, 31, 40]`.

| Étape | bas | haut | milieu | Valeur au milieu | Décision |
|---|---|---|---|---|---|
| 1 | 0 | 7 | 3 | 15 | 15 < 23 → à droite, `bas = 4` |
| 2 | 4 | 7 | 5 | 23 | trouvé ! |

**2 comparaisons** au lieu de 6 pour la recherche linéaire.

```
def chercher_binaire(liste, cible):
    bas, haut = 0, len(liste) - 1
    while bas <= haut:
        milieu = (bas + haut) // 2
        if liste[milieu] == cible:
            return milieu
        elif liste[milieu] < cible:
            bas = milieu + 1
```

Exemple pas à pas

# Exemple pas à pas (suite)

```
        else:
            haut = milieu - 1
    return -1

print(chercher_binaire([3, 8, 11, 15, 19, 23, 31, 40], 23))  # 5
print(chercher_binaire([3, 8, 11, 15, 19, 23, 31, 40], 12))  # -1
```

**Invariant** : si la cible est dans la liste, elle est entre `bas` et `haut`.

# À toi, avec aide

Fais la trace de `chercher_binaire([2, 5, 8, 12, 16, 23, 38, 56, 72, 91], 56)`.

Remplis : `bas`, `haut`, `milieu`, valeur au milieu, décision, pour chaque étape. Combien de comparaisons ?

À toi, avec aide · réponse

# Indices progressifs

## Indice 1

Départ : bas = 0, haut = 9. Le milieu est (0 + 9) // 2.

## Indice 2

Le milieu est 4, la valeur est 16. 16 < 56 : on va à droite.

## Indice 3

Recalcule `milieu` avec les nouvelles bornes.

À toi, avec aide · réponse

# Réponse

| Étape | bas | haut | milieu | Valeur | Décision |
|---|---|---|---|---|---|
| 1 | 0 | 9 | 4 | 16 | 16 < 56 → `bas = 5` |
| 2 | 5 | 9 | 7 | 56 | trouvé |

**2 comparaisons**, indice 7.

# À toi, en solo

Écris `premiere_position(liste_triee, cible)` : la recherche binaire doit renvoyer la **première** position de la cible quand elle apparaît plusieurs fois. Exemple : `[1, 2, 2, 2, 5]`, cible 2 → 1.
Indice de méthode : quand tu trouves la cible, ne t'arrête pas, continue vers la gauche.

À toi, en solo · réponse

# Pistes de correction

```
def premiere_position(liste_triee, cible):
    bas, haut = 0, len(liste_triee) - 1
    resultat = -1
    while bas <= haut:
        milieu = (bas + haut) // 2
        if liste_triee[milieu] == cible:
            resultat = milieu
            haut = milieu - 1      # on cherche encore plus à gauche
        elif liste_triee[milieu] < cible:
            bas = milieu + 1
        else:
            haut = milieu - 1
    return resultat

print(premiere_position([1, 2, 2, 2, 5], 2))  # 1
print(premiere_position([1, 2, 2, 2, 5], 3))  # -1
```

# Pièges classiques

| Piège | Pourquoi c'est un problème | Comment l'éviter |
|---|---|---|
| **Binaire sur liste non triée** | Résultat faux sans erreur | Vérifier ou trier d'abord |
| **Boucle infinie** | `bas` ou `haut` ne changent pas | Toujours `milieu + 1` ou `milieu - 1` |
| **Erreur de un** | `bas < haut` au lieu de `bas <= haut` rate le dernier élément | Tester avec 1 et 2 éléments |
| **Trier pour une seule recherche** | Le tri coûte plus que la recherche | Trier seulement si on cherche souvent |

Mini-boss

# Mini-boss

Compare les deux recherches sur le papier **et** en code.

1. Écris un compteur de comparaisons dans les deux fonctions.
2. Lance-les sur `list(range(0, 1000, 2))` en cherchant 998, 0, 501 (absent).
3. Remplis un tableau avec les comparaisons de chaque méthode.
4. Écris 3 phrases : quand choisis-tu l'une ou l'autre ?

Mini-boss · réponse

# Pistes de correction

```
def lineaire(liste, cible):
    comparaisons = 0
    for i in range(len(liste)):
        comparaisons += 1
        if liste[i] == cible:
            return i, comparaisons
    return -1, comparaisons

def binaire(liste, cible):
    bas, haut, comparaisons = 0, len(liste) - 1, 0
    while bas <= haut:
        milieu = (bas + haut) // 2
        comparaisons += 1
        if liste[milieu] == cible:
            return milieu, comparaisons
        elif liste[milieu] < cible:
            bas = milieu + 1
        else:
            haut = milieu - 1
```

Mini-boss · réponse

# Pistes de correction (suite)

```
    return -1, comparaisons

donnees = list(range(0, 1000, 2))
for c in (998, 0, 501):
    print(c, lineaire(donnees, c), binaire(donnees, c))
```

Conclusion attendue : linéaire si les données sont non triées, petites, ou cherchées une seule fois ; binaire si triées et consultées souvent.

Quiz éclair

# Quiz éclair

**1.** La recherche binaire exige :

A. une liste courte
B. une liste de textes
C. une liste triée

**2.** À chaque étape, la recherche binaire :

A. divise l'espace de recherche par deux
B. regarde un élément de plus
C. trie la liste

**3.** Pour 1 000 000 d'éléments, la binaire fait environ :

A. 1 000 comparaisons

# Quiz éclair (suite)

B. 20 comparaisons
C. 500 000 comparaisons

**4.** 🔁 *Révision mission 04* : Un cas de base sert à :

A. accélérer le code
B. nommer la fonction
C. arrêter la récursion

**5.** 🔁 *Révision mission 03* : Un invariant est :

A. une propriété vraie à chaque tour
B. une variable
C. un test

Quiz éclair · réponse

# Réponses

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

# Je réfléchis

1. Quelle était ta stratégie dans le jeu d'échauffement ? Ressemble-t-elle à la recherche binaire ?
2. Dans la vraie vie, où utilises-tu spontanément une recherche « par moitiés » (dictionnaire, annuaire) ?

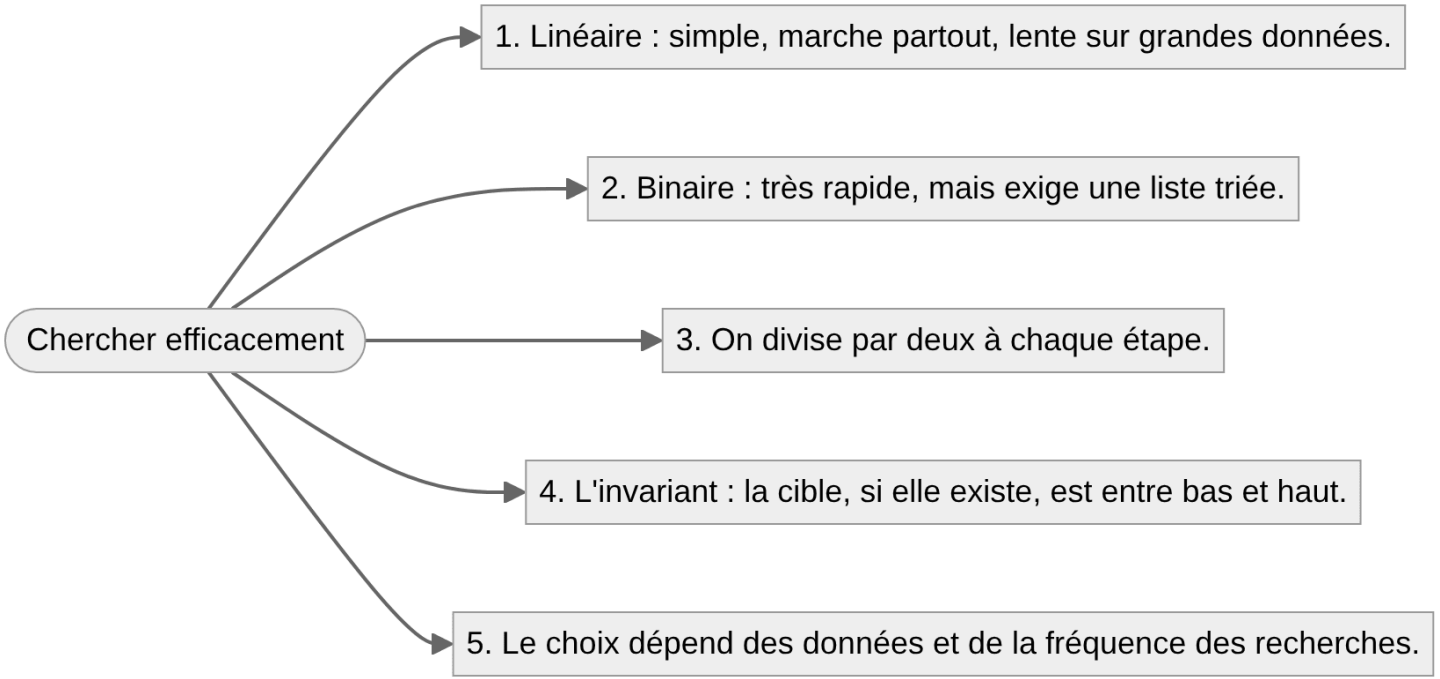
À retenir

# À retenir

1. Linéaire : simple, marche partout, lente sur grandes données.
2. Binaire : très rapide, mais exige une liste triée.
3. On divise par deux à chaque étape.
4. L'invariant : la cible, si elle existe, est entre `bas` et `haut`.
5. Le choix dépend des données et de la fréquence des recherches.

À retenir

# Le schéma de la mission



MISSION ACCOMPLIE

# +100 XP

## Badge débloqué : Chasseur de trésors

Prochaine mission : Mettre de l'ordre : les tris

