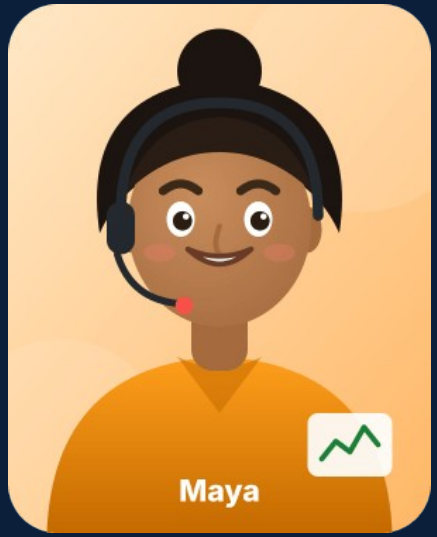
HEF-010 · MISSION 06 / 10

# Mettre de l'ordre : les tris

*Trier, c'est rendre la recherche facile. Il existe plusieurs méthodes, avec des qualités différentes.*





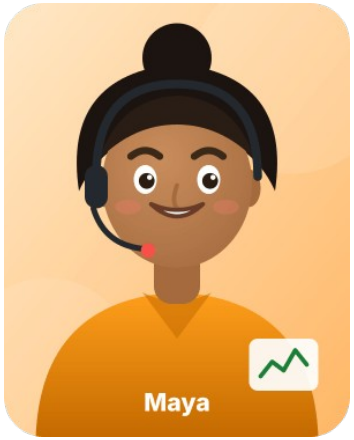


**Badge : Rangeur·se de génie**

75 min · débutant

# L'histoire

Pour utiliser la recherche binaire de la dernière mission, Kodi doit **trier** 100 000 membres. Il demande à Sam : « Comment on fait ? »




> *« Tu peux le faire de dix façons, dit Sam. Prenons les trois plus simples. Tu vas comprendre ce qu'elles coûtent. »*

**Dans cette mission, tu codes trois tris classiques et tu apprends à les comparer.**

Ta mission

# À la fin, je serai capable de…

1. expliquer le tri par sélection et par insertion
2. coder un tri simple
3. faire la trace d'un tri étape par étape
4. expliquer ce qu'est un tri stable
5. reconnaître que les tris simples deviennent lents sur de grandes listes

Défi d'échauffement

# Défi d'échauffement

Tu as un jeu de 8 cartes mélangées dans la main. **Comment les ranges-tu par ordre croissant ?** Décris ton geste, étape par étape (surtout, pas de « je les vois et je les place »). Combien de mouvements fais-tu environ ?

Le concept

# Tri par sélection

Principe : on cherche le **plus petit** élément de la partie non triée, on le place au début, puis on recommence sur le reste.

```
def tri_selection(liste):
    liste = list(liste)          # copie : on ne modifie pas l'original
    for i in range(len(liste)):
        mini = i
        for j in range(i + 1, len(liste)):
            if liste[j] < liste[mini]:
                mini = j
        liste[i], liste[mini] = liste[mini], liste[i]
    return liste
```

Le concept

# Tri par insertion

Principe : comme avec des cartes. On prend un élément et on l'**insère à sa place** dans la partie déjà triée.

```
def tri_insertion(liste):
    liste = list(liste)
    for i in range(1, len(liste)):
        courant = liste[i]
        j = i - 1
        while j >= 0 and liste[j] > courant:
            liste[j + 1] = liste[j]    # on décale vers la droite
            j = j - 1
        liste[j + 1] = courant
    return liste
```

Le concept

# L'invariant des deux tris

À chaque tour, la partie **à gauche de i est déjà triée**. À la fin, tout est trié.

## Tri stable

Un tri est **stable** si deux éléments égaux gardent leur ordre d'origine. Utile pour trier d'abord par nom, puis par ville : les noms restent ordonnés dans chaque ville. L'insertion est stable. La sélection (telle qu'écrite ici) ne l'est pas.

## Le coût

Pour n éléments, ces tris font de l'ordre de n × n opérations. 1 000 éléments : un million d'opérations. 100 000 éléments : dix milliards. **C'est pour cela qu'on utilise des tris plus malins** (fusion, tri rapide), que tu verras avec la complexité.

Le concept

# Le coût (suite)

Dans la vraie vie, tu utilises `sorted(liste)` en Python, qui est rapide et stable. Ici, on code pour **comprendre**.

Exemple pas à pas

# Exemple pas à pas

### Tri par insertion de `[5, 2, 4, 1]`

| Tour | Élément pris | Liste après insertion |
|---|---|---|
| départ | : | `[5, 2, 4, 1]` |
| i=1 | 2 | `[2, 5, 4, 1]` |
| i=2 | 4 | `[2, 4, 5, 1]` |
| i=3 | 1 | `[1, 2, 4, 5]` |

À chaque tour, la partie gauche est triée.

```
def tri_insertion(liste):
    liste = list(liste)
    for i in range(1, len(liste)):
        courant = liste[i]
```

# Exemple pas à pas (suite)

```
        j = i - 1
        while j >= 0 and liste[j] > courant:
            liste[j + 1] = liste[j]
            j = j - 1
        liste[j + 1] = courant
    return liste

print(tri_insertion([5, 2, 4, 1]))   # [1, 2, 4, 5]
```

**Observation** : si la liste est déjà presque triée, l'insertion fait très peu de décalages. Elle est excellente dans ce cas.

À toi, avec aide

# À toi, avec aide

Fais la trace du **tri par sélection** sur `[64, 25, 12, 22, 11]`. À chaque tour, écris quel est le minimum trouvé et l'état de la liste.

# Indices progressifs

## Indice 1

Tour 0 : le minimum de toute la liste est 11. Il va en position 0.

## Indice 2

Après l'échange du tour 0 : `[11, 25, 12, 22, 64]`.

## Indice 3

Tour 1 : le minimum de `[25, 12, 22, 64]` est 12.

À toi, avec aide · réponse

# Réponse

| Tour | Minimum trouvé | Liste après échange |
|---|---|---|
| 0 | 11 | `[11, 25, 12, 22, 64]` |
| 1 | 12 | `[11, 12, 25, 22, 64]` |
| 2 | 22 | `[11, 12, 22, 25, 64]` |
| 3 | 25 | `[11, 12, 22, 25, 64]` |
| 4 | 64 | `[11, 12, 22, 25, 64]` |

# À toi, en solo

Écris `tri_bulles(liste)` : on parcourt la liste en échangeant les voisins mal rangés, et on recommence tant qu'il y a eu un échange. Renvoie une nouvelle liste triée.

Puis réponds : pourquoi s'arrête-t-il si la liste est déjà triée dès le premier passage ?

À toi, en solo · réponse

# Pistes de correction

```
def tri_bulles(liste):
    liste = list(liste)
    echange = True
    while echange:
        echange = False
        for i in range(len(liste) - 1):
            if liste[i] > liste[i + 1]:
                liste[i], liste[i + 1] = liste[i + 1], liste[i]
                echange = True
    return liste

print(tri_bulles([5, 1, 4, 2, 8]))  # [1, 2, 4, 5, 8]
print(tri_bulles([1, 2, 3]))        # [1, 2, 3]
```

Si aucun échange n'a lieu pendant un passage, echange reste False : la liste est triée, la boucle s'arrête.

# Pièges classiques

| Piège | Pourquoi c'est un problème | Comment l'éviter |
|---|---|---|
| **Modifier la liste d'origine** | Effet de bord surprenant | Travailler sur une copie `list(liste)` |
| **Mauvaises bornes de boucle** | Erreur d'index ou élément oublié | Tester avec 0, 1, 2 éléments |
| **Oublier l'échange** | Éléments dupliqués ou perdus | Utiliser l'échange `a, b = b, a` |
| **Utiliser un tri lent sur de grandes données** | Programme très lent | Utiliser `sorted()` ou un tri plus efficace |

Mini-boss

# Mini-boss

Tu dois trier des membres `[("Léa", 3), ("Omar", 1), ("Sara", 3), ("Zoé", 2)]` par nombre de missions **décroissant**, et à égalité garder l'ordre d'origine.

1. Écris une version de `tri_insertion` qui prend une fonction `cle` (par exemple `cle=lambda m: m[1]`) et un paramètre `decroissant`.
2. Vérifie que le résultat est `[("Léa", 3), ("Sara", 3), ("Zoé", 2), ("Omar", 1)]`.
3. Explique pourquoi la stabilité est importante ici.

Mini-boss · réponse

# Pistes de correction

```
def tri_insertion(liste, cle=lambda x: x, decroissant=False):
    liste = list(liste)
    for i in range(1, len(liste)):
        courant = liste[i]
        j = i - 1
        while j >= 0 and ((cle(liste[j]) < cle(courant)) if decroissant else (cle(
   liste[j]) > cle(courant))):
            liste[j + 1] = liste[j]
            j = j - 1
        liste[j + 1] = courant
    return liste

membres = [("Léa", 3), ("Omar", 1), ("Sara", 3), ("Zoé", 2)]
print(tri_insertion(membres, cle=lambda m: m[1], decroissant=True))
```

On ne décale que si l'élément est **strictement** plus petit (ou plus grand) : les égaux ne se croisent jamais, donc Léa reste avant Sara.

Quiz éclair

# Quiz éclair

**1.** Le tri par insertion fonctionne :

A. en cherchant le plus grand
B. comme on range des cartes dans sa main
C. au hasard

**2.** Un tri stable :

A. est toujours le plus rapide
B. ne modifie pas la liste
C. garde l'ordre des éléments égaux

**3.** Pour n éléments, les tris simples font environ :

A. n × n opérations

# Quiz éclair (suite)

B. n opérations
C. 2 opérations

**4.** 🔁 *Révision mission 05* : La recherche binaire exige :

A. une liste courte
B. une liste triée
C. des nombres

**5.** 🔁 *Révision mission 04* : Un cas de base arrête :

A. la compilation
B. le tri
C. la récursion

Quiz éclair · réponse

# Réponses

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

# Je réfléchis

1. Ton geste du début ressemble-t-il à la sélection ou à l'insertion ?
2. Pourquoi vaut-il mieux utiliser `sorted()` en production ?

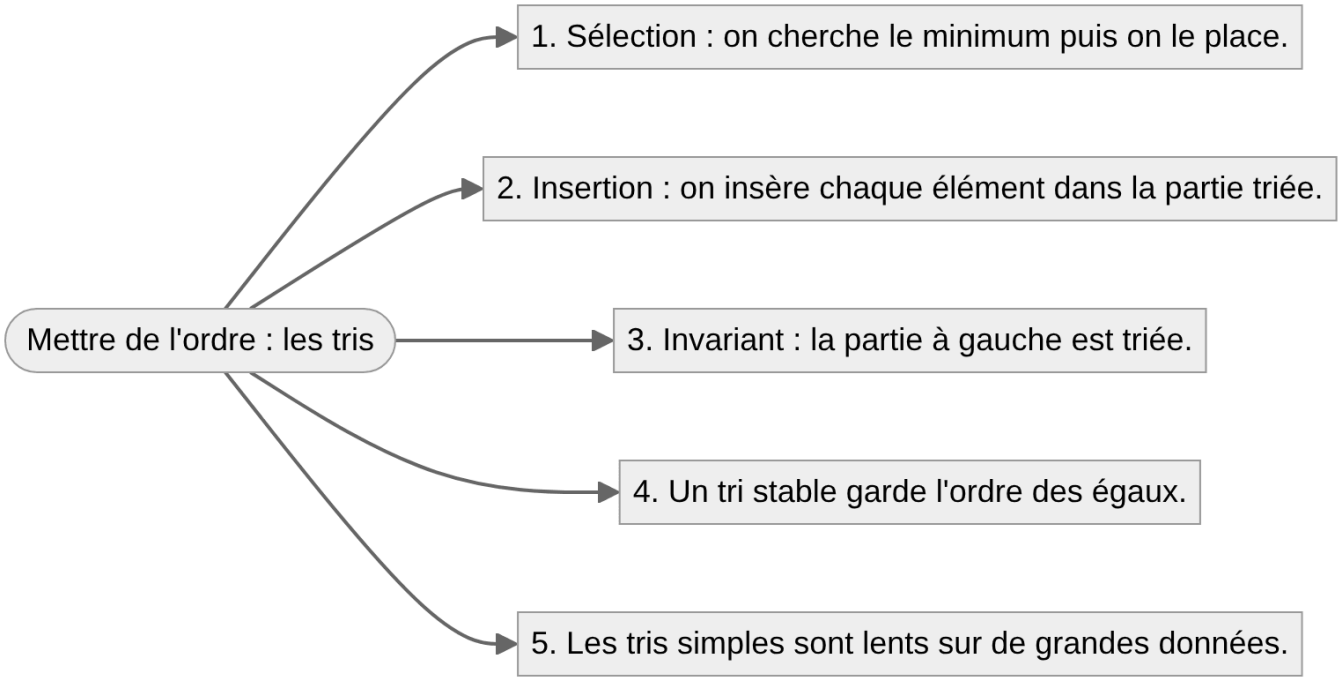
À retenir

# À retenir

1. Sélection : on cherche le minimum puis on le place.
2. Insertion : on insère chaque élément dans la partie triée.
3. Invariant : la partie à gauche est triée.
4. Un tri stable garde l'ordre des égaux.
5. Les tris simples sont lents sur de grandes données.

À retenir

# Le schéma de la mission



MISSION ACCOMPLIE

# +100 XP

## Badge débloqué : Rangeur·se de génie

Prochaine mission : Glouton et diviser pour régner

