Algorithme de Tri

1. **Qu'est-ce qu'un algorithme de tri ?**

   Un algorithme de tri est un algorithme qui organise les éléments d'une liste dans un ordre particulier. L'objectif est de placer les éléments dans un ordre croissant, décroissant ou selon un critère spécifique, comme lexicographique ou numérique. Les algorithmes de tri sont utilisés dans de nombreuses applications informatiques pour organiser les données de manière efficace afin de faciliter leur recherche, leur traitement ou leur présentation.

2. **Différents types d'algorithmes de tri :**

   Il existe de nombreux algorithmes de tri, chacun avec ses propres avantages et inconvénients en termes de vitesse d'exécution, utilisation de la mémoire et complexité. Voici quelques-uns des types d'algorithmes de tri les plus courants :

Tri par insertion

Principe **:** Cet algorithme consiste à insérer chaque élément de la liste dans la partie déjà triée de la liste, de manière à ce que la partie triée reste triée à chaque étape.

Exemple **:**
Considérons la liste
`[5, 2, 4, 6, 1, 3]`. Le tri par insertion fonctionnerait comme suit :

- L'élément `2` est inséré dans la partie triée `[5]`, devenant `[2, 5, 4, 6, 1, 3]`.
- Ensuite, `4` est inséré après `2`, devenant `[2, 4, 5, 6, 1, 3]`.
- Ce processus se poursuit jusqu'à ce que toute la liste soit triée.

```
Algorithme tri_insertion(liste):
    Pour chaque élément de la liste à partir de l'indice 1
        élément_actuel <- élément à cet indice
        index <- indice précédent à celui de l'élément_act
        Tant que l'index >= 0 et élément_actuel < liste[in
            Déplacer liste[index] vers la droite
```

```
            Décrémenter l'index
        Placer élément_actuel à la position index + 1
```

Tri par sélection :

Principe : Cet algorithme sélectionne successivement l'élément le plus petit et le place à sa position correcte dans le tableau.

Exemple :
Prenons la liste
`[64, 25, 12, 22, 11]`. Le tri par sélection fonctionnerait comme suit :

- Il trouve le plus petit élément (`11`) et l'échange avec le premier élément.
- Ensuite, il trouve le deuxième plus petit élément (`12`) et l'échange avec le deuxième élément, et ainsi de suite jusqu'à ce que la liste soit triée.

```
Algorithme tri_selection(liste):
    Pour chaque élément de la liste:
        min_index <- index de l'élément actuel
        Pour chaque élément restant de la liste:
            Si cet élément est plus petit que liste[min_in
                min_index <- index de cet élément
        Échanger l'élément actuel avec liste[min_index]
```

Tri à bulles :

Principe : Cet algorithme compare les éléments adjacents et les échange s'ils ne sont pas dans le bon ordre. Ce processus est répété jusqu'à ce que la liste soit entièrement triée.

Exemple :
Prenons la liste
`[5, 1, 4, 2, 8]`. Le tri à bulles fonctionnerait comme suit :

- Il compare les paires adjacentes et les échange si nécessaire. Après la première itération, la plus grande valeur (`8`) est déplacée à la fin.
- Ce processus se répète jusqu'à ce que tous les éléments soient triés.

```
Algorithme tri_bulles(liste):
    Pour i allant de 0 à longueur(liste) - 1:
```

```
        Pour j allant de 0 à longueur(liste) - i - 1:
            Si liste[j] > liste[j + 1]:
                Échanger liste[j] et liste[j + 1]
```

Tri par fusion :

Principe : Cet algorithme divise récursivement la liste en deux moitiés, trie chaque moitié séparément, puis fusionne les deux moitiés triées pour former une seule liste triée.

Exemple :
Prenons la liste
`[38, 27, 43, 3, 9, 82, 10]`. Le tri par fusion fonctionnerait comme suit :

- Divisez la liste en deux moitiés `[38, 27, 43, 3]` et `[9, 82, 10]`.
- Triez chaque moitié séparément. Après le tri, ils deviendraient `[3, 27, 38, 43]` et `[9, 10, 82]`.
- Fusionnez les deux moitiés triées pour obtenir la liste triée complète.

```
Algorithme tri_fusion(liste):
    Si la longueur de la liste est inférieure ou égale à 1
        Retourner la liste
    Diviser la liste en deux moitiés
    Trier récursivement les deux moitiés
    Fusionner les deux moitiés triées
```

Tri rapide (Quick Sort) :

Principe : Cet algorithme choisit un élément pivot, divise la liste en deux parties autour du pivot, trie récursivement les deux parties, puis concatène les parties triées avec le pivot entre elles.

Exemple :
Prenons la liste
`[10, 7, 8, 9, 1, 5]`. Le tri rapide fonctionnerait comme suit :

- Choisissez un pivot, par exemple `7`.
- Divisez la liste en deux parties `[1, 5]` et `[10, 8, 9]` autour du pivot.
- Triez récursivement chaque partie. Après le tri, elles deviendraient `[1, 5]` et `[8, 9, 10]`.

- Concaténez les parties triées avec le pivot entre elles pour obtenir la liste triée complète.

```
Algorithme tri_rapide(liste):
    Si la longueur de la liste est inférieure ou égale à 1
        Retourner la liste
    Choisir un pivot
    Diviser la liste en deux parties autour du pivot
    Trier récursivement les deux parties
    Concaténer les parties triées avec le pivot entre elle
```

Tri par tas :

Principe : Cet algorithme utilise une structure de données de tas pour organiser les éléments. Il convertit initialement le tableau en un tas, puis répète le processus d'extraction du maximum pour obtenir les éléments triés.

Exemple :
Considérons la liste
`[4, 10, 3, 5, 1]`. Le tri par tas fonctionnerait comme suit :

- Convertissez d'abord la liste en un tas max. Après cela, la liste ressemblerait à `[10, 5, 3, 4, 1]`.
- Répétez l'étape d'extraction du maximum jusqu'à ce que le tas soit vide. Cela donnerait `[1, 3, 4, 5, 10]`, qui est la liste triée.

```
Algorithme tri_par_tas(liste):
    Construire un tas max à partir de la liste
    Pour i allant de la longueur(liste) - 1 à 0:
        Échanger liste[0] avec liste[i]
        Réduire la taille du tas de 1
        Entasser_max(liste, 0)
```

Tri par comptage :

Principe : Cet algorithme convient particulièrement lorsque les données à trier sont dans un petit intervalle entier. Il compte le nombre d'occurrences de chaque élément distinct dans la liste, puis reconstruit le tableau trié.

Exemple :
Prenons la liste
`[4, 2, 2, 8, 3, 3, 1]`. Le tri par comptage fonctionnerait comme suit :

- Comptez le nombre d'occurrences de chaque élément. Vous obtiendrez `[1: 1, 2: 2, 3: 2, 4: 1, 8: 1]`.
- Reconstruisez la liste triée à partir de ces comptages. Vous obtiendrez `[1, 2, 2, 3, 3, 4, 8]`.

```
Algorithme tri_comptage(liste):
    Trouver la valeur maximale dans la liste
    Initialiser un tableau de comptage de la taille de la
    Pour chaque élément de la liste:
        Incrementer le compteur correspondant à cet élémen
    Pour chaque index dans le tableau de comptage:
        Pour chaque occurrence de cet index:
            Ajouter l'index à la liste dans l'ordre
```

Tri radix (ou tri par base) :

Principe : Cet algorithme trie les éléments en traitant les chiffres individuels ou les positions des chiffres.

Exemple :
Prenons la liste
`[170, 45, 75, 90, 802, 24, 2, 66]`. Le tri radix fonctionnerait comme suit :

- Triez les éléments selon leur chiffre des unités (0 à 9).
- Ensuite, triez-les selon leur chiffre des dizaines.
- Répétez ce processus pour chaque position de chiffre jusqu'à ce que toute la liste soit triée.

```
Algorithme tri_radix(liste):
    Trouver le nombre maximum de chiffres dans la liste
    Pour i allant de 1 à ce nombre maximum:
        Utiliser un tri de comptage stable pour trier la l
```