Python – Le tri fusion

Fusion de deux tableaux triées

Le cœur du tri par fusion repose sur la fusion de tableaux triés.

Exercice 1 : Compléter la fonction `fusion(tab1, tab2)` qui prend deux tableaux triés et renvoie un tableau trié correspondant à leur fusion. Le principe est le suivant:

1. On crée un tableau `tab` dont la longueur est la somme de celle des deux autres.
2. On note `i1` l'indice pour parcourir `tab1` et `i2` celui pour `tab2`.
3. Si `tab1[i1] < tab[i2]`, on copie `tab1[i1]` en position `i1+i2` dans `tab` et on avance `i1`.
4. Sinon on copie `tab2[i2]` et on avance `i2`.
5. Dès qu'on est arrivé à la fin d'un des deux tableaux, on copie les valeurs restantes de l'autre.

```
def fusion(tab1, tab2):
    n1 = len(tab1)
    n2 = len(tab2)
    tab = [0] * ...
    i1 = 0
    i2 = 0
    # On continue jusqu'à avoir fini un des tableaux
    while ...:
        if tab1[i1] < tab2[i2]:
            tab[i1+i2] = ...
            ... += 1
        else:
            tab[i1+i2] = ...
            ... += 1
    # On copie le reste de tab1 si nécessaire
    while ...:
        tab[i1+i2] = tab1[i1]
        i1 += 1
    # On copie le reste de tab2 si nécessaire
    while ...:
        tab[i1+i2] = tab2[i2]
        i2 += 1
    return tab
```

```
>>> fusion([1, 2, 3], [4, 5, 6])
[1, 2, 3, 4, 5, 6]
>>> fusion([1, 20, 30], [4, 5, 6])
[1, 4, 5, 6, 20, 30]
>>> fusion([1, 20, 30], [4, 5, 60])
[1, 4, 5, 20, 30, 60]
>>> fusion([10, 20, 30], [4, 5, 60])
[4, 5, 10, 20, 30, 60]
```

Tri fusion

Le tri fusion est un algorithme de tri récursif qui fonctionne de la manière suivante :

- Si le tableau est de longueur 1, ou moins, on le renvoie.
- Sinon, on fait une copie de la première moitié du tableau et une autre de la deuxième.
- On trie les deux sous-tableaux puis on les fusionne.

Exercice 2 : Compléter la fonction ci-dessous.

```
def tri_fusion(tab):
    n = len(tab)
    if ...:
        # On copie les tableaux
        tab1 = [tab[i] for i in range(0, n//2)]
        tab2 = [tab[i] for i in range(..., ...)]
        # On fusionne les deux tableaux triés
        return ...
    else:
        return tab
```

```
>>> tri_fusion([9, 4, 1, 5, 2, 6, 7, 8])
[1, 2, 4, 5, 6, 7, 8, 9]
>>> tri_fusion([1])
[1]
>>> tri_fusion([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> tri_fusion([])
[]
```

Comparaison avec les autres algorithmes de tri

Nous allons comparer cet algorithme de tri avec les algorithmes de tri par sélection et par insertion. Pour cela, il faut rajouter cela au début de votre fichier.

```
import random
import time
import matplotlib.pyplot as plt

def echange(tab, i, j):
    tab[i], tab[j] = tab[j], tab[i]
```

Exercice 3 : Compléter la fonction `tri_selection(tab)`.

```
def tri_selection(tab):
    n = len(tab)
    for i in range(...):
        m = i
        for j in range(..., n):
            if ... < tab[m]:
                m = ...
        if i != m:
            echange(tab, ..., ...)
```

Exercice 4 : Compléter la fonction `tri_insertion(tab)`.

```
def tri_insertion(tab):
    n = len(tab)
    for i in range(1, n):
        v = tab[i]
        j = ...
        while j > ... and ... > v:
            tab[j] = tab[...]
            j = ...
        tab[j] = ...
```

Pour comparer les temps de calcul, nous devons générer des tableaux aléatoirement :

```
def genere_tableau(n):
    return [random.random() for _ in range(n)]

def genere_tableau_presque_trie(n, k):
    t = list(range(n))
    for _ in range(k):
        # on fait k échanges au hasard
        echange(t, random.randint(0, n-1), random.randint(0, n-1))
    return t
```

Voici la fonction pour mesurer le temps d'exécution :

```
def temps_tri(t, algo_tri):
    debut = time.time()
    algo_tri(t)
    return time.time()-debut
```

```
>>> temps_tri(genere_tableau(1000), tri_fusion)
0.005249500274658203
>>> temps_tri(genere_tableau(10000), tri_fusion)
0.07259964942932129
>>> temps_tri(genere_tableau(100000), tri_fusion)
0.41570019721984863
>>> temps_tri(genere_tableau(1000000), tri_fusion)
4.687780380249023
```

Pour obtenir des résultats plus précis, il faut faire la moyenne sur plusieurs exécution de l'algorithme de tri.

Exercice 5 : Écrire une fonction `moyenne(liste)` qui renvoie la moyenne des valeurs d'une liste non vide.

```
>>> moyenne([1])
1.0
>>> moyenne([0, 10, 20, 30, 40])
20.0
>>> moyenne([0.5, 0.127, 0.1127, 0.983])
0.43067500000000003
```

La fonction `comparaison_temps(n, k)` compare le tri d'un tableau de `n` éléments. Le tri est refait `k` fois et on prend la moyenne pour chaque algotirhme de tri.

```python
def comparaison_temps(n, k, presque_trie=False):
    t_fusion = []
    t_selection = []
    t_insertion = []
    t_python = []
    for i in range(k):
        if presque_trie:
                t = genere_tableau_presque_trie(n, 10)
            else:
                t = genere_tableau(n)
        t_fusion.append(temps_tri(t.copy(), tri_fusion))
        t_selection.append(temps_tri(t.copy(), tri_selection))
        t_insertion.append(temps_tri(t.copy(), tri_insertion))
        t_python.append(temps_tri(t.copy(), sorted))
    print(f"Tri fusion        : {moyenne(t_fusion):.4}")
    print(f"Tri par sélection : {moyenne(t_selection):.4}")
    print(f"Tri par insertion : {moyenne(t_insertion):.4}")
    print(f"Tri python        : {moyenne(t_python):.4}")
```

Exercice 6 : Comparer le temps d'exécution avec `n` valant 1 000 et 10 000. Vous pouvez diminuer la valeur de `k` si le temps de calcul est trop long.

Exercice 7 : Refaire la même chose en prenant des tableaux presque triés pour comparer les différences de temps de calcul.

Nous allons rajouter la visualisation des temps de calcul avec des graphiques.

```python
def graphique_temps(n_max, k, points=10, presque_trie=False):
    t_taille = [] # pour les abscisses
    # Tableaux des ordonnées
    t_fusion = []
    t_selection = []
    t_insertion = []
    pas = n_max//points # distance entre les abscisses des points
    n = pas
    for _ in range(points):
        t_taille.append(n)
        # somme des temps de calcul pour des listes de taille n
        temps_fusion = 0
        temps_selection = 0
        temps_insertion = 0
        for _ in range(k):
            if presque_trie:
                t = genere_tableau_presque_trie(n, 10)
            else:
                t = genere_tableau(n)
            temps_fusion += temps_tri(t.copy(), tri_fusion)
            temps_selection += temps_tri(t.copy(), tri_selection)
            temps_insertion += temps_tri(t.copy(), tri_insertion)
        # On rajoute les temps moyens aux listes des ordonnées
        t_fusion.append(temps_fusion/k)
        t_selection.append(temps_selection/k)
```

```
        t_insertion.append(temps_insertion/k)
        n += pas
    # affichage des courbes
    plt.plot(t_taille, t_fusion, label="fusion")
    plt.plot(t_taille, t_selection, label="selection")
    plt.plot(t_taille, t_insertion, label="insertion")
    plt.legend()
    plt.show()
```

Exercice 8 : Comparer les temps d'exécution avec `graphique_temps(1000, 10, 10)`.