Lab Algorithmes de Tri

"Il faut faire de l'ordre avec du désordre." « Marc Caussidière »

I. Algorithmes de Tri

a. Tri par selection

Sur un tableau de n éléments (numérotés de 0 à n-1), le principe du tri par sélection est le suivant :

- rechercher le plus petit élément du tableau, et l'échanger avec l'élément d'indice 0 ;
- rechercher le plus petit élément de la portion du tableau comprise entre les indices 1 et n-1, et l'échanger avec l'élément d'indice 1 ...
- continuer de cette façon jusqu'à ce que le tableau soit entièrement trié.

Écrire la fonction `tri_selection(t)` qui tri le tableau `t` en utilisant le tri par sélection.

b. Tri à bulles

Le tri à bulles est un algorithme de tri qui consiste à faire remonter progressivement les plus grands éléments d'un tableau (comme des bulles d'air remontent à la surface d'un verre d'eau gazifiée).
Le tri à bulles est une succession d'étapes. Une étape du tri à bulles consiste à parcourir tous les éléments du tableau et à échanger dans le tableau l'élément courant avec l'élément suivant si l'élément courant est strictement plus petit que l'élément suivant.

Écrire une fonction `tri_a_bulles(t)` qui implémente le tri à bulles.

c. Tri par insertion

Le tri par insertion permet de trier une liste L d'éléments. Il consiste à ajouter un à un les éléments de L dans une liste R initialement vide, de sorte que la liste R soit toujours triée.

Implémenter la fonction `tri_insertion(t)` qui prend en paramètre un tableau `t` et qui renvoie un nouveau tableau trié contenant les éléments de `t`.

d. Tri rapide

Le principe du tri rapide consiste à choisir un élément p du tableau, appelé pivot, puis à trier le tableau en mettant les éléments plus petits que p à gauche de p et les éléments plus grands que p à droites de p. Ensuite, on recommence le processus sur le tableau de gauche d'une part (c'est à dire les éléments situés à gauche du pivot) et sur le tableau de droite d'autre part. L'algorithme s'arête quand le tableau est complètement trié.

Écrire une fonction `tri_rapide(t)` qui implémente le tri à rapide.

e. Tri fusion

Le tri fusion consiste à couper le tableau en 2 de tailles identiques (à un élément près), à trier le tableau de gauche en utilisant l'algorithme de tri fusion, à trier le tableau de droite avec le même algorithme, puis à fusionner les deux tableaux.

Écrire une fonction `tri_fusion(t)` qui implémente le tri fusion.

f. Tri par dénombrement

Ici, on suppose que tous les nombres sont compris entre 0 et M, où M est fixé. Cette contrainte supplémentaire va nous permettre d'optimiser cet algorithme de tri, pour peu que M soit assez petit.
Afin de trier un tableau t, le principe est le suivant :

- on crée un tableau `tiroirs` constitué de M+1 zéros ;
- on modifie le tableau `tiroirs` de manière à ce que `tiroirs[k]` soit égal au nombre d'éléments de valeur k dans le tableau .
- à l'aide du tableau `tiroirs`, on trie le tableau t en renvoyant un tableau contenant dans l'ordre :

`tiroirs[0]` 0, `tiroirs[1]` 1, `tiroirs[2]` 2, etc ...

Écrire une fonction `tri_denombrement(t,N)` qui implémente le tri par dénombrement.

II. Comparaison des temps d'éxecution

Afin de tester quel sont les méthodes de tri les plus performantes pour un tableau t donné, écrivez le script python qui réalise l'algorithme suivant :

```
Pour N dans [5,50,500,5000,50000,500000] faire
    T <= Creer_Tableau_Aleatoire(N,M)
    Pour Fonction dans Fonctions_de_tri :
        Temp <= copy(T)
        R <= Fonction(Temp)
    Fin Pour
Fin Pour
```

- La fonction **Creer_Tableau_Aleatoire(N,M)** est une fonction qui génére un tableau T de taille N remplis aléatoirement d'entier compris entre 0 et M.
- Ajoutez la fonction *sorted* native de Python dans la liste des Fonctions_de_tri.
- Récupérez le temps consommé par chaque exécution d'une fonction de tri.

Comparez les temps d'exécution. Peut-on-deviner quel algorithme Python utilise pour trier les tableaux ?