TD 4 - Quelques algorithmes de tri

Dans ce TP, les listes d'éléments à trier sont des listes d'entiers.

Vous devez implémenter et tester tout les programmes de ce TP en utilisant Python. Vous devez évaluer la complexité de chacun de vos algorithmes. Vous devez comparer par la théorie et par la pratique les temps d'exécution de vos différents algorithmes de tris.

Exercice 1: Le tableau est-il trié ?

Écrire une fonction `est_trie(t)`, qui prend en paramètre un tableau `t` et qui renvoie `true` si le tableau est trié et `false` sinon.

Exercice 2: Recherche dichotomique

Implémenter une fonction, de complexité optimale, qui prend en paramètre un tableau trié t et un élément e et qui renvoie -1 si l'élément e n'existe pas dans le tableau ou qui renvoie la position de sa première occurrence si il existe.

Tester l'algorithme avec le tableau `t=[1,3,3,5,6,7,7]` et la valeur `3`.

Exercice 3: Tri par sélection.

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 ;
- rechercher le plus petit élément de la portion du tableau comprise entre les indices 2 et n-1, et l'échanger avec l'élément d'indice 2 ;
- continuer de cette façon jusqu'à ce que le tableau soit entièrement trié.

Écrire une procédure `tri_selection(t)` qui tri le tableau `t` en utilisant le tri par sélection. Tester votre procédure sur plusieurs exemples. Combien d'échanges fait-il au maximum ? Ce résultat vous semble-t-il optimal ?

Exercice 4: Tri par insertion (ou tri du joueur de cartes)

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`.

Tester l'algorithme sur la liste `[2,7,1,2,8,7,5]`.

Implémenter une procédure `tri_insertion_en_continu()` qui demande à l'utilisateur des entiers, qui affiche au fur et à mesure la liste triée des entiers tapés par l'utilisateur et qui s'arrête quand l'utilisateur tape -1.

Exercice 5: 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 de champagne).

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. Par un exemple, voici ce que donne le tri à bulles sur la liste `[5,1,3,7,2]` :

Première étape :

$$[5, 1, 3, 7, 2] \to [1, 5, 3, 7, 2] \to [1, 3, 5, 7, 2] \to [1, 3, 5, 7, 2] \to [1, 3, 5, 2, 7]$$

Deuxième étape :

$$[1, 3, 5, 2, 7] \to [1, 3, 5, 2, 7] \to [1, 3, 5, 2, 7] \to [1, 3, 2, 5, 7]$$

Troisième étape :

$$[1, 3, 2, 5, 7] \to [1, 3, 2, 5, 7] \to [1, 2, 3, 5, 7]$$

Quatrième étape :

$$[1, 2, 3, 5, 7] \to [1, 2, 3, 5, 7]$$

Écrire une procédure `tri_a_bulles(t)` qui implémente le tri à bulles. Tester votre tableau sur plusieurs exemples.

Modifier votre procédure en une procédure `trace_bulles(t)` qui affiche avec la commande `print` les différentes étapes du tri à bulles, comme effectué sur l'exemple ci-dessus.

Exercice 6: 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 procédure `tri_rapide(t)` qui implémente le tri à rapide. Tester votre tableau sur plusieurs exemples.

Exercice 7: 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 procédure `tri_fusion(t)` qui implémente le tri fusion. Tester votre tableau sur plusieurs exemples.

Exercice 8: 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 t (le faire en une seule lecture du tableau t).
— 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_par_denombrement(t,N)` qui implémente le tri par dénombrement.

Exercice 9: Tri radix

Ici, on suppose que tous les nombres du tableau sont compris entre 0 et 10^{m-1}.

Écrire une fonction `convertir_liste(n,m)` qui transforme un entier n en un tableau à m éléments listant son écriture en base 10 (renvoyer une erreur si ce n'est pas possible). Exemple : `convertir_liste(823,4)` doit renvoyer `[0,8,3,2]`.

Écrire une fonction `convertir_entier(l)` qui transforme une liste l de chiffres compris entre 0 et 9 en un entier dont l'écriture en base 10 est donnée par l. Exemple : `convertir_entier([0,8,3,2])` doit renvoyer `832`.

Le principe du tri radix sur un tableau t est le suivant :
— remplacer les entiers du tableau t par leur écriture en base 10 donnée par `convertir_liste(n,m)` ;
— trier, en utilisant un tri stable, les nombres par rapport aux chiffres des unités ;
— trier, en utilisant un tri stable, les nombres par rapport aux chiffres des dizaines ;
— continuer à trier, toujours en utilisant des tris stables, décimale par décimale ;
— reconvertir t en une liste d'entiers Python.

Tester cet algorithme sur la papier avec la liste `[764,199,20,438,199]` (m=3). Expliquer pourquoi cet algorithme de tri marche. Écrire une fonction `tri_radix(t,m)` qui implémente le tri radix.

Exercice 10: Tri du singe

Un singe trie les cartes de la façon suivante : il prend les cartes, les jette en l'air, les ramasse puis regarde si elles sont triées. Si oui, il s'arrête, sinon il relance les cartes.

Implémenter et tester ce tri sur le tableau `t=[3,1,9,4,4]`.