CAPES MATHS OPTION INFORMATIQUE

ALGORITHMIQUES DE TRI

Test si liste triée

Avant de commencer à écrire des algorithmes de tri, on s'intéresse à tester si une liste est triée (pas besoin de la triée si elle l'est déjà). Donner la fonction Python qui teste si une liste passée en paramètre est triée et donner la complexité de cette fonction.

Tri par insertion

Le tri par insertion est l'algorithme utilisé par la plupart des joueurs lorsqu'ils trient leur « main » de cartes à jouer. Le principe consiste à prendre le premier élément du sous-tableau non trié et à l'insérer à sa place dans la partie triée du tableau.

a. Dérouler le tri par insertion du tableau [5.1, 2.4, 4.9, 6.8, 1.1, 3.0].

b. Ecrire en Python la procédure de tri par insertion, par ordre croissant, d'un tableau de réels :

> **Procédure** tri_par_insertion (tab : tableau de n réels)
> **Précondition** : tab[0], tab[1], … tab[n-1] initialisés
> **Postcondition** : tab[0] ≤ tab[1] ≤ … ≤ tab[n-1]

c. Un algorithme de tri est dit « stable » s'il préserve toujours l'ordre initial des ex-aequos. Dans notre exemple, l'algorithme est stable si des valeurs identiques restent dans leur ordre d'apparition avant le tri. L'algorithme de tri par insertion est-il stable ?

d. Donner l'invariant de boucle correspondant à cet algorithme, en démontrant qu'il vérifie bien les 3 propriétés d'un invariant de boucle : initialisation, conservation, terminaison.

e. Evaluer le nombre d'affectations de réels pour un tableau de taille *n*, dans le cas le plus défavorable (tableau trié dans l'ordre décroissant).

Tri par sélection

Le principe du tri par sélection est le suivant. Pour chaque élément i de 1 à n-1, on échange tab[i] avec l'élément minimum de tab[i..n]. Nous devons donc rechercher, pour chaque itération le minium d'un sous-tableau de plus en plus petit. Les éléments à gauche de i sont à leur place définitive et donc le tableau est complémentent trié lorsque i arrive sur l'vant dernier élément (le dernier élément étant forcément le plus grand

a. Ecrire en Python la procédure de tri par sélection, par ordre croissant, d'un tableau de réels :

b. Donner la complexité de cet algorithme de tri.

c. Donner l'invariant de boucle correspondant à cet algorithme, en démontrant qu'il vérifie bien les 3 propriétés d'un invariant de boucle : initialisation, conservation, terminaison.

Tri à bulles

Le tri à bulles est un algorithme de tri qui s'appuie sur des permutations répétées d'éléments contigus qui ne sont pas dans le bon ordre.

Procédure tri_bulles(tab : tableau [1..n] de réels)
Précondition : tab est un tableau contenant n réels
Postcondition : les éléments de tab sont triés dans l'ordre croissant
Variables locales : i, j : entiers, e : réel
Début

```
1   i ← 1
2   Tant que (i <= n-1) Faire
3       j← n
4       Tant que (j >= i+1) Faire
5           Si tab[j] < tab[j-1] Alors
6               {on permute les deux éléments}
7               e ← tab[j-1]
8               tab[j-1] ← tab[j]
9               tab[j] ← e
10          Fin Si
11          j ← j-1
12      Fin Tant que
13      i ← i + 1
14  Fin Tant que
```

Fin tri_bulles

a. Soit le tableau suivant : {53.8, 26.1, 2.5, 13.6, 8.8, 4.0}. Donnez les premiers états intermédiaires par lesquels passe ce tableau lorsqu'on lui applique la procédure tri_bulles.

b. Complétez la phrase suivante de sorte à ce qu'elle corresponde à l'invariant de boucle du Tant que interne (boucle sur j, lignes 4 à 12 de l'algorithme).

> « Lorsqu'on vient de décrémenter j (ligne 11), le _____ du sous-tableau tab[_____**...** _____] se trouve en position _____. De plus, si j>1, les éléments du sous-tableau tab[_____**...**_____] occupent les mêmes positions qu'avant le démarrage de la boucle sur j. »

c. Déduisez-en la propriété que présente le tableau lorsqu'on a terminé cette boucle interne, c'est-à-dire lorsqu'on arrive sur la ligne 13.

d. En utilisant la propriété précédente, on peut montrer que l'invariant de boucle du Tant que externe (boucle sur i) est le suivant : « Juste avant d'incrémenter i (ligne 13) : tab est trié entre les indices 1 et i, et tous les éléments restants sont supérieurs ou égaux à tab[i] ». Donnez la première étape de cette démonstration (initialisation). Les étapes de conservation et de terminaison ne sont pas demandées.

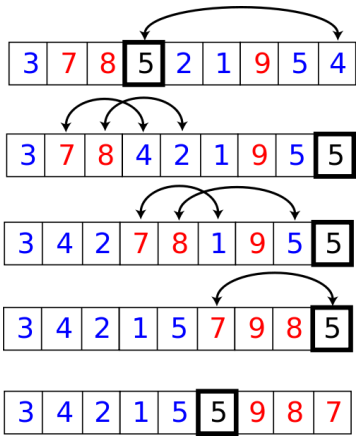
e. Calculez le nombre total d'affectations de réels réalisées par la procédure tri_bulles lors du tri complet d'un tableau de *n* réels, dans le cas le plus défavorable.

Tri rapide *(quick sort)*

Cette méthode de tri consiste à placer un élément du tableau (appelé pivot) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui sont inférieurs au pivot soient à sa gauche et que tous ceux qui sont supérieurs au pivot soient à sa droite. Cette opération s'appelle le *partitionnement*. Pour chacun des sous-tableaux, on définit un nouveau pivot et on répète l'opération de partitionnement. Ce processus est répété récursivement, jusqu'à ce que l'ensemble des éléments soit trié.

Concrètement, pour partitionner un sous-tableau :

- on place le pivot arbitrairement à la fin (peut être fait aléatoirement), en l'échangeant avec le dernier élément du sous-tableau (étape 1 ci-contre)
- on place tous les éléments inférieurs au pivot en début du sous-tableau (étapes 2 et 3)
- on place le pivot à la fin des éléments déplacés (étapes 4 et 5)



Ecrire en Python la procédure de partitionnement et la procédure récurvise de tri rapide.

Tri par fusion interne

a. Ecrire en Python une version **récurvise** de l'algorithme du tri par fusion d'un tableau de réels.

b. Ecrire en notation algorithmique une version **itérative** de l'algorithme du tri par fusion d'un tableau de réels.

Tri par comptage

Soit N un entier naturel non nul. On cherche à trier une liste L d'entiers naturels strictement inférieurs à N.

a. Ecrire une fonction comptage, d'arguments L et N, renvoyant une liste P de longueur N dont l'élément d'indice k désigne le nombre d'occurrences de l'entier k dans la liste L.

b. Utiliser la liste P pour déduire une fonction triComptage, d'arguments L et N, renvoyant une liste M triée dans l'ordre croissant.

c. Tester la fonction triComptage sur une liste de 20 entiers inférieurs ou égaux à 5, choisis aléatoirement.

d. Quelle est la complexité temporelle de cet algorithme ? La compararer à la complexité d'un tri par insertion ou d'un tri fusion.

Tri stupide

Analyser la complexité du tri suivant :

```
def triStupide (tab) :
   while not estTrie(tab) :
      melangeAleatoire(tab)
```

Où `estTrie` teste si le tableau est trié et `melangeAleatoire` mélange les éléments du tableau de manière aléatoire.

Tri par base (tri radix)

Dans le tri par base, on trie les éléments par clés selon l'ordre lexicographique. Avec des nombres, les éléments sont triés avec les clés définies comme les chiffres qui composent les nombres :

1. Prendre le chiffre le moins significatif comme clé
2. Trier (par comptage) les éléments
3. Répéter avec les chiffres plus significatifs

Exemple :

liste	6**1**	60**1**	11**1**	1**1**	**1**	6**0**	1**6**	**6**	6**6**
chiffre	0	1	2	3	4	5	6	7	8	9
compt. chiffre	1	5	0	0	0	0	3	0	0	0
compt. cumulé	1	6	6	6	6	6	9	9	9	9
liste	**6**0	**6**1	6**0**1	1**1**1	**1**1	**0**1	**1**6	**0**6	**6**6
chiffre	0	1	2	3	4	5	6	7	8	9
compt. chiffre	3	3	0	0	0	0	3	0	0	0
compt. cumulé	3	6	6	6	6	6	9	9	9	9
liste	**6**01	**0**01	**0**06	**1**11	**0**11	**0**16	**0**60	**0**61	**0**66
chiffre	0	1	2	3	4	5	6	7	8	9
compt. chiffre	7	1	0	0	0	0	1	0	0	0
compt. cumulé	7	8	8	8	8	8	9	9	9	9
liste	1	6	11	16	60	61	66	111	601	

a. Ecrire en Python une fonction `valeurBase` qui prend en paramètre deux nombres entiers positifs n et i et qui calcule et retourne la valeur du ième chiffre de n en partant des unités. Exemple valeurBase(61,0)=1 et valeurBase(11,2)=0.

b. Ecrire la fonction Python de tri par base.

c. Donner la complexité de ce tri.