DS 2

Informatique pour tous, deuxième année

Julien Reichert

Durée : 2 heures maximum.

Attention : La question de cours est sur six points, et un maximum de quatorze points peut être obtenu pour les autres exercices, valant chacun quatre points (barème prévisionnel).

Exercice 1 : Écrire un des algorithmes de tri au programme (insertion, sélection, bulles, fusion ou rapide), au choix dans sa version en place ou non en place. Prouver sa terminaison lorsqu'il est récursif ou contient une boucle conditionnelle ; calculer sa complexité dans le pire des cas par une analyse rigoureuse, ainsi que sa complexité dans le meilleur des cas en précisant pour quel genre de liste la complexité asymptotique est extrémale. Prouver aussi la correction de l'algorithme. En plus de cela, l'algorithme doit avoir un argument booléen précisant si le tri doit se faire dans l'ordre croissant ou non. La fonction `reversed` et la méthode `reverse` sont interdites (notamment en raison de conflits de types).

Exercice 2 : Écrire un programme établissant un classement général en tenant compte de possibles égalités. Le programme aura pour paramètre une liste de couples (chaîne de caractères,valeur) et imprimera les chaînes de caractères dans l'ordre décroissant des valeurs associées à raison d'une par ligne en commençant les lignes par le classement.

Exemple de résultat attendu :

```
>>> classement([('a',42),('b',57),('c',3),('d',20),('e',3)])

1 b
2 a
3 d
4 c
4 e
```

Exercice 3 : Écrire une fonction `tri_nity` qui trie (si possible en place) une matrice (liste de listes toutes de même taille). Le résultat doit être une liste de listes dont la lecture ligne par ligne donne une séquence croissante. Calculer la complexité de la fonction.

Exercice 4 : Écrire une fonction `tri_llian` qui trie une liste à ceci près que tous les `42` sont placés en tête de liste indépendamment du reste (qui est croissant, de son côté). Il est demandé que l'on ne copie nulle part la liste de départ.

Exercice 5 : Soient les trois algorithmes suivants, agissant sur une liste. Déterminer pourquoi les deux premiers ne sont pas des tris corrects, prouver que le troisième en est un, prouver également sa terminaison et calculer sa complexité.

On suppose déjà écrites une fonction `engendrer_transpositions(n)` qui retourne la liste des couples `(i,j)` pour `0` $\leq$ `i < j < n`, une fonction `melange(l)` qui mélange son argument de façon pseudo-aléatoire (il s'agit en fait de la fonction `shuffle` du module `random`, par exemple) et une fonction `croissante(l)` qui détermine si son argument est une liste croissante.

```
def petit_tri_ah_non(liste):
    transpo = engendrer_transpositions(len(liste))
    melange(transpo)
    while not (croissante(liste)):
        (i,j) = transpo.pop()
        liste[i], liste[j] = liste[j], liste[i]

def grand_tri_ah_non(liste):
    transpo = engendrer_transpositions(len(liste))
    melange(transpo)
    while not (croissante(liste)):
        (i,j) = transpo.pop()
        if liste[i] > liste[j]:
            liste[i], liste[j] = liste[j], liste[i]

def tri_castin(liste):
    transpo = engendrer_transpositions(len(liste))
    melange(transpo)
    k = 0
    while not (croissante(liste)):
        (i,j) = transpo[k]
        if liste[i] > liste[j]:
            liste[i], liste[j] = liste[j], liste[i]
            k = 0
        else:
            k += 1
```

Exercice 6 : Double dichotomie (plus difficile) Écrire un programme qui calcule en temps logarithmique l'indice de la première position et celui de la dernière position d'un élément dans une liste triée.