2I003 Complexité des listes, tris 5

Exercice 1 – Complexité du tri à bulles

On rappelle l'algorithme de tri à bulles :

```
def Push(T, k):
    j = 1
    while j < k:
        if T[j] < T[j - 1]:
            tmp = T[j - 1]
            T[j - 1] = T[j]
            T[j] = tmp
        j = j + 1

def BubbleSort(T):
    i = 0
    n = len(T)
    while i < n:
        Push(T, n - i)
        i = i + 1
```

On souhaite étudier la complexité de l'algorithme `BubbleSort` en nombre de comparaisons ($<$) entre éléments du tableau en fonction de la taille n de T.

Question 1

Pour $1 \leq k < n$, combien `Push(T,k)` effectue-t-il de comparaisons entre éléments du tableau ?

Question 2

Calculer la complexité de `BubbleSort`.

Question 3

Est-ce que le tri à bulle peut-être utlisé pour trier une liste simplement chaînée ou doublement chaînée ? Quelle est alors la complexité obtenue ?

Exercice 2 – Tri fusion récursif

Le but de cet exercice est d'étudier l'algorithme de tri fusion récursif d'une liste L. Le code est le suivant :

```
def TriFusionRec(L):
        print "Appel␣de␣TriFusion␣pour␣L=", L
        if (len(L)>1):
                moitie=len(L)/2
                L1 = TriFusionRec(L[ : moitie])
                L2 = TriFusionRec(L[moitie : ])
                L = fusion(L1,L2)
        print "Renvoi␣de␣L=", L
        return L
```

`fusion` est la fonction de fusion de deux listes vue en cours (cours 5 sur les tris).

Question 1
Exécuter `TriFusionRec(L)` pour la liste $L = (7, 9, 2, 1, 6, 4, 3, 1)$. Précisez l'arbre des appels.

Question 2
Démontrez la terminaison et la validité de la fonction `TriFusionRec(L)`.

Question 3
On suppose dans cette question que la liste L est une liste chaînée (simplement chaînée ou doublement chaînée circulaire). Quelle est la complexité du tri fusion récursif ?

Question 4
Est-ce que le tri fusion est envisageable pour trier un tableau ? Quelle est alors sa complexité ?

Exercice 3 – Tri rapide pour un tableau

On suppose pour cet exercice que `tab` est un tableau de n entiers. On considère la fonction `partition` dont le code suit :

```
def partition(tab,debut,fin):
    i = debut-1   # indice du plus petit element -1
    pivot = tab[fin]     # le pivot est le dernier element
    print "Entree␣:␣tab=", tab,"i=", i
    for j in range(debut,fin):
        if  (tab[j] <= pivot):
            i = i+1
            tab[i],tab[j] = tab[j],tab[i]
        print "tab=", tab,"i=", i, "j=", j
    tab[i+1],tab[fin] = tab[fin],tab[i+1]
    print "Sortie␣:␣tab=", tab,"i+1=", i+1
    return ( i+1 )
```

Question 1
Exécutez l'appel `partition(tab, 0, 6)` pour le tableau `tab=[7,5,9,2,1,6,4]`. Que fait cette fonction ?

Question 2
Exprimez l'invariant de boucle de la fonction `partition`. En supposant que cet invariant de boucle est correct, étudiez la sortie de boucle, et en déduire la validité de la fonction `partition`.

Question 3
Evaluez la complexité de la fonction `partition`.

Question 4
Le principe du tri rapide pour un tableau $tab[d \cdots f]$ avec $f - d > 0$ consiste à partitionner le tableau comme vu précédemment, puis à trier récursivement les deux sous-tableaux obtenus en observant que le pivot est bien à sa place. Le code suit :

```
def quickSort(tab,debut=0,fin=len(tab)-1):
    if debut< fin:
        print "Appel␣tab=", tab[debut:fin+1], "debut=", debut, "fin=", fin
        indPivot= partition(tab,debut,fin)
        quickSort(tab, debut,indPivot-1)
        quickSort(tab, indPivot+1, fin)
        print "Retour␣tab=", tab[debut:fin+1], "debut=", debut, "fin=", fin
```

Exécutez `quickSort(tab)` pour `tab=[7,5,9,2,1,6,4]`. Vous n'effectuerez que les affichages de `quickSort`. Donnez également l'arbre des appels avec les tableaux obtenus.

Question 5
Démontrez la validité et la terminaison de la fonction `quickSort`.

Question 6
Evaluez la complexité de la fonction `quickSort` dans le pire et le meilleur des cas.

Exercice 4 – Tri par paquets

On considère le tri par paquets dont le code suit :

```
def triPaquet(L, N):
        max = elemMax(L)
        K = int(ceil(float(max+1)/N))
        tab =[]
        for i in range(N):
                tab.append([])
        for elem in L:
                tab[elem//K].append(elem)
        for i in range(N):
                insertionSort(tab[i])
        R=[]
        for i in range(N):
                R=R+tab[i]
        return R
```

N est un entier positif strictement. La fonction `elemMax(L)` renvoie la valeur maximale d'un entier de L. La valeur K est initialisée à $K = \lceil \frac{max+1}{N} \rceil$. `range(N)` désigne la liste $(0, \cdots, N-1)$. La fonction `insertionSort(tab[i])` renvoie la liste $tab[i]$ triée par insertion. $+$ est l'opérateur de concaténation de deux listes.

Question 1
Exécuter cette fonction pour la liste $L = (2, 8, 4, 1, 5, 9, 6, 7, 3)$ et $N = 3$.

Question 2
Explicitez le fonctionnement de ce tri. S'agit-il d'un tri stable ?

Question 3
Comment doit-on implémenter L et tab pour que la complexité des opérations de manipulation de ces structures soit la meilleure possible. Quelle est alors la complexité dans le pire des cas du tri par paquets ? Dans le meilleur des cas ?

Exercice 5 – Tri Radix

Le but de cet exercice est d'étudier le tri Radix d'une liste d'entiers L.

Question 1
Soit $B \in \mathbb{N}^\star$ et un entier $x \in \mathbb{N}$. Soit $x = \sum_{i=0}^{n(x)-1} x_i B^i$ la décomposition de x en base B, où $n(x)$ désigne le nombre maximum de termes dans la décomposition et $x_i \in \{0, \cdots, B-1\}$. Montrez que $n(x) = \lceil \frac{\log(x+1)}{\log B} \rceil$.

Question 2
Montrez que, pour tout $i \in \{0, \cdots, n(x)-1\}$, $x_i = \lfloor \dfrac{x \bmod B^{i+1}}{B^i} \rfloor$.

Question 3
Soit maintenant la fonction `TriParUnite` dont le code suit. Ici, l'appel à `insertionSortBase(L, B, PB)` est une variation du tri par insertion sur les valeurs $\lfloor \frac{L[i] \text{ mod } (B * PB)}{PB} \rfloor$.

```
def TriParUnite(L, B):
        max = elemMax(L)
        nbIter =int(ceil( log(max+1)/log(B)))
        PB =1
        for i in range(nbIter):
                   L=insertionSortBase(L, B, PB)
                   PB = PB*B
        return L
```

Exécutez cette fonction sur la liste $L = (78, 34, 12, 169, 902, 99, 194)$ pour $B = 10$.

Question 4
Montrez que la fonction `TriParUnite(L, B)` trie les éléments de L. Est-ce que la stabilité du tri par insertion est importante ?

Question 5
Quelle est la complexité de ce tri (en fonction du tri utilisé à chaque itération) ? Est-il stable ? Est-ce un tri de comparaison ?

Question 6
Pour améliorer la complexité du tri par unité, nous allons maintenant remplacer l'appel à un algorithme de tri par un tri par paquets, en en prenant B. On obtient alors le tri suivant :

```
def TriRadix(L, B):
        max = elemMax(L)
        nbIter =int(ceil( log(max+1)/log(B)))
        PB =1
        for i in range(nbIter):
                tab =[]
                for j in range(B):
                        tab.append([])
                for elem in L:
                        index = (elem % (B*PB))//PB
                        tab[index].append(elem)
                L=[]
                for i in range(B):
                        L=L+tab[i]
                PB = PB*B
        return L
```

Exécutez cette fonction sur la liste $L = (78, 34, 12, 169, 902, 99, 194)$ pour $B = 10$.

Question 7
Quelle est la complexité de ce nouveau tri si tab est un tableau de listes doublement chaînées circulaires ? Est-il stable ? Est-ce un tri de comparaison ?

Exercice 6 – Tri par fusions - Extrait de l'examen de Mai 2013
Dans tout cet exercice, l'ordre considéré est l'ordre croissant ("trié" signifie donc "trié en ordre croissant").
On dit qu'une liste L est *triée par paquets de* k si les k premiers éléments sont triés, puis les k suivants et ainsi de suite jusqu'aux p derniers (avec $p \leq k$). Par exemple, la liste `maL=(4,5,1,3,8,9,1,2,6,11,7)` est triée par

paquets de 2 puisque les listes (4,5), (1,3), (8,9), (1,2), (6,11), (7) sont triées mais elle n'est pas triée par paquets de 3.

Remarques :

— toute liste est triée par paquets de 1 ;

— si une liste de taille n est triée par paquets de k avec $k \geq n$ alors elle est triée ;

— si une liste est triée alors elle est triée par paquets de k pour tout $k \geq 1$.

On rappelle la définition de la fonction `fusion` vue en cours :

```
def fusion(L1,L2):
  if (L1 == []):
    return L2
  if (L2 == []):
    return L1
  if (L1[0]<= L2[0]):
    R=fusion(L1[1: ], L2)
    R.insert(0, L1[0])
    return R
  R=fusion(L1, L2[1: ])
  R.insert(0, L2[0])
  return R
```

Notations : si $\texttt{L} = (a_0, \ldots, a_{n-1})$ alors $\texttt{L[i]} = a_i$, $\texttt{L[i:j]} = (a_i, \ldots, a_{j-1})$ et $\texttt{L[i:]} = (a_i, \ldots, a_{n-1})$

Rappels : la fonction `fusion` se termine et, si `L1` et `L2` sont deux listes triées, alors `fusion(L1,L2)` est une liste triée.

Question 1

Donner le résultat de la fusion des listes (1,5,8,2,12) et (3,7,4,9,15).

On considére la fonction `FK` ainsi définie :

```
def FK(k,L):
  if len(L) <= k:
    res = L
  elif len(L) <= 2*k:
    res = fusion(L[0:k],L[k:])
  else:
    res = fusion(L[0:k],L[k:2*k])+FK(k,L[2*k: ])
  print res
  return res
```

Question 2

On considére la liste maL=(4,5,1,3,8,9,1,2,6,11,7). Exécuter l'appel de FK(2,maL), en précisant les appels récursifs à `FK` et les messages successivement affichés.

Question 3

Montrer que FK(k,L) se termine.

Indication : faire un raisonnement par récurrence sur $|L|$.

Question 4

Montrer que, si L est triée par paquets de k alors FK(k,L) est composée des mÃªmes éléments que L et est triée par paquets de $2k$.

Indication : faire un raisonnement par récurrence sur $|L|$. Pour la base, considérer $|L| \leq k$ et $k < |L| \leq 2k$.

On considére la fonction `triK` ainsi définie :

```
def triK(L):
 k = 1
```

```
while k < len(L):
  L = FK(k,L)
  k = 2*k
return L
```

Question 5

On considére la liste `maL=(4,5,1,3,8,9,1,2,6,11,7)`. Exécuter l'appel de `triK(maL)`, en précisant les valeurs successives de k et de L.

On note L_i et k_i les valeurs de L et k à la fin de l'itération i. Initialement $L_0 = L$ et $k_0 = 1$.

Question 6

1. Montrer que, pour $i \geq 0$, à la fin de l'itération i, si elle existe, on a $k_i = 2^i$ et L_i triée par paquets de k.
2. En déduire que `triK(L)` se termine et renvoie la liste L triée.

Question 7

On suppose que la fusion de deux listes de tailles p et q est en $\mathcal{O}(p+q)$ et que la concaténation de deux listes est en $\mathcal{O}(1)$. On considére une liste L de taille n.

1. Soit m le nombre d'itérations dans `triK(L)`. Montrer que $2^{m-1} < n \leq 2^m$ et en déduire que $m < 1 + \log_2 n$.
2. Montrer que, pour une liste de taille n, `FK` est en $\mathcal{O}(n)$ et `triK` est en $\mathcal{O}(n \log n)$.