

## Algorithmique III. L2 Informatique I41.

### TD 8. Le tri fusion et le tri rapide<sup>1</sup>

**EXERCICE 1.** On considère la liste  $L = [3, 2, 1, 7, 5, 6, 9, 4, 8, 10, 3, 2, 8]$ . Appliquez pas-à-pas l'algorithme du [tri fusion](#) à la liste  $L$  en décomposant chaque étape de séparation mais pas les fusionnements.

**EXERCICE 2.** Écrivez un algorithme `Copier( $X, i, Y, j, n$ )` qui copie  $n$  valeurs consécutives du tableau  $X$  en partant de l'indice  $i$  dans le tableau  $Y$  à partir de l'indice  $j$ . La copie s'arrête si la fin de l'un ou l'autre des tableaux a été atteinte. Faites la preuve d'arrêt et la preuve de justesse de votre algorithme.

**EXERCICE 3.** On veut écrire une version itérative du [tri fusion](#). On suppose que la longueur du tableau  $T$  à trier est toujours une puissance de 2 (quitte à compléter les cases surnuméraires avec une valeur constante supposée strictement supérieure aux  $n$  valeurs utiles). L'algorithme consiste à fusionner les paires de sous-tableaux successifs de taille 1, 2, 4, 8, etc. (voir l'exemple en table 1).

taille 1	$T[4, 1, 7, 2, 5, 6, 8, 3]$
taille 2	$T[1, 4, 2, 7, 5, 6, 3, 8]$
taille 4	$T[1, 2, 4, 7, 3, 5, 6, 8]$
tableau trié	$T[1, 2, 3, 4, 5, 6, 7, 8]$

TABLE 1. Illustration du tri fusion itératif.

Écrivez l'algorithme `TriFusionIt(@T)` qui trie itérativement le tableau  $T$  de longueur  $n := 2^k$ . On utilisera sans l'écrire une variation de l'algorithme de fusionnement `Fusionner(@T, i, j)` qui fusionne les deux sous-tableaux adjacents  $T[i : j - 1]$  et  $T[j : 2j - (i + 1)]$  tous deux de taille  $j - i$ .

**EXERCICE 4.** On considère une liste  $L$  dont la structure de données associée  $L$  est une [liste chaînée](#), une liste est donc une *référence*, elle n'est que la clé pour accéder à l'objet référencé noté ici  $L$ . Les expressions  $L$ »val et  $L$ »suiv désignent respectivement la valeur contenue dans la cellule en tête de la liste  $L$  et la sous-liste suivante. On note  $[]$  la liste vide. Un *atome* désigne ici une liste  $A$  ne contenant qu'une cellule, i.e. telle que la liste suivante est vide.

(1) Écrivez un algorithme `Separer(@L)` qui renvoie la liste constituée par les cellules en positions paires extraites de la liste  $L$ . À l'issue de l'exécution de l'algorithme, la liste  $L$  ne contient plus que les cellules en positions impaires. On utilise, sans le définir, l'algorithme `InsTete(@L, A)` qui insère un atome  $A$  en tête de liste  $L$ , autrement dit  $A$ »suiv  $\leftarrow L$  puis  $L \leftarrow A$ . NB. La liste retour contiendra les valeurs dans l'ordre inverse d'apparition.

(2) Quelle est la complexité en espace et en temps de cet algorithme ?

**EXERCICE 5.** (1) On considère la liste  $L = [5, 2, 1, 7, 3, 6, 9, 5, 1, 4, 8, 6, 9]$ . Appliquez pas-à-pas l'algorithme [Partitionner](#) (rappelé ci-dessous) à la liste  $L$  en décomposant chaque étape. La liste est indexée en partant de 1.

(2) Quelle est la valeur renvoyée par l'algorithme ?

(3) Montrez qu'à la sortie de la boucle principale (#07), on a toujours  $j = i$  ou  $j = i - 1$ , dans le premier cas uniquement si  $L[j]$  égale la valeur du pivot. On admettra qu'après chaque échange #08 dans la boucle principale, la proposition suivante est satisfaite :

$$(i < j) \wedge (L[p:i] \leq x \leq L[j:r]). \quad (1)$$

Indication : étudiez le partitionnement des deux listes  $L = [4, 5, 2, 4, 1, 3]$  et  $L = [4, 5, 2, 4, 4, 1, 3]$ .

On rappelle l'algorithme `Partitionner` :

```
ALGORITHME Partitionner(@L, p, r):entier
DONNÉES
· L: liste de valeurs
· p, r: entiers
VARIABLES
· i, j, x: entiers
DEBUT
01> · x ← L[p]
```

1. Version du 5 juillet 2022, 12 : 19

```

02> · i ← p
03> · j ← r
04> · TQ (L[j] > x) FAIRE
05> ·   · j ← j - 1
06> · FTQ
07> · TQ (i < j) FAIRE
08> ·   · Echanger(L, i, j)
09> ·   · Reculer(L, x, j)
10> ·   · Avancer(L, x, i)
11> · FTQ
   · RENVOYER(j)
   FIN

```

Il utilise ici deux algorithmes auxiliaires *Reculer* et *Avancer* pour déplacer les deux sentinelles  $j$  et  $i$  le long de la liste :

ALGORITHME Reculer(L, x, @j)	ALGORITHME Avancer(L, x, @i)
DONNÉES	DONNÉES
· L: liste de valeurs	· L: liste de valeurs
· x: valeur	· x: valeur
· j: entier	· j:entier
DEBUT	DEBUT
· j ← j - 1	· i ← i + 1
· TQ (L[j] > x) FAIRE	· TQ (L[i] < x) FAIRE
·   · j ← j - 1	·   · i ← i + 1
· FTQ	· FTQ
FIN	FIN