Algorithmique IV (UE-41) - TD 8.

TD 8. Le tri fusion et le tri rapide [1]

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.

Solution. Dans la table 1, la sous-liste qui va être partitionnée est sur fond gris et la position de la séparation ($\longleftrightarrow$) est matérialisée par une barre verticale |. Les sous-listes qui vont fusionner ($\mapsto\!\!\leftarrow$) sont sur fond orange et bleu respectivement, puis jaune une fois fusionnées.

EXERCICE 2. Écrivez un algorithme Copier(X, i, Y, j, n) qui copie $X[i+k]$ dans $Y[j+k]$ pour $k \in [\![0, n-1]\!]$. La copie soit s'arrêter si la fin de l'un ou l'autre des tableaux X ou Y a été atteinte. Faites la preuve d'arrêt et la preuve de correction partielle de votre algorithme.

Solution. Sauf mention du contraire, les listes en algorithmique sont indexées de 1 à leur taille.

```
ALGORITHME Copier(X,i,@Y,j,n)
DONNEES
 · X,Y: listes
 · i,j,n: entiers
VARIABLE
 · k: entier
DEBUT
 · k ← 0
 · TQ ((i + k) <= #X) ET ((j + k) <= #Y) ET (k < n)) FAIRE
 ·  · Y[j + k] ← X[i + k]
 ·  · k ← k + 1
 · FINTQ
FIN
```

Algo. 1. Copie de sous-listes.

i	1	2	3	4	5	6	7	8	9	10	11	12	13
$\longleftrightarrow$	3	2	1	7	5	6	9	4	8	10	3	2	8
$\longleftrightarrow$	3	2	1	7	5	6	9	4	8	10	3	2	8
$\longleftrightarrow$	3	2	1	7	5	6	9	4	8	10	3	2	8
$\longleftrightarrow$	3	2	1	7	5	6	9	4	8	10	3	2	8
$\mapsto\!\!\leftarrow$	3	2	1	7	5	6	9	4	8	10	3	2	8
$\longleftrightarrow$	2	3	1	7	5	6	9	4	8	10	3	2	8
$\mapsto\!\!\leftarrow$	2	3	1	7	5	6	9	4	8	10	3	2	8
	2	3	1	7	5	6	9	4	8	10	3	2	8
$\mapsto\!\!\leftarrow$	2	3	1	7	5	6	9	4	8	10	3	2	8
$\longleftrightarrow$	1	2	3	7	5	6	9	4	8	10	3	2	8
$\longleftrightarrow$	1	2	3	7	5	6	9	4	8	10	3	2	8
$\mapsto\!\!\leftarrow$	1	2	3	7	5	6	9	4	8	10	3	2	8
	1	2	3	7	5	6	9	4	8	10	3	2	8
$\mapsto\!\!\leftarrow$	1	2	3	7	5	6	9	4	8	10	3	2	8
	1	2	3	7	5	6	9	4	8	10	3	2	8
$\mapsto\!\!\leftarrow$	1	2	3	7	5	6	9	4	8	10	3	2	8
$\longleftrightarrow$	1	2	3	5	6	7	9	4	8	10	3	2	8
$\longleftrightarrow$	1	2	3	5	6	7	9	4	8	10	3	2	8
$\longleftrightarrow$	1	2	3	5	6	7	9	4	8	10	3	2	8
$\mapsto\!\!\leftarrow$	1	2	3	5	6	7	9	4	8	10	3	2	8
	1	2	3	5	6	7	9	4	8	10	3	2	8
$\mapsto\!\!\leftarrow$	1	2	3	5	6	7	9	4	8	10	3	2	8
$\longleftrightarrow$	1	2	3	5	6	7	9	4	8	10	3	2	8
$\longleftrightarrow$	1	2	3	5	6	7	9	4	8	10	3	2	8
$\mapsto\!\!\leftarrow$	1	2	3	5	6	7	9	4	8	10	3	2	8
	1	2	3	5	6	7	9	4	8	10	2	3	8
$\mapsto\!\!\leftarrow$	1	2	3	5	6	7	9	4	8	10	2	3	8
	1	2	3	5	6	7	9	4	8	10	2	3	8
$\mapsto\!\!\leftarrow$	1	2	3	5	6	7	9	4	8	10	2	3	8
	1	2	3	5	6	7	9	2	3	4	8	8	10
$\mapsto\!\!\leftarrow$	1	2	3	5	6	7	9	2	3	4	8	8	10
	1	2	2	3	3	4	5	6	7	8	8	9	10

Table 1. Trace du tri fusion d'une liste

Notons que les deux premières conditions de la conjonction assurent simplement que l'on ne déborde pas des listes X et Y. Pour l'arrêt, la variable k étant initialisée à 0 avant la boucle et incrémentée à chaque passage, la suite $n-k$ est strictement décroissante et n'est pas minorée, la condition $n-k>0$ sera donc invalidée.

On définit les intervalles $I_0 := \varnothing$ et $I_k := [\![0, k-1]\!]$ pour tout entier $k \geqslant 1$. Considérons le prédicat $P(k)$ définit sur I_n suivant : en entrant

1. Version du 24 mars 2025, 17 : 55

dans la boucle `TQ`, on a

$$\forall r \in I_k \quad Y[j+r] = X[i+r].$$

$P(0)$ est vrai puisque $I_0 = \varnothing$. Supposons que $P(k)$ soit vrai avant d'entrer dans la boucle, montrons qu'après le passage dans la boucle, $P(k)$ est encore vrai. La première instruction de la boucle assure que $Y[j+k] = X[i+k]$ ce qui permet d'affirmer que $P(k+1)$ est vraie et l'incrémentation de k qui suit assure que $P(k)$ est vraie. Quand on sort de la dernière boucle on a $k = n$ et $P(n)$ est satisfaite, autrement dit on a bien copié les n valeurs :

$$\forall r \in [\![0, n-1]\!] \quad Y[j+r] = X[i+r].$$

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 2).

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

Table 2. 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 $j-i$.

Solution. La seule difficulté concerne la gestion des différents indices pour indexer correctement les éléments du tableau T lors de l'appel à l'algorithme de fusionnement. La variable k est l'exposant de 2 qui fixe la taille du tableau à trier. Elle détermine le nombre de séries de fusionnement sur des sous-tableaux de la même taille.

Rappelons que $\ll$ désigne l'opérateur bit-à-bit de décalage *à gauche* où $(x \ll k)$ est l'entier dont les bits sont ceux de x décalés de k positions vers la gauche (la variable x n'est pas modifiée par l'opérateur).

```
ALGORITHME TriFusion(@T)
DONNEES
 · T: tableau
VARIABLE
 · i,k,r: entiers
DEBUT
 · k ← log(#T)/log(2)
 · r ← 0
 · TQ (r < k) FAIRE
 ·  · i ← 1
 ·  · TQ (i < #T) FAIRE
 ·  ·  · FUSIONNER(T, i, i + (1 << r))
 ·  ·  · i ← i + (1 << (r + 1))
 ·  · FINTQ
 ·  · r ← r + 1
 · FINTQ
FIN
```

Algo. 2. Tri fusion itératif.

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 référencée 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` ← `L` puis

`L` ← `A` (la liste retour contiendra donc les valeurs dans l'ordre inverse d'apparition).

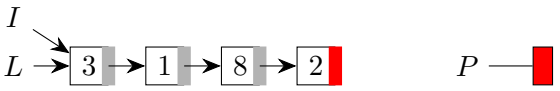
(2) Quelle est la complexité en espace et en temps de cet algorithme ?

(3) Écrivez un algorithme `LievreTortue`(@L) qui renvoie la liste constituée par la deuxième moitié des cellules de la liste L. À l'issue de l'exécution de l'algorithme, la liste L ne contient plus que la première moitié des cellules. Indication : parcourez la liste L avec deux pointeurs `lievre` et `tortue` simultanément, le premier avançant de deux cellules pendant que le second avance d'une cellule.

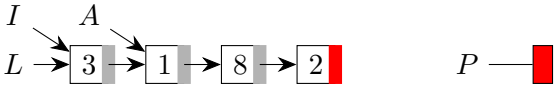
Solution. (1) Dans l'algorithme plus bas, la liste P est initialisée à la liste vide [], la référence L n'est pas modifiée (le passage par adresse est tout de même spécifié au cas où une autre structure de liste était utilisée), c'est une référence auxiliaire I qui balaie les cellules de la liste L. La liste A décrit les atomes "pairs" à décrocher de L et à insérer dans P.

Dans l'exemple qui suit et qui illustre l'algorithme, une cellule est matérialisée par une boite. Elle contient une valeur et une référence (adresse) vers la cellule suivante représentée par une flèche sur une bande grise, ou une simple bande rouge s'il n'y a pas de cellule suivante (la référence/adresse "nulle").

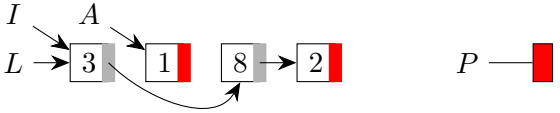
Après avoir initialisé les listes P et I et juste avant d'entrer dans la boucle principale, la situation générale est schématiquement la suivante :



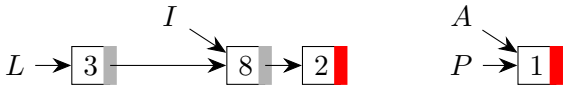
Illustrons un passage dans la boucle. Après l'instruction #01, les états des différentes listes sont :



Après les instructions #02 et #03 :



Après les instructions #04 et #05 :



L'algorithme 4 réalise la séparation de la liste d'entrée.

```
ALGORITHME Separer(@L):liste
DONNEES
 · L:liste
VARIABLES
 · I, P, A:listes
DEBUT
 · P ← []
 · I ← L
 · TQ ((I != []) ET (I»SUIV != [])) FAIRE
01> ·  · A ← I»SUIV
02> ·  · I»SUIV ← A»SUIV
03> ·  · A»SUIV ← []
04> ·  · InsTete(P,A)
05> ·  · I ← I»SUIV
 · FTQ
 · RENVOYER P
FIN
```

Algo. 3. Séparation d'une liste en deux.

(2) À chaque étape de la boucle principale, on décroche exactement une cellule de la liste L que l'on insère dans la liste P initialement vide. Comme l'insertion se fait en tête de liste, cette opération a un coût de $\Theta(1)$. Il faut donc parcourir les n cellules de L pour achever le découpage de L, ce qui se fait en $n\Theta(1) = \Theta(n)$. Pour la mémoire, l'algorithme se fait sur place puisqu'aucune nouvelle cellule n'est créée.

(3) L'idée est simple, quand le lièvre arrive au bout de la liste L, la tortue est à la moitié. Il suffit donc de déclarer un pointeur `tortue` et un pointeur `lievre` partant respectivement de la première et de la deuxième cellule de la liste `L`, puis avançant respectivement d'une et de deux cellules à chaque étape.

Le «décrochage» de la deuxième moitié de la liste L se fait via l'instruction `B>`, il ne faut évidemment pas le faire avant d'avoir conservé la référence de cette deuxième moitié d'où l'instruction `A>` placée *avant*.

NB. On pourrait faire partir les deux concurrents de la première cellule, mais il faudrait alors s'assurer que le lièvre a pu avancer avant de faire avancer la tortue. En faisant partir le lièvre de la deuxième cellule, on évite une comparaison dans la boucle.

```
    ALGORITHME LievreTortue(@L):liste
    DONNEES
     · L:liste
    VARIABLES
     · tortue, lievre, milieu:listes
    DEBUT
     · SI ((L = []) OU (L»SUIV = [])) ALORS
     ·  · RENVOYER []
     · FSI
     · tortue ← L
     · lievre ← L»SUIV
     · TQ ((lievre != []) ET (lievre»SUIV != [])) FAIRE
     ·  · lievre ← (lievre»SUIV)»SUIV
     ·  · tortue ← tortue»SUIV
     · FTQ
A>   · milieu ← tortue»SUIV
B>   · tortue»SUIV ← []
     · RENVOYER milieu
    FIN
```

Algo. 4. Séparation d'une liste en deux avec l'algorithme du lièvre et de la tortue.

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. L'indexation de la liste L commence en 1. Quelle est la valeur renvoyée par l'algorithme ?

(2) Étudiez le partitionnement des deux listes $[2, 1, 2, 4, 3]$ et $[2, 1, 2, 2, 3]$.

(3) Montrez qu'à la sortie de l'algorithme, après 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 :

$$(i < j) \ \wedge \ (L[p{:}i] \leqslant x \leqslant L[j{:}r]). \tag{1}$$

L'algorithme 5 réalise le partitionnement.

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

Algo. 5. Partitionnement d'une liste pour le tri rapide.

Il utilise ici deux algorithmes auxiliaires `Reculer` et `Avancer` pour déplacer les deux sentinelles j et i le long de la liste (cf. Algo. 6).

Solution. **(1)** On résume les étapes du partitionnement dans la table 3. La valeur x du pivot est en bleu (ici $x = 5$). La variable i est sur fond jaune et la variable j sur fond rouge et les deux sont sur fond orange si elles coïncident. L'indice q de la partition est sur fond gris. Ici $q = 6$.

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

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

Algo. 6. Déplacement des deux sentinelles.

	1	2	3	4	5	6	7	8	9	10	11	12	13
L	5	2	1	7	3	6	9	5	1	4	8	6	9
01>03	**5**	2	1	7	3	6	9	5	1	4	8	6	9
04>06 (←)	**5**	2	1	7	3	6	9	5	1	4	8	6	9
08 (⇌)	4	2	1	7	3	6	9	5	1	**5**	8	6	9
09 (←)	4	2	1	7	3	6	9	5	1	**5**	8	6	9
10 (→)	4	2	1	7	3	6	9	5	1	**5**	8	6	9
08 (⇌)	4	2	1	1	3	6	9	5	7	**5**	8	6	9
09 (←)	4	2	1	1	3	6	9	5	7	**5**	8	6	9
10 (→)	4	2	1	1	3	6	9	5	7	**5**	8	6	9
08 (⇌)	4	2	1	1	3	5	9	6	7	**5**	8	6	9
09 (←)	4	2	1	1	3	5	9	6	7	**5**	8	6	9
10 (→)	4	2	1	1	3	5	9	6	7	**5**	8	6	9

Table 3. Partitionnement de la liste L.

(2) Avec le même codage que dans la table 3, on trouvera la trace du partitionnement de ces deux listes en table 4.

(3) On étudie tout d'abord le cas où l'on n'entre pas dans la boucle principale (07), donc parce que $i \geqslant j$. Dans ce cas, la variable j a été décrémentée jusqu'à atteindre la valeur $i = p$ grâce à la boucle (04), ce

	1	2	3	4	5
L	2	1	2	4	3
01>03	**2**	1	2	4	3
04>06 (←)	**2**	1	2	4	3
08 (⇌)	2	1	**2**	4	3
09 (←)	2	1	**2**	4	3
10 (→)	2	1	**2**	4	3

	1	2	3	4	5
L	2	1	2	2	3
01>03	**2**	1	2	2	3
04>06 (←)	**2**	1	2	2	3
08 (⇌)	2	1	2	**2**	3
09 (←)	2	1	2	**2**	3
10 (→)	2	1	2	**2**	3

Table 4. Partitionnements des listes $[2, 1, 2, 4, 3]$ et $[2, 1, 2, 2, 3]$.

qui suppose que

$$\forall i \in [\![p+1, r]\!] \quad L[i] > x$$

la valeur pivot x n'apparaît donc qu'une seule fois dans la liste et en première position $i = p$. On sort ainsi de la boucle (04), puis de la boucle principale avec $j = i = p$ et par conséquent $L[j] = L[p] = x$.

Étudions à présent le cas général où l'on entre dans la boucle principale (07). Si $j = i+1$ après l'échange, on sort de la boucle avec $i = j-1$ puisque i et j sont systématiquement modifiées. Si $j > i + 1$ après l'échange et que la valeur du pivot apparaît une *unique* fois à une position k dans l'intervalle $[\![i + 1, j - 1]\!]$, et que les valeurs à sa gauche (resp. à sa droite) sont strictement inférieures (resp. strictement supérieures) à x, alors j sera décrémentée jusqu'à k et i sera incrémentée jusqu'à k, on sortira donc de la boucle avec $i = j$ et $L[j] = x$. Dans tous les autres cas où l'on sort de la boucle, i a nécessairement été incrémentée après la position j puisque $L[j] < x$ et si $L[j] = x$, i aura rencontré une autre occurence de la valeur de x, contredisant l'hypothèse que l'on sort de la boucle.