Informatique

8

Tris

Cours

Algorithmes de Tri ***3***

1.I. Contexte **3**

1.II. Quelques définitions **3**

1.III. Fonctions natives de python **4**

1.IV. Tris au programme **5**

1.IV.1 Tri par insertion 5

1.IV.1.a Tri avec liste auxiliaire 5

1.IV.1.a.i Principe 5

1.IV.1.a.ii Complexité 5

1.IV.1.b Tri en place 6

1.IV.1.b.i Principe 6

1.IV.1.b.ii Complexité 6

1.IV.1.b.iii Stabilité 6

1.IV.2 Tri rapide (quicksort), ou par pivot 7

1.IV.2.a Choix du pivot 7

1.IV.2.b Tri rapide avec listes auxiliaires 7

1.IV.2.c Tri en place 8

1.IV.2.d Remarque 12

1.IV.2.e Complexité 13

1.IV.2.e.i Meilleur des cas 13

1.IV.2.e.ii Pire des cas 13

1.IV.2.f Stabilité 13

1.IV.2.g Application à la détermination de la médiane d'une liste 14

1.IV.3 Tri fusion (merge-sort) 15

1.IV.3.a Tri avec listes auxiliaires 15

1.IV.3.a.i Principe 15

1.IV.3.a.ii Exemple 15

1.IV.3.b Tri en place 16

1.IV.3.b.i Principe 16

1.IV.3.b.ii Exemple 16

1.IV.3.c Complexité 17

1.IV.3.d Stabilité 17

1.IV.4 Tri par sélection 18

1.IV.4.a Principe 18

1.IV.4.b Exemple en place 18

1.IV.4.c Complexité 18

1.IV.4.d Stabilité 18

1.IV.5 Tri par comptage 19

1.IV.5.a Principe 19

1.IV.5.b Exemple 19

1.IV.5.c Complexité 19

1.IV.5.d Stabilité 19

Algorithmes de Tri

1.I. Contexte

L'objectif de ce paragraphe est de vous permettre de mettre en place des algorithmes qui réalisent un tri ordonné d'une liste aléatoire. Nous allons voir dans un premier temps que Python propose des fonctions qui effectuent un tri directement, des sortes de « boîtes noires » dont on ne sait pas grand-chose. Ensuite, nous aborderons les principales techniques de tri que vous avez à connaître et nous discuterons de leur complexité en temps.

Evidemment, un algorithme de tri est efficace si, lorsqu'il retourne la liste triée, il a effectué ce tri en un temps le plus court possible.

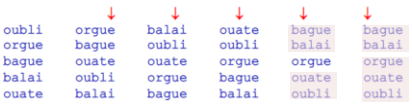
Le caractère aléatoire de l'organisation initiale de la liste à trier va nous conduire à discuter des complexités dans le meilleur et dans le pire des cas. Il est évident que nous comparerons alors les pires des cas, mais il ne faudra jamais oublier qu'il est possible, selon l'organisation initiale de la liste à trier, que la performance d'un algorithme censé être plus lent qu'un autre pourra s'inverser…

1.II. Quelques définitions

Voici quelques définitions à connaître :

- Clefs : c'est ce qui est utilisé pour trier des éléments. Par exemple :
  - On peut trier des mots à l'aide de leur première lettre. La clé est ainsi la première lettre
  - On peut trier des couples (4,5) (1,2) (1,3) (2,3) (3,1) selon leur premier terme, ce sera donc la clé, ou selon leur deuxième terme…
  - On peut trier des élèves ayant une multitude de caractéristiques, par leur âge, taille…
- Tri comparatif : Tri fondé sur la comparaison entre les « clefs » des éléments pour les trier
- Tri itératif : Tri basé sur un ou plusieurs parcours itératifs de la liste à trier
- Tri récursif : Tri basé sur une méthode récursive
- Tri en place : Tri qui utilise une quantité constante d'éléments en mémoire
- Tri avec listes auxiliaires : Tri qui crée des listes auxiliaires pour réaliser le tri (quantité de mémoire utilisée non constante – éventuelle complexité en temps)
- Tri stable (ex : tri insertion & tri fusion si bien codés) : Tri qui conserve l'ordre relatif des éléments de même clef. On veut par exemple trier selon leur premier élément les couples suivants (4,5) (1,**2**) (1,**3**) (2,3) (3,1), on applique un algorithme de tri qui renvoie :
  - (1,**2**) (1,**3**) (2,3) (3,1) (4,5) ⇒ Stable
  - (1,**3**) (1,**2**) (2,3) (3,1) (4,5) ⇒ Non stable

  Application classique : Tri lexicographique de mots de même taille, on réalise les tris successifs de ces mots de la dernière à la première lettre. Le résultat est correct si les tris successifs utilisés sont stables, ce qui mémorise le travail des étapes précédentes…



↓	↓	↓	↓	↓
oubli	orgue	balai	ouate	bague	bague
orgue	bague	oubli	oubli	balai	balai
bague	ouate	ouate	orgue	orgue	orgue
balai	oubli	orgue	bague	ouate	ouate
ouate	balai	bague	balai	oubli	oubli

  Remarque : on peut vérifier assez simplement par des exemples si le tri utilisé semble stable en introduisant dans la liste à trier deux nombres égaux de types différents, par exemple un 1 (entier) et un 1.0 (flottant)

```
>>> f_tri_insertion_aux([3,2,1,1.0])
[1, 1.0, 2, 3]
```

1.III. Fonctions natives de python

Introduisons ici deux fonctions que python propose nativement : « `.sort` » et « `sorted(L)` ».

Voici un exemple de réalisation de tri avec ces fonctions :

```
from random import randint as rd
n = 1000
L = [rd(1,n) for i in range(n)]
LL = sorted(L)
L.sort()
```

`L.sort()` trie directement la liste L en la modifiant, c'est un tri « en place ».
`LL = sorted(L)` permet de créer une nouvelle liste triée tout en gardant la liste initiale non modifiée.

Il est fort probable que ces deux fonctions soient interdites d'utilisation le jour du concours. En effet, l'un des objectifs de la matière est de vous apprendre à réfléchir pour un jour, résoudre de nouveaux problèmes !

1.IV. Tris au programme

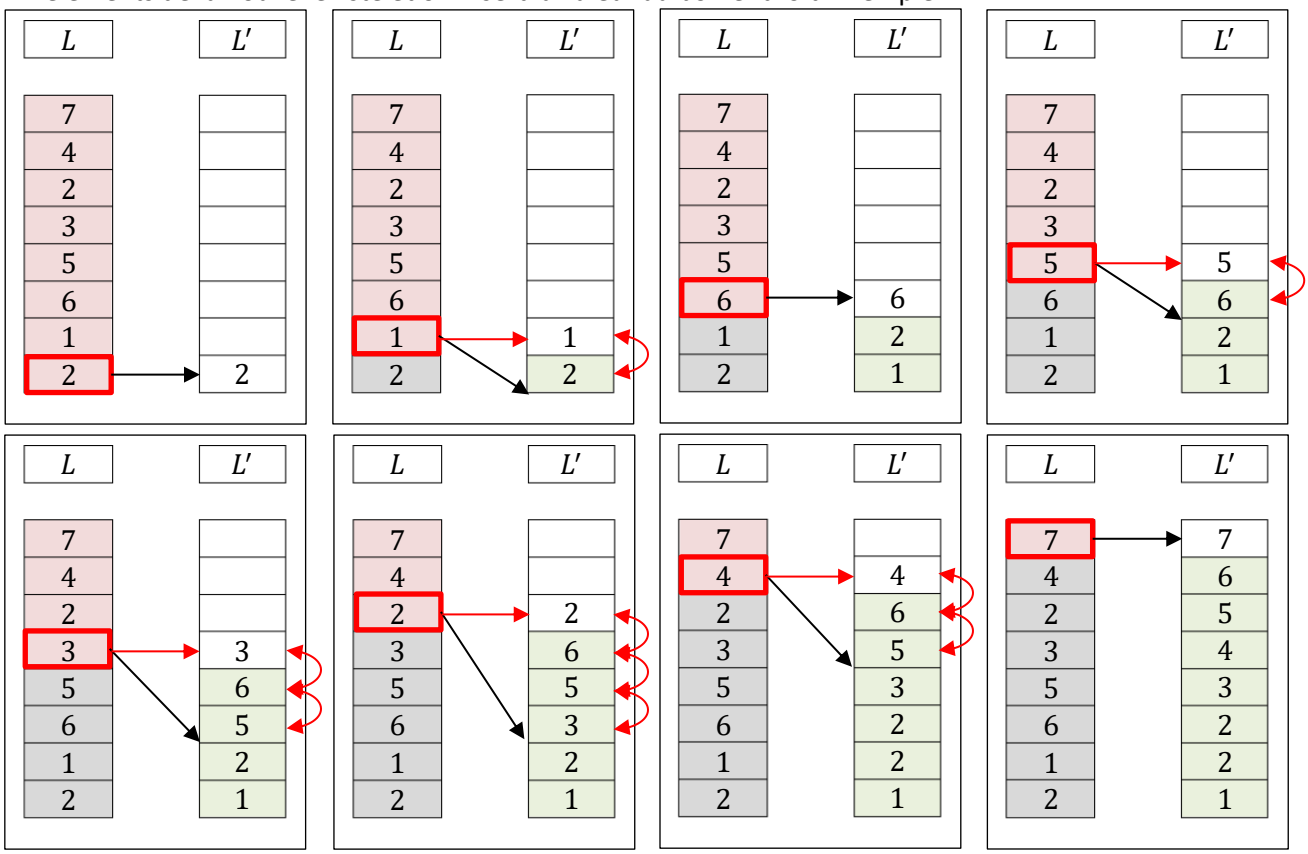
Nous allons aborder quelques manières de trier que vous avez à connaître. Nous verrons ici les principes, et en TP vous les coderez directement. Appelons L la liste à trier et L' la liste triée.

1.IV.1 Tri par insertion

1.IV.1.a Tri avec liste auxiliaire

1.IV.1.a.i Principe

Le tri par insertion, comme son nom l'indique, consiste à créer une nouvelle liste vide qui sera remplie avec les éléments de la liste à trier. Ensuite, à chaque nouveau terme à insérer, on parcourt les éléments de la nouvelle liste et on insert la valeur au bon endroit. Exemple :



1.IV.1.a.ii Complexité

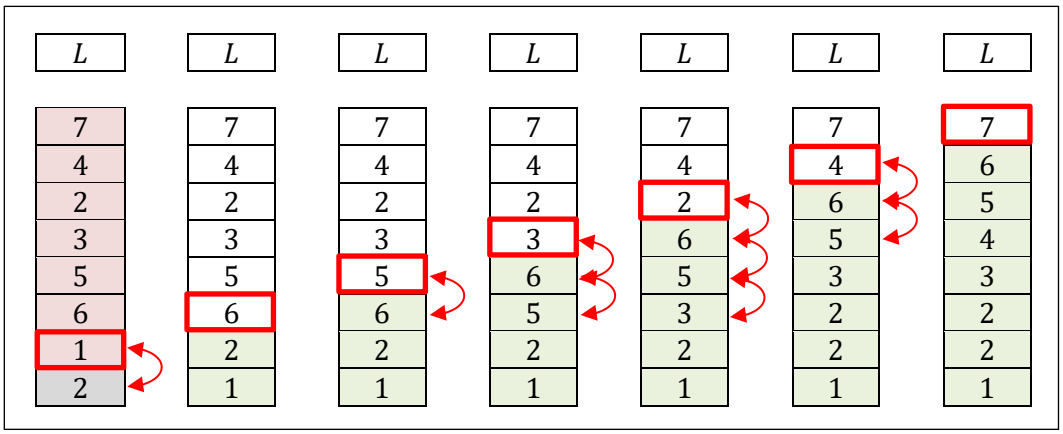
A la première étape, il faut simplement affecter la première valeur de L à L'. Ensuite, pour chaque indice i de la valeur de L de 1 à $n-1$:

Meilleur des cas	1 test sur la liste L'	$O\left(\sum_{i=1}^{n-1} 1\right) = O(n-1) = \boldsymbol{O(n)}$
Pire des cas	$i-1$ tests sur la liste L'	$O\left(\sum_{i=1}^{n-1}(i-1)\right) = O\left(\sum_{i=0}^{n-2} i\right) = O\left(\frac{1+(n-2)}{2}(n-1)\right) = O\left(\frac{(n-1)^2}{2}\right) = \boldsymbol{O(n^2)}$

1.IV.1.b Tri en place

1.IV.1.b.i Principe

Le tri en place consiste à prendre chaque valeur d'une liste à trier, de la comparer à la valeur précédente, et d'intervertir les deux si la valeur actuelle est plus faible que la valeur d'avant. On procède alors ainsi de suite jusqu'à ce que la valeur déplacée initialement soit supérieure à la précédente. On commence à partir de la seconde valeur dans la liste. Il n'est pas nécessaire de créer une seconde liste, on trie directement la liste initiale. Exemple :



1.IV.1.b.ii Complexité

La première étape consiste à s'intéresser à la seconde valeur de L. Alors, pour chaque indice i de la valeur de L de 1 à $n-1$:

Meilleur des cas	1 test sur la liste L'	$O\left(\sum_{i=1}^{n-1} 1\right) = O(n-1) = \boldsymbol{O(n)}$
Pire des cas	$i-1$ tests sur la liste L'	$O\left(\sum_{i=1}^{n-1}(i-1)\right) = O\left(\sum_{i=0}^{n-2} i\right) = O\left(\frac{1+(n-2)}{2}(n-1)\right) = O\left(\frac{(n-1)^2}{2}\right) = \boldsymbol{O(n^2)}$

1.IV.1.b.iii Stabilité

Pour les deux algorithmes proposés ci-dessus, si on veille à ne descendre un terme que tant qu'il est supérieur strictement à celui d'en dessous, on remarquera que cet algorithme est stable.

1.IV.2 Tri rapide (quicksort), ou par pivot

1.IV.2.a Choix du pivot

Ce tri est basé sur le choix d'un pivot (une des valeurs de la liste à trier) et le partage des autres valeurs en fonction de celui-ci. Naïvement, on considère que le pivot est la première valeur de la liste. On peut effectuer un autre choix, nous en parlerons lors de l'étude de la complexité de ce tri.

1.IV.2.b Tri rapide avec listes auxiliaires

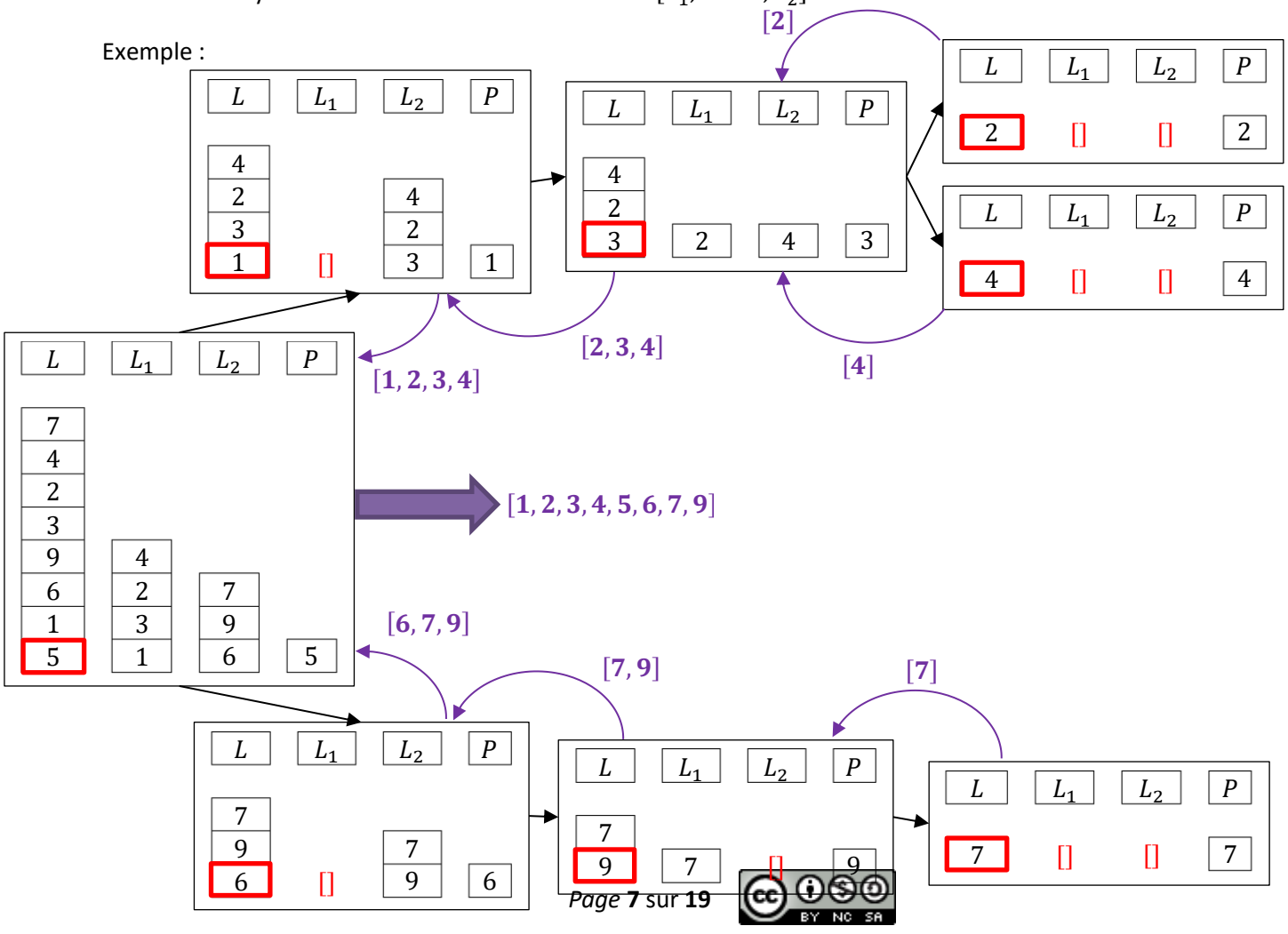
Le tri rapide a été inventé par Hoare vers 1960. Le principe est de partager une liste en 2 listes telles que dans la première, toutes les valeurs prises soient inférieures à celles de la seconde. La stratégie est dite de « Diviser pour régner ». On décompose un problème en deux problèmes plus simples. A la fin, on regroupe les résultats de chaque sous problème pour arriver au résultat.

Il est alors possible d'appliquer récursivement cette démarche afin d'obtenir, à la fin, une liste triée.

Démarche :

- Traiter les cas de base : Si la liste est vide, la renvoyer
- Choisir un élément de L appelé « pivot » (naïvement, première valeur)
- Créer les listes L_1 et L_2 telles que $\begin{cases}\forall i \neq 0, L_1[i] < Pivot \\ \forall i \neq 0, L_2[i] \geq Pivot\end{cases}$
- Appliquer récursivement le tri aux listes L_1 et L_2 pour obtenir deux listes L'_1 et L'_2 triées
- Renvoyer les listes combinées dans l'ordre : $[L'_1, Pivot, L'_2]$

Exemple :



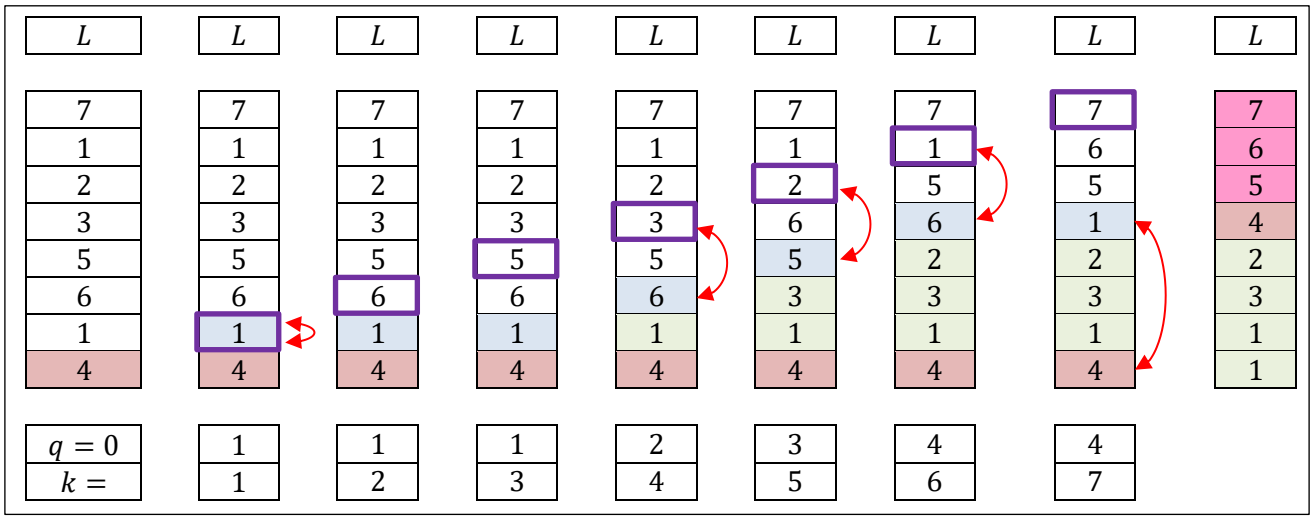
1.IV.2.c Tri en place

On peut réaliser un tri rapide en place, c'est-à-dire sans utiliser de mémoire additionnelle pour stocker des listes auxiliaires.

Le principe de la réalisation de ce tri consiste à appliquer la démarche suivante par récurrence :

- Considérer une portion de liste entre les indices i et j inclus. A la première itération, c'est la liste entière
- Si cette sous liste contient un seul terme, ne rien exécuter, elle est déjà « triée »
- Sinon :
    - Choisir le pivot : Naïvement, le premier. Sinon, en choisir un et l'échanger avec la première valeur de la portion de liste traitée
    - Appeler p l'indice du pivot : $p = i$
    - Définir un indice q qui au départ vaut p
    - Etudier chaque valeur de la sous liste étudiée pour un indice k entre les indices $i + 1$ et j inclus et procéder ainsi :
        - $L[k] \geq Pivot \rightarrow RAS$
        - $L[k] < Pivot$:
            - $q \mathrel{+}= 1$
            - $L[q], L[k] = L[k], L[q]$
    - A la fin, on échange le pivot avec le terme à la position q : $L[p], L[q] = L[q], L[p]$
    - Appeler récursivement cette procédure sur les « sous » listes de part et d'autre du pivot de la portion de liste étudiée

Exemple pour la première itération : $\begin{cases} i = 0 \\ j = len(L) - 1 = 7 \end{cases}$

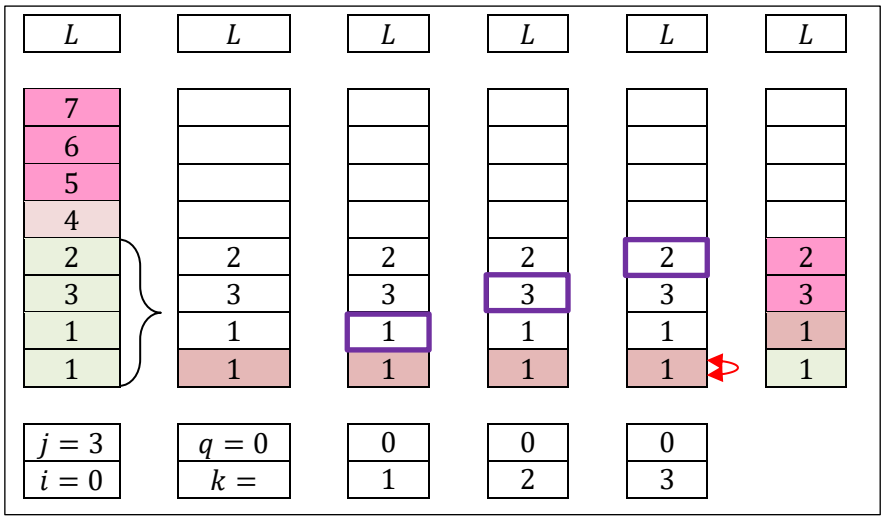


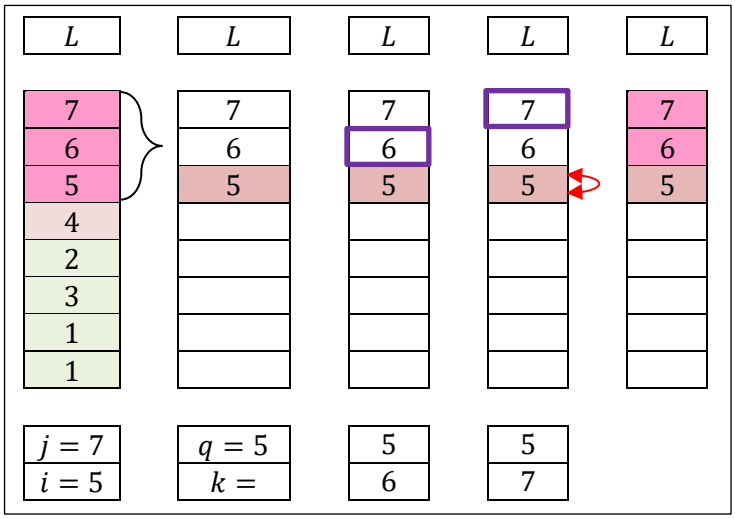
Après cette étape, on obtient un pivot définitivement placé, et deux sous listes avant et après le pivot telles :

- Qu'avant, toutes les valeurs sont inférieures ou égales au pivot
- Après, elles sont supérieures strictes au pivot

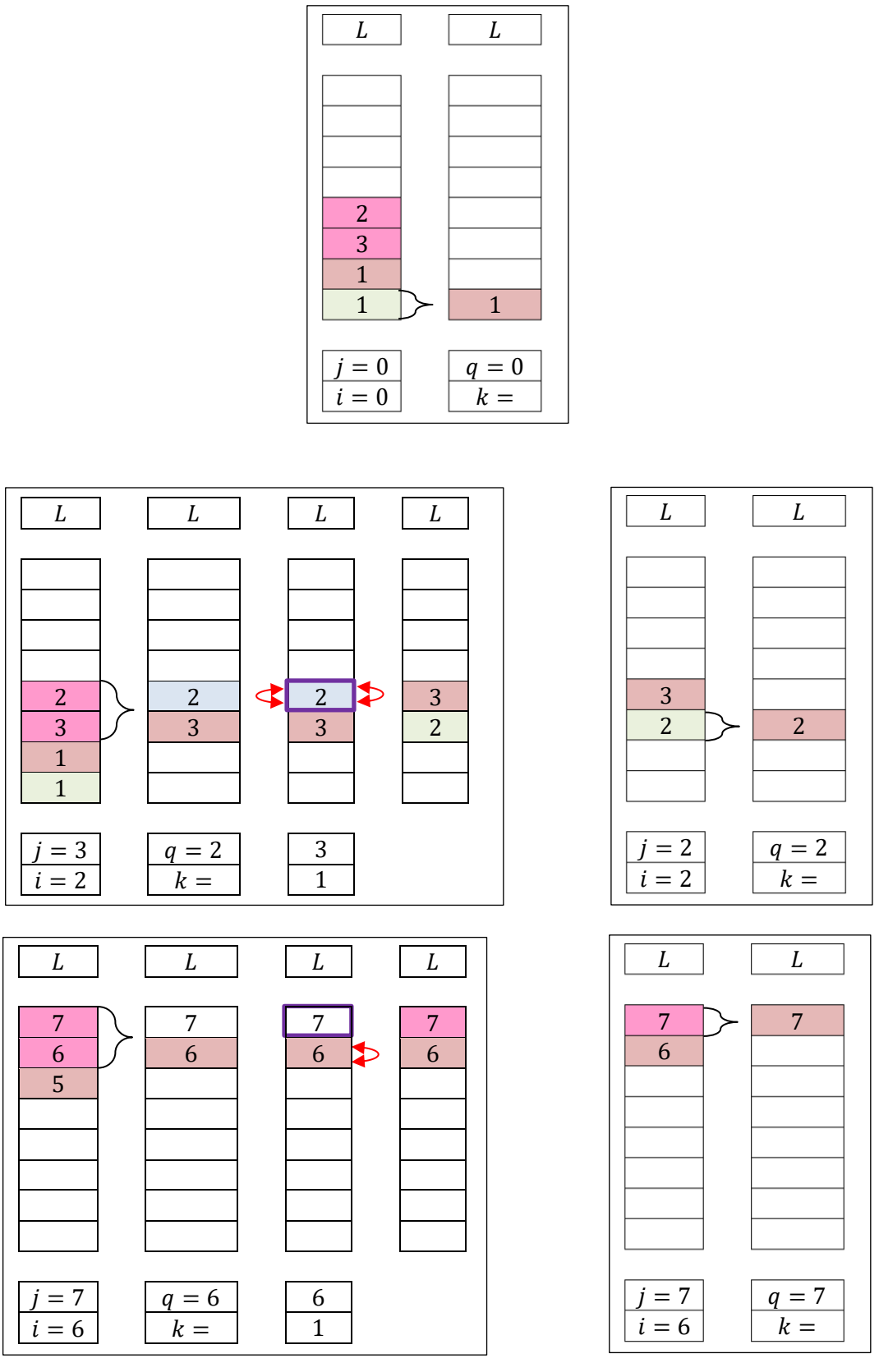
Il suffit alors de procéder de même sur les deux sous listes et ainsi de suite par récursivité. Il n'y a alors aucun renvoie, chaque sous liste étant directement triée.

Etape 2 de l'exemple :

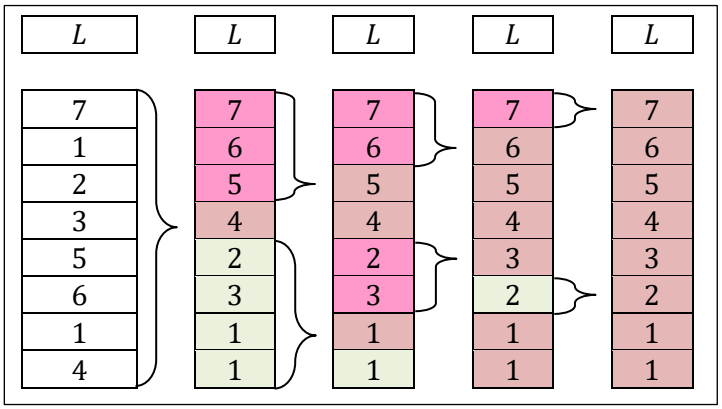




Etape 3 de l'exemple :



Récapitulons l'état de la liste L à chaque appelle de tri :



Remarque : Chaque accolade pointe sur le pivot placé après traitement de la liste concernée

1.IV.2.d Remarque

A chaque étape, le pivot est définitivement placé. Chaque terme devient à un moment le pivot, est est placé.

1.IV.2.e Complexité

1.IV.2.e.i Meilleur des cas

Dans le meilleur des cas, à chaque division, on a autant de termes dans chacune des sous listes (le pivot est la médiane de la liste). Ainsi, à chaque étape, il s'appelle 2 fois à l'ordre $n/2$. Pour chaque exécution à l'ordre n, il parcourt la liste de ses n éléments. La complexité à l'ordre n est donc $O(n)$. Ainsi, nous sommes dans le cas $\alpha = 1$ des démonstrations de complexité. On a :

$$C(n) = n \ln n$$

1.IV.2.e.ii Pire des cas

Dans le pire des cas, à chaque division, une sous liste possède 0 termes, l'autre les $n - 1$ restants (le pivot est un extremum de la liste).

On est donc le cadre de l'auto-appel 1 fois au rang $n - 1$ avec une complexité à l'épate n de $O(n^1)$.

La complexité vaut donc :

$$C(n) = n^2$$

Le pire des cas est atteint lorsque la liste est déjà presque triée.

En faisant une pré-étude de la liste avant de la trier, si on se rend compte qu'elle est organisée, on comprend alors qu'il peut être utile de prendre un pivot au milieu de la liste dans le but de diviser en 2 ☺. Autrement dit, l'idéal serait de prendre la médiane de la liste pour être dans le meilleur des cas…

1.IV.2.f Stabilité

L'algorithme avec listes auxiliaires proposé ci-dessus sera stable si, lorsqu'un pivot possède des ex aequo, ceux-ci sont placés dans la liste des éléments supérieurs.

L'algorithme en place proposé n'est pas stable à cause des échanges réalisés. Exemple sur la liste [2,7,7.0,1]. Le 1 étant plus petit que le pivot 2, il va être échangé avec le premier 7 pour donner à la fin de la première itération [1,2,7.0,7].

Remarque : La stabilité serait obtenue à deux conditions :

- Comme pour l'algorithme en place, il faudrait que les ex aequo aux pivots soient laissés à leur place
- Mais en plus, il ne faudrait pas échanger deux termes mais « descendre » chaque terme plus petit que le pivot au bon endroit. Mais attention, la descente est couteuse : Une descente de n places est en $O(n)$
    - Dans le meilleur des cas, aucune descente, on reste en $O(n)$ à chaque étape, soit $O(n \ln n)$ pour le tri
    - Dans le pire des cas, il faudra faire descendre environ $n/2$ termes d'environ $n/2$ places (ex : [1,2,2,2,0,0,0] devra faire descendre 3 termes 0 de 3 places), ce qui conduit un algorithme en $O(n^2)$ à chaque étape $(\alpha = 2)$, et un tri « rapide » en $O(n^3)$, ce qui explique pourquoi on réalise des échanges $\big(O(1)\big)$ dans l'algorithme.

1.IV.2.g Application à la détermination de la médiane d'une liste

Nous verrons en TP qu'il est possible d'utiliser l'algorithme de tri rapide en place en ne traitant à chaque étape que la sous liste contenant l'indice milieu de la liste jusqu'à ce que le pivot se retrouve à cette position. Ce sera alors la médiane de la liste.

Cette méthode permettra donc de trouver la médiane d'une liste sans la trier ☺.

Cette méthode effectue :

- Dans le meilleur des cas : 1 seul auto-appel au rang n/2 avec une étape en $O(n)$, soit une complexité en $O(n)$
- Dans un cas intermédiaire où le pivot serait bien choisi : des auto-appels une fois au rang n/2 avec un travail en $O(n)$ à chaque étape, soit une complexité en $O(n \ln n)$
- Dans le pire des cas : des auto-appels une fois au rang n-1 avec un travail en $O(n)$ à chaque étape, soit une complexité en $O(n^2)$

1.IV.3 Tri fusion (merge-sort)

1.IV.3.a Tri avec listes auxiliaires

1.IV.3.a.i Principe

Cet algorithme emploi lui aussi une stratégie dite de « diviser pour régner ». Il consiste à diviser récursivement une liste en deux puis à fusionner les sous listes obtenues en les triant.

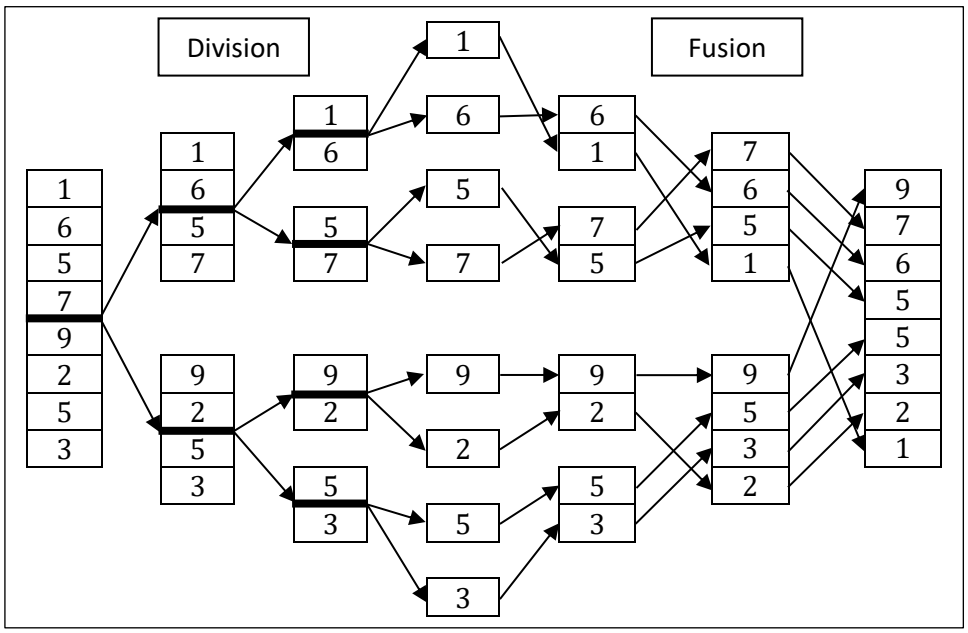
Soit une liste L à trier :

- Traiter le cas de base : Si L ne contient qu'un terme, elle est triée
- Partager L en 2 listes L_1 et L_2 de tailles identiques (à 1 près)
- Appliquer récursivement la procédure aux listes L_1 et L_2 pour les trier
- En supposant que L_1 et L_2 ont été triées à l'étape précédente, les fusionner de manière ordonnée

On peut comparer cet algorithme avec le tri rapide ainsi :

- Le tri rapide effectue des travaux de trie autour du pivot puis appelle récursivement ce travail de tri – On parle de récursivité sur les résultats
- Le tri fusion appelle récursivement une fonction qui divise le problème et trie à la fin, puis recombine les résultats – On parle de récursivité sur les données

1.IV.3.a.ii Exemple



1.IV.3.b Tri en place

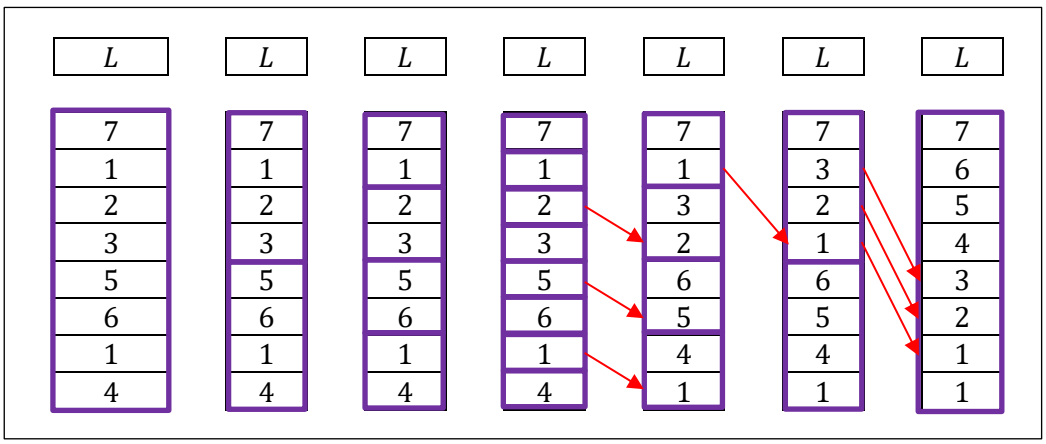
1.IV.3.b.i Principe

Soit une liste L à trier.

- Considérer une portion de liste dans l'intervalle d'indices Python $[i, j[$. A la première itération, c'est la liste entière.
- Traiter le cas de base : Si $j = i + 1$, la portion de liste traitée ne contient qu'un terme, elle est déjà triée
- Déterminer un indice milieu (gauche ou droite) m entre i et j
- Appeler récursivement la fonction de tri fusion sur les deux portions de L dans les intervalles d'indices $[i, m[$ et $[m, j[$
- En supposant que les portions de listes sur les deux intervalles $[i, m[$ et $[m, j[$ ont chacune été triées en place à l'étape précédente, appliquer une fusion ordonnée en place des termes de L entre i et j

Remarque : la fusion ordonnée consiste, par exemple, à comparer les deux sous listes à trier, et à « descendre » les termes les plus petits de la seconde partie de liste « au-dessus » vers la première partie « en dessous » lorsque cela est nécessaire, en décalant tous les autres « vers le haut »

1.IV.3.b.ii Exemple



1.IV.3.c Complexité

Le principe de la division par deux est toujours respecté. On appelle donc toujours deux fois la fonction à l'ordre n/2.

Le travail au rang n est toujours en $O(n)$:

- Dans le meilleur des cas, le maximum d'une des deux listes est inférieur au minimum de l'autre. Il suffit d'empiler les deux listes. C'est donc un travail à l'étape n sur n termes (ou /2 si on suppose qu'on ajoute à une liste les n/2 autres termes), en $O(n)$.
- Dans le pire des cas, il faut parcourir les n termes et les empiler les uns après les autres. On a aussi une complexité à l'étape n en $O(n)$.

Comme pour le tri rapide, on a donc le cas $\alpha = 1$ des démonstrations de complexité.

$$C(n) = n \ln n$$

Remarque : Pour obtenir cette complexité sur la version en place, on suppose que la fusion ordonnée implémentée est en $O(n)$, ce qui n'est pas toujours si évident du fait de décalages

1.IV.3.d Stabilité

La stabilité dépend de la réalisation de la fusion ordonnée, qu'elle soit en place ou avec listes auxiliaires. Lors du parcours de gauche à droite des deux listes à fusionner, la stabilité est obtenue si, pour des es aequo, le terme de la liste de gauche est privilégié.

1.IV.4 Tri par sélection

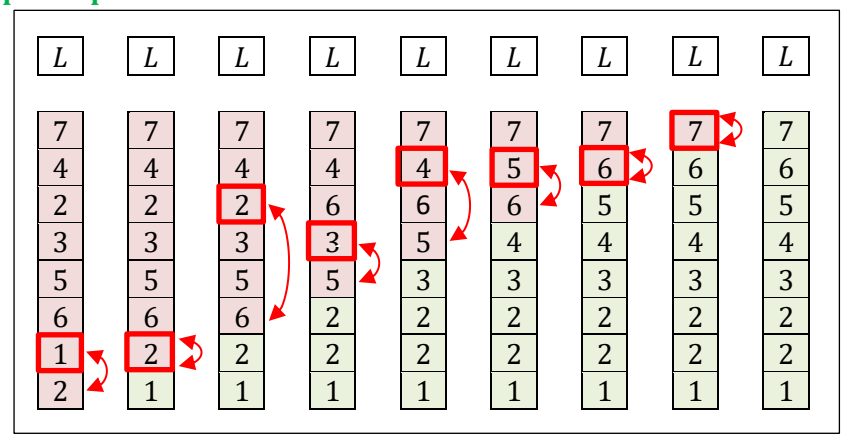
1.IV.4.a Principe

Le principe de ce tri est de procéder de la sorte :

- Sélectionner le minimum de L et échanger ce minimum avec le premier terme de L (indide $i = 0$)
- Sélectionner le minimum de L privée de son premier terme et échanger ce minimum avec le second terme de L
- Etc.

Il est aussi facile de programmer ce tri en place ou avec listes auxiliaires.

1.IV.4.b Exemple en place



1.IV.4.c Complexité

Soit n la taille de la liste L. Pour i dans $[0, n-2]$ (il est inutile de réaliser l'étape de l'algorithme sur le dernier terme de L), on cherche le minimum d'une liste de taille $n-i$, travail en $O(n-i)$, soit un travail en :

$$O\left(\sum_{i=0}^{n-2}(n-i)\right)$$

$$\sum_{i=0}^{n-2}(n-i) = \sum_{i=0}^{n-2} n - \sum_{i=0}^{n-2} i = n(n-1) - \left(\frac{0+(n-2)}{2}\right)(n-1) = \frac{2n^2-2n}{2} - \frac{n^2-3n+2}{2}$$
$$= \frac{n^2+n-2}{2}$$

Finalement, le tri sélection est en $O(n^2)$.

1.IV.4.d Stabilité

Lorsqu'il y a des exæquo, on garde leur ordre d'apparition en sélectionnant le **premier minimum** de L dans l'algorithme proposé.

1.IV.5 Tri par comptage

Ce tri est très intéressant dès lors que les éléments à trier sont connus (ie : on peut en dresser une liste prédéterminée). Prenons pour exemple une liste L de n entiers positifs (la valeur minimale possible est donc 0).

1.IV.5.a Principe

Le principe est le suivant :

- **Initialisation** : Déterminer la valeur maximale m de la liste L et créer une liste D remplie de $m+1$ valeurs nulles (chaque élément $D[i]$ représente le nombre d'apparition de la valeur i dans L)
- **Dénombrement** : compter les apparitions de chaque terme de L par un parcours de la liste L en incrémentant de 1 l'élément $D\big[L[i]\big]$ – A la fin de cette étape, D contient le nombre d'apparition de chaque élément de L
- **Reconstruction** : Créer une nouvelle liste contenant dans l'ordre, les éléments de L autant de fois qu'ils apparaissent, à l'aide de la liste D

Remarque : on pourra regarder le TD « Tri radix » pour aller plus loin avec un tri par dénombrement utilisant un dictionnaire des valeurs apparaissant dans L, et définissant leur ordre à respecter).

1.IV.5.b Exemple

<table>
<tr><td colspan="2">$L = [3,2,3,4,5,6,1,2,5,3,6,7,9,2]$</td></tr>
<tr><td>Initialisation</td><td>$m = 9$</td></tr>
<tr><td>Dénombrement</td><td>
<table>
<tr><td>i</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr>
<tr><td>$D[i]$</td><td>0</td><td>1</td><td>3</td><td>3</td><td>1</td><td>2</td><td>2</td><td>1</td><td>0</td><td>1</td></tr>
</table>
</td></tr>
<tr><td>Reconstruction</td><td>$[1,2,2,2,3,3,3,4,5,5,6,6,7,9]$</td></tr>
</table>

1.IV.5.c Complexité

Cet algorithme, s'il est bien programmé, fait apparaître 3 étapes à la suite de complexité linéaire :

- Détermination du maximum : $O(n)$
- Création de D : $O(m)$
- Parcours de L pour dénombrement : $O(n)$
- Reconstruction : $O(m)$

Soit $O(2n+2m) = O(n+m)$.

1.IV.5.d Stabilité

On ne peut parler de stabilité dans ce tri. En effet, on reconstruit de toute pièce une nouvelle liste image de la liste initiale, les éléments ne sont pas gardés.