TP 7 - Corrigé
Algorithmes de tri

Les solutions données dans ce corrigé ne sont bien sûr que des propositions, et sont sans nul doute perfectibles.

1 Tri à bulles

Q1 Ci-dessous l'exécution de l'algorithme de tri à bulles.

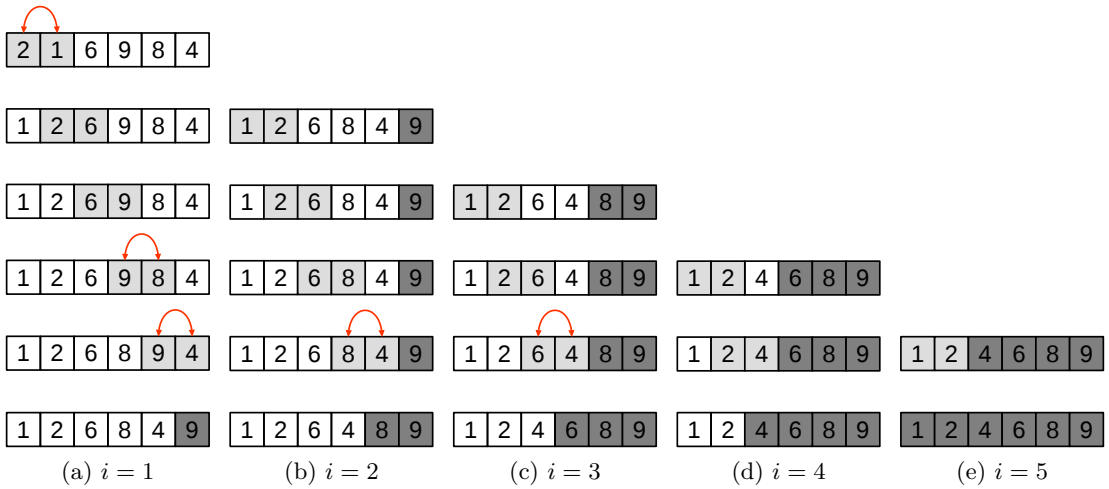

FIGURE 1 – Exemple d'exécution de l'algorithme de tri à bulles. Les cases gris clair représentent les éléments comparées, les flèches rouges les échanges d'éléments, et les case gris sombre les éléments placés définitivement.

Q2 On peut écrire cette fonction de manière concise de la manière suivante.

Listing 1 – est_trie

```
1  def est_trie(T):
2      N = len(T)
3      res = True
4      i = 0
5      while res and i < N-1:
6          res = (T[i] <= T[i+1])
7          i += 1
8      return res
```

Q3 Il suffit d'adapter l'algorithme en pseudo-code.

Listing 2 – tri_bulles

```
def tri_bulles(T):
    N = len(T)
    for i in range(N-1,0,-1):
        for j in range(0,i):
            if T[j] > T[j+1]:
                T[j],T[j+1] = T[j+1],T[j]
```

Pour tester la fonction de tri, on peut utiliser le code suivant.

Listing 3 – Test du tri à bulles

```
import numpy.random as rd

res = True
    for n in range(1000):
        T = list(rd.randint(1000,size=100))
        tri(T)
        if not est_trie(T):
            res = False
            break
    if res:
        print("La fonction de tri pourrait etre correcte.")
    else:
        print("La fonction de tri n'est pas correcte.")
```

Remarque : ce test ne garantit pas que l'implémentation du tri à bulles est correcte. En revanche si une erreur est constatée on est assuré que l'implémentation est incorrecte.

2 Tri par insertion

Q4 Ci-dessous l'exécution de l'algorithme de tri par insertion.

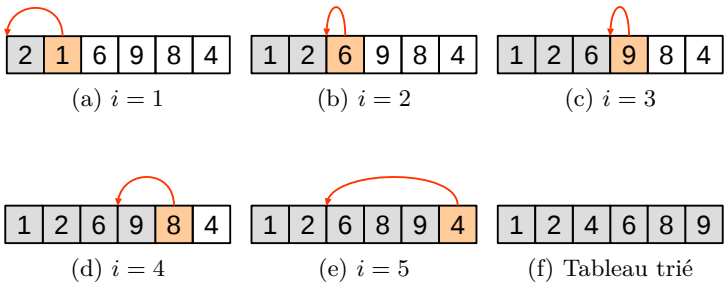

FIGURE 2 – Exemple d'exécution de l'algorithme de tri par insertion. Les cases gris clair représentent les éléments déjà triés, et la case orange l'élément à placer à la bonne position parmi les éléments précédents.

Q5 Là aussi il suffit d'adapter l'algorithme en pseudo-code.

Listing 4 – tri_insertion

```
def tri_insertion(T):
    N = len(T)
    for i in range(1,N):
        x = T[i]
        j = i
        while j > 0 and T[j-1] > x:
            T[j-1] = T[j]
            j -= 1
        T[j] = x
```

3 Tri rapide version facile

Q6 Ci-dessous l'exécution de l'algorithme de tri rapide.

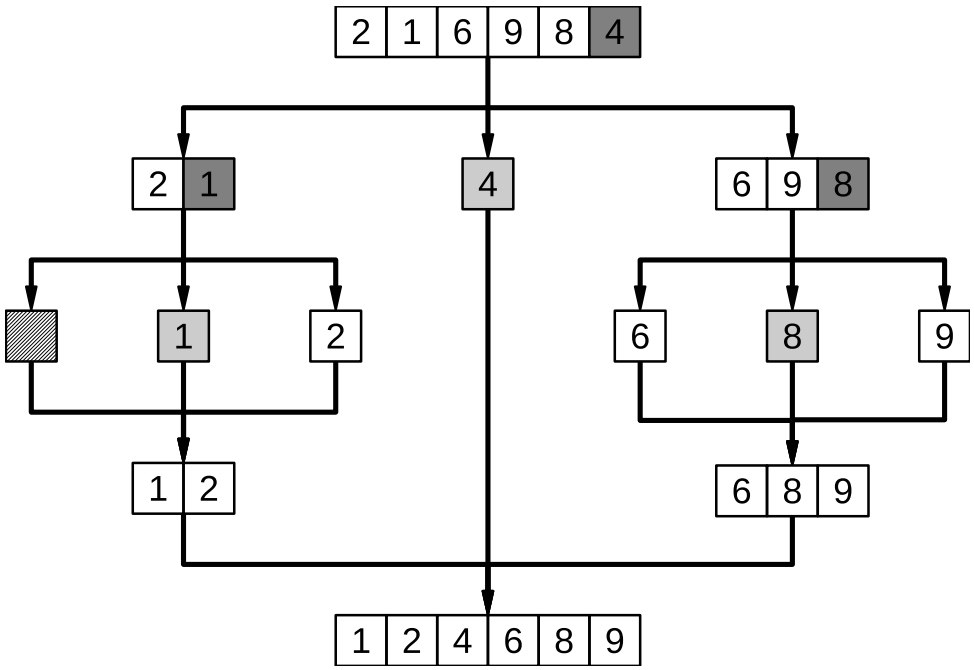

FIGURE 3 – Arbre représentant l'exécution de l'algorithme de tri rapide. En gris les pivots.

Q7 Il n'y a pas de difficulté particulière.

Listing 5 – partitionnement (version simple)

```
def partitionnement(T):
    N = len(T)
    pivot = T[-1]
    Tinf = []
    Tsup = []
```

```
6       for i in range(N-1):
7           if T[i] < pivot:
8               Tinf.append(T[i])
9           else:
10              Tsup.append(T[i])
11      return (Tinf,Tsup)
```

Q8 Une fois n'est coutume il suffit d'adapter l'algorithme en pseudo-code.

Listing 6 – tri_rapide (version en place)

```
1   def tri_rapide(T):
2       N = len(T)
3       if N > 1:
4           pivot = T[-1]
5           (Tinf,Tsup) = partitionnement(T)
6           Ttrie = tri_rapide(Tinf)+[pivot]+tri_rapide(Tsup)
7           return Ttrie
8       else:
9           return T
```

4 Tri rapide version « en place »

Q9 Ci-dessous l'exécution de l'algorithme de partitionnement en place.

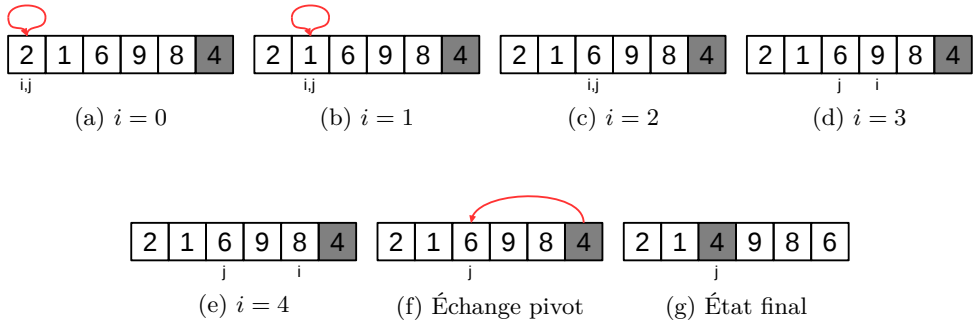

FIGURE 4 – Exemple d'exécution du partitionnement en place, pour $debut = 0$ et $fin = 5$. En-dessous du tableau on indique les valeur des indices i et j, en gris foncé le pivot, et les échanges d'éléments sont représentés par les flèches rouges.

Q10 Ci-dessous l'arbre d'exécution de la fonction de tri rapide en place.

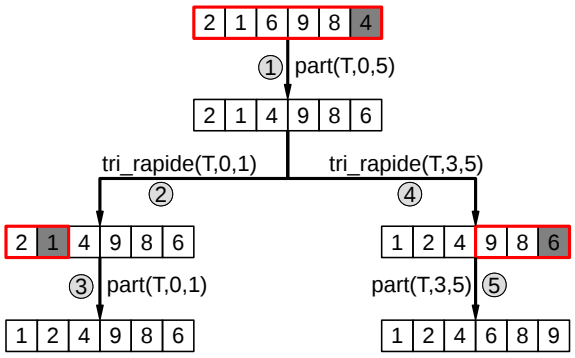

FIGURE 5 – Arbre représentant l'exécution d'un appel à la fonction de tri rapide en place. La fonction partitionnement a été abrégée en « part ».

Q11 Oubli dans l'énoncé du TP : il fallait bien sûr également implémenter la fonction `partitionnement_en_place`. Aucune des deux fonctions ne présentait de difficulté particulière, car il suffisait d'adapter les algorithmes donnés en pseudo code.

Listing 7 – partitionnement_en_place

```
1 def partitionnement_en_place(T, debut, fin):
2     j = debut
3     for i in range(debut,fin):
4         if T[i] <= T[fin]:
5             T[j],T[i] = T[i],T[j]
6             j += 1
7     T[j],T[fin] = T[fin],T[j]
8     return j
```

Listing 8 – tri_rapide_en_place

```
1 def tri_rapide_en_place(T,debut,fin):
2     if debut < fin:
3         position_pivot = partitionnement_en_place(T,debut,fin)
4         tri_rapide_en_place(T,debut,position_pivot-1)
5         tri_rapide_en_place(T,position_pivot+1,fin)
```