Chapitre 8 : Algorithmes de tris

Mots clés : *tri par insertion, tri par sélection, tri fusion, tri rapide*

I Introduction

Près d'un quart des opérations effectuées par un ordinateur consistent à trier.

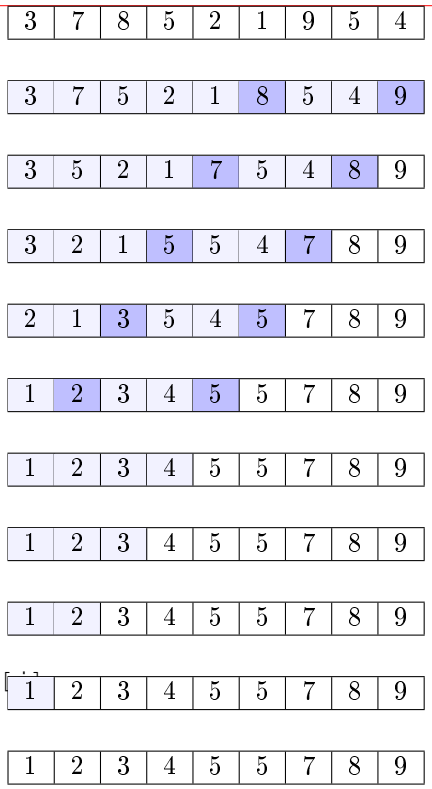
Dans tout ce chapitre, les listes sont considérées comme des listes non vides de nombres entiers, que l'on souhaite trier par **ordre croissant**.

II Tri à bulles

Le principe du tri à bulles est le suivant :

- à chaque étape i, on fait remonter les éléments supérieurs à leur successeur, ce qui a pour conséquence de mettre le plus grand élément de $L[:n-i]$ en position $n-1-i$:
  - on compare chaque élément de la liste avec son suivant et s'il est plus grand que son suivant alors on échange les deux
  - on répète cela jusqu'à la position $n-i-1$
- on renvoie la liste (étape facultative)

```
def tri_a_bulles(L: list) -> list:
    n = len(L)
    for i in range(n):
        for j in range(0, n-i-1):
            if L[j] > L[j+1] :
                L[j], L[j+1] = L[j+1], L[j]
    return L  # facultatif
```

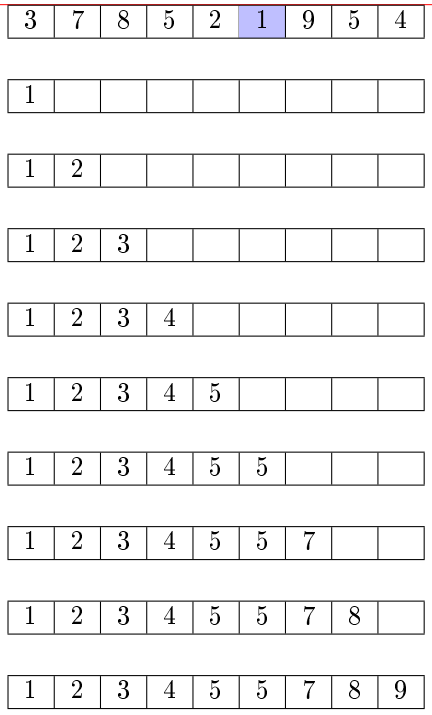


III Tri par sélection

Le principe du tri par sélection est le suivant :

- à chaque étape i entre 0 et $n-1$
  - rechercher le (premier) plus petit élément de la liste `L[i:]`
  - échanger ce minimum avec l'élément d'indice i,

```
def tri_selection(L: list) -> list:
```



Version récursive.

Compléter les fonctions suivantes pour mettre en place la version récursive de l'algorithme de tri par sélection.

```
def recherche_min(L: list) -> int:
    m = L[0]
    pos = 0
    for j in range(1, len(L)):



    return pos
```

```
def tri_selection_rec(L):
    if len(L) <= 1:
        return L
    else:
        pos = recherche_min(L)

        return [L[0]] +
```

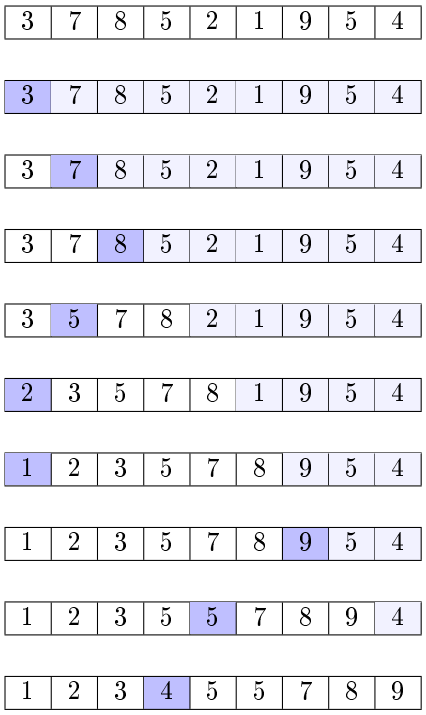
IV Tri par insertion

Le tri par insertion est le tri naturel pour trier une main de cartes, au fur et à mesure que les cartes sont distribuées.

Tri par insertion.
On parcourt le tableau à trier du début à la fin :

- on insère le $i-$ème élément à sa place parmi ceux qui précèdent :
  - on trouve son emplacement en le comparant aux éléments précédents,
  - on décale les éléments afin de pouvoir l'insérer
- au moment où l'on considère le $i-$ème élément, les éléments qui le précèdent sont déjà triés.

```
def tri_insertion(L: list) -> list:
  for i in range(len(L)):
```



V Tri fusion

Diviser pour régner : version récursive naturelle. Compléter la fonction `fusionner(A, B)` qui à partir de deux listes **triées** `A` et `B` renvoie une liste triée obtenue à partir des deux listes.

```
def fusionner(A: list, B:list) -> list:
    if len(A) == 0: # liste vide
        return B
    if len(B) == 0:
        return A
    if A[0] <= B[0]:
        return [ A[0] ] + fusionner( A[1:], B)  # concatène deux listes
    else:
        return
```

```
>>> fusionner([3], [2])
[2, 3]
```

```
>>> fusionner([1, 4, 7], [2, 3, 9, 11])
[1, 2, 3, 4, 7, 9, 11]
```

- Si le tableau n'a qu'un élément, il est déjà trié.
- Sinon : on sépare le tableau en deux parties "égales".
- On trie les deux parties à l'aide du tri fusion.
- On fusionne les parties triées.

```
def tri_fusion(L: list) -> list:
    if len(L) <= 1:
        return L
    else:
        return
```