Un algorithme de Tri prend en paramètre un tableau d'éléments d'ordonnables, par exemple une liste de nombres et renvoie le tableau trié dans le sens croissant.

Exercice 1

Écrire un algorithme de tri de son choix. Quels sont sa complexité dans le pire des cas ? Dans le meilleur des cas ?

Exercice 2

Tri par insertion. C'est celui que l'on utilise habituellement dans la vie courante, par exemple, pour trier un paquet de carte :

On constitue (imaginairement) 2 tas :

- l'un dans la main droite, contenant toutes les cartes avant tri,
- l'autre dans la main gauche, contenant les cartes déjà triées,

Initialement la main gauche est vide.

Chaque étape consiste à :

- prendre la première carte du tas non trié
- L'insérer progressivement à sa bonne place dans le tas trié, en la faisant descendre d'une position tant que sa valeur reste inférieure à celle de la carte située en dessous-d'elle.

Après chaque étape le tas non trié contient une carte de moins, le tas trié une carte de plus.

A la fin du tri la main droite est vide. La main gauche contient toutes les cartes, triées.

Exemple : (en gras la partie du tableau non encore ordonnée (main droite))

Tableau :	**3**	**2**	**1**	**5**	**4**
Insertion 1	**3**	**2**	**1**	**5**	**4**
Insertion 2	3	**2**	**1**	**5**	**4**
	2	3	**1**	**5**	**4**
Insertion 3	2	3	**1**	**5**	**4**
	2	**1**	3	**5**	**4**
	1	2	3	**5**	**4**
Insertion 4	1	2	3	**5**	**4**
Insertion 5	1	2	3	5	**4**
	1	2	3	**4**	5
Tableau trié :	1	2	3	4	5

Remarque : La première insertion est toujours inutile : on l'évitera.

1. Ecrire une fonction `triInsertion` prenant en paramètre une liste de nombres et qui applique un tri par insertion pour le trier dans le sens croissant. La fonction débutera ainsi :

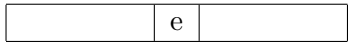
```
def triInsertion(T):
    n = len(T):
    for k in range(1,n):
        ...
```

2. Quel invariant de boucle permet de justifier qu'à la fin de l'algorithme le tableau est trié ?
3. Complexité dans le meilleur des cas : quel est le meilleur des cas ? Quelle est alors la complexité ?
4. Complexité dans le pire des cas : quel est le pire des cas ? Quelle est alors la complexité ?

Exercice 3

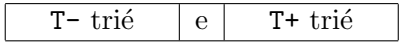
Tri rapide. Ce tri particulièrement rapide, est basé sur le principe de diviser pour régner :

- Choisir arbitrairement un élément `e` dans le tableau `T`,



- Décomposer les autres éléments du tableau en deux sous-tableaux :
  - `T-` constitué des éléments inférieurs à e,
  - `T+` constitué des éléments supérieurs à e.

- Après appels récursifs sur les deux sous-tableaux `T-` et `T+`, le tableau `T` trié s'obtient en concaténant les sous-tableaux `T-` trié, suivi de `e`, suivi de `T+` trié.



Le tableau `T` est alors trié.

Exemple : (1) Tri rapide à la main de `[7, 3, 6, 4, 2, 5, 1]` :

– Appel sur `T = [7, 3, 6, 4, 2, 5, 1]`
`e = 4, T1 = [3, 2, 1], T2 = [7, 6, 5]`

– Appel récursif sur `T1 = [3, 2, 1]` :
`e = 2, T11 = [1], T12 = [3]`

$\Longrightarrow$ `T1` devient : `T11 + [2] + T12 = [1, 2, 3]`

– Appel récursif sur `T2 = [7,6,5]`
`e = 6, T21 = [5], T22 = [7]`

$\Longrightarrow$ `T2` devient : `T21 + [6] + T22 = [5,6,7]`

$\Longrightarrow$ `T` devient : `T1 + [4] + T2 = [1,2,3,4,5,6,7]`

1. Ecrire une fonction `triRapide` prenant en paramètee une liste de nombres `T` et qui renvoie un tableau trié contenant les mêmes éléments que `T`.

2. Lorsque l'élément `e` est chaque fois choisi médian, quelle et la complexité de l'algorithme ?

3. Lorsque l'élément `e` est chaque fois choisi extremal, quelle est la compléxité de l'algorithme ?

Exercice 4

Tri fusion. Le tri fusion est un autre exemple d'algorithme de tri basé sur le principe de "Diviser pour régner".

Il est basé sur le fait que fusionner deux tableaux triés en un seul tableau trié `T1`, `T2` se fait en temps linéaire (sur le nombre total d'éléments, dans le pire et le meilleur des cas) :

```
i1, i2 = 0
T = tableau vide
TANT QUE l'on n'a pas atteint la fin d'un des tableaux:
    SI T1[i1] <= T2[i2] ALORS
        Insérer T1[i1] à la fin de T
        incrémenter i1
    SINON
        Insérer T2[i2] à la fin de T
        incrémenter i2
FIN TANT QUE
Insérer les éléments restants du tableau non vide en fin de
T
```

On met alors en oeuvre la récursivité :

– Décomposer le tableau en deux sous-tableaux de même longueur (± 1)
– Appliquer l'algorithme récursivement sur chacun des 2 sous-tableaux
– Puis fusionner les 2 sous-tableaux triés en un tableau trié.

1. Ecrire une fonction `fusion` prenant en paramètre deux listes triées, et qui les fusionne en une liste triée dans le sens croissant, qu'elle renverra.
2. Ecrire une fonction `triFusion` qui exécute le Tri fusion.
3. Quelle est sa complexité dans le pire des cas ? dans le meilleur des cas ?