Une opération de grand usage en informatique est le tri d'un tableau de nombres. C'est à dire, donné un tableau de nombres (une liste de nombres en `python`), le modifier pour ordonner toutes ses valeurs, par exemple dans le sens croissant. (Ou alternativement retourner un autre tableau contenant exactement les mêmes valeurs mais ordonnées dans le sens croissant.)

1 Le tri à bulle

1.1 Principe

C'est l'algorithme de tri le plus simple :

- Parcourir les éléments du tableau de gauche à droite.
  - Dès que l'on rencontre deux éléments consécutifs qui ne sont pas dans le bon ordre, on échange leur position. C'est à dire :
    `SI tableau[i] > tableau[i+1]`
    `ALORS : échanger liste[i] et liste[i+1]`
- Recommencer tant que l'on a changé quelque chose.

Exemple :

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

1.2 Algorithme

Le principe illustré ci-dessus conduit à l'algorithme général suivant pour le tri d'un tableau unidimensionnel (une liste en `python`) :

Algorithme du Tri à bulle `(Paramètre : un tableau T)`

```
N = longueur(T)
Changement = VRAI
TANT QUE Changement est VRAI :
    Changement = FAUX
    POUR  i variant de 0 à N-2
        SI T[i] > T[i+1]
        ALORS  échanger T[i] et T[i+1]
               Changement = VRAI
    FIN POUR
FIN TANT QUE
```

On peut l'améliorer : on démontre qu'à la fin du premier parcours du tableau, le dernier élément du tableau est la plus grande valeur : il est donc à sa place définitive. En effet, supposons que le maximum soit disposé à l'indice k, alors durant le premier parcours du tableau il sera successivement déplacé à l'indice $k+1$, $k+2$, ..., jusqu'à l'indice $N-1$ (en effet, toutes les comparaisons `T[k] > T[k+1]`, `T[k+1] > T[k+2]`, ..., `T[N-2] > T[N-1]` étantnt successivement vérifiées).

Ainsi après le i-ème parcours du tableau, tous les i derniers éléments sont à leurs places définitives. Donc à chaque parcours de tableau, le parcours pourra s'arrêter un indice avant le précédent. L'algorithme devient :

Algorithme du Tri à bulle `(Paramètre : un tableau T)`

```
N = longueur(T)
Changement = VRAI
TANT QUE Changement est VRAI :
    Changement = FAUX
    POUR  i variant de 0 à N-2
        SI T[i] > T[i+1]
        ALORS  échanger T[i] et T[i+1]
               Changement = VRAI
    FIN POUR
    N = N-1
FIN TANT QUE
```

Il doit son nom au fait que les éléments remontent progressivement, de proche en proche, un peu comme des bulles d'air remontant progressivement à la surface d'un liquide.

1.3 Code Python

```
def tri_bulle(T):
    N = len(T)
    changement = True
    while changement == True:
        changement = False
        for k in range(N-1):
            if T[k] > T[k+1]:
                print(T, " échange :",k,k+1 ) # Affichage (optionnel)
                T[k], T[k+1] = T[k+1], T[k]
                changement = True
        N = N-1
        print('---------------') # Affichage fin de parcours (opt.)
    return T
```

Exemple d'utilisation :

```
In[1]: T = [3, 2, 1, 5, 4]
In[2]: tri_bulle(T)
[3, 2, 1, 5, 4]    échange : 0 1
[2, 3, 1, 5, 4]    échange : 1 2
[2, 1, 3, 5, 4]    échange : 3 4
---------------
[2, 1, 3, 4, 5]    échange : 0 1
---------------
---------------
Out[2]: [1, 2, 3, 4, 5]
```

1.4 Complexité

Soit `N` la longueur du tableau, c'est à dire son nombre d'éléments. Puisque comme vu plus haut, après chaque passage un nouvel élément est à sa place définitive, la boucle `while` s'effectuera au plus `N-1` fois. A chaque exécution de la boucle, il y aura 2 affectations et les opérations dans la boucle `for` (1 comparaison et au plus 1 échange et une affectation pour chaque passage dans la boucle `for`). Le nombre total d'opérations élémentaires est donc majoré par :

$$\sum_{k=0}^{N-2}(2+k.3) = 2.(N-1) + 3\sum_{k=0}^{N-2} k = 2.(N-1) + \frac{3}{2}(N-1)(N-2) = \frac{3}{2}N^2 - \frac{5}{2}N + 1.$$

C'est un polynôme de variable N et de degré 2. Le nombre d'opérations nécessaires au tri d'un tableau de longueur N est au plus quadratique dans le pire des cas.
Le pire des cas ici c'est lorsque le tableau est trié mais dans le sens décroissant. Il y a alors exactement $\frac{3}{2}N^2 - \frac{5}{2}N + 1$ opérations : **dans le pire des cas la complexité est quadratique**, on écrit : **complexité d'ordre $O(N^2)$ dans le pire des cas**.

Exemple : le pire des cas.

```
In[3] : T = [5, 4, 3, 2, 1]
In[4]: tri_bulle(T)
[5, 4, 3, 2, 1]     échange : 0 1
[4, 5, 3, 2, 1]     échange : 1 2
[4, 3, 5, 2, 1]     échange : 2 3
[4, 3, 2, 5, 1]     échange : 3 4
---------------
[4, 3, 2, 1, 5]     échange : 0 1
[3, 4, 2, 1, 5]     échange : 1 2
[3, 2, 4, 1, 5]     échange : 2 3
---------------
[3, 2, 1, 4, 5]     échange : 0 1
[2, 3, 1, 4, 5]     échange : 1 2
---------------
[2, 1, 3, 4, 5]     échange : 0 1
---------------
---------------
Out[4]: [1, 2, 3, 4, 5]
```

Dans le meilleur des cas, lorsque le tableau est déjà trié, la boucle `while` se déroule une seule fois, il y a un seul parcours du tableau (boucle `for`) (sans modification). Le nombre d'opérations est un polynôme de variable N et de degré 1 : la complexité dans le meilleur des cas est linéaire, d'ordre $O(N)$.

Exemple : le meilleur des cas.

```
In[5]: T = [1, 2, 3, 4, 5]
In [6]: tri_bulle(T)
---------------
Out[28]: [1, 2, 3, 4, 5]
```