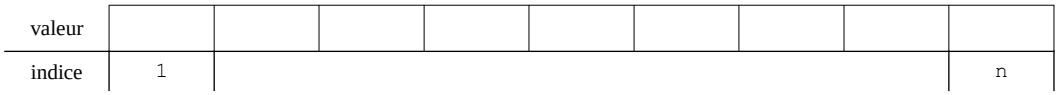
Le tri à bulles

On considère le tableau T de taille n :



Pour i ∈ {1, 2 , … , n}, T(i) est la valeur du tableau T situé à l'indice i.

Objectif : on souhaite trier le tableau par ordre croissant : T(1) ⩽ T(2) ⩽ … ⩽ T(n).
Le tri se fera par comparaison et permutation.

I. Le tri à bulles (bubble-sort)

a) Après le premier passage, la valeur située à l'indice n dans le tableau T est la plus grande valeur du tableau.

Algorithme :
 	
Bubble sort
 	
T est un tableau de taille n
	`for i in reverse 2, ... , n loop`
	`  for j in 1, ... , i - 1 loop`
	`  if T(j) > T(j + 1) then`
	`  T(j) ↔ T(j + 1)`
	`  end if`
	`  end loop`
	`end loop`

Preuve :

▪ Terminaison

Il s'agit de deux boucles finies imbriquées, donc l'algorithme termine.

▪ Validité

Invariants :

- P_i : dans la tranche d'indices allant de i + 1 à n, on trouve les plus grandes valeurs du tableau rangées en ordre croissant.
- N_j : dans la tranche d'indices allant de 1 à j, T(j) est le plus grand élément de la tranche

```
-- Pn est trivialement vraie : de n + 1 à n la tranche contient un élément
for i in reverse 2, ... , n loop
      -- Pi est vraie
      -- N1 est trivialement vraie : T(1) est le plus grand élément de la tranche allant de 1 à 1
      for j in 1, ... , i - 1 loop
            -- Nj est vraie
            if T(j) > T(j + 1) then
                  T(j) ↔ T(j + 1)
            end if
            -- Nj+1 est vraie : Si T(j + 1) est plus grand que T(j) on change sinon rien.
      end loop
      -- Ni est vraie : T(i) est le plus grand élément de la tranche allant de 1 à i
      -- Pi-1 est vraie : on a fait remonter le plus grand élément de la tranche allant de 1 à i
            cet élément est le plus grand des éléments de 1 à i et devient le plus petit des éléments
            de i à n. Ainsi la tranche de i à n est triée dans l'ordre croissant.
end loop
-- P1 est vraie. Donc la tranche de 2 à n contient les plus grandes valeurs triées dans l'ordre croissant.
      Et donc T(1) étant plus petit que toutes les autres valeurs, T est trié dans l'ordre croissant.
```

b) Il y a $\sum_{i=1}^{n-1} k = \frac{n(n-1)}{2}$ comparaisons d'éléments.

- Il y a donc, au plus, $\frac{n(n-1)}{2}$ échanges d'éléments dans le tableau.
  Le pire cas est atteint lorsque le tableau est dans l'ordre décroissant et que les valeurs sont toutes différentes.
  Par exemple : `T = [10, 8, 7, 6, 4, 2, 0]`
- Et, au minimum, aucun échange.
  Le meilleur cas est atteint lorsque le tableau est déjà trié.
  Par exemple : `T = [2, 3, 4, 4, 5, 6]`

II. Améliorations

a)

Algorithme :
 	
Bubble sort
 	
Amélioration sur le nombre de comparaisons
 	
T est un tableau de taille n
	sup ← **n**
	while **sup** ⩾ **2** loop
	  borne ← **0**
	  for **j** in **1, ... , sup - 1** loop
	  if T(j) > T(j + 1) then
	  T(j) ↔ T(j + 1)
	  borne ← **j**
	  end if
	  end loop
	  sup ← **borne**
	end loop

Soit `k`, le plus grand indice à partir duquel les échanges ne se font plus, les éléments de la tranche allant de `1` à `k - 1` sont plus petits que les éléments (triés) de la tranche allant de `k + 1` à `n`.

Ce tri est optimisé pour les comparaisons (il en fait moins), le nombre d'échanges reste le même.

b) On peut effectuer le tri dans les 2 sens et mémoriser le plus grand indice à partir duquel les échanges ne se font plus lorsque l'on va de gauche à droite et le plus petit indice à partir duquel les échanges ne se font plus lorsque l'on va de droite à gauche.

Algorithme :
 	
Shaker sort
 	
T est un tableau de taille n
	sup ← **n**
	inf ← **1**
	borne ← **0**
	while **sup** > **inf** loop
	  for **i** in **inf, ... , sup - 1** loop
	  if T(i) > T(i + 1) then
	  T(i) ↔ T(i + 1)
	  borne ← **i**
	  end if
	  end loop
	  sup ← **borne**
	  for **j** in reverse **inf + 1, ... , sup** loop
	  if T(j) < T(j - 1) then
	  T(j) ↔ T(j - 1)
	  borne ← **j**
	  end if
	  end loop
	  inf ← **borne**
	end loop

c) Il y a également, au plus, $\frac{n(n-1)}{2}$ échanges d'éléments dans le tableau. Le pire cas est atteint lorsque le tableau est dans l'ordre décroissant et que les valeurs sont toutes différentes. Par exemple : `T = [10, 8, 7, 6, 4, 2, 0]`.

Le shaker sort n'apporte pas, non plus, d'amélioration sur le nombre d'échanges réalisé par l'algorithme. Par contre, il diminue significativement le nombre de comparaisons et donc le temps d'exécution.

III. Scripts de test

```
import random, time
import pylab as plt


def triBulles(tableau):
    """ Tri bulles sur la liste passée en paramètre.
    Retourne le nombre de comparaisons et d'échanges effectués."""
    comparaisons, echanges = 0, 0
    for i in range(len(tableau) - 1, 0, -1):
        for j in range(0, i):
            if tableau[j] > tableau[j + 1]:
                tableau[j], tableau[j + 1] = tableau[j + 1], tableau[j]
                echanges += 1
            comparaisons += 1
    return comparaisons, echanges


def triCocktail(tableau):
    """ Tri cocktail sur la liste passée en paramètre.
    Retourne le nombre de comparaisons et d'échanges effectués."""
    comparaisons, echanges = 0, 0
    sup = len(tableau) - 1
    inf = 0
    borne = -1
    while sup > inf:
        # Tri de gauche à droite
        for i in range(inf, sup):
            if tableau[i] > tableau[i + 1]:
                tableau[i], tableau[i + 1] = tableau[i + 1], tableau[i]
                echanges += 1
                borne = i
            comparaisons += 1
        sup = borne
        # Tri de droite à gauche
        for j in range(sup, inf, -1):
            if tableau[j] < tableau[j - 1]:
                tableau[j], tableau[j - 1] = tableau[j - 1], tableau[j]
                echanges += 1
                borne = j
            comparaisons += 1
        inf = borne
    return comparaisons, echanges


def complexiteTri(tri, tailleEchantillon, tailleTableau, nbDouble):
    """ Test la complexité du tri bulles
    On simule un échantillon de tailleEchantillon tableau de taille tailleTableau
    On calcule le nb moyen de comparaisons, d'echanges et le temps moyen du tri.
    Puis on double la taille nbDouble fois et recueille les mêmes informations."""
    # On utilise des tableaux pour les tracés
    tailleTableaux = []
    comparaisons, echanges, durees = [], [], []
    # On créé nbDouble échantillons
    for i in range(nbDouble):
        tailleTableaux.append(tailleTableau)
        sommeComparaisons, sommeEchanges, sommeDurees = 0, 0, 0
        # On créé des échantillons de tailleEchantillon
        for j in range(tailleEchantillon):
            tableau = [random.randrange(0, 20000) for i in range(tailleTableau)]
            start_time = time.perf_counter()
            nbComparaisons, nbEchanges = tri(tableau)
            duree = time.perf_counter() - start_time
            sommeComparaisons += nbComparaisons
            sommeEchanges += nbEchanges
            sommeDurees += duree
        moyComparaisons = sommeComparaisons / tailleEchantillon
        moyEchanges = sommeEchanges / tailleEchantillon
        moyDurees = round(sommeDurees / tailleEchantillon, 8)
        comparaisons.append(moyComparaisons)
        echanges.append(moyEchanges)
        durees.append(moyDurees)
        # impression des résultats
        print("Pour", tailleEchantillon, "tableaux de taille", tailleTableau, ": ")
        print("    nb moyens comparaisons du tri :", moyComparaisons)
        print("    nb moyens échanges du tri :", moyEchanges)
        print("    durée moyenne du tri (en sec) :", moyDurees, '\n')
        tailleTableau *= 2        # On double la taille du tableau
    return tailleTableaux, comparaisons, echanges, durees
```

Représentation graphique

```
def plotComplexite(resultatsTri1, resultatsTri2):
    """ Procédure permettant d'obtenir une représentation graphique
    Comparaisons entre les tris"""
    plt.figure("Comparaison Tris")
    plt.subplot(1, 3, 1)
    plt.title('Comparaisons')
    plt.xlabel("Taille du tableau"), plt.ylabel("Nb de comparaisons")
    plt.plot(resultats1[0], resultats1[1], 'r', label = "Bubble sort")
    plt.plot(resultats2[0], resultats2[1], 'g', label = "Shaker sort")
    plt.ylim(ymin = 0), plt.xlim(xmin = 0)
    plt.xticks(rotation = 90), plt.legend(loc = 'upper left')
    plt.subplot(1, 3, 2)
    plt.title('Echanges')
    plt.xlabel("Taille du tableau"), plt.ylabel("Nb d'échanges")
    plt.plot(resultats1[0], resultats1[2], 'r', label = "Bubble sort")
    plt.plot(resultats2[0], resultats2[2], 'g', label = "Shaker sort")
    plt.ylim(ymin = 0), plt.xlim(xmin = 0)
    plt.xticks(rotation = 90), plt.legend(loc = 'upper left')
    plt.subplot(1, 3, 3)
    plt.title('Durées du tri')
    plt.xlabel("Taille du tableau"), plt.ylabel("Durée du tri en sec")
    plt.plot(resultats1[0], resultats1[3], 'r', label = "Bubble sort")
    plt.plot(resultats2[0], resultats2[3], 'g', label = "Shaker sort")
    plt.ylim(ymin = 0), plt.xlim(xmin = 0)
    plt.xticks(rotation = 90), plt.legend(loc = 'upper left')
    plt.show()
```

Implémentations originales des tris en Python

- Tri Bulles

```
def triBulles(tableau):
    """ Procédure implémentant le tri bulles sur la liste passée en paramètre.
    Tri en place."""
    for i in range(len(tableau) - 1, 0, -1):
        for j in range(0, i):
            if tableau[j] > tableau[j + 1]:
                tableau[j], tableau[j + 1] = tableau[j + 1], tableau[j]
```

- Tri Cocktail

```
def triCocktail(tableau):
    """ Procédure implémentant le tri cocktail sur la liste passée en paramètre.
    Tri en place."""
    sup = len(tableau) - 1
    inf = 0
    borne = -1
    while sup > inf:
        # Tri bulle de gauche à droite
        for i in range(inf, sup):
            if tableau[i] > tableau[i + 1]:
                tableau[i], tableau[i + 1] = tableau[i + 1], tableau[i]
                borne = i
        sup = borne
        # Tri bulle de droite à gauche
        for j in range(sup, inf, -1):
            if tableau[j] < tableau[j - 1]:
                tableau[j], tableau[j - 1] = tableau[j - 1], tableau[j]
                borne = j
        inf = borne
```

IV. Résultats comparaison des tris

Bubble Sort	Shaker Sort
`>>> complexiteTri(triBulles, 100, 100, 10)`	`>>> complexiteTri(triCocktail, 100, 100, 10)`
Pour **100** tableaux de taille **100** : nb moyens comparaisons du tri : **4950.0** nb moyens échanges du tri : **2486.11** durée moyenne du tri (en sec) : **0.00074214**	Pour **100** tableaux de taille **100** : nb moyens comparaisons du tri : **3382.05** nb moyens échanges du tri : **2478.42** durée moyenne du tri (en sec) : **0.00060968**
Pour 100 tableaux de taille **200** : nb moyens comparaisons du tri : **19900.0** nb moyens échanges du tri : **9945.01** durée moyenne du tri (en sec) : **0.00298818**	Pour 100 tableaux de taille **200** : nb moyens comparaisons du tri : **13365.09** nb moyens échanges du tri : **9881.62** durée moyenne du tri (en sec) : **0.00241946**
Pour 100 tableaux de taille **400** : nb moyens comparaisons du tri : **79800.0** nb moyens échanges du tri : **39821.82** durée moyenne du tri (en sec) : **0.0123644**	Pour 100 tableaux de taille **400** : nb moyens comparaisons du tri : **53705.64** nb moyens échanges du tri : **40006.17** durée moyenne du tri (en sec) : **0.01031692**
Pour 100 tableaux de taille **800** : nb moyens comparaisons du tri : **319600.0** nb moyens échanges du tri : **159424.25** durée moyenne du tri (en sec) : **0.05347027**	Pour 100 tableaux de taille **800** : nb moyens comparaisons du tri : **213836.54** nb moyens échanges du tri : **159697.27** durée moyenne du tri (en sec) : **0.04504276**
Pour 100 tableaux de taille **1600** : nb moyens comparaisons du tri : **1279200.0** nb moyens échanges du tri : **638436.81** durée moyenne du tri (en sec) : **0.23214567**	Pour 100 tableaux de taille **1600** : nb moyens comparaisons du tri : **853767.38** nb moyens échanges du tri : **638981.27** durée moyenne du tri (en sec) : **0.18900067**
Pour 100 tableaux de taille **3200** : nb moyens comparaisons du tri : **5118400.0** nb moyens échanges du tri : **2559796.83** durée moyenne du tri (en sec) : **0.94635251**	Pour 100 tableaux de taille **3200** : nb moyens comparaisons du tri : **3412197.94** nb moyens échanges du tri : **2555997.0** durée moyenne du tri (en sec) : **0.76286488**
Pour 100 tableaux de taille **6400** : nb moyens comparaisons du tri : **20476800.0** nb moyens échanges du tri : **10244544.67** durée moyenne du tri (en sec) : **3.77353053**	Pour 100 tableaux de taille **6400** : nb moyens comparaisons du tri : **13644446.47** nb moyens échanges du tri : **10225369.32** durée moyenne du tri (en sec) : **3.02681951**
Pour 100 tableaux de taille **12800** : nb moyens comparaisons du tri : **81913600.0** nb moyens échanges du tri : **40925849.95** durée moyenne du tri (en sec) : **15.11234434**	Pour 100 tableaux de taille **12800** : nb moyens comparaisons du tri : **54595860.96** nb moyens échanges du tri : **40938709.3** durée moyenne du tri (en sec) : **12.01310291**
Pour 100 tableaux de taille **25600** : nb moyens comparaisons du tri : **327667200.0** nb moyens échanges du tri : **163688366.34** durée moyenne du tri (en sec) : **61.38221671**	Pour 100 tableaux de taille **25600** : nb moyens comparaisons du tri : **218335746.32** nb moyens échanges du tri : **163737322.26** durée moyenne du tri (en sec) : **48.68246473**
Pour 100 tableaux de taille **51200** : nb moyens comparaisons du tri : **1310694400.0** nb moyens échanges du tri : **655237699.07** durée moyenne du tri (en sec) : **252.58576397**	Pour 100 tableaux de taille **51200** : nb moyens comparaisons du tri : **873578395.39** nb moyens échanges du tri : **655247329.1** durée moyenne du tri (en sec) : **195.63896141**

On vérifie que, lorsque l'on double la taille du tableau, le nombre de comparaisons, d'échanges et la durée du tri sont multipliés par 4. Ce qui caractérise une complexité quadratique.

```
>>> plotComplexite(resultatsTriBulles, resultatsTriCocktail)
```