I Trouver les deux valeurs les plus proches dans un tableau

1. On se donne un tableau `T` unidimensionnel. Écrire une fonction `distance_min1(T)` qui renvoie les deux éléments qui sont les plus proches ie dont la valeur absolue de la différence est minimale. On indiquera les valeurs obtenues ainsi que les indices correspondants.
2. Pour un tableau à n cases, montrer que le nombre de comparaisons $C(n)$ faites dans cette fonction est tel que la suite $\left(\frac{C(n)}{n^2}\right)$ est bornée : on dit que la **complexité est quadratique**.
3. Faire la même question pour un tableau bidimensionnel en écrivant une fonction `distance_min2(T)`. Ici, `T` sera donc une matrice pas nécessairement carrée par exemple de la forme

```
T=[[1,2,3],[6,4,3],[3,8,9],[3,-2,0]]:
```

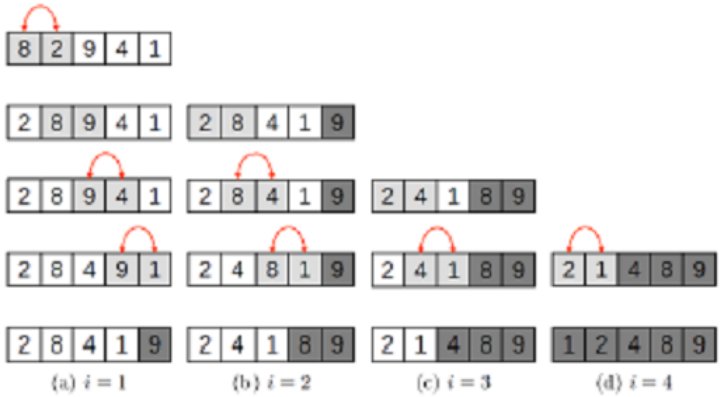
chaque élément de `T` désignera une ligne du tableau. Le nombre de ligne est le nombre d'éléments de `T`, le nombre de colonnes est le nombre d'éléments de `T[0]`.

II Recherche d'un mot dans un texte

1. Écrire une fonction `est_ici(texte,motif)` qui, étant données deux chaines de caractères `texte` et `motif`, renvoie `True` ou `False` selon que mot est ou n'est pas dans texte. On n'utilisera pas de slicing mais on fera deux méthodes, l'une sans booléen explicite, l'autre avec.
2. Écrire une fonction `est_sous_mot(texte,motif)` qui renvoie `True` ou `False` selon que motif est dans texte ou pas.
3. Écrire une fonction `position_sous_mot(texte,motif)` qui renvoie la liste de toutes les occurences de l'indice de position du mot `motif` dans `texte`.

III Tri à bulles

Le tri à bulles est un algorithme de tri classique. Son principe est simple, et il est très facile à implémenter. On considère un tableau de nombres T, de taille N. L'algorithme parcourt le tableau, et dès que deux éléments consécutifs ne sont pas ordonnés, les échange. Après un premier passage, on voit que le plus grand élément se situe bien en dernière position. On peut donc recommencer un tel passage, en s'arrêtant à l'avant-dernier élément, et ainsi de suite.
Au i-ème passage on fait remonter le i-ème plus grand élément du tableau à sa position définitive, un peu à la manière de bulles qu'on ferait remonter à la surface d'un liquide, d'où le nom d'algorithme de tri à bulles.



1. Appliquer l'algorithme de tri à Bulles «à la main» au tableau ci-dessous :

2	1	6	9	8	4

2. Ecrire une fonction `est_trie(T)` qui renvoie `True` ou `False` selon que le tableau `T` est trié ou pas.
3. Écrire une fonction `TriBulles(T)` triant le tableau `T` par l'algorithme de tri à bulles [1]
4. Vérifier que la fonction ci-dessus est correcte en utilisant la fonction `EstTrie` sur des tableaux de nombres aléatoires. On rappelle que la bibliothèque random permet de créer des nombres aléatoires. `random.randint(a,b)` renvoie un entier aléatoire compris entre a et b.
5. Montrer que la complexité est quadratique (notion définie dans I)
6. Ecrire une fonction améliorée `TriBulles2(T)` qui sort de la fonction dès que le tableau a été parcouru sans faire d'échange (auquel cas, il est trié et donc inutile de continuer).
7. Une variante du tri à bulles est le tri cocktail : il consiste à changer de direction à chaque passage. Lors du premier parcours, on se déplace du début du tableau vers la fin. Puis lors du second parcours on part de la fin du tableau pour arriver au début. A la troisième itération on part du début et ainsi de suite ... C'est une légère amélioration car il permet non seulement aux plus grands éléments de migrer vers la fin de la série mais également aux plus petits éléments de migrer vers le début.

   Écrire une fonction `TriCocktail(T)` qui étant donné un tableau `T` le renvoie trié selon la méthode du tri cocktail.

   Justifier en quoi cette variante peut être intéressante selon le type de tableau à trier mais vérifier néanmoins que la complexité reste quadratique.

1. cette fonction doit agir sur place concernant le tableau `T` ie doit le modifier par effet de bord.