Le but de ce TP est de travailler sur la complexité d'algorithmes souvent déjà étudiés.

Exercice 1 :

Cet exercice se fera uniquement sans ordinateur.

Dans cet exercice, L désigne une liste triée de nombres et a un nombre.

(1) a. Écrire un algorithme « naïf » qui prend en arguments L et a et renvoie le plus petit indice i tel que $L[i] = a$.

b. Analyser le complexité de cet algorithme.

(2) a. Écrire un algorithme plus rapide.

b. Analyser la complexité de celui-ci.

Exercice 2 :

Un problème classique en informatique consiste à rechercher, non pas une seule valeur, mais une séquence de valeurs dans un tableau. Cela revient à chercher une occurrence d'un tableau dans un autre ou, pour les chaînes de caractères, une occurrence d'un mot dans un texte.

On souhaite écrire une fonction recherche_mot qui, étant donnés deux tableaux m et t, détermine la position de la première occurrence de m dans t, si elle existe, et renvoie non sinon. Ainsi, pour lest tableaux $m = [1, 2, 3]$ et $t = [2, 1, 4, 1, 2, 6, 1, 2, 3, 7]$, recherche_mot renvoie 6 :

$$[2,\ 1,\ 3,\ 1,\ 2,\ 6,\ 1, 2, 3,\ 7].$$

(1) Écrire (sur papier) un algorithme répondant à ce problème.

(2) Montrer la terminaison et la validité de cet algorithme.

(3) Analyser la complexité de cet algorithme.

(4) Implémenter cet algorithme en Python.

(5) Évaluer le temps d'exécution de cet algorithme avec différentes listes. Est-ce cohérent avec la complexité théorique ?

Exercice 3 :

On se donne l'algorithme suivant :

```
def etape_bulle(L):
    echange_effectue = False
    for i in range(len(L)-1):
        if L[i] > L[i+1]:
            tmp = L[i+1]
            L[i+1] = L[i]
            L[i] = tmp
            echange_effectue = True
    return echange_effectue
```

(1) Tester la fonction **echange_effectue** sur la liste $L = [5, 1, 4, 2, 8]$.

(2) Quel est le but de celui-ci ?

(3) Déterminer la complexité de cet algorithme.

(4) Tester cette fonction sur la liste L jusqu'à ce qu'elle renvoie **False**.

(5) En déduire la fonction **tri_bulle** qui prend en argument une liste L et la tri.

(6) Déterminer la complexité de cet algorithme.

(7) Pourquoi l'appelle-t'on **tri à bulle** ?

(8) Programmer et tester cette fonction.

Exercice 4 :

Dans cette question, on cherche à écrire l'algorithme de tri par insertion que l'on a déjà vu en TP. On se donne la description de l'algorithme par Wikipedia :

Dans l'algorithme, on parcourt le tableau à trier du début à la fin. Au moment où on considère le i-ème élément, les éléments qui le précèdent sont déjà triés. Pour faire l'analogie avec l'exemple du jeu de cartes, lorsqu'on est à la i-ème étape du parcours, le i-ème élément est la carte saisie, les éléments précédents sont la main triée et les éléments suivants correspondent aux cartes encore mélangées sur la table.

L'objectif d'une étape est d'insérer le i-ème élément à sa place parmi ceux qui précèdent. Il faut pour cela trouver où l'élément doit être inséré en le comparant aux autres, puis décaler les éléments afin de pouvoir effectuer l'insertion. En pratique, ces deux actions sont fréquemment effectuées en une passe, qui consiste à faire « remonter » l'élément au fur et à mesure jusqu'à rencontrer un élément plus petit.

Exemple pour la liste $[9, 6, 1, 4, 8]$:

Étape	Liste au départ	Liste à l'arrivé.
Étape 1	9 6 1 4 8	6,9,1,4,8
Étape 2	6,9,1,4,8	1,6,9,4,8
Étape 3	1,6,9,4,8	1,4,6,9,8
Étape 4	1,4,6,9,8	1,4,6,8,9

(1) Écrire l'algorithme en **Python**.

(2) Étudier la complexité de cet algorithme.

(3) Programmer et tester cette fonction.

Exercice 5 :

En utilisant une des fonctions précédentes, proposer une méthode qui étant donné une liste de nombres, renvoie la médiane de l'ensemble des nombres d'une liste. On pourra aussi indiquer les quartiles. On programmera cette fonction en Python.