Les exercices se feront sur papier, on ne programmera que le dernier lorsqu'il sera écrit sur papier.

Exercice 1 :

(1) Analyser les complexités des algorithmes d'opérations sur les matrices. (on supposera que toutes les matrices sont carrées de tailles $n \times n$)

(2) Analyser les complexités des algorithmes d'opérations sur les polynômes. (On supposera que tous les polynômes sont de taille n)

Exercice 2 :

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 un indice i tel que $L[i] = a$.

b. Analyser la complexité de cet algorithme.

(2) a. Écrire un algorithme plus rapide (on pensera à la dichotomie).

b. Analyser la complexité de celui-ci (on considèrera une liste de taille une puissance de 2 : $len(L) = 2^k$)

Exercice 3 :

On se donne l'algorithme suivant :

```
def fonction(x, n):
    for i in range(n):
        print(x ** i)
```

(1) Expliquer ce que fait cet algorithme.

(2) Quel est la complexité de celui-ci ?

(3) Donner un algorithme qui fera le même affichage avec une complexité de l'ordre de n opérations.

(4) Modifier cet algorithme pour afficher seulement $x**n$.

(5) Existe-t-il un algorithme permettant de faire le même affichage plus rapidement ?

Exercice 4 :

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 les 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) Analyser la complexité de cet algorithme.

(3) Implémenter cet algorithme en Python.

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