Exercice 1.

I. Résultats concernant la recherche séquentielle

Le tableau doit être parcouru complètement lorsque l'élément recherché se trouve à la fin du tableau ou s'il n'existe pas dans le tableau.
Le meilleur des cas est lorsque l'élément recherché se trouve au début du tableau.

II. Résultats concernant la recherche dichotomique

Le tableau doit être parcouru totalement s'il se trouve à l'une des extrémités du tableau ou s'il n'existe pas.
Le meilleur des cas est lorsque l'élément recherché se trouve au centre du tableau.

III. Comparatif des deux méthodes de recherche

	Séquentielle				Dichotomique	
	Existe		Existe pas		Existe	Existe pas
	Trié	Non trié	Trié	Non trié		
Temps (en s)	0,08	0,12 *	0,10	0,17	0,01	0,01

* Très variable en fonction des exécutions

IV. Code de l'implémentation des recherches

Recherche séquentielle

```
# Recherche de manière séquentiel un élément dans un tableau
# Donne l'indice de l'élément recherché ou False s'il n'existe pas
def recherche_seq(tab, elemRech):

    i = 0
    for elem in tab:
        if(elem == elemRech):
            return i
        i = i + 1
    return False
```

Recherche dichotomique

```
# Recherche de manière dichotomique un élément dans un tableau trié
# Donne l'indice de l'élément recherché ou False s'il n'existe pas
def recherche_dich(tab, elemRech):
    m = len(tab) // 2
    if(elemRech != tab[m] and len(tab) == 1):
        return False
    if(elemRech == tab[m]):
        print("oui")
        return m
    if(tab[m] > elemRech):
        return recherche_dich(tab[0:m], elemRech)
    else:
        return recherche_dich(tab[m:len(tab)], elemRech)
```

Exercice 2.

Implémentation de la somme des éléments d'une matrice

Code de l'implémentation

Représentation sous la forme d'un tableau à deux dimensions

Dans cette algorithme, la complexité est quadratique.

```
# Additinne les élements d'une matrice
def somme_elements_matrice(M):

    acc = 0
    for i in range(len(M)):
        for j in range(len(M[i])):
            acc = acc + M[i][j]
    return acc
```

Représentation sous la forme d'un tableau à une dimension

Pour cette algorithme, la complexité est linéaire.

```
# Additionne les éléments d'une matrice qui est sous la représentation "tableau
à une dimension"
def somme_elements_matrice_1dim(M_1dim):

    acc = 0
    for ligne in M_1dim:
        acc = acc + ligne[2]
    return acc
```

Résultats comparatifs

100x100	Représentation en deux dim.				Représentation en une dim.			
% de 0	10	20	30	50	10	20	30	50
Temps (en s)	0,004	0,004	0,004	0,005	0,001	0,001	0,0008	0,0007

1000x1000	Représentation en deux dim.				Représentation en une dim.			
% de 0	10	20	30	50	10	20	30	50
Temps (en s)	0,3	0,3	0,3	0,3	0,1	0,1	0,1	0,1

Pour la matrice de 1000x1000, la différence des temps d'exécution en fonction du nombre de 0 n'est pas significative.
Mais pour une matrice de 100x100, on remarque que le temps d'exécution est divisé par 10 lorsque le pourcentage de 0 dépasse les 30% avec la représentation sous la forme d'un tableau à une dimension.

Implémentation du produit de deux matrices

Code de l'implémentation

Représentation sous la forme d'un tableau à deux dimensions

```
# Algorithme qui prend en argument deux matrices et retourne la matrice
# produit.

def m_mul_matrix(Ma, Mb):


    if(np.size(Ma, 1) != np.size(Mb, 0)):
        print("Les matrices ne sont pas compatibles pour le produit.")
        return []

    result = np.zeros(shape=(len(Ma), len(Mb[0])))

    for i in range(len(Ma)): # nombre de ligne de A
        for j in range(len(Mb[0])): # nombre de colonne de B
            for k in range (len(Ma[0])): # nombre de colonne de A
                result[i][j] += Ma[i][k]*Mb[k][j]

    return result
```

Représentation sous la forme d'un tableau à une dimension

Pour cette algorithme, la complexité est linéaire.

```
# Additionne les éléments d'une matrice qui est sous la représentation "tableau
à une dimension"
def somme_elements_matrice_1dim(M_1dim):

    acc = 0
    for ligne in M_1dim:
        acc = acc + ligne[2]
    return acc
```

Résultats comparatifs

Représentation sous la forme d'un tableau à deux dimensions

En ce qui concerne l'algorithme multipliant deux matrices qui sont sous la représentation d'un tableau à deux dimensions, peu importe le nombre de 0 dans la matrice le temps d'exécution est le même.
Donc, voici les temps d'exécution en fonction de la taille des matrices :

100x100 :	1 s
150x150 :	3,5 s
200x200 :	10 s
300x300 :	30 s

Représentation sous la forme d'un tableau à une dimension

Comparatif avec le même nombre de 0 que précédemment :

Taille	Temps d'exécution
100x100 :	0,004 s
150x150 :	0,008 s
200x200 :	0,013 s
300x300 :	0,031 s

Comparatif en fonction du nombre de 0 d'un matrice de 1000x1000 :

Taille	% de 0	Temps d'exécution
1000x1000	10%	0,362
1000x1000	20%	0,361
1000x1000	30%	0,367
1000x1000	50%	0,372

On constate qu'il n'y a aucune évolution du temps d'exécution en fonction du nombre de 0.