Exercices : Algorithmes de parcours sur les tableaux

Exercice 1

1. Écrire ci-contre une fonction `recherche` implémentant l'algorithme de recherche d'une occurrence d'une valeur `val` parmi les éléments d'un tableau `tab`.
   Cette fonction doit retourner :
   - le plus petit indice `index` tel que `tab[index] = val` si `val` est présent dans `tab`
   - `-1` sinon

```
def recherche(val, tab) :
    for i in range(len(tab)):
        if tab[i] == val:
            return i
    return -1
```

2. S'agit-il d'un parcours par indice ou par élément ? par indice

3. Si la valeur recherchée est présente aux indices 1234, 784 989 et 3 785 654, quel sera l'indice retourné par la fonction `recherche` ? 1234

4. Dans un tableau de taille 100 000 000, où doit être positionnée la valeur cherchée pour que la recherche soit rapide ? au début

5. Dans un tableau de taille 100 000 000, où doit être positionnée la valeur cherchée pour que la recherche soit lente ? à la fin

6. Si la valeur cherchée n'est pas présente parmi les éléments du tableau, la recherche est-elle rapide ou lente ? Justifier en quelques mots. Lente : on parcourt le tableau en entier

7. Sur un tableau T de taille 10 000 000 et sur une machine donnée, `recherche(7, T)` a pris 31 ms. Combien de temps prendra `recherche(8, T)` sur la même machine ? On ne peut pas savoir : cet algorithme a un temps d'exécution très variable selon l'emplacement de la valeur cherchée dans le tableau.

Exercice 2

1. En vous inspirant de l'algorithme de calcul d'une moyenne des éléments d'un tableau, écrire ci-contre une fonction `compter_occurences` permettant de compter le nombre d'occurrences de `val` dans le tableau `tab` (autrement dit, compter combien de fois `elt` est présent dans `tab`).

```
def compter_occurences(val, tab):
    compteur = 0
    for elt in tab:
        if elt == val:
            compteur = compteur + 1
    return compteur
```

   En particulier se poser la question : " Vais-je faire un parcours par élément ou par indice ?"

   Voici quelques assertions :

```
T = [1, 7, 4, 7, 5, 3, 8, 9]
assert(compter_occurences(7, T)==2)
assert(compter_occurences(4, T)==1)
assert(compter_occurences(77, T)==0)
```

2. Est-ce que `compter_occurences` fonctionne avec un tableau de chaînes de caractères ?
   oui

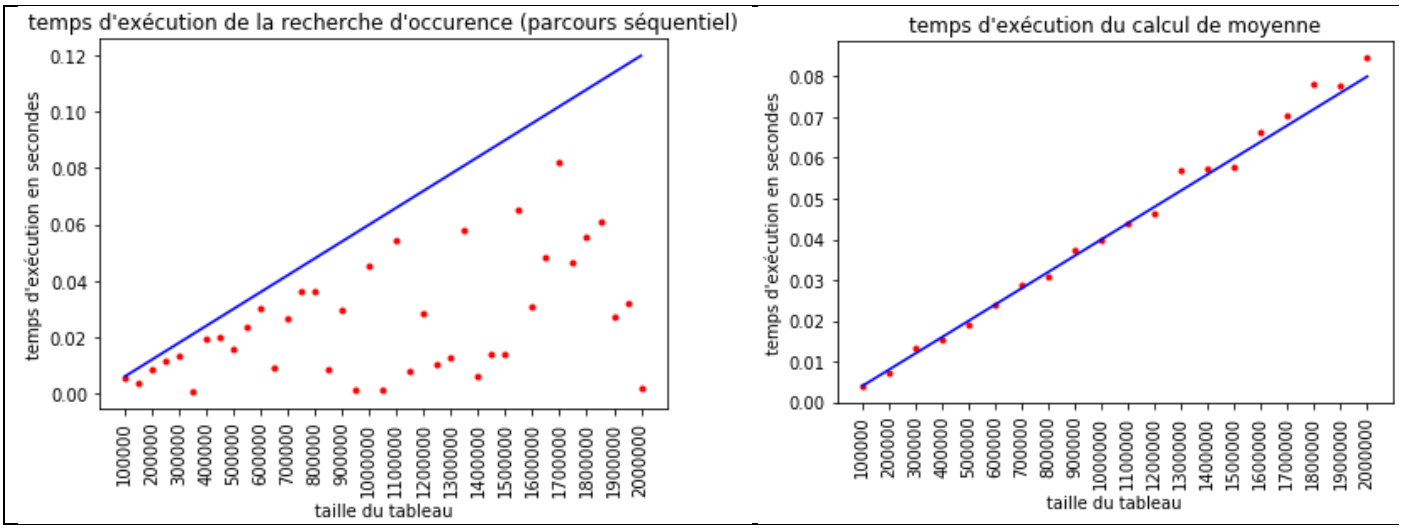
3. Est-ce qu'on parcourt systématiquement tout le tableau ? oui
   Quel est le coût de l'algorithme que vous avez implémenté ? linéaire ? quadratique ? linéaire
   Est-ce valable tout le temps ou seulement dans le pire des cas ? **tout le temps**

4. *Pour cette question on admet que le temps d'exécution est comme le coût (*).*
   Sur un tableau d'entiers T1 de taille 1 000 000 et sur une machine donnée, `compter_occurences(10, T1)` a pris 150 ms.
   Sur un tableau d'entiers T2 de taille 15 000 000 et sur la même machine, combien de temps prendra approximativement `compter_occurences(1000, T2)`?
   Seule la taille du tableau influence le temps d'exécution, pas la valeur recherchée.
   Le temps d'exécution est donc multiplié environ par 15 soit : 2250 ms.

(*) : *en réalité on ne passe pas forcément aussi simplement de la complexité algorithmique au temps d'exécution sur machine.*

Exercice 3

En reprenant les deux exercices ci-dessus, comparer brièvement les deux graphiques ci-desssous :
- expliquer le point commun
- expliquer la différence



- Point commun : temps d'exécution est majoré par une fonction linéaire. On a un coût linéaire dans le pire des cas.
- Différence : pour la recherche d'occurrence, le temps d'exécution est parfois beaucoup plus court (lorsque la valeur cherchée est en début de tableau). Pour la recherche d'occurrence on fait donc souvent mieux que le pire des cas.
  Au contraire, pour le calcul de moyenne, on parcourt tout le temps l'intégralité du tableau. On est tout le temps dans le pire des cas (qui est donc le cas normal !)