

Algorithmes de parcours sur les tableaux

I : Recherche d'une occurrence

On souhaite écrire une fonction recherche vérifiant la spécification suivante :

- paramètres : un tableau `tableau` et un élément `elem` de type quelconque
- valeur de retour : un entier `index`
- postcondition : si `elem` est présent dans `tableau` alors `index` correspond au plus petit indice de `elem` dans `tableau`. Si `elem` n'est pas présent alors `index` est égal à -1.

Exemples d'assertions vérifiées par recherche :

```
assert recherche( [ 3, 5, 9, 1, 7, 8 ] , 9 ) == 2
assert recherche( [ "a", "d", "gh", "b", "ko", "d" ] , "d" ) == 1
assert recherche( [ 3, 5, 9, 1, 7, 8 ] , 6 ) == -1
```

Ici on a besoin de l'indice on utilise donc un parcours par indice

```
def recherche(tableau, elem):
    for ... in ..... :
        if ..... == ..... :
            return .....
    return ...
```

II : Recherche d'un extremum

On souhaite écrire une fonction trouver_max vérifiant la spécification suivante :

- paramètres : un tableau `tableau` de nombres
- valeur de retour : un nombre `maximum`
- précondition : le tableau `tableau` est non vide
- postcondition : `maximum` est égal au plus grand des nombres présents dans le tableau

Exemples d'assertions vérifiées par trouver_max :

```
assert trouver_max( [ 3, 5, 9, 1, 17, 8 ] ) == 17
assert trouver_max( [-2, -5, -7, 0, -4] ) == 0
```

Ici on n'a pas besoin de l'indice on peut donc utiliser un parcours par élément

```
def trouver_max(tableau):
    maximum = .....
    for ... in ..... :
        if ..... > ..... :
            ..... = .....
    return .....
```

III : Calcul de moyenne

On souhaite écrire une fonction calculer_moyenne vérifiant la spécification suivante :

- paramètres : un tableau `tableau` de nombres
- valeur de retour : un nombre `moyenne`
- précondition : ...
- postcondition : `moyenne` est égal à la moyenne des nombres présents dans le tableau.

Exemples d'assertions vérifiées par calculer_moyenne :

```
assert round(calculer_moyenne([ 3, 5, 9, 1, 17, 7 ]), 2) == 7.0
assert round(calculer_moyenne([-2, -5, 7, 0, -4, 3, 1]), 2) == 0.0
```

Ici on n'a pas besoin de l'indice on peut donc utiliser un parcours paélément

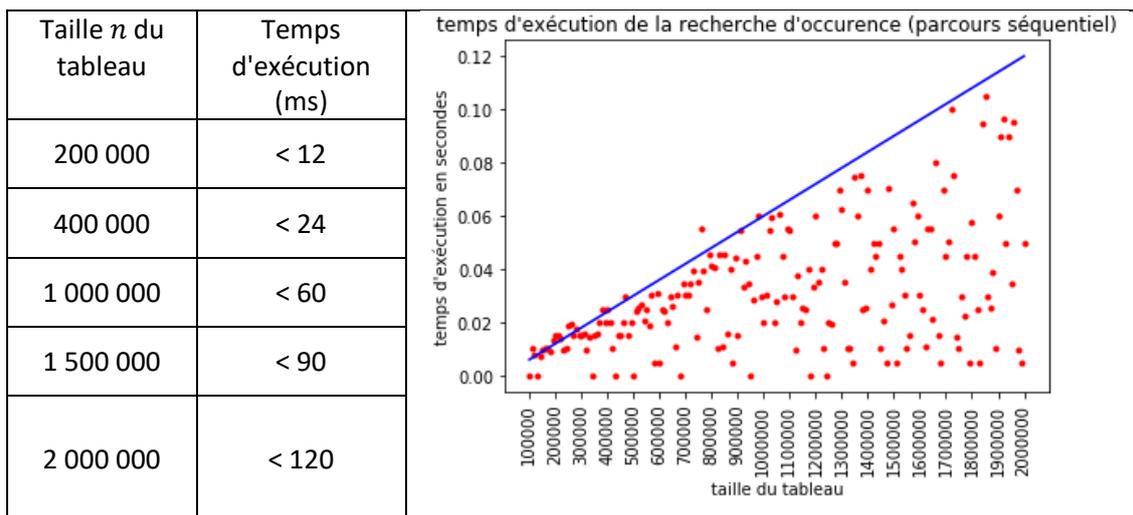
```
def calculer_moyenne(tableau):

    somme = ...
    for ..... in ..... :
        somme = ..... + .....
    moyenne = ..... / .....
    return .....
```

IV : Complexité des algorithmes de parcours séquentiel : linéaire en $O(n)$

Observation expérimentale sur des tableaux de nombres aléatoires

Prenons comme exemple la recherche d'une occurrence par parcours séquentiel. En faisant tourner cet algorithme sur des tableaux de taille n on obtient – avec une implémentation en python sur un PC familial – les temps d'exécution suivants (en millisecondes) :



Démonstration de la complexité linéaire (pour la recherche)

```
def recherche(tab, elem):

    for i in range(len(tab)) :           # \
        if tab[i] == elem :             # }
            return i                     # /

    return -1                             #
```