

DEVOIR SURVEILLÉ 2 – 1 HEURE

CI 2 : ALGORITHMIQUE ET PROGRAMMATION

ÉLÉMENTS DE CORRIGÉS


Exercice

Question 1

Écrire la fonction permettant de permuter les valeurs du tableau. Cette fonction devra correspondre aux spécifications suivantes :

- nom de la fonction : *permute* ;
- arguments de la fonction : un tableau, les deux indices à permuter.

Correction



```
def permute(tab, i, j):
    tab[i], tab[j] = tab[j], tab[i]
    return tab
```

Question 2

Soit le tableau *tab* = [2, 3, 1, 4]. Pour chaque valeur de *i* et pour chaque valeur de *j*, indiquer le contenu du tableau *tab*. Pour répondre à la question on utilisera le tableau donné en fin de sujet. On le remplira à partir de la double barre. Les colonnes *i*, *j*, indice et *tab* seront à remplir intégralement. Les * sont à remplacer par les valeurs correspondantes.


Question 3

Écrire la fonction qui vérifie si un tableau est trié. Cette fonction devra correspondre aux spécifications suivantes :

- nom de la fonction : *is_sorted* ;
- argument de la fonction : un tableau ;
- retour de la fonction : la fonction doit retourner la valeur booléenne *True* si le tableau est trié. Elle devra retourner la valeur booléenne *False* si le tableau n'est pas trié.

On utilisera une boucle **Pour**.

Correction



```
def is_sorted(tab):
    for i in range(0, len(tab)-1):
        if tab[i] > tab[i+1]:
            return False
    return True
```

Question 4

Réécrire la fonction précédente en utilisant exclusivement une boucle **Tant que**.

Correction



```
def is_sorted_while(tab):
    i=1
    while tab[i-1]<=tab[i] and i<len(tab)-1:
        i=i+1
    return tab[i-1]<=tab[i]
```

Question 5

Des deux structures proposées, estimez laquelle peut être la plus efficace ? (c'est-à-dire, laquelle nécessite, le cas échéant, le moins d'opérations).

Correction

Pour les deux boucles proposées, la même portion de tableau sera parcourue afin de savoir si le tableau est trié.

Correction



```
def recherche_dichotomique(x, a):
    g, d = 0, len(a)-1
    while g <= d:
        m = (g + d) // 2
        if a[m] == x:
            return m
        if a[m] < x:
            g = m+1
        else:
            d = m-1
    return None
```

Question 6

De quel type sont les variables x et a ?

Correction

La variable x représente le tableau à trier. La variable a représente la valeur à tester.

Question 7

Quelle opération est effectuée ligne 4 ? Expliquer ce choix.

Correction

En ligne 4 on effectue une division euclidienne. Ainsi on ne récupère que la partie entière d'un nombre. Cela est nécessaire car ce nombre représente par la suite un index du tableau. Ce doit donc être nécessairement un nombre entier.

Question 8

Pour chaque itération de la boucle while, quelle est l'étendue de la zone de recherche ?

Correction

A chaque incrémentation de la boucle while la zone de recherche est divisée par 2.

Question 9

Compléter l'algorithme (en recopiant les lignes qui vous semblent nécessaires) pour qu'il renvoie l'index de l'élément recherché.

Question 10

L'algorithme a-t-il le comportement souhaité ? Si ce n'est pas le cas, compléter l'algorithme.

Question 11

A partir des fonctions définies précédemment, écrire une fonction permettant, à partir d'un tableau d'entier quelconque (trié ou non trié), de dire si un élément appartient au tableau ou non :

- données d'entrées de la fonction : un tableau, un nombre ;
- données de sortie de la fonction : un booléen.

Correction



```
def recherche_dichotomique(tab, nb):
    if is_sorted(tab):
        return(recherche_dichotomique(tab, nb))
    else :
        tri (tab)
        return(recherche_dichotomique(tab, nb))
```

Question 12

Quel est l'intérêt de trier un tableau dans le cas où on cherche une valeur dans le tableau ? Quel est l'intérêt de le trier lorsqu'on cherche plusieurs valeurs ? Commenter.

Correction

Pour être trié, le tableau est parcouru $(n-1)+(n-2)+\dots+2+1$ fois. Une fois trié, dans le pire des cas, une recherche dichotomique demande ** redivision du tableau.

Ainsi pour rechercher une seule fois une valeur dans un tableau, il est moins coûteux de ne pas trier le tableau.

NOM :

Commentaires	i	j	indice	tab
Instant initial	–	–	–	[2,3,1,4]
Pour i allant de 0 à 3 :	–	–	–	[2,3,1,4]
Pour $i = 0$	0	–	–	[2,3,1,4]
Affectation	0	–	0	[2,3,1,4]
Pour j allant de 1 à 3	0	–	0	[2,3,1,4]
Pour $j = 1$	0	1	0	[2,3,1,4]
tab[1]<tab[0] $\Rightarrow 3 < 2$ est faux	0	1	0	[2,3,1,4]
Pas d'affectation de l'indice	0	1	0	[2,3,1,4]
Pour $j = 2$	0	2	0	[2,3,1,4]
tab[2]<tab[0] $\Rightarrow 1 < 2$ est vrai	0	2	0	[2,3,1,4]
Affectation de indice	0	2	2	[2,3,1,4]
Pour $j = 3$	0	3	2	[2,3,1,4]
tab[3]<tab[0] $\Rightarrow 4 < 2$ est faux	0	3	2	[2,3,1,4]
Affectation ? Non	0	3	2	[2,3,1,4]
Permutation (Oui)	0	3	2	[1,3,2,4]
Pour $i = 1$	1	3	2	[1,3,2,4]
Affectation	1	3	1	[1,3,2,4]
Pour j allant de 2 à 3				
Pour $j = 2$	1	2	1	[1,3,2,4]
tab[2]<tab[1] $\Rightarrow 2 < 3$ est vrai	1	2	1	[1,3,2,4]
Affectation ? Oui	1	2	2	[1,3,2,4]
Pour $j = 3$	1	3	2	[1,3,2,4]
tab[3]<tab[2] $\Rightarrow 4 < 2$ est Fau	1	3	2	[1,3,2,4]
Affectation ? Non				[1,3,2,4]
Pour $j = *$	*	*	*	*
tab[*]<tab[*]	*	*	*	*
Affectation ?	*	*	*	*
Permutation ?				[1,2,3,4]