TP 15 : Complexité d'un algorithme
Parcours d'un tableau

Exercice 1.

1. On considère la fonction définie ci-dessous :

```
def conversion(n):
    h = n // 3600
    m = (n - 3600*h) // 60
    s = (n - 3600*h) % 60
    return h,m,s
```

a. Quel(s) est/sont le(s) paramètre(s), et leur type, de la fonction définie ci-dessus ?

b. Que retourne cette fonction pour n = 7345 ?

c. Que permet de faire cette fonction ?

2. Déterminer la complexité de cette fonction.

Exercice 2.

1. On considère la fonction définie ci-dessous :

```
def puissanceMoinsUn(n):
   if n%2==0:
      res = 1
   else:
      res = -1
   return res
```

a. Quel(s) est/sont le(s) paramètre(s), et leur type, de la fonction définie ci-dessus ?

b. Que retourne cette fonction pour n = 13, puis n = 18 ?

c. Que permet de faire cette fonction ?

2. Déterminer la complexité de cette fonction.

Exercice 3.

1. On considère la fonction définie ci-dessous :

```
def factorielle(n):
    fact = 1
    i = 2
    while i <= n:
        fact = fact * i
        i = i + 1
    return fact
```

a. Quel(s) est/sont le(s) paramètre(s), et leur type, de la fonction définie ci-dessus ?

b. Que retourne cette fonction pour n = 6 ?

c. Que permet de faire cette fonction ?

2. Déterminer la complexité de cette fonction.

Exercice 4. *Recherche d'un maximum d'un tableau*

Algorithme :

Pour connaître la valeur maximale des valeurs contenues dans la liste L , il faut évaluer tour à tour tous les éléments e de L .

La valeur maximale à une étape donnée est mémorisée dans une variable m , initialisée avec le premier élément de la liste L .

Si l'élément e est supérieur à m , m prend alors la valeur de e.

1. Implémenter en Python une fonction `maximum(L)`
2. Calculer l'ordre de complexité de cet algorithme.

Exercice 5.

On veut créer une fonction `separer()` permettant, à partir d'une liste de nombres, d'obtenir deux listes. La première comporte les nombres inférieurs ou égaux à un nombre donné, la seconde les nombres qui lui sont strictement supérieurs. Par exemple :

```
separer([45, 21, 56 ,12, 1, 8, 30, 22, 6, 33], 30)
```

doit renvoyer `[21, 12, 1, 8, 30, 22, 6], [45, 56, 33]`

1. Sur une feuille, décrire en langage naturel l'algorithme de cette fonction.
2. Implémenter en Python la fonction `separer(L,e)`
3. Calculer l'ordre de complexité de cet algorithme.

Exercice 6.

On veut créer une fonction `plus_proche_i(L,n)` permettant de rechercher la plus proche valeur d'un nombre dans une liste et son indice dans la liste. Par exemple :

`plus_proche_i([45, 21, 56 ,12, 1, 8, 30, 22, 6, 33], 20)` doit renvoyer `1, 21`

1. Sur une feuille, décrire en langage naturel l'algorithme de cette fonction.
2. Implémenter en Python la fonction `plus_proche_i(L,n)`
3. Calculer l'ordre de complexité de cet algorithme.

Exercice 7.

On veut créer une fonction `lex_avant(mot1, mot2)` permettant de déterminer si le `mot1` est classé avant le `mot2` dans l'ordre lexicographique (celui du dictionnaire). Par exemple : `lex_avant('Information', 'informatique')` doit renvoyer `True`.

Afin de ne pas tenir compte de la casse (lettres majuscules ou minuscules, on pourra utiliser la fonction `.lower()`

1. Sur une feuille, décrire en langage naturel l'algorithme de cette fonction.
2. Implémenter en Python la fonction `lex_avant(mot1, mot2)`
3. Calculer l'ordre de complexité de cet algorithme.