Le petit point de cours, informatique (2), une correction

1. (a) Expliquez ce que renvoie la fonction suivante où `L` est une liste de nombres entiers compris (au sens large) entre 0 et 1000.

```
def Occurence(L):
        K=[0 for k in range(1001)]
        for i in range(len(L)):
                K[L[i]]=K[L[i]]+1
        return K
```

   Cette fonction va créer une liste `Occurence(L)` de comportant 1001 nombres telle qu'à la place `v`, pour `v` entre 0 et 1000, la liste `Occurence(L)` contient le nombre de fois où la valeur `v` se trouve dans la liste `L`.

   (b) Quelle est la complexité de la fonction `Occurence` en fonction de n qui est la longueur de la liste `L` ?

   Pour chaque bloc de la boucle, il y a deux opérations élémentaires (une addition et une affectation). La complexité est donc linéaire, c'est à dire, en $O(n)$.

   (c) Proposer une fonction `TriOcc` qui permet de trier par ordre croissant une liste `L` de nombres entiers compris (au sens large) entre 0 et 1000 qui utilise la fonction `Occurence`.

```
def TriOcc(L):
        N=[]
        M=Occurence(L)
        for i in range(len(M)):
                if M[i]!=0 :
                        for j in range(M[i]) :
                                N.append(i)
        return N
```

2. Proposer une fonction Python `Maxliste`, qui mange une liste de nombres `L` et qui renvoie une liste de deux éléments dont le premier est le maximum de la liste et le second un indice où ce maximum est atteint.

```
def Maxliste(L):
        max=L[0]
        for i in range(1,len(L)):
                if L[i]>max:
                        max=L[i]
                        place=i
        return [max,place]
```

3. (a) Que fait la fonction suivante où `L` est une liste, en expliquant !

```
def function(L):
        for i in range(1,len(L)):
                insertion=L[i]
                j=i
                while j>0 and L[j-1]>insertion :
                        L[j]=L[j-1]
                        j=j-1
                L[j]=insertion
        return L
```

La fonction précédente tri la liste `L` par insertion.

- `insertion` est la valeur que l'on insère, les i précédentes valeurs, à savoir `L[0]`,...,`L[i-1]` ayant déjà été triées dans l'ordre croissant.
- La séquence

```
while j>0 and L[j-1]>insertion :
        L[j]=L[j-1]
        j=j-1
```

décale d'une place vers la droite les valeur triées qui sont supérieures à la valeur `insertion`. Le compteur `j=j-1` permet de trouver la place d'insertion.
- `L[j]=insertion` affecte à la place d'insertion déterminée par le compteur précédent la valeur que l'on insère.

(b) Évaluez la complexité de `function` en fonction de n qui est la longueur de la liste `L`.

On va évaluer la complexité **dans le pire des cas** (la liste `L` en entrée est déjà triée par ordre décroissant.)

- Le bloc

```
                while j>0 and L[j-1]>insertion :
                L[j]=L[j-1]
                j=j-1
```

comporte 2 opérations élémentaires (comparaison et affectation).

- A i fixé, le bloc

```
insertion=L[i]
j=i
while j>0 and L[j-1]>insertion :
        L[j]=L[j-1]
        j=j-1
L[j]=insertion
```

comporte alors $2 + 4i$ opérations (tests et affectations)

- La complexité maximale est donc : $C_{max} = \sum_{i=1}^{n}(2 + 4i) = O(n^2)$.

Dans le meilleur des cas (liste `L` triée) la complexité est en $O(n)$: 4 opérations seulement dans le bloc précédent.