Corrigé — L'Ordre, l'Ordre, l'Ordre.

I Algorithmes de cours

Question 1. *Second Minimum*

coeff
1

Question 2. *Recherche linéaire - version for*

coeff
1

II L'ordre ...

Question 3. *Minimum*

coeff	initialisation	parcours	calcul indice	retour
1	2	3	3	1

```
def minimum(l) :
    if len(l) == 0 :
        return None
    im = 0
    for i in range(len(l)) :
        if l[i] < l[im] :
            im = i
    return im
```

Question 4. *Filtrage*

Implémenter une fonction `plus_petits(l,x)` qui prend une liste d'entiers `l` et un entier `x` et renvoie la liste des entiers de `l` plus petits ou égaux à `x`.

coeff	initialisation	parcours	ajout	retour
2	1	3	4	1

```
def plus_petits(l,x) :
    lpp = []
    for y in l :
        if y < x :
            lpp.append(y)
    return lpp
```

III … l'ordre …

Question 5. *Nadine*

coeff	1)	2)	3)	4)	5)
2	1	2	2	2	2

1. Ce sont les valeurs de `l` : `4, 13, 0, 26, 25, 6, 10, 16`.
2. La boucle 2 fait `n` itération, ici `8`, quelque soit l'itération de la boucle 1. La boucle 1, elle, fait aussi `n` itération, ici `8`. Donc la ligne 7 qui est exécutée à chaque itération de la boucle 2, sera exécutée $8 \times 8 = 64$ fois.
3. Le raisonnement est le même que précédemment, sauf que, cette fois ci, la ligne 8 n'est pas exécutée à chaque tour de la boucle 2. On peut faire le calcul à chaque itération, ce qui donne : `1,4,0,7,6,2,3,5` pour chaque valeur de `x`. On peut aussi avancer sur la suite et remarqué que le nombre d'exécution de la ligne 8 est exactement la valeur de `c` après la boucle 2, donc le nombre d'éléments plus petits que `x` ainsi que sa position dans `l_res`. On peut donc remarquer que, comme toutes les valeurs de `c` entre `0` et `7` seront produite, il y aura `0+1+2+3+4+5+6+7 = 7*8 /2 = 28` exécutions de la ligne `8`.
4. `l_res = [0,4,None,None,13,None,25,26]`
5. Comme on l'a dit au dessus, la boucle 2 compte le nombre d'éléments, `c`, plus petits que `x` et place `x` à l'indice `c` de `l_res`. Et cela pour chaque élément `x` de la liste `l`. A la fin, chaque élément de `l` a été replacé dans `l_res` de telle manière à ce que `l_res` soit triée.

Question 6. *Est triée*

coeff	parcours	test	booléen
2	3	3	3

On cherche un indice `0 < i < len(l)` qui soit plus petit que le précédent.

```
def est_triee(l) :
    for i in range(1,len(l)) :
        if l[i] < l[i-1] :
            return False
    return True
```

IV … et l'ordre.

Question 7. *Ordre lexicographique*

coeff	cas prefixe	cas égalité	resultat
2	3	3	3

On cherche un indice commun différent dans `s1` et `s2` avec une recherche linéaire version **`while`**. A la fin, si le score est nul, on vérifie si l'un des deux est un préfixe stricte.

```
def lexico(s1,s2) :
    i = 0
    dif = 0
    while i < len(s1) and i < len(s2) and dif == 0:
        if ord(s1[i]) < ord(s2[i]) :
            dif = -1
        elif ord(s2[i]) < ord(s1[i]) :
            dif = 1
        i = i + 1
    if dif == 0 and len(s1) < len(s2) :
        dif = -1
    elif dif == 0 and len(s2) < len(s1) :
        dif = 1
    return dif
```

Question 8. *Tri Sélection - Simulation*

coeff
1

`(5,0), (5,1), (3,2), (7,3), (4,4), (5,5), (7,6), (7,7).`

Question 9. *Tri Sélection - Code*

coeff	T1	T2	T3	T4
2	2	2	2	3

```
def tri_selection(l) :
    n = len(l)
    dernier_trie = ""
    for nb_tours in range(n) :
        min_tour = l[nb_tours]
        imin_tour = nb_tours
        for i in range(n) :
            s = l[i]
            if lexico(dernier_trie,s) == -1 and lexico(s,min_tour) == - 1:
                min_tour = s
                imin_tour = i
        l[imin_tour] = l[nb_tours]
        l[nb_tours] = min_tour
        dernier_trie = min_tour
    return None
```

Question 10. *Tri Insertion*

coeff	parcours	insertion	résultat
3	2	5	2

A chaque étape on regarde l'indice `i`. On cherche en arrière, depuis `i`, l'indice auquel placer l'élément `l[i]`. On fait descendre par échange cet élément au fur et à mesure de la recherche.

Dans un parcours de la liste, on adapte une recherche linéaire version **`while`**.

```
def tri_insertion(l) :
    n = len(l)
    for i in range(n) :
        s = l[i]
        j = i
        while j > 0 and lexico(s,l[j-1]) == -1 :
            l[j] = l[j-1]
            l[j-1] = s
            j -= 1
    return None
```