NOM :..... Prénom :.....

CPGE PSI* 2026/2027 — **Informatique**
Lycée La Fayette — **Nathalie Planche**

Devoir surveillé n°1 - Test de rentrée - Jeudi 3 septembre 2026

Version n°2

Exercice 1 : Ecrire une fonction `minimum(L)` qui renvoie **un couple** constitué de la valeur de l'élément minimum de la liste `L`, et de la position de ce minimum dans la liste (si ce minimum est atteint plusieurs fois, l'indice renvoyé sera le plus petit).

```
def minimum(L):
    mini = L[0]          #initialisation du minimum
    indice = 0           #initialisation de l'indice où il est
    for i in range(len(L)):
        if L[i]<mini:    #cas où il faut changer
            mini = L[i]
            indice = i
    return mini,indice   #on retourne un tuple
```

Exercice 2 : 1. Ecrire une fonction `comptage(L)` qui à partir d'une liste `L`, renvoie **un dictionnaire** dont les clefs sont les éléments de la liste, et les valeurs sont le nombre d'occurrences de l'élément dans la liste.

Exemple : si `L=[5,0,3,5]`, alors la fonction pourra renvoyer le dictionnaire `{5:2,0:1,3:1}`.

```
def comptage(L):
    d={}
    for elt in L:
        if elt in d:
            d[elt]+=1
        else:
            d[elt]=1
    return d
```

ou

```
def comptage(L)
    d={elt:0 for elt in L}
    for elt in L:
        d[elt]+=1
    return d
```

2. Donner, en justifiant, la complexité en temps dans le pire des cas, en fonction de la taille n de la liste `L`.

si $n=$len(L), la boucle for tourne n fois, les opérations à l'intérieur sont en temps constant car "if elt in d" est en temps constant. Donc $O(n)$

le remplissage de d est en $O(n)$ (première boucle for)
deuxième boucle for en $O(n)$
Complexité totale :
$2O(n)=O(n)$

Exercice 3 : 1. Compléter le script suivant pour que la fonction trie la liste `L` selon les valeurs croissantes, en suivant le principe du **tri par insertion**.

Remarque : cette fonction ne retourne rien, quand on l'exécute elle modifie la liste passée en paramètre.

```
1  def tri_insertion (L) :
2      n=len(L)
3      for k in range(1, n) :
4          x = L[k] # élément à placer
5          j = k
6          # on cherche la place de x parmi le début de la liste , triée
7          while j>0 and x<L[j-1]: # tant que pas au début de la liste et
8                                  # que la future place de x n'est pas trouvée
9              L[j] = L[j-1] # on décale le terme d'avant
10             j = j-1 # on décale vers la gauche le compteur qui cherche la future place de x
11         L[j] = x # on place x dans la position libérée
```

2. Donner, en justifiant, la complexité en temps dans le pire des cas, en fonction de la taille n de la liste `L`.

• la boucle for tourne n-1 fois
• à l'itération k, dans le pire des cas la boucle while tourne k fois (c'est le cas où il faut tout décaler)
Le reste des opérations est en temps constant
Donc l'ordre de grandeur de la complexité est
$\sum_{k=1}^{n-1} k = \frac{(n-1)n}{2} = O(n^2)$ complexité quadratique

Exercice 4 : A faire uniquement si les autres exercices sont finis

Programmer l'algotihme de recherche dichotomique dans un tableau trié. La fonction devra renvoyer la position de l'élément s'il est présent, et `False` sinon.

Savez-vous justifier la complexité de cet algorithme ?

```
def dicho(L,elt):
    d,f=0,len(L)-1
    while d<=f:
        milieu=(d+f)//2
        if L[milieu]==elt:
            return milieu
        if L[milieu]<elt:
            d=milieu+1
        if L[milieu]>elt:
            f=milieu-1
    return False
```

Complexité

Si la liste est de longueur $n=2^p$, alors la boucle while s'exécute p fois dans le pire des cas (car la longueur (f-d) est divisée par 2 à chaque tour)

Donc complexité en $O(p)$.

Or $n=2^p \Leftrightarrow p=\log_2(n)$

Donc $C(n)=O(\log_2(n))$

complexité logarithmique.