Épreuve d'informatique – Propagation d'une épidémie

Partie I. Tri

1.

```
def plusGrand(L):
  a = L[0]
  for v in L:
    if v>a:
      a = v
  return a
```

Cette fonction contient une boucle `for` de n itérations, sa complexité est donc $O(n)$.

2.

```
def moyenne(L):
  S = 0
  for v in L:
    S+=v
  return S/len(L)
```

Pour la même raison que dans la question précédente, la complexité de cette fonction est $O(n)$.

3.

itération	L
1	[2,5,3,1,4]
2	[2,3,5,1,4]
3	[1,2,3,5,4]
4	[1,2,3,4,5]

4. Soit $P(i)$ la proposition « la liste `L[0:i+1]` (avec la convention Python) est triée par ordre croissant à l'issue de l'itération i ».
   - La proposition $P(0)$ est vraie car la liste `L[0:1]` représente une liste de 1 élément qui est donc forcément triée.
   - Supposons que $P(i-1)$ soit vraie, c'est-à-dire qu'après l'itération $i-1$, la liste `L[0,i]` est triée. L'itération i place l'élément `L[i]` à sa place dans la liste `L[0:i]` que l'on a supposée triée. À l'issue de cette itération, la liste `L[0:i+1]` est donc triée et $P(i)$ est vraie.
   - Lors de la dernière itération de la boucle, $i = n-1$ et donc à l'issue de cette itération, $P(n-1)$ est vraie, c'est-à-dire que `L[0:n]` (soit la liste `L` dans son ensemble) est triée.
5. Cet algorithme est le *tri par insertion.*
6. - Dans le meilleur des cas, la liste `L` est déjà triée et la boucle `while` ne sera jamais exécutée. Dans ce cas la complexité est $O(n)$.
   - Dans le pire des cas, la liste `L` est triée à dans l'ordre décroissant. Dans ce cas, l'itération i la boucle `while` comportera i itérations, et la complexité sera $C(n) = 1+2+3+4+\ldots+n-1 = \frac{n(n-1)}{2} = O(n^2)$.
7. Il suffit de comparer les deuxièmes éléments de chaque élément de `L` :

```
def tri_chaine(L):
    n=len(L)
    for i in range(1,n):
        j=i
        x=L[i]
        while 0<j and x[1]<L[j-1][1]:
            L[j]=L[j-1]
            j=j-1
        L[j]=x
```

8. Dans la table `palu`, aucun des attributs `nom`, `iso` ou `annee` ne peut servir de clé primaire, car aucun de ces attributs ne peut identifier un enregistrement de manière unique. Par contre le couple `(nom, annee)` pourrait servir de clé primaire.
9.

```
SELECT * FROM palu WHERE annee=2010 AND deces>1000;
```

10.

```
SELECT nom, cas/pop AS incidence FROM palu JOIN demographie
ON iso=pays AND annee=periode
WHERE annee=2011;
```

11.

```
SELECT nom FROM palu WHERE annee=2010 ORDER BY cas DESC LIMIT 1;
```

12. Il suffit d'utiliser la fonction `trie_chaine` définie à la question 7 : `>>>trie_chaine(deces2010)`.

Partie II. Modèle à compartiments

13. Le vecteur X et la fonction f sont :

$$X(t) = \begin{pmatrix} S(t) \\ I(t) \\ R(t) \\ D(t) \end{pmatrix} \quad \text{et} \quad f : \begin{array}{ccc} \mathbb{R}^4 & \to & \mathbb{R}^4 \\ \begin{pmatrix} S(t) \\ I(t) \\ R(t) \\ D(t) \end{pmatrix} & \mapsto & \begin{pmatrix} -rS(t)I(t) \\ rS(t)I(t) - (a+b)I(t) \\ aI(t) \\ bI(t) \end{pmatrix} \end{array}$$

14. La ligne 4 doit renvoyer le vecteur défini à la question précédente, elle peut donc s'écrire :

```
return np.array([-r*X[0]*X[1],
                 r*X[0]*X[1]-(a+b)*X[1],
                 a*X[1],
                 b*X[1]
                ])
```

15. Les deux simulations sont différentes car celle faite avec $N = 7$ a un pas beaucoup plus grand que celle faite avec $N = 250$. Elle sera donc beaucoup moins précise. C'est évidemment la simulation faite avec $N = 250$ qui a nécessité le temps de calcul le plus long.
16. Pour compléter la ligne 7, on procède comme à la question 14, il faut juste utiliser `Itau` à la place de `X[1]` lorsque l'on a $I(t-\tau)$ dans l'équation. On obtient :

```
return np.array([-r*X[0]*Itau,
                 r*X[0]*Itau-(a+b)*X[1],
                 a*X[1],
                 b*X[1]
                ])
```

Pour la ligne 28 il faut penser à passer en paramètre la valeur de `Itau` à la fonction `f`. On peut obtenir cette valeur à partir des précédentes valeurs stockées dans `XX` si $i > p$ et dans `X0` sinon. On peut l'écrire de la manière suivante :

```
if i>=p:
  Itau = XX[-p][1]
else:
  Itau = X0[1]
X = X + dt*f(X,Itau)
```

Partie III. Modélisation dans des grilles

17. Cette fonction renvoie une grille de taille $n \times n$ contenant uniquement des 0, c'est-à-dire uniquement des individus sains.
18.

```
def init(n):
  G = grille(n)
  ligne = randrange(n)
  colonne = randrange(n)
  G[ligne][colonne] = 1
  return G
```

19.

```
def compte(G):
  tot = [0]*4
  for ligne in G:
    for v in ligne:
      tot[v]+=1
  return tot
```

20. La fonction `est_exposee` renvoie un booléen (`True` ou `False`).
21. Ligne 12 :

```
return (G[0][j-1]-1)*(G[1][j-1]-1)*(G[1][j]-1)*
            (G[1][j+1]-1)*(G[0][j+1]-1) == 0
```

Ligne 23 :

```
return (G[i+1][j-1]-1)*(G[i+1][j]-1)*(G[i+1][j+1]-1)*
       (G[i-1][j-1]-1)*(G[i-1][j]-1)*(G[i-1][j+1]-1)*
       (G[i][j+1]-1)*(G[i][j-1]-1) == 0
```

22.

```
def suivant(G, p1, p2):
  n = len(G)
  G2 = grille(n)
  for i in range(n):
    for j in range(n):
      if G[i][j] == 1:          # La case est infectee
        if bernouilli(p1)==1:
          G2[i][j] = 3          # La case devient décédée
        else:
          G2[i][j] = 2          # La case devient rétablie
      # La case est saine et exposée
      elif G[i][j] == 0 and est_exposee(G,i,j) and bernouilli(p2)==1:
        G2[i][j] = 1            # La case devient infectée
      # Dans tous les autres cas, pas de changement
      else:
        G2[i][j] = G[i][j]
 return G2          # Renvoie la nouvelle grille
```

23. À la fin d'une simulation il ne peut plus y avoir de case infectée. On a donc $x_1 = 0$.
24.

```
def simulation(n,p1,p2):
   G = init(n)
   stat = [n**2-1, 1, 0, 0]
   while stats[1]!=0:       # tant qu'il reste des cases infectées
     G = suivant(G, p1, p2)
     stats = compte(G)
   # Calcule les proportions
   prop = []
   for i in range(4):
      prop.append(stats[i]/n**2)
```

```
    return prop
```

25. On forcément $x_0 + x_1 + x_2 + x_3 = 1$. Le nombre de personnes atteintes par la maladie est la somme du nombre de personnes décédées et de personnes rétablies. Donc $x_{\text{atteintes}} = x_2 + x_3$.
26.

```
def seuil(Lp2, Lxa):
  imin = 0
  imax = len(Lp2)-1
  while imax-imin > 1 :
    imed = (imin + imax)//2
    if Lxa[imed]>0.5:
      imax = imed
    else
      imin = imed
  return [Lp2[imin], Lp2[imax]]
```

27. Le test de la ligne 8 ne peut pas être supprimé car il permet de s'assurer que l'on ne vaccine pas deux fois le même individu.
28. L'appel de la fonction `init_vac(5,0.2)` renvoie une grille de 5×5 individus parmi lesquels 20 % (5 individus) sont immunisés, un individu est infecté et les autres sont sains.