

Partie I. Tri et bases de données

Q1 –  $\frac{i}{L} \parallel \begin{array}{c} 1 \\ [2, 5, 3, 1, 4] \end{array} \parallel \begin{array}{c} 2 \\ [2, 3, 5, 1, 4] \end{array} \parallel \begin{array}{c} 3 \\ [1, 2, 3, 5, 4] \end{array} \parallel \begin{array}{c} 4 \\ [1, 2, 3, 4, 5] \end{array}$

- Q2 – «la liste L[0:i+1] est triée par ordre croissant» est un invariant de sortie :
- À l'issue du tour de boucle i=1, on a échangé L[0] et L[1] si L[1] < L[0], et on les a laissés en place sinon. Donc L[0:2] est bien triée par ordre croissant.
  - Si à la fin du tour i-1, L[0:i] est triée par ordre croissant, on note  $x_0 \leq x_1 \leq \dots \leq x_{i-1}$  les éléments de L[0:i]. On va alors décaler d'un rang vers la droite tous les éléments précédents  $x = L[i]$  qui lui sont strictement plus grands et le placer à droite du premier plus petit. On aura alors dans L[0:i+1] :  $x_0 \leq x_1 \leq \dots \leq x_{k-1} \leq x \leq x_k \leq \dots \leq x_{i-1}$  donc L[0:i+1] est triée par ordre croissant, ce qui établit la récurrence. Ainsi, à la fin de l'algorithme (i = n - 1), L[0:n] = L est triée par ordre croissant.

- Q3 – Dans le meilleur cas, la liste est déjà triée par ordre croissant et le contenu de la boucle while n'est jamais exécuté : on obtient une complexité en O(n). Dans le pire des cas, la liste est triée dans l'ordre inverse, chaque boucle while a i - 1 tour donc on obtient une complexité en  $O\left(\sum_{i=1}^{n-1} (i-1)\right) + O(n) = O(n^2)$ . L'algorithme de tri fusion est plus rapide dans le pire des cas car il s'exécute en O(n ln n) dans tous les cas.

Q4 – En reprenant la fonction proposée (tri insertion) :

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

Q5 – Aucun attribut seul ne peut servir de clé primaire (plusieurs années pour un pays, plusieurs pays pour une année). Cependant, les couples (nom, annee) et (iso, annee) peuvent servir de clé primaire.

Q6 –

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

Q7 –

```
SELECT nom, cas/pop*100000 AS 'Taux d'indidence' FROM palu
JOIN demographie ON iso = pays and annee = periode
WHERE annee = 2011;
```

Q8 –

```
SELECT nom FROM palu p
WHERE annee = 2010
AND (SELECT COUNT(*) FROM palu WHERE cas > p.cas and annee = 2010) = 1;
```

(fonctionne sous mysql et postgresql mais pas sous sqlite...) La requête suivante fonctionne sous sqlite :

```
SELECT nom FROM palu
WHERE annee = 2010
AND cas = (SELECT MAX(cas) FROM palu
WHERE annee = 2010
AND cas != (SELECT MAX(cas) FROM palu WHERE annee = 2010)
);
```

Q9 –

```
tri_chaine(decès2010)
```

Partie II. Modèle à compartiments

Q10 – Le vecteur X = (S, I, R, D) et la fonction f sur  $\mathbb{R}^4$  par  $f(S, I, R, D) = (-rSI, rSI - (a+b)I, aI, bI)$  conviennent.

Q11 –

```
def f(X):
    """ Fonction définissant l'équation différentielle """
    global r, a, b
    S, I, R, D = X
    return np.array([-r*S*I, r*S*I-(a+b)*I, a*I, b*I])
```

Q12 – L'erreur par rapport aux solutions exactes est bien plus importante pour N = 7 que pour N = 250. Le temps de calculs le plus long est celui pour N = 250.

Q13 –

```
def f(X, Itau):
    """
    Fonction définissant l'équation différentielle
    Itau est la valeur de I(t - p * dt)
    """
    global r, a, b
    S, I, R, D = X
    return np.array([-r*S*Itau, r*S*Itau-(a+b)*I, a*I, b*I])
```

```
# Methode d'Euler
for i in range(N):
    t = t + dt
    Itau = XX[-p-1][1]
    X = X + df * f(X, Itau)
    tt.append(t)
    XX.append(X)
```

Q14 – Il suffit de remplacer la ligne Itau = XX[-p-1][1] par :

```
Itau = dt * sum( XX[-j-1][1] * h(j*dt) for j in range(p) )
```

Partie III. Modélisation dans des grilles

Q15 – La fonction grille(n) renvoie une liste de n listes contenant n fois 0.

### ❑ Q16 -

```
def init(n):
    G = grille(n)
    i = rd.randrange(n)
    j = rd.randrange(n)
    G[i][j] = 1
    return G
```

### ❑ Q17 -

```
def compte(G):
    n = len(G)
    nombres = [0] * 4
    for i in range(n):
        for j in range(n):
            nombres[ G[i][j] ] += 1
    return nombres
```

❑ Q18 - La fonction `est_exposee` renvoie un booléen.

### ❑ Q19 -

```
elif i == 0:
    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

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

### ❑ Q20 -

```
def suivant(G, p1, p2):
    n = len(G)
    for i in range(n):
        for j in range(n):
            if G[i][j] == 1: # case infectée
                if bernoulli(p1) == 1: G[i][j] = 3
                else: G[i][j] = 2
            elif G[i][j] == 0 and est_exposee(G,i,j): # case saine
                if bernoulli(p2) == 1: G[i][j] = 1
```

### ❑ Q21 -

```
def simulation(n, p1, p2):
    G = init(n)
    G1 = []
    while G != G1:
        G1 = G.copy()
        suivant(G, p1, p2)
    n = compte(G)
    return [ n[i]/4 for i in range(4) ]
```

On pourrait éviter la copie en renvoyant dans `suivant` un booléen vrai s'il y a eu un changement et faux sinon.

❑ Q22 - Comme les cases infectées évoluent systématiquement, il n'y en a plus à la fin de la simulation :  $x_1 = 0$ .

Comme toutes les cases sont dans un et un seul des 4 états,  $x_0 + x_1 + x_2 + x_3 = 1$ .

Les cases qui ont été atteintes par la maladie sont les cases décédées ou rétablies (qui le sont restées).

Donc  $x_{atteinte} = x_2 + x_3$ .

### ❑ Q23 -

```
def seuil(Lp2, Lxa):
    def dichotomie(indmin, indmax):
        if indmax - indmin == 1: return (Lp2[indmin], Lp2[indmax])
        else:
            indmilieu = (indmax + indmin) // 2
            if Lxa[indmilieu] > .5: dichotomie(indmin, indmilieu)
            else: dichotomie(indmilieu, indmax)
    return dichotomie(0, len(Lp2)-1)
```

❑ Q24 - On ne peut pas supprimer le test de la ligne 8 car dans `init()` on a une case infectée qui ne doit pas être vaccinée, et au fur et à mesure de la vaccination, il ne faut pas vacciner de nouveau des cases qui le sont déjà.

❑ Q25 - `init_vac(5, 0.2)` renvoie une grille de taille  $5 \times 5$  dont exactement 5 cases sont à 2, une autre est à 1 et toutes les autres sont à 0.