Nom : .....

Prénom : .....

2023/12/06

Classe : MP/PSI

Lycée Loges

ITC : DS n° 2 - 2 heures

type Mines/Centrale

calculatrice interdite

Exercice I : .... /-

Exercice II : .... /-

Note finale : .... / -

Les deux exercices sont indépendants.

⚠ Faites attention à la lisibilité de votre code et vos explications

Exercice I

MODÉLISATION DE LA PROPAGATION D'UNE ÉPIDÉMIE

L'étude de la propagation des épidémies joue un rôle important dans les politiques de santé publique. Les modèles mathématiques ont permis de comprendre pourquoi il a été possible d'éradiquer la variole à la fin des années 1 970 et pourquoi il est plus difficile d'éradiquer d'autres maladies comme la poliomyélite ou la rougeole. Ils ont également permis d'expliquer l'apparition d'épidémies de grippe tous les hivers. Aujourd'hui, des modèles de plus en plus complexes et puissants sont développés pour prédire la propagation d'épidémies à l'échelle planétaire telles que le SRAS, le virus H5N1 ou le virus Ebola. Ces prédictions sont utilisées par les organisations internationales pour établir des stratégies de prévention et d'intervention.

Le travail sur ces modèles mathématiques s'articule autour de trois thèmes principaux : traitement de données, simulation numérique (par plusieurs types de méthodes), identification des paramètres intervenant dans les modèles à partir de données expérimentales. Ces trois thèmes sont abordés dans le sujet. *Les parties sont indépendantes.*

Dans tout le problème, on peut utiliser une fonction traitée précédemment. On suppose que les bibliothèques numpy et random ont été importées par :

```
import numpy as np
import random as rd
```

I Partie I : tri

Dans le but ultérieur de réaliser des études statistiques, on souhaite se doter d'une fonction tri. On se donne la fonction `tri` suivante, écrite en `Python` :

```
1 def tri(L):
2     n = len(L)
3     for i in range(1,n):
4         j = i
5         x = L[i]
6         while 0 < j and x < L[j-1]:
7             L[j] = L[j-1]
8             j = j-1
9         L[j] = x
```

☐ **Q1 -** Lors de l'appel `tri(L)` lorsque L est la liste `[5, 2, 3, 1, 4]`, donner le contenu de la liste L à la fin de chaque itération de la boucle `for`.

☐ **Q2 -** Soit L une liste non vide d'entiers ou de flottants. Montrer que « la liste `L[0:i+1]` (avec la convention `Python`) est triée par ordre croissant à l'issue de l'itération i » est un invariant de boucle. En déduire que `tri(L)` trie la liste L.

☐ **Q3 -** Évaluer la complexité dans le meilleur et dans le pire des cas de l'appel `tri(L)` en fonction du nombre n d'éléments de L. Citer un algorithme de tri plus efficace dans le pire des cas. Quelle en est la complexité dans le meilleur et dans le pire des cas ?

On souhaite, partant d'une liste constituée de couples (chaîne, entier), trier la liste par ordre croissant de l'entier associé suivant le fonctionnement suivant :

```
>>> L = [['Bresil', 76], ['Kenya', 26017], ['Ouganda', 8431]]
>>> tri_chaine(L)
>>> L
[['Bresil', 76], ['Ouganda', 8431], ['Kenya', 26017]]
```

☐ **Q4 -** Écrire en `Python` une fonction `tri_chaine` réalisant cette opération.

II Partie II : Modèle à compartiments

On s'intéresse ici à une première méthode de simulation numérique.

Les modèles compartimentaux sont des modèles déterministes où la population est divisée en plusieurs catégories selon leurs caractéristiques et leur état par rapport à la maladie. On considère dans cette partie un modèle à quatre compartiments disjoints : sains (S, "susceptible"), infectés (I, "infected"), rétablis (R, "recovered", ils sont immunisés) et décédés (D, "dead"). Le changement d'état des individus est gouverné par un système d'équations différentielles obtenues en supposant que le nombre d'individus nouvellement infectés (c'est-à-dire le nombre de ceux qui quittent le compartiment S) pendant un intervalle de temps donné est proportionnel au produit du nombre d'individus infectés avec le nombre d'individus sains.

En notant S(t), I(t), R(t) et D(t) la fraction de la population appartenant à chacune des quatre catégories à l'instant t, on obtient le système :

$$\left.\begin{aligned} \frac{d}{dt}S(t) &= -rS(t)I(t) \\ \frac{d}{dt}I(t) &= rS(t)I(t) - (a+b)I(t) \\ \frac{d}{dt}R(t) &= aI(t) \\ \frac{d}{dt}D(t) &= bI(t) \end{aligned}\right\} \quad (1)$$

avec r le taux de contagion, a le taux de guérison et b le taux de mortalité. On suppose qu'à l'instant initial $t = 0$, on a $S(0) = 0{,}95$, $I(0) = 0{,}05$ et $R(0) = D(0) = 0$.

☐ **Q5 -** Préciser un vecteur X et une fonction f (en donnant son domaine de définition et son expression) tels que le système différentiel (1) s'écrive sous la forme

$$\frac{d}{dt}X = f(X).$$

☐ **Q6 -** Compléter la ligne 4 du code suivant (on précise que `np.array` permet de créer un tableau `numpy` à partir d'une liste donnant ainsi la possibilité d'utiliser les opérateurs algébriques).

```
1  def f(X):
2      """ Fonction definissant l'equation differentielle """
3      global r, a, b
4      # a completer
5
6  # Parametres
7  tmax = 25.
8  r = 1.
9  a = 0.4
10 b = 0.1
11 X0 = np.array([0.95, 0.05, 0., 0.])
12
13 N = 250
14 dt = tmax/N
15
16 t = 0
17 X = X0
18 tt = [t]
19 XX = [X]
20
21 # Methode d'Euler
22 for i in range(N):
23     t = t + dt
```

```
24      X= X + dt * f(X)
25      tt.append(t)
26      XX.append(X)
```

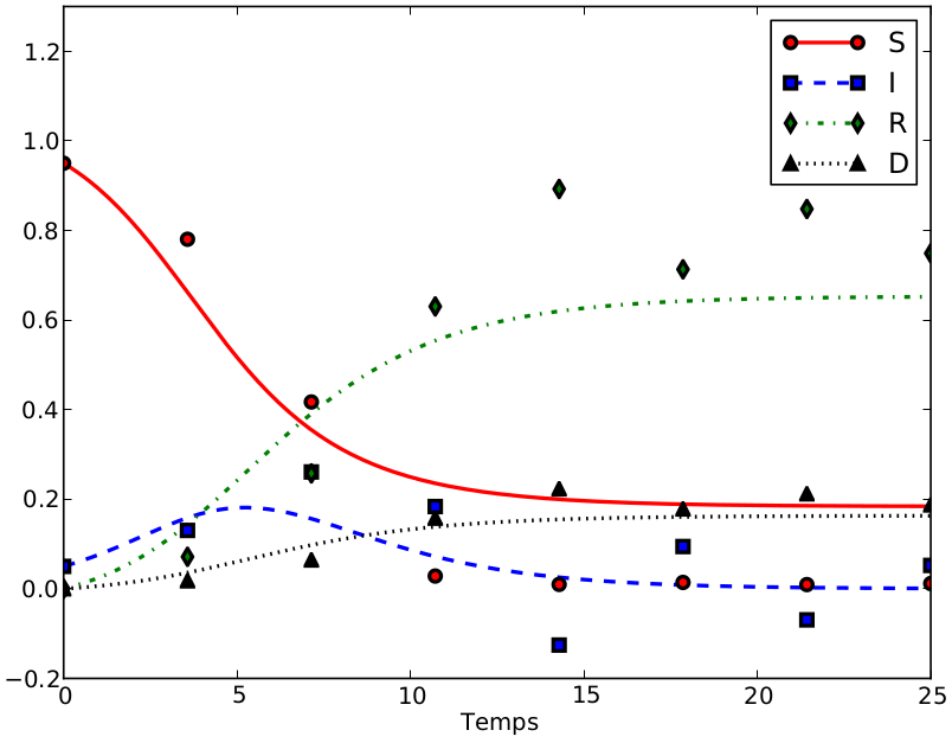

FIGURE 1 – Représentation graphique des quatre catégories S, I, R et D en fonction du temps pour $N = 7$ (points) et $N = 250$ (courbes)

☐ **Q7 -** La figure 1 représente les quatre catégories en fonction du temps obtenues en effectuant deux simulations : la première avec $N = 7$ correspond aux points (cercle, carré, losange, triangle) et la seconde avec $N = 250$ correspond aux courbes. Expliquer la différence entre ces deux simulations. Quelle simulation a nécessité le temps de calcul le plus long ?

En pratique, de nombreuses maladies possèdent une phase d'incubation pendant laquelle l'individu est porteur de la maladie mais ne possède pas de symptômes et n'est pas contagieux. On peut prendre en compte cette phase d'incubation à l'aide du système à retard suivant :

$$\begin{cases} \frac{d}{dt}S(t) = -rS(t)I(t-\tau) \\ \frac{d}{dt}I(t) = rS(t)I(t-\tau) - (a+b)I(t) \\ \frac{d}{dt}R(t) = aI(t) \\ \frac{d}{dt}D(t) = bI(t) \end{cases}$$

où τ est le temps d'incubation. On suppose alors que pour tout $t \in [-\tau,0]$, $S(t) = 0{,}95$, $I(t) = 0{,}05$ et $R(t) = D(t) = 0$.

En notant $tmax$ la durée des mesures et N un entier donnant le nombre de pas, on définit le pas de temps $dt = tmax/N$. On suppose que $\tau = p \times dt$ où p est un entier ; ainsi p est le nombre de pas de retard.

Pour résoudre numériquement ce système d'équations différentielles à retard (avec $tmax = 25$, $N = 250$ et $p = 50$), on a écrit le code suivant :

```
1   def f(X, Itau):
2       """ Fonction definissant l'equation differentielle
```

```
3       Itau est la valeur de I(t - p * dt ) """
4       global r, a, b
5       # a completer
6
7   # Parametres
8   tmax = 25.
9   r = 1.
10  a = 0.4
11  b = 0.1
12  X0 = np.array([0.95, 0.05, 0., 0.])
13
14  N = 250
15  dt = tmax/N
16  p = 50
17
18  t = 0
19  X = X0
20  tt = [t]
21  XX = [X]
22
23  # Methode d'Euler
24  for i in range(N):
25      t = t + dt
26      # a completer
27      tt.append(t)
28      XX.append(X)
```

☐ **Q8 -** Compléter les lignes 5 et 26 du code précédent (utiliser autant de lignes que nécessaire).

On constate que le temps d'incubation de la maladie n'est pas nécessairement le même pour tous les individus. On peut modéliser cette diversité à l'aide d'une fonction positive d'intégrale unitaire (dite de densité) $h : [0,\tau] \to \mathbb{R}_+$ telle que représentée sur la figure 2. On obtient alors le système intégro-différentiel :

$$\begin{cases} \frac{d}{dt}S(t) = -rS(t)\int_0^\tau I(t-s)h(s)ds \\ \frac{d}{dt}I(t) = rS(t)\int_0^\tau I(t-s)h(s)ds - (a+b)I(t) \\ \frac{d}{dt}R(t) = aI(t) \\ \frac{d}{dt}D(t) = bI(t) \end{cases}$$

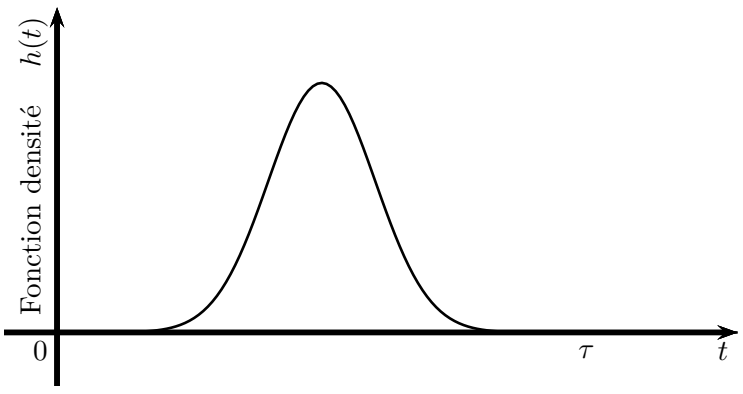

FIGURE 2 – Exemple d'une fonction de densité.

On supposera à nouveau que pour tout $t \in [-\tau,0]$, $S(t) = 0{,}95$, $I(t) = 0{,}05$ et $R(t) = D(t) = 0$. Pour

j entier compris entre 0 et N, on pose $t_j = j \times dt$. Pour un pas de temps dt donné, on peut calculer numériquement l'intégrale à l'instant t_i $(0 \leqslant i \leqslant N)$ à l'aide de la méthode des rectangles à gauche en utilisant l'approximation :

$$\int_0^\tau I(t_i - s)h(s)ds \approx dt \times \sum_{j=0}^{p-1} I(t_i - t_j)h(t_j).$$

☐ **Q9 -** On suppose que la fonction h a été écrite en `Python`. Expliquer comment modifier le programme de la question précédente pour résoudre ce système intégro-différentiel (on explicitera les lignes de code nécessaires).

III Partie III : Modélisation dans des grilles

On s'intéresse ici à une seconde méthode de simulation numérique (dite par *automates cellulaires*).

Dans ce qui suit, on appelle *grille de taille* $n \times n$ une liste de n listes de longueur n, où n est un entier strictement positif.

Pour mieux prendre en compte la dépendance spatiale de la contagion, il est possible de simuler la propagation d'une épidémie à l'aide d'une grille. Chaque case de la grille peut être dans un des quatre états suivants : saine, infectée, rétablie, décédée. On choisit de représenter ces quatre états par les entiers :

0 (Sain), 1 (Infecté), 2 (Rétabli) et 3 (Décédé).

L'état des cases d'une grille évolue au cours du temps selon des règles simples. On considère un modèle où l'état d'une case à l'instant $t+1$ ne dépend que de son état à l'instant t et de l'état de ses huit cases voisines à l'instant t (une case du bord n'a que cinq cases voisines et trois pour une case d'un coin). Les *règles de transition* sont les suivantes :

- une case décédée reste décédée ;
- une case infectée devient décédée avec une probabilité p_1 ou rétablie avec une probabilité $(1 - p_1)$;
- une case rétablie reste rétablie ;
- une case saine devient infectée avec une probabilité p_2 si elle a au moins une case voisine infectée et reste saine sinon.

On initialise toutes les cases dans l'état sain, sauf une case choisie au hasard dans l'état infecté.

☐ **Q10 -** On a écrit en `Python` la fonction `grille(n)` suivante

```
def grille(n):
    M = []
    for i in range(n):
        L = []
        for j in range(n): L.append(0)
        M.append(L)
    return M
```

Décrire ce que retourne cette fonction.

On pourra dans la question suivante utiliser la fonction `randrange(p)` de la bibliothèque `random` qui, pour un entier positif p, renvoie un entier choisi aléatoirement entre 0 et $p-1$ inclus.

☐ **Q11 -** Écrire en `Python` une fonction `init(n)` qui construit une grille `G` de taille $n \times n$ ne contenant que des cases saines, choisit aléatoirement une des cases et la transforme en case infectée, et enfin renvoie `G`.

☐ **Q12 -** Écrire en `Python` une fonction `compte(G)` qui a pour argument une grille `G` et renvoie la liste `[n0, n1, n2, n3]` formée des nombres de cases dans chacun des quatre états.

D'après les règles de transition, pour savoir si une case saine peut devenir infectée à l'instant suivant, il faut déterminer si elle est exposée à la maladie, c'est-à-dire si elle possède au moins une case infectée dans son voisinage. Pour cela, on écrit en `Python` la fonction `est_exposee(G, i, j)` suivante.

```
1  def est_exposee(G, i, j):
2      n = len(G)
3      if i == 0 and j == 0:
4          return (G[0][1]-1)*(G[1][1]-1)*(G[1][0]-1) == 0
5      elif i == 0 and j == n-1:
6          return (G[0][n-2]-1)*(G[1][n-2]-1)*(G[1][n-1]-1) == 0
7      elif i == n-1 and j == 0:
8          return (G[n-1][0]-1)*(G[n-2][1]-1)*(G[n-2][0]-1) == 0
9      elif i == n-1 and j == n-1:
10         return (G[n-1][n-2]-1)*(G[n-2][n-2]-1)*(G[n-2][n-1]-1) == 0
11     elif i == 0:
12         # a completer
13     elif i == n-1:
14         return
           ↪  (G[n-1][j-1]-1)*(G[n-2][j-1]-1)*(G[n-2][j]-1)*(G[n-2][j+1]-1)*(G[n-1][j+1]-1)
           ↪  == 0
15     elif j == 0:
16         return (G[i-1][0]-1)*(G[i-1][1]-1)*(G[i][1]-1)*(G[i+1][1]-1)*(G[i+1][0]-1) == 0
17     elif j == n-1:
18         return
           ↪  (G[i-1][n-1]-1)*(G[i-1][n-2]-1)*(G[i][n-2]-1)*(G[i+1][n-2]-1)*(G[i+1][n-1]-1)
           ↪  == 0
19     else:
20         # a completer
```

☐ **Q13 -** Quel est le type du résultat renvoyé par la fonction `est_exposee` ?

☐ **Q14 -** Compléter les lignes 12 et 20 de la fonction `est_exposee`.

☐ **Q15 -** Écrire une fonction `suivant(G, p1, p2)` qui fait évoluer toutes les cases de la grille `G` à l'aide des règles de transition et renvoie une nouvelle grille correspondant à l'instant suivant. Les arguments `p1` et `p2` sont les probabilités qui interviennent dans les règles de transition pour les cases infectées et les cases saines. On pourra utiliser la fonction `bernoulli(p)` suivante qui simule une variable aléatoire de Bernoulli de paramètre p : `bernoulli(p)` vaut 1 avec la probabilité p et 0 avec la probabilité $(1-p)$.

```
def bernoulli(p):
    x = rd.random()
    if x <= p:
        return 1
    else:
        return 0
```

On reproduit ci-dessous le descriptif de la documentation `Python` concernant la fonction `random` de la bibliothèque `random` :

```
random.random()
    Return the next random floating point number in the range [0.0, 1.0).
```

Avec les règles de transition du modèle utilisé, l'état de la grille évolue entre les instants t et $t+1$ tant qu'il existe au moins une case infectée.

☐ **Q16 -** Écrire en `Python` une fonction `simulation(n, p1, p2)` qui réalise une simulation complète avec une grille de taille $n \times n$ pour les probabilités `p1` et `p2`, et renvoie la liste `[x0, x1, x2, x3]` formée des proportions de cases dans chacun des quatre états à la fin de la simulation (une simulation s'arrête lorsque la grille n'évolue plus).

☐ **Q17 -** Quelle est la valeur de la proportion des cases infectées `x1` à la fin d'une simulation ? Quelle relation vérifient `x0`, `x1`, `x2` et `x3` ? Comment obtenir à l'aide des valeurs de `x0`, `x1`, `x2` et `x3` la valeur `x_atteinte` de la proportion des cases qui ont été atteintes par la maladie pendant une simulation ?

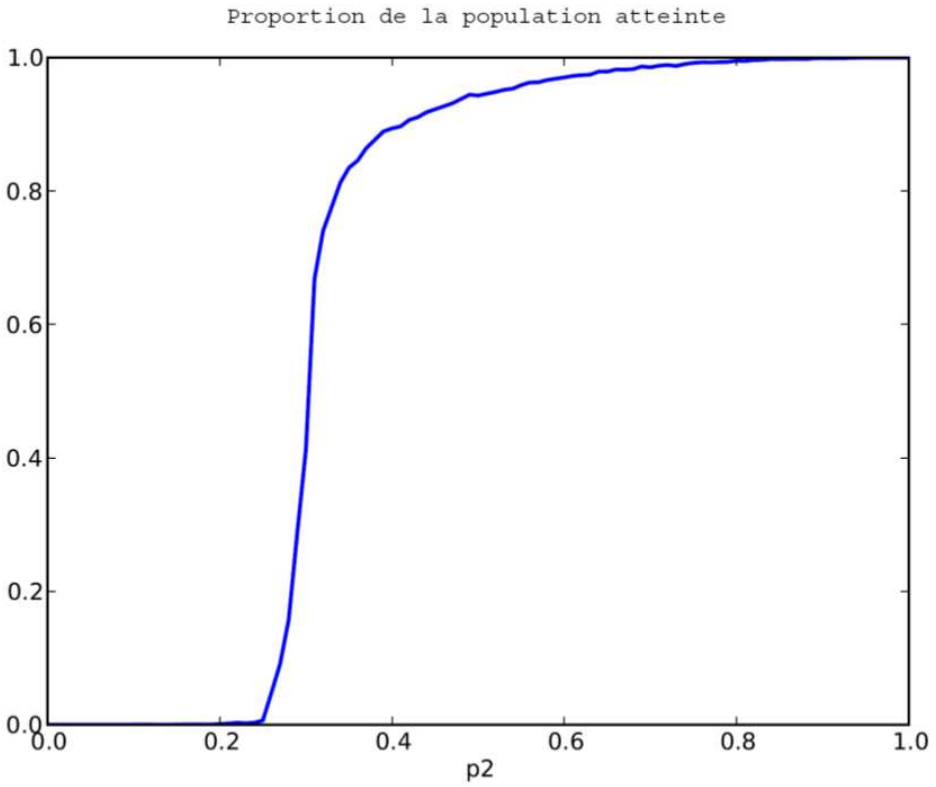

FIGURE 3 – Représentation de la proportion de la population qui a été atteinte par la maladie pendant la simulation en fonction de la probabilité p_2.

On fixe `p1` à 0,5 et on calcule la moyenne des résultats de plusieurs simulations pour différentes valeurs de `p2`. On obtient la courbe de la figure 3.

☐ **Q18 -** On appelle *seuil critique de pandémie* la valeur de `p2` à partir de laquelle plus de la moitié de la population a été atteinte par la maladie à la fin de la simulation. On suppose que les valeurs de `p2` et `x_atteinte` utilisées pour tracer la courbe de la figure 3 ont été stockées dans deux listes de même longueur `Lp2` et `Lxa`. Écrire en `Python` une fonction `seuil(Lp2, Lxa)` qui détermine par dichotomie un encadrement `[p2cmin, p2cmax]` du seuil critique de pandémie avec la plus grande précision possible. On supposera que la liste `Lp2` croît de 0 à 1 et que la liste `Lxa` des valeurs correspondantes est croissante.

Pour étudier l'effet d'une campagne de vaccination, on immunise au hasard à l'instant initial une fraction q de la population. On a écrit la fonction `init_vac(n, q)`.

```
1  def init_vac(n, q):
2      G = init(n)
3      nvac = int(q * n**2)
4      k = 0
5      while k < nvac:
6          i = rd.randrange(n)
7          j = rd.randrange(n)
8          if G[i][j] == 0:
9              G[i][j] = 2
10             k += 1
11     return G
```

☐ **Q19 -** Peut-on supprimer le test en ligne 8 ?

☐ **Q20 -** Que renvoie l'appel `init_vac(5, 0.2)` ?

Exercice II

Au cours du développement des fonctions nécessaires à la manipulation des nombres premiers on s'aperçoit que le choix des algorithmes pour évaluer chaque fonction est primordial pour garantir des performances acceptables. On souhaite donc mener des tests à grande échelle pour évaluer les performances réelles du code qui a été développé. Pour ce faire on effectue un grand nombre de tests sur une multitude d'ordinateurs. Les données sont ensuite centralisées dans une base de données composée de deux tables.

La première table est `ordinateur` et permet de stocker des informations sur les ordinateurs utilisés pour les tests. Ses attributs sont :

- `nom CHAR`, clé primaire, le nom de l'ordinateur.
- `gflops INTEGER`, la puissance de l'ordinateur en milliards d'opérations flottantes par seconde.
- `ram INTEGER`, la quantité de mémoire vive de l'ordinateur en Go.

Exemple du contenu de cette table :

nom	gflops	ram
nyarlathotep114	69	32
nyarlathotep119	137	32
...		
shubniggurath42	133	16
azathoth137	85	8

La seconde table est `fonctions` et stoke les informatioons sur les tests effectués pour différentes fonctions en cours de développement. Ses attributs sont :

- `id INTEGER`, l'identifiant du test effectué.
- `nom CHAR`, le nom de la fonction testée.
- `algorithme CHAR`, le nom de l'algorithme qui permet le calcul de la fonction testée.
- `teste_sur TEXT`, le nom du PC sur lequel le test a été effectué.
- `temps_exec INTEGER`, le temps d'exécution du test en millisecondes.

Exemple du contenu de cette table :

id	nom	algorithme	teste_sur	temps_exec
1	li	rectangle	nyarlathotep165	2638
2	li	rectangle	shubniggurath28	736
3	li	trapezes	nyarlathotep165	4842
...				
2154	Ei	puiseux	nyarlathotep145	2766
2155	aleatoire	bbs	azathoth145	524

Q1. Expliquer pourquoi il n'est pas possible d'utiliser l'attribut `nom` comme clé primaire de la table `fonction`.

Écrire les requêtes SQL permettant d'obtenir les tables de données suivantes :

Q2. Donner les noms des PC ayant une mémoire vive de 32 Go.

Q3. Donner les noms des PC et les noms des fonctions qui y sont testées.

Q4. Donner la liste des algorithmes permettant le calcul des fonctions testées. On ne souhaite pas avoir de doublons dans les données obtenues.

Q5. Donner le nom des ordinateurs ayant un temps d'exécution supérieur strictement à 1 000 millisecondes pour la fonction `li`.

Q6. Pour la fonction nommée `Ei`, trier les résultats des tests du plus lent au plus rapide. Pour chaque test retenir le nom de l'algorithme utilisé, le nom du PC sur lequel il a été effectué et la puissance du PC.

Q7. Donner les noms des PC sur lesquels l'algorithme `rectangles` n'a pas été testé pour la fonction nommée `li`

Q8. Combien de PC ont été testé avec la fonction `li` ?

Q9. Donner les noms des fonctions ainsi que la moyenne de leurs temps d'exécution.

Q10. Donner les noms des PC dont le temps d'exécutions pour la fonction `li` est supérieur à la moyenne du temps d'exécution de cette fonction.

Q11. Quel résultat obtient-on avec la requête suivante ?

```
SELECT teste_sur
FROM fonctions
GROUP BY teste_sur
HAVING COUNT(id) =
        SELECT MAX(nbr)
        FROM SELECT COUNT(id) AS nbr
                    FROM fonction
                    GROUP BY teste_sur
```