DS4 : corrigé

MP Lycée Camille Jullian

1er avril 2022

Partie I : Préliminaires.

Q1

Pas besoin d'importer tout le module si on ne veut vraiment se servir que des fonctions citées :

```
from math import log, sqrt, floor, ceil
print(log(0.5))
```

Q2

```
def sont_proches(x,y) :
    atol=10**(-5)
    rtol=10**(-8)
    return abs(y-x) <= atol+abs(y)*rtol
```

Q3

L'appel mystere(1001,10) va calculer récursivement 1+mystere(100.1,10), puis 2+mystere(10.01,10) et enfin 3+mystere(1.001,10) pour finalement renvoyer 3.

Q4

De façon générale, le nombre d'étapes récursives effectuées par mystere va correspondre à l'entier n tel que $b^n \leqslant x \leqslant b^{n+1}$, donc la fonction renvoie $\lfloor \log_b(x) \rfloor$. Autrement dit, mystere renvoie le nombre de chiffres en base b du nombre x, diminué d'une unité.

Q5

La variable x1 contient simplement la dernière valeur calculée, soit $10^5 \times 10^{-5}$, ce qui est logiquement égal à 1 (pas de raison que le calcul ne soit pas précis ici). Par contre, x2 a été obtenu en additionnant 100 000 fois de suite la valeur 0.00001, ce qui devrait normalement donner aussi un résultat égal à 1, mais en pratique les approximations de Python lors du calcul des sommes de flottants s'accumulent pour donner la valeur inexacte affichée dans l'énoncé. La morale, c'est qu'il vaut mieux éviter d'additionner le pas à chaque étape quand on écrit des programmes si on souhaite garder une précision optimale.

Partie II. Génération de nombres premiers.

II.a Approche systématique.

Q6

Comme 4 Go correspondent à 32 Gigabits, on pourra donc stocker 10^9 booléens au maximum.

Q7

Un booléen ne nécessite théoriquement qu'un seul bit pour être stocké (deux valeurs possibles), on peut gagner un facteur 32.

Q8

Il faut faire attention à plusieurs choses : créer une liste contenant bien N valeurs et gérer correctement les indices Python (la case i de la liste correspondra au caractère premier ou non de l'entier $i+1$), puis surtout écrire une boucle qui marque comme non premiers les multiples de i sans faire des tests inutiles (on sait très bien où se trouvent les multiples de i dans la liste, il ne faut surtout pas parcourir toute la liste pour tester si chaque indice est multiple de i ou non !). Un dernier détail déjà signalé en cours : une ligne du genre « if b !=False » quand b est une variable booléenne, c'est très moche, un simple « if b » fait exactement la même chose !

```
def erato_iter(N) :
    L=[True for i in range(N)]
    L[0]=False
    for in range(2,int(N**0.5)+1) :
        if L[i-1] :
            for j in range(2,N//i+1) :
                L[j*i-1]=False
    return L
```

Q9

On a une double boucle, dont la deuxième ne s'effectue que pour les nombres premiers inférieurs ou égaux à $\sqrt{N}$, et on effectuera dans ce cas un nombre de calcul proportionnel à $\dfrac{N}{p}$, ce qui donne une complexité en $O\left(\displaystyle\sum_{p\leqslant\sqrt{N}} \frac{N}{p}\right) = O(N\ln(\ln(\sqrt{N})))$ en utilisant la formule donnée dans l'énoncé (la somme ne portant que sur les entiers premiers). On peut se contenter pour simplifier de $O(N\ln(\ln(N)))$.

Q10

La base ne change en fait rien au raisonnement, le nombre de chiffres n est proportionnel à $\ln(N)$, donc on aura du $O(b^n\ln(n))$, où b est la base choisie (2 est le plus logique pour un informaticien mais si on choisit la base 10 on ne sera pas pénalisé !).

II.b Génération rapide de nombres premiers.

Q11

L'énoncé est manifestement mal fichu : le principe semble être de créer un nombre A en affectant à chacun de ses bits un 0 ou un 1 aléatoirement, mais si l'algorithme tourne vraiment pour i variant de 1 (et pas de 0) à $N-1$, le bit de poids faible restera forcément nul, ce qui est idiot. Respectons quand même l'énoncé : la valeur maximale de A est donc $\displaystyle\sum_{i=1}^{N-1} 2^i = 2^N - 2$.

Q12

On suppose pour l'extraction de la partie décimale du temps d'horloge que celui-ci est toujours donné avec sept décimales pour simplifier l'écriture de la commande. Notons en passant l'incohérence

du programme de l'énoncé avec ce qu'on a calculé pour la question précédente : la boucle commence bien à $i = 0$ et pas à $i = 1$, ce qui donne plutôt une valeur maximale de A égale à $2^N - 1$.

```
from time import time
def bbs(N) :
    p1,p2=24375763,28972763
    M=p1*p2
    t=time()
    xi=(t-int(t))*10**7
    A=0
    for i in range(N) :
        if xi%2==0 :
            A=A+2**i
        xi=(xi**2)%M
    return A
```

Q13

On veut appliquer le programme bbs à un nombre N tel que que $2^N - 1 < n_{max}$ (pour que le nombre obtenu soit nécessairement plus petit que n_{max}), donc pour $N \leqslant \log_2(n_{max})$. On peut si on le veut calculer cette valeur à l'aide du programme mystère donné dans l'énoncé, histoire qu'il serve à quelque chose. La boucle « while True » permet de créer un environnement qui va répéter les mêmes opérations tant que le return ne sera pas effectué, donc ici tant qu'on n'a pas trouvé un entier A convenable. C'est bien sûr dangereux, on a un risque de boucle infinie si on n'atteint jamais le return.

```
def premier_rapide(n_max) :
    N=mystere(n_max)
    while True :
        A=bbs(N)
        if (2**(A-1)%A==1) and (3**(A-1)%A==1) and (5**(A-1)%A==1) and (7**(A-1)%A==1) :
            return A
```

Q14

On revient à des choses assez classiques, le programme va bien sûr utiliser le crible programmé en première partie pour savoir si les entiers obtenus sont réellement premiers :

```
def stats_bbs_fermat(N,nb) :
    n,L=0,[]
    crible=erato_iter(N)
    for i in range(nb) :
        A=premier_rapide(N)
        if not crible[A-1] :
            n+=1
            L.append(A)
    return n/nb,L
```

Partie III. Compter les nombres premiers.

III.a Calcul de $\pi(n)$ via un crible.

Q15

En gros, on nous demande de prendre la liste obtenue via le crible et de la parcourir une seule fois, en mettant à jour en cours de route la valeur de $\pi(n)$. Tout ça ne devrait pas être trop dur à programmer :

```
def Pi(N) :
    L=erato_iter(N)
    L2,p=[],0
    for i in range(N) :
        if L[i] :
            p+=1
        L2.append([i+1,p])
    return L2
```

Q16

Il faut simplement faire attention ici de bien démarrer à l'indice 5392...

```
def verif_Pi(N) :
    L=Pi(N)
    for i in range(5392,N) :
        n,p=L[i]
        if n/(log(n)-1)>=p :
            return False
    return True
```

III.b Calcul d'une valeur approchée de $\pi(n)$.

Q17

En supposant l'intervalle d'intégration fixé, la complexité de la méthode des rectangles dépend linéairement du nombre de rectangles, donc on a ici du $O\left(\frac{1}{pas}\right)$.

Q18

La complexité est la même pour les rectangles centrés que pour les rectangles à droite (on ne calcule pas les mêmes valeurs de la fonction, c'est tout), mais aussi pour les trapèzes (une valeur de plus à calculer).

Q19

Un programme qu'il faut savoir écrire, comme je vous l'ai déjà dit. Attention à bien faire des rectangles à droite, donc à faire varier les abscisses points dont on va calculer les valeurs entre $a+pas$ (ne pas partir à a) et b.

```
def inv_ln_rect_d(a,b,pas) :
    s=0
    for i in range(1,(b-a)/pas+1) :
        s+=1/log(a+i*pas)
    return s
```

Q20

Il faut faire attention à bien découper le calcul en deux intervalles quand $x > 1$, donc à calculer l'intégrale entre 0 et $1 - pas$, puis entre $1 + pas$ et x.

```
def li_d(x,pas) :
```

```
if x==1 :
    return -float('inf')
elif x<1 :
    return inv_ln_rect_d(0,x,pas)
inv_ln_rect_d(0,1-pas,pas)+inv_ln_rect_d(1+pas,x,pas)
```

Q21

Le graphe donné dans la figure 1 de l'énoncé montre que la fonction s'annule aux environs de $x = 1.4$. Il est donc tout à fait normal que l'écart **relatif** devienne très grand au voisinage de cette valeur (l'écart **absolu** n'est jamais égal à 0).

Q22

On constate plus précisément que cet écart semble très proche de 1 quand x est plus grand que 1. Si on regarde bien la figure 4 de l'énoncé, on se rend compte que le fait d'utiliser des rectangles à droite des deux côtés brise la symétrie de l'approximation. Ainsi, quand on calcule notre valeur approchée, on devrait négliger l'intégrale comprise entre $1-\varepsilon$ et $1+\varepsilon$ (qui tend bien vers 0 quand ε tend vers 0), mais à la place on néglige l'intégrale comprise $1-\varepsilon$ et $1+2\varepsilon$ (c'est un peu plus compliqué que ça car on a du coup « un terme en plus » à droite qu'à gauche, mais ce dernier va tendre vers 0 quand on diminue le pas), autrement dit, on a un écart qui correspond à la limite quand ε tend vers 0 de $\displaystyle\int_{1+\varepsilon}^{1+2\varepsilon} \frac{1}{\ln(x)}\,dx$. Cette intégrale est comprise entre $\dfrac{\varepsilon}{\ln(1+2\varepsilon)}$ et $\dfrac{\varepsilon}{\ln(1+\varepsilon)}$, expressions qui ont pour limites respectives $\dfrac{1}{2}$ et 1 quand ε tend vers 0. On pourrait prouver que la limite est bien égale à 1, mais on a déjà fait trop de maths pour une question d'un sujet d'info.

Q23

Le plus simple est de ne pas utiliser la méthode des rectangles à droite pour la partie d'intégrale à droite de 1, mais celle des rectangles à gauche, pour restaurer la symétrie. Sinon, on peut aussi salement ajouter 1 à la valeur calculée par notre précédent programme quand $x > 1$...

Q24

Pour récompenser les élèves ayant miraculeusement réussi à atteindre cette question en moins d'une heure et demie, un programme bien long et pénible à écrire. Il est simple de d'abord calculer le fonction EI, en calculant les puissances et factorielles au sein de la boucle pour éviter un programme de complexité trop grande. Attention à garder en permanence pendant le calcul les **deux** dernières valeurs calculées pour la somme, puisqu'il faut tester si elles sont suffisamment proches pour arrêter le calcul.

```
def Ei(x) :
    g=0.577215664901
    fact,puiss=1,x
    u,v=g+log(x),g+log(x)+x
    for i in range(2,101) :
        fact,puiss=fact*i,puiss*x
        u,v=v,v+puiss/(i*fact)
        if sont_proches(u,v) :
            return v
    return False
```

Il ne reste plus qu'à écrire la fonction demandée en faisant attention aux condition imposées : $\ln(x)$ doit être strictement positif et inférieur à 40.

```
def li_dev(x) :
```

```
z=log(x)
if z<=0 or z>40 :
        return False
return Ei(z)
```

IV. Analyse de performance de code.

Q25

Une clé primaire doit prendre une valeur différente pour chaque ligne de la table, ce qui n'est manifestement pas le cas de l'attribut fonction.

Q26

1. SELECT COUNT(*), AVG(ram) FROM ordinateurs

   Notons que le fait de « mélanger » deux requêtes ne pose ici aucun problème puisqu'elles n'affichent chacune qu'une seule valeur de toute façon.

2. Il y a plein de façons de faire, on peut utiliser un EXCEPT si on connait, mais je ne vous en ai pas reparlé, donc je vais privilégier le NOT IN que pour le coup il vaut mieux avoir en tête :

   SELECT nom FROM ordinateurs WHERE nom NOT IN (SELECT teste_sur FROM fonctions WHERE nom='li' AND algorithme='rectangles')

3. Une jointure pou terminer, attention au fait que les deux tables ont un attribut nom qui ne représente pas du tout la même chose (surtout pas de NATURAL JOIN ici, ça ferait n'importe quoi).

   SELECT algorithme, ordinateurs.nom, gflops FROM ordinateurs JOIN fonctions
   ON test_sur=ordinateurs.nom WHERE fonctions.nom='Ei' ORDER BY temps_exec DESC