Exercice 1 : Une fonction inconnue

On donne ci-dessous le code d'une fonction `mystere` qui s'applique sur deux listes triées `li1` et `li2` :

```
def mystere(li1:list,li2:list) -> list:
    li1.append(float('infinity'))
    li2.append(float('infinity'))
    li=[]
    i=j=0
    while len(li)<len(li1)+len(li2):
        if li1[i]<=li2[j]:
            li.append(li1[i])
            i+=1
        else:
            li.append(li2[j])
            j+=1
    return li
```

où `float('infinity')` désigne un nombre de type `float` de valeur $+\infty$.

1. Faire tourner l'algorithme à la main sur l'appel `mystere([1,5,7,8,9,11],[2,4,6])`. On précisera à chaque itération l'état de la liste `li` et des variables `i` et `j`.

mystere([1,5,7,8,9,11],[2,4,6]) :

	Li	*i*	*j*
entrée 1er tour	[]	0	0
sortie 1er tour	[1]	*1*	*0*
sortie 2e tour	*[1,2]*	*1*	*1*
sortie 3e tour	*[1,2,4]*	*1*	*2*
sortie 4e tour	*[1,2,4,5]*	*2*	*2*
sortie 5e tour	*[1,2,4,5,6]*	*2*	*3*
sortie 6e tour	*[1,2,4,5,6,7]*	*3*	*3*
sortie 7e tour	*[1,2,4,5,6,7,8]*	*4*	*3*
sortie 8e tour	*[1,2,4,5,6,7,8,9]*	*5*	*3*
sortie 9e tour	*[1,2,4,5,6,7,8,9,11]*	*6*	*3*

2. Quelle est l'opération réalisée par la fonction précédente ? En utilisant un variant de boucle, montrer que la terminaison du programme.

Cette fonction permet de fusionner deux listes triées en une liste triée.
$N = len(li1) + len(li2) - len(li)$ est un variant de boucle puisqu'à chaque tour de boucle, li gagne un élément donc N décroit strictement en restant positif. Ainsi, le programme termine.

3. Proposer une version récursive de cette fonction.

```python
1  def fusion_rec(li1,li2):
2      if li1==[]:
3          return(li2)
4      elif li2==[]:
5          return(li1)
6      elif li1[0]<=li2[0]:
7          return([li1[0]]+fusion_rec(li1[1:],li2))
8      else:
9          return([li2[0]]+fusion_rec(li1,li2[1:]))
```

4. Établir la preuve de correction :

Si $li1$ ou $li2$ est vide, le programme renvoie la liste triée non vide.
sinon, soit l'invariant « Si `fusion_rec(li1,li2)` renvoie la fusion des deux listes $li1$ et $li2$ pour un nombre total d'éléments de $N-1$ », soit $n1$ et $n2$ les cardinaux respectifs de $li1$ et $li2$ avec $n1+n2=N$, alors `fusion_rec(li1[1:],li2)` et `fusion(li1,li2[1:])` renvoient des listes triées par hypothèse.
Alors ligne 6, on cherche $m=\mathrm{Min}(li1[0],li2[0])$ par exemple $m=li1[0]$. Comme les listes $li1$ et $li2$ sont triées, on a $m\leqslant \mathrm{Min}(li1[1:],li2)$ donc `[li1[0]]+fusion_rec(li1[1:],li2)` renvoie bien une liste triée et l'appel de la fonction conserve donc l'invariant. A la sortie, le programme renvoie bien la fusion triée des deux listes, le programme est correct.

5. On appelle p et q les longueurs respectives de `li` et `li2`. Justifier que la complexité temporelle $C(p+q)$ de la fonction `mystere` vérifie $C(p+q)=O(p+q)$. Qu'en est-il de la fonction `mystere_rec` ? Justifier.

- A chacun des $p+q$ tours, il y a une comparaison d'où une complexité en $O(p+q)$.
- Pour la fonction récursive, il y a une comparaison à chaque appel et l'appel récursif à deux listes de longueur totale $p+q-1$. D'où $C(p+q)=1+C(p+q-1)$.
D'où $C(p+q)=p+q-1+C(1)=p+q$. On obtient une complexité linéaire.

6. Écrire une fonction de tri récursif permettant de trier une liste quelconque et qui utilise l'opération réalisée à la question 2 :

```python
def tri_fusion(L:list):
    if len(L)==1: # surtout pas len(L)==0 sinon tourne a l'infini.
        return L
    else:
        n=len(L)//2
        return(fusion_rec(tri_fusion(L[:n]),tri_fusion(L[n:])))
```

Exercice 2 : Etude d'un algorithme de tri

On donne ci-dessous le code d'un algorithme de tri :

```
1   def tri(li:list) -> list:
2       for i in range(1,len(li)):
3           print('i=',i,li)
4           j=i-1
5           temp=li[i]
6           while j>=0 and li[j]>temp:
7               li[j+1]=li[j]
8               j=j-1
9           li[j+1]=temp
10      return None
```

7. Cet algorithme de tri crée-t-il une copie de la liste à trier ou la modifie-t-il ? Justifier.

Cet algorithme agit sur place, il modifie la liste `li` mais ne renvoie rien. Il ne crée pas une nouvelle liste.

8. Quel est le nom de ce tri ? Justifier en expliquant en détail la nature de l'opération réalisée par la boucle ligne 6.

C'est le tri par insertion. A la ligne 6, on prend l'élément numéro i et on l'insère parmi les éléments placés devant, qui sont déjà triés. Pour cela, on déplace tous les éléments placés devant qui lui sont supérieurs d'une case vers la droite, puis on place l'élément en question à la dernière place restante.

9. Proposer une version récursive `inserer_rec` permettant de réaliser l'opération réalisée dans la boucle ligne 6 par récursivité.

```
def inserer_rec(x:float,L:list) -> list:  Insère x dans la liste triée L
    if L==[]:
        return([x])
    else:
        if x<L[0]:
            return [x]+L
        else:
            return [L[0]]+inserer_rec(x,L[1:])
```

10. Établir la preuve de correction de la fonction `inserer_rec` :

Si $L = []$, on renvoie juste $[x]$ ce qui insère x.
Par récurrence, supposons l'invariant : « `inserer_rec(x,L)` insère x dans une liste triée L de taille $N-1$ » et soit L une liste triée de longueur N.
Alors soit $x < L[0]$ et on renvoie $[x]$ suivi de la liste triée L de sorte que la nouvelle liste obtenue est bien triée.
Sinon, `inserer_rec(x,L[1:])` insère x dans $L[1:]$ de taille $N-1$ selon l'hypothèse, que l'on concatène avec $[L[0]]$ qui reste le minimum de $L \cup [x]$. Ainsi, `inserer_rec(x,L)` renvoie $[L[0],...x,...,L[n-1]]$ triée.
Dans les deux cas, l'insertion de x est correcte.

11. Dans la fonction itérative, justifier l'invariant de boucle : « à la fin du i^e tour de boucle,la liste `li[:i+1]` est triée ».

- Pour $i=1$, $j=0$, $temp = li[0]$.
"$li[0] > temp$" est faux donc le programme ne rentre pas dans la boucle ligne 6 et $li[1]$ prend la valeur temp ligne 9.
Donc $li[:2] = [li[0], li[1]]$ est bien triée.
- Supposons qu'en sortie de la boucle i, $li[:i+1]$ soit triée.
A l'entrée du $(i+1)^e$ tour de boucle, j prend la valeur i et $temp = li[i+1]$.
Ligne 6, tant que $li[j] > temp$, on décale d'une case vers la droite. Posons donc $j_0 = \mathrm{Min}\{j \ / \ li[j] > temp\}$.
Alors $li[j_0 - 1] \leqslant temp$ et on pose $li[j_0] = temp$ à la ligne 9.
Ainsi $li[0] \leqslant ... \leqslant li[j_0-1] \leqslant temp = li[j_0] \leqslant li[j_0+1] \leqslant ... \leqslant li[i+1]$.
Donc $li[:i+2]$ est triée ce qui établit la récurrence.

12. En déduire la preuve de correction de la fonction `tri`.

A la sortie du dernier tour de boucle, $li[:n]$ est li est triée.

13. Quelle est la complexité temporelle de cet algorithme dans le pire des cas (à définir) ?

Dans le pire des cas, celui d'une liste décroissante, il y a i échanges au i^etour de boucle donc la complexité est en $\sum_{i=0}^{n-1} O(i) = O(n^2)$ quadratique.

14. Quelle est la complexité temporelle de cet algorithme dans le meilleur des cas (à définir) ?

Dans le meilleur des cas, celui d'une liste déjà triée, il y a n tours de boucles et aucun échange à chaque tour donc la complexité est en $\sum_{i=0}^{n-1} O(1) = O(n)$ linéaire.

15. Donner le nom et la complexité temporelle d'un algorithme de tri de meilleure complexité temporelle dans le pire des cas.

Le tri fusion a une complexité dans le pire des cas qui est meilleure puisqu'elle est en $O(n\ln(n))$.

Exercice 3

Partie I. Généralités

16. Dans un programme `Python` on souhaite pouvoir faire appel aux fonctions `log`, `sqrt`, `floor` et `ceil` du module `math` (`round` est disponible par défaut). Écrire des instructions permettant d'avoir accès à ces fonctions et d'afficher le logarithme népérien de 0.5.

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

17. Écrire une fonction `sont_proches(x, y)` qui renvoie `True` si la condition suivante est remplie et `False` sinon

$$|x - y| \leqslant atol + |y| \times rtol$$

où $atol$ et $rtol$ sont deux constantes, à définir dans le corps de la fonction, valant respectivement 10^{-5} et 10^{-8}. Les paramètres `x` et `y` sont des nombres quelconques.

```python
def sont_proches(x:float,y:float) -> bool:
atol, rtol = 1e-5, 1e-8
return abs(x-y) <= atol + abs(y)*rtol
```

18. On donne la fonction `mystere` ci-dessous. Que renvoie `mystere(1001,10)` ? Le paramètre `x` est un nombre strictement positif et **b** un entier naturel non nul.

```python
def mystere(x:int,b:int) -> int:
    if x < b:
        return 0
    else :
        return 1 + mystere(x / b, b)
```

> *La valeur retournée par* `mystere(1001,10)| est 3. En effet,`
> $\text{mystere}(1001,10) = 1 + \text{mystere}(100.1,10) = 2 + \text{mystere}(10.01,10) = 3 + \text{mystere}(1.001,10)3 + 0 = 3$

19. Exprimer ce que renvoie `mystere` et exprimer la valeur renvoyée en fonction de la partie entière d'une fonction usuelle.

> `mystere(x,b)| renvoie la plus grande valeur de` $k \in \mathbb{N}$ `telle que` $b^k \leqslant x$ `ie telle que` $b^k \leqslant x < b^{k+1}$.
>
> `On a donc` $\text{mystere}(x,b) = \begin{cases} \lfloor \frac{\ln(x)}{\ln(b)} \rfloor = \lfloor \log_b(x) \rfloor & \texttt{si } x \geqslant b \\ 0 & \texttt{sinon} \end{cases}$

20. On donne le code suivant :

```python
pas=1e-5

x2 = 0
for i in range(100000):
    x1 = (i + 1) * pas
    x2 = x2 + pas

print("x1:", x1)
print("x2:", x2)
```

L'exécution de ce code produit le résultat :

```
x1: 1.0
x2: 0.9999999999980838
```

Commenter brièvement.

> *- Au dernier tour de boucle,* x_1 *contient* $10^5 * 10^{-5} = 1$
> *- De même,* x_2 *contient* $\sum_{k=0}^{10^5-1} 10^{-5}$ *et n'affiche pas exactement 1 à cause des erreurs d'arrondis qui s'accumulent dans le calcul de la somme.*

Partie II. Génération de nombres premiers

Le crible d'Ératosthène est un algorithme qui permet de déterminer la liste des nombres premiers appartenant à l'intervalle $1n$. Son pseudo-code s'écrit comme suit :

liste_bool ← *liste de N booléens initialisés à Vrai;*
Marquer comme Faux le premier élément de liste_bool;
Pour *l'entier* $i \leftarrow 2$ **à** $\lfloor\sqrt{N}\rfloor$
  Si *i n'est pas marqué comme Faux dans liste_bool*
  Marquer comme Faux tous les multiples de i différents de i dans liste_bool ;
retourner *liste_bool*

À la fin de l'exécution, si un élément de `liste_bool` vaut Vrai alors le nombre codé par l'indice considéré est premier. Par exemple pour N=4 une implémentation Python du crible renvoie [False True True False].

21. Sachant que le langage `Python` traite les listes de booléens comme une liste d'éléments de 32 bits, quel est (approximativement) la valeur maximale de `N` pour laquelle `liste_bool` est stockable dans une mémoire vive de 4 Go ?

Un Go, c'est 10^9 *octets soit* 8.10^9 *bits donc 4 go représentent* 32.10^9 *bits. Ainsi si chaque booléen est codé sur 32 bits, cela fait* $N = \frac{32.10^9}{32} = 10^9$ *éléments.*

22. Quel facteur peut-on gagner sur la valeur maximale de `N` en utilisant une bibliothèque permettant de coder les booléens non pas sur 32 bits mais dans le plus petit espace mémoire possible pour ce type de données (on demande de le préciser) ?

Les booléens peuvent être codés sur un seul bit (soit 0 soit 1). On peut dans ce cas travailler avec 32 fois plus d'éléments.
Si on code les booléens sur 1 bit alors la valeur maximale de N est 32.10^9

23. Écrire la fonction `erato_iter(N)` qui implémente l'algorithme ci-dessus pour un paramètre `N` qui est un entier supérieur ou égal à 1.

```python
def erato_iter(N):
    liste_bool = N * [True]
    liste_bool[0] = False
    i=2
    while i**2 <= N:
        if liste_bool[i - 1]:
            for k in range(2, N//i + 1):
                liste_bool[k*i - 1] = False
        i += 1
    return liste_bool
```

24. Quelle est la complexité algorithmique du crible d'Ératosthène en fonction de N ? On admettra que :

$$\sum_{\substack{p<N \\ p\ premier}} \frac{1}{p} \simeq \ln(\ln(N)) \qquad (1)$$

La réponse devra être justifiée.

Comptons les affectations.
On remarque que pour chaque i *tel que* $list\_bool[i]$ *vaut* $True$*, autrement dit tel que* i *est premier, il y a* $\left\lfloor \frac{N}{i} \right\rfloor$ *affectations soit approximativement* $\frac{N}{i}$.
Au total, il y en a $\sum_{\substack{p premier \\ p\leqslant\sqrt{N}}} \frac{N}{p} \underset{N}{\sim} \ln(\ ln(\sqrt{N})) = O(N\ln(ln((N)))$.

25. Quand on traite des nombres entiers il est intéressant d'exprimer la complexité d'un algorithme non pas en fonction de la valeur N du nombre traité mais de son nombre de chiffres n. Donner une approximation du résultat de la question précédente en fonction de n en précisant la base choisie.

Si n est le nombre de chiffres de N, on a $10^{n-1} \leqslant N \leqslant 10^n$, on a donc

$$\underbrace{10^{n-1}\ln(\ln(10^{n-1}))}_{=O(n10^n)} \leqslant N\ln(\ln(N)) \leqslant \underbrace{10^n \ln(\ln(10^n))}_{=O(n10^n)}.$$

La complexité de l'algorithme en fonction du nombre de chiffres n de N est $\boxed{O(n10^n)}$