Il est attendu des réponses construites. La notation finale tiendra compte de la précision, du soin et de la clarté des programmes ainsi que de la rédaction en général pour les questions plus théoriques.
Les programmes Python devront être indentés correctement et les variables utilisées devront être sans ambiguités.
Bon travail !

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` :

```
1   def mystere(li1:list,li2:list)->list:
2       li1.append(float('infinity'))
3       li2.append(float('infinity'))
4       li=[]
5       i,j=0,0
6       while len(li)<len(li1)+len(li2):
7           if li1[i]<=li2[j]:
8               li.append(li1[i])
9               i+=1
10          else:
11              li.append(li2[j])
12              j+=1
13      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`.

	Valeur de Li	i	j
entrée 1er tour			
sortie 1er tour			
sortie 2^e tour			
sortie 3^e tour			
$\vdots$			

2. Quelle est l'opération réalisée par la fonction précédente ? En utilisant un variant de boucle, montrer que la terminaison du programme.
3. Proposer une version récursive `mystere_rec(li1:list,li2:list) -> list` de cette fonction.
4. Établir la preuve de correction de la fonction récursive `mystere_rec`.

5. On appelle p et q les longueurs respectives de `li1` et `li2`. Montrer 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.
6. Écrire une fonction de tri récursif permettant de trier une liste quelconque et qui utilise l'opération réalisée à la question 2.

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.
8. Quel est le nom de ce tri ? Justifier en expliquant en détail la nature de l'opération réalisée par la boucle ligne 6.
9. Proposer une version récursive `inserer_rec(x:float, L:list)` permettant de réaliser l'opération réalisée dans la boucle ligne 6 par récursivité.
10. Montrer que la fonction « `inserer_rec(x,L)` insère `x` dans la liste triée `L` » est un invariant de boucle. En déduire la preuve de correction de la fonction `inserer_rec`.
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 ».
12. En déduire la preuve de correction de la fonction `tri`.
13. Quelle est la complexité temporelle de cet algorithme dans le pire des cas (à définir) ?
14. Quelle est la complexité temporelle de cet algorithme dans le meilleur des cas (à définir) ?
15. Donner le nom et la complexité temporelle d'un algorithme de tri de meilleure complexité temporelle dans le pire des cas.

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.

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 (appelées tolérance absolue et tolérance relative), à définir dans le corps de la fonction, valant respectivement 10^{-5} et 10^{-8}. Les paramètres `x` et `y` sont des nombres quelconques.

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.

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

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

20. On donne le code suivant :

```
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.

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 $[\![1, n]\!]$. 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* ← 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, quelle est (approximativement) la valeur maximale de `N` pour laquelle `liste_bool` est stockable dans une mémoire vive de 4 Go ?

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) ?

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.

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

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

La réponse devra être justifiée.

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. Montrer que la complexité est un $O(N\ln(\ln(N)))$.