NOM : **Prénom :** **Classe :**

Devoir Surveillé 2

Les réponses sont à écrire exclusivement sur ce document

Les différentes questions sont indépendantes les unes des autres.

1 Un problème de rendu de monnaie

On s'intéresse ici au nombre de manière différentes de rendre une somme n en fonction d'une liste de types de pièces disponible stockés dans une liste L. On considérera que l'on dispose de suffisamment de pièces de chaque type pour rendre la somme n.

On va écrire une fonction `pieces(n,L)` qui va renvoyer le nombre de manières différentes d'obtenir n avec des pièces ayant des valeurs dans L.

Question 1. Montrer que `pieces(3,[1,2])` vaut 2
Que vaut `pieces(2,[0.5,1])` ?

Question 2. Compléter le programme suivant :

```
def pieces(n,L):
    if n==0:
        return
    if L=[] or n<0:
        return
    else:
        M=L[0]
        return
```

Question 3. Montrer que ce programme se termine.

2 Transformations de listes récursives

Question 4. Écrire une fonction récursive `swap2` qui prend en argument une liste `L` et qui renvoie cette liste où l'ordre des éléments est modifié de sorte à ce que cette liste contienne : le second élément de L puis le premier, puis le quatrième, puis le troisième, puis le sixième, puis le cinquième etc...

Ainsi `swap2([1,4,9,16,25,36,49])` renverra `[4,1,16,9,36,25,49]`

Question 5. Réécrire cette fonction dans un cas plus général : écrire une fonction récursive `swap` qui prend en arguments une liste `L` et un entier strictement positif `k` qui renvoie la liste constituée du $k^{\text{ème}}$ élément de L, puis du $k-1^{\text{ème}}$, ..., puis du premier, puis du $2k^{\text{ème}}$ ... du $k+1^{\text{ème}}$ etc...

Ainsi `swap([1,4,9,16,25,36,49],3)` renverra `[9,4,1,36,25,16,49]`

Question 6. On considère dans cette question que l'affectation ou l'ajout d'un élément à une liste a un coût de 1, et qu'un slicing de liste a un coût égal à la longueur de la liste slicée : ainsi `L[a:b]` a un coût de $b-a$ et `L[:]` a un coût de n=`len(L)`

Exprimer et justifier la complexité (asymptotique) de l'algorithme écrit précédemment en fonction de n et de k. Si besoin, on pourra considérer que n est grand devant k.

3 Question de cours : tri par insertion

Question 7. Écrire une fonction `tri_insertion(L)` qui prend en argument une liste de nombres `L` et la trie dans l'ordre croissant en utilisant l'algorithme du tri par insertion.

Question 8. Donner la complexité dans le pire et le meilleur des cas du tri par insertion en fonction de `n` la longueur de la liste `L` à trier.

Il n'est pas demandé de justifier votre réponse.

4 Tri à bulles

Le tri à bulles consiste à effectuer dans un premier temps $n-1$ comparaisons et les échanges correspondants nécessaires entre éléments consécutifs permettant de placer le plus grand élément d'une liste de longueur n à trier en queue de celle-ci. Si un échange a été effectué, un booléen est mis à vrai ; sinon, il lui est affecté la valeur « faux ».

Tant que ce booléen est vrai, on recommence cet algorithme en effectuant $n-2$ comparaisons et échanges éventuels et en plaçant le plus grand élément restant en avant-dernière position et ainsi de suite avec $n-3$ comparaisons et échanges éventuels etc...

Question 9. Écrire une fonction `tri_bulle(L)` qui trie dans l'ordre croissant une liste `L` comme décrit précédemment.

Question 10. Montrer que cet algorithme s'arrête.

Question 11. Quelle est la complexité de cet algorithme en fonction de n, où n est la taille de `t`, dans le pire des cas ? Justifier la réponse.

Question 12. Quelle est la complexité de cet algorithme en fonction de n, où n est la taille de `t`, dans le meilleur des cas ? Quelles sont les listes correspondantes ?

Question 13. Montrer que cet algorithme trie effectivement `L` par ordre croissant. On utilisera un invariant de boucle et on remarquera qu'après k passages dans la boucle principale du programme, les k derniers éléments sont les plus grands de la liste `L` et sont triés dans l'ordre croissant.