Exercice 1 : Questions diverses

1. Voici cette fonction classique :

```
def mini(L):
    """renvoie le minimum et un indice de ce minimum"""
    if L == []:
        return(None) # si la liste est vide, on ne renvoie rien
    else:
        indice = 0
        mini = L[0] # initialisation de l'indice et du minimum
        n = len(L)
        for i in range(1, n):
            a = L[i]
            if a < mini: # si l'on trouve un élément plus petit, c'est le nouveau minimum
                indice = i
                mini = a
    return((mini, indice))
```

2. Comme nous l'avons vu :

```
def comptage(L):
    """fonction qui renvoie un dictionnaire permettant de connaitre le nombre d'occurrences des éléments
     de la liste L"""
    d = {} # création d'un dictionnaire vide
    for i in L:
        if i in d:
            d[i] += 1 # si l'élément est déjà une clé, on ajoute 1
        else:
            d[i] = 1 # sinon on crée l'entrée correspondante dans le dictionnaire
    return(d)
```

3. On reprend l'algorithme déjà étudié :

```
def tri_bulles (L):
    """ordonne la liste L dans l'ordre croissant en utilisant le tri bulle"""
    n =len(L) # la longueur de la liste
    trier = False # cette variable prend la valeur True lorsque l'on fait un parcours sans échange
    while trier == False:
        trier = True # pour l'instant, on n'a pas fait d'échange
        for i in range(n - 1): # on parcourt la liste
            if L[i] > L[i + 1]:
                L[i], L[i +1] = L[i + 1], L[i] # on échange les paires mal rangées
                trier = False # on indique que la liste n'est pas triée car on a fait un échange
    return(L) # quand on sort de la boucle, la liste est triée
```

D'après le cours, on sait que la complexité du tri bulle est $O(n^2)$ où n est la longueur de la liste.

4. De façon classique, le plus simple est de faire :

```
import matplotlib.pyplot as plt
import numpy as np

X = np.linspace(-7, 3, 100000)
Y = np.sqrt(np.sin(X ** 2 - X + 1) + 4)
plt.plot(X, Y)
plt.show()
```

5. (a) En faisant quelques essais, il apparait clairement que cette fonction renvoie 2^n.

   (b) La quantité $n-i$ est un variant de boucle. En effet, $n-i$ est un entier naturel qui décroit strictement à chaque passage dans la boucle puisque n ne varie pas et i augmente de 1. La boucle *while* se termine.

   (c) Prenons comme invariant de boucle la propriété définie pour $i \in [\![0, n]\!]$:

$$\mathcal{H}_i \ : \ "p = 2^i"$$

   • Initialement, on a $p = 1$ et $i = 0$ donc avant de rentrer dans la boucle, il est vrai que $p = 2^i$.

   • On suppose que $\mathcal{H}_i$ est vraie pour $i \in [\![0, n-2]\!]$ fixé. On a donc $p = 2^i$. Lors du passage numéro $i+1$ dans la boucle, on a p qui devient $2p$ et i qui augmente de 1, en notant p' et i' ces deux nouvelles valeurs, on a bien :

$$p' = 2p = 2 \times 2^i = 2^{i+1} = 2^{i'}$$

   Ce qui montre que $\mathcal{H}_{i+1}$ est vraie également.

   • Enfin, lorsque l'on sort de la boucle, on a $i = n$ et il est toujours vrai que $p = 2^i$ donc $p = 2^n$ comme voulu.

Exercice 2 : Tri des crêpes

1. C'est une recherche de maximum classique, on remarque que la i-ième crêpe a pour indice $i-1$.

```
def grande(L, i):
    """renvoie l'indice de la plus grande crêpes parmi les i premières"""
    valmax = L[0] # initialisation de la valeur maximale
    imax = 0 # indice du max
    for k in range(1, i): # on examine les i premières crêpes
        if L[k] > valmax: # si l'on en trouve une plus grande, on change l'indice
            valmax = L[k]
            imax = k
    return(imax)
```

2. On propose la fonction suivante :

```
def retourner(L, i):
    """retourne la pile de crêpes située au-dessus de la place i"""
    for k in range(i // 2 + 1): # on échange la crêpe à la place k avec la crêpe i - k
        L[k], L[i - k] = L[i - k], L[k]
    return(L)
```

3. (a) La méthode que l'on devine grâce à l'exemple est la suivante, on place la spatule sous la crêpe la plus grande qui sera placée ainsi sur le sommet de la pile. On place ensuite la spatule en-dessous du tas afin que la crêpe la plus grande finisse en bas, comme voulu. On réitère le procédé en triant à présent le tas restant, c'est-à-dire en ignorant la crêpe la plus grande.

   (b) Voici la traduction en Python :

```
def tricrepes(L):
    """réalise le tri des crêpes de la liste L"""
    n = len(L)
    for k in range(n, 1, -1): # pour diminuer la taille du tas trié au fur et à mesure
        g = grande(L, k) # la plus grande parmi les non triées
        L = retourner(L, g) # on la place au-dessus
        L = retourner(L, k - 1) # on la place en-dessous des non triées
    return(L)
```

   (c) Pour chacune des n crêpes, il y a 2 étapes pour la placer correctement. Cependant lorsque que l'on a déjà trié $n-2$ crêpes en $2n-4$ étapes, il reste les 2 crêpes du dessus de la pile à trier. Deux cas se présentent, soit les 2 crêpes sont bien placées et dans ce cas, il n'y a rien à faire. Soit elles sont dans le mauvais ordre et dans ce cas un seul retournement suffit. On a justifié que le nombre maximal d'étapes pour trier n crêpes est $2n-3$.

4. Lorsque l'on cherche la crêpe la plus grande de la pile, il y a $n-1$ comparaisons (dans la fonction *grande*). Puis, on cherche la crêpe la plus grande parmi celles restantes : $n-2$ comparaisons. On poursuit le procédé, il y a au total :

$$\sum_{k=1}^{n-1} k = \frac{(n-1)n}{2} \text{ comparaisons}$$

   La complexité de ce tri très spécial est en $O(n^2)$.

5. Il s'agit d'adapter le principe vu précédemment. Lorsque la crêpe que l'on veut bien placer est sur le dessus de la pile, on peut faire une étape en plus pour la retourner si la face cramée est visible.

   Il semble judicieux d'ajouter un signe à la taille des crêpes. Par exemple, -5 signifie que la crêpe de taille 5 a son côté cramé face visible.

   Il n'y a pas grand chose à modifier dans les fonctions :

```
def grande2(L, i):
    """renvoie l'indice de la plus grande crêpes parmi les i premières"""
    # on compare cette fois-ci les valeurs absolues
    valmax = abs(L[0]) # initialisation de la valeur maximale
    imax = 0 # indice du max
    for k in range(1, i): # on examine les i premières crêpes
        if abs(L[k]) > valmax: # si l'on en trouve une plus grande, on change l'indice
            valmax = abs(L[k])
            imax = k
    return(imax)


def retourner2(L, i):
    """retourne la pile de crêpes située au-dessus de la place i"""
    for k in range(i // 2 + 1): # on échange la crêpe à la place k avec la crêpe i - k
        L[k], L[i - k] = L[i - k], L[k]
    return(L)

def tricrepes2(L):
    """ réalise le tri des crêpes de la liste L"""
    n = len(L)
    for k in range(n, 1, -1): # pour diminuer la taille du tas trié au fur et à mesure
        g = grande2(L, k) # la plus grande parmi les non triées
        L = retourner2(L, g) # on la place au-dessus
        if L[0] < 0: # si la crêpe est mal orientée, on la retourne
            L[0] = -L[0]
        L = retourner2(L, k - 1) # on la place en-dessous des non triées
    return(L)
```