Exercice 8. Tri de crêpes (DS de l'année 2021-2022)

Question 1.a –

```
def maxi(L, i):
    j = i
    for k in range(i+1, len(L)):
        if L[k] > L[j]:
            j = k
    return j
```

Question 1.b –

```
def spatule(L, i):
    taille_sous_liste = len(L) - i
    for k in range(taille_sous_liste // 2):
        k1 = i + k
        k2 = len(L) - 1 - k
        tmp = L[k1]
        L[k1] = L[k2]
        L[k2] = tmp
```

Question 2.a – On va faire une sorte de tri par sélection : on place la plus grande crêpe à l'indice 0, puis la deuxième plus grande à l'indice 1, et ainsi de suite. Pour cela, on distingue plusieurs cas :

- Si $n = 0$ ou $n = 1$, il n'y a rien à faire.
- Si $n \geqslant 2$, on commence par repérer l'indice j de la crêpe avec le plus grand diamètre (opération 1). Avec un coup de spatule à l'indice j (opération 2), on place la crêpe sur le dessus de la pile. Avec un deuxième coup de spatule à l'indice 0 (opération 2), on place la crêpe en dessous de la pile. Il suffit alors de recommencer récursivement en ne considérant que les $n-1$ crêpes sur le dessus de la pile (on ne touche plus à la crêpe qui se trouve en dessous).

  Remarque : pour diminuer le nombre d'appels à la fonction `spatule`, on pourrait faire des cas spéciaux lorsque $j = n-1$ (un coup de spatule suffit) et lorsque $j = 0$ (les appels à la fonction `spatule` ne sont pas utiles).

Question 2.b – Notons a_n et b_n le nombre maximal d'utilisations des opérations 1 et 2 pour une liste de taille n. On a :

$$a_0 = a_1 = 0, \qquad b_0 = b_1 = 0.$$

Lorsque $n \geqslant 2$, on utilise une fois l'opération 1, puis deux fois l'opération 2, puis on trie une pile de taille $(n-1)$. Pour tout $n \geqslant 2$, on a donc :

$$a_n = a_{n-1} + 1 \qquad b_n = b_{n-1} + 2$$

En résumé :

$$\begin{array}{lll} a_0 = 0, & \text{et} & a_n = n-1 \text{ pour tout } n \geqslant 1 \\ b_0 = 0, & \text{et} & b_n = 2n-2 \text{ pour tout } n \geqslant 1 \end{array}$$

Question 3.a – Notez que la description faite dans la question 2.a correspond à une fonction récursive alors que le code ci-dessus est une fonction itérative. Il faut donc adapter la procédure en conséquence.

```
def tri_crepes(L):
    for i in range(len(L)-1):
        j = maxi(L, i)
        spatule(L, j)
        spatule(L, i)
```

Question 3.b – Les fonctions `maxi` et `spatule` s'exécutent en temps linéaire en `len(L)`. Dans la fonction `tri_crepes`, il y a un nombre linéaire de tours de boucle et chaque tour de boucle s'exécute en temps linéaire. Finalement, la complexité est quadratique en `len(L)`.

Question 4 –

```
def tri_crepes_bis(L):
    T = L[:]
    tri_crepes(T)
    return T
```

Question 5 – On va générer toutes les permutations de la liste `L = [n-1, n-2, ..., 0]` et déterminer le nombre minimal d'appels à la fonction `spatule` pour trier chaque permutation. Pour cela on va procéder à l'envers : on part de `L`, on génère toutes les listes obtenues avec un appel à la fonction `spatule` (il y en a `n`), puis on recommence avec ces nouvelles listes jusqu'à ce que toutes les permutations aient été atteintes.

Le nombre d'appels à la fonction `spatule` nécessaires pour atteindre une permutation `P` sera stocké dans un dictionnaire. Étant donné que les clés d'un dictionnaire ne peuvent pas être des listes, on utilise `tuple(P)` comme clé.

```
def make_dict_nb_etapes(n):
    """
    n: int
    Return: int, dict[tuple, int]
    ----
    Renvoie le nombre maximal d'étapes ainsi qu'un dictionnaire qui associe à
    chaque liste L le nombre d'étapes nécessaires pour obtenir L.
    """
    L = [i for i in range(n, 0, -1)]
    d = {}
    etape = 0
    d[tuple(L)] = etape
    derniers_vus = [L[:]]
    while derniers_vus != []:
        etape += 1
        nouveaux_vus = []
        for M in derniers_vus:
            for i in range(n):
                N = M[:]
                spatule(N, i)
                Nt = tuple(N)
                if Nt not in d:
                    d[Nt] = etape
                    nouveaux_vus.append(N)
        derniers_vus = nouveaux_vus
    return etape-1, d
```

```
def nb_max_op2(n):
    n, _ = make_dict_nb_etapes(n)
    return n
```