DS 5 d'informatique - MPSI - Algorithmes de tri - Correction

Question 1.a –

```
import random
import time
```

Question 1.b –

```
from math import floor
```

Question 1.c –

```
import matplotlib.pyplot as plt
```

Question 2.a – La signature de la fonction `phi` est :

```
phi(L: list[int]) -> list[int]
```

Question 2.b – Dans la fonction `phi`, on a :

→ Une duplication de liste (ligne 2) dans laquelle une liste de taille 101 est créée.

→ Une boucle `for` ligne 3 qui fait `len(L)` tours.

Comme le nombre de tours de boucle est fini et que toutes les opérations utilisées sont élémentaires :

La fonction `phi` termine pour toute entrée `L`.

Question 2.c – Toutes les lignes de la fonction s'exécutent en temps constant et la boucle `for` de la ligne 3 fait `len(L)` tours. Ainsi :

La complexité de la fonction `phi` est en $\mathcal{O}(n)$ avec `n = len(L)`.

Question 3 – Soit `n = len(L)` et numérotons les tours de la boucle `for` de `i = 0` à `i = n - 1`. Pour chaque `i` $\in [\![0, n-1]\!]$, on note C_i la valeur de `C` à la fin du tour de boucle `i` et on définit l'invariant de boucle :

$$(\mathcal{P}_i) : C_i = \varphi\Big(L[0 : i+1]\Big)$$

Montrons $(\mathcal{P}_i)$ par récurrence sur `i` $\in [\![0, n-1]\!]$

Initialisation. Pour `i = 0`, on s'intéresse à C_0 qui est la valeur de `C` à la fin du premier tour de boucle. D'après les lignes 2, 3, 4, la liste C_0 contient uniquement des 0, sauf la case d'indice `L[0]` qui contient un 1. On a bien $(\mathcal{P}_0)$:

$$C_0 = \varphi\Big([L[0]]\Big) = \varphi\Big(L[0 : 1]\Big)$$

Hérédité. Soit `i` $\in [\![0, n-2]\!]$. On suppose $(\mathcal{P}_{i-1})$ et on montre $(\mathcal{P}_i)$. On remarque que :

→ Par hypothèse de récurrence :

$$C_{i-1} = \varphi\Big(L[0 : i]\Big)$$

→ D'après la ligne 4 du programme, la liste C_i contient les mêmes éléments que C_{i-1} sauf l'élément d'indice `L[i]` auquel on a ajouté 1.

→ La liste $\varphi\Big(L[0 : i+1]\Big)$ est égale à $\varphi\Big(L[0 : i] + [L[i]]\Big)$, c'est à dire qu'elle contient les mêmes éléments que $\varphi\Big(L[0 : i]\Big)$ sauf l'élément d'indice `L[i]` auquel on a ajouté 1.

Finalement, on a bien :

$$(\mathcal{P}_i) : C_i = \varphi\Big(L[0 : i+1]\Big)$$

Conclusion. Si la liste `L` est vide, alors `phi` renvoie une liste de taille 101 composée uniquement de zéros ; ce qui est correct. Sinon, on se place à la fin du dernier tour de boucle. D'après $(\mathcal{P}_{n-1})$:

$$C_{n-1} = \varphi\Big(L[0 : n]\Big) = \varphi(L)$$

En d'autres termes :

L'appel à `phi(L)` renvoie φ`(L)`.

Question 4.a –

```
1 def triComptage(L):
2     C = phi(L)
3     M = []
4     for i in range(len(C)):
5         for _ in range(C[i]):
6             M.append(i)
7     return M
```

Question 4.b – D'après la question 2.b, l'appel à la fonction `phi` de la ligne 2 termine. De plus, toutes les autres opérations sont élémentaires et la boucle `for` de la ligne 4 fait 101 tours. Pour la boucle `for` de la ligne 5, le résultat de la question 3 stipule que `C` = φ`(L)`, ce qui implique que tous les éléments de `C` sont inférieurs ou égaux à `len(L)`. Ainsi, pour un `i` fixé, la boucle `for` de la ligne 5 fait moins de `len(L)` tours. En conclusion :

L'appel à `triComptage` termine pour toute entrée `L`.

Question 4.c – Soit `n = len(L)`. D'après la question 2.c, la ligne 2 s'exécute en temps $\mathcal{O}$`(n)`. De plus, toutes les autres opérations s'exécutent en temps constant. D'après la question 3, pour chaque `i` $\in [\![$`0, 100`$]\!]$, `C[i]` est le nombre d'occurrences de `i` dans `L`. On en déduit que la ligne 6 est exécutée exactement une fois pour chaque élément de `L`. En conclusion :

La complexité de la fonction `triComptage` est en $\mathcal{O}$`(n)` avec `n = len(L)`.

Question 5.a –

```
def intToBin(n):
    assert n >= 0
    s = ""
    while n > 0:
        if n % 2 == 0:
            s += "0"
        else:
            s += "1"
        n = n // 2
    return s
```

Question 5.b –

```
def listToBin(L):
    M = []
    for n in L:
        M.append(intToBin(n))
    return M
```

Question 6.a –

```
def tailleMax(M):
    maxi = -1
    for s in M:
        if len(s) > maxi:
            maxi = len(s)
    return maxi
```

Question 6.b –

```
def completer(M):
    m = tailleMax(M)
    N = []
    for s in M:
        N.append(s + "0" * (m - len(s)))
    return N
```

Question 7.a –

```
def separerBit(N, i):
    N0 = []
    N1 = []
    for s in N:
        if s[i] == "0":
            N0.append(s)
        elif s[i] == "1":
            N1.append(s)
    return N0, N1
```

Question 7.b – Soit `m` $\in \mathbb{N}$ la taille des éléments de `N`. Pour tout élément `s` de `N`, `s[0]` correspond donc au bit de poids faible et `s[m-1]` correspond au bit de poids fort.

Solution 1 (version récursive). L'idée est de séparer les éléments de `N` par rapport au bit numéro `m-1` avec la fonction `separerBit`. On obtient alors deux listes qu'on trie récursivement en utilisant uniquement les bits `0` à `m-2`.

```
def triBinStr_rec0(N, i):
    if i == -1:
        return N[:]
    N1, N2 = separerBit(N, i)
    return triBinStr_rec0(N1, i-1) + triBinStr_rec0(N2, i-1)
```

```
def triBinStr_rec(N):
    if len(N) == 0:
        return []
    else:
        return triBinStr_rec0(N, len(N[0]) - 1)
```

Solution 2 (version itérative). On utilise la fonction `separerBit` pour trier la liste en regardant d'abord le bit numéro `0`, puis on trie le résultat en regardant le bit numéro `1`, puis on trie le résultat en regardant le bit numéro `2`, ..., jusqu'au bit numéro `m-1`. Étant donné que le dernier tri se fait via le bit de poids fort, c'est bien ce bit qui aura le plus d'influence sur le résultat final.

```
def triBinStr_it(N):
    if len(N) == 0:
        return []
    for i in range(len(N[0])):
        N0, N1 = separerBit(N, i)
        N = N0 + N1
    return N
```

Question 8.a –

```
def binToInt(s):
    n = 0
    for i in range(len(s)):
        if s[i] == "1":
            n += 2**i
    return n
```

Question 8.b –

```
def triBin(L):
    M = listToBin(L)
    N = completer(M)
    S = triBinStr_it(N)
    T = []
    for s in S:
        T.append(binToInt(s))
    return T
```

Question 9 –

```
def peigne(L, p):
    assert p > 0
    b = False
    for i in range(len(L) - p):
        if L[i] > L[i+p]:
            L[i], L[i+p] = L[i+p], L[i]
            b = True
    return b
```

Question 10.a – Comme $p_0 = n$, appeler `peigne` avec p_0 revient à appeler `peigne(L, len(L))`. La boucle `for` de cette fonction fait alors 0 tour et `L` n'est pas modifiée. En conclusion :

Appeler la fonction `peigne` avec p_0 n'a pas d'intérêt car cet appel ne modifie pas `L`.

Question 10.b – Lors d'un appel à `peigne(L, 1)`, la fonction parcourt les indices $\texttt{i} \in [\![0, \texttt{len(L)} - 2]\!]$ en comparant `L[i]` avec `L[i+1]`. Cet appel renvoie `False` si et seulement si la ligne « `b = True` » n'est jamais exécutée, c'est à dire si et seulement si :

$$\forall \texttt{i} \in [\![0, \texttt{len(L)} - 2]\!] : \texttt{L[i]} \leqslant \texttt{L[i + 1]}.$$

En d'autres termes, c'est équivalent au fait que la liste soit triée.

Question 10.c –

```
def triPeigne(L, R):
    p = len(L)
    b = True
    while p > 1 or b:
        p = max(1, floor(p / R))
        b = peigne(L, p)
```

Question 11.a – Le type de retour de `triPeigne` est `NoneType`, ce qui signifie que le programme de l'énoncé affiche uniquement `None`. Ainsi, exécuter ce programme ne permet pas de savoir si la fonction a trié correctement la liste.

Question 11.b –

```
L = [1, 5, 2, -5]
triPeigne(L, 1.05062025)
print(L)
```

Question 12 –

```
def listeAlea():
    L = []
    for _ in range(random.randint(500, 1000)):
        L.append(random.randint(-1000, 1000))
    return L
```

Question 13 –

```
def chrono(R):
    T = 0
    for _ in range(800):
        L = listeAlea()
        t_ini = time.perf_counter()
        triPeigne(L, R)
        T += time.perf_counter() - t_ini
    return T
```

Question 14 –

```
def traceChrono(a, b, m):
    pas = (b-a)/(m-1)
    X = [a + i*pas for i in range(m)]
    Y = []
    for R in X:
        Y.append(chrono(R))
    plt.figure()
    plt.plot(X, Y, "o", color = "black")
    plt.show()
```

Question 15 – D'après les courbes présentées dans l'énoncé, il semble pertinent d'utiliser un facteur de réduction d'environ 1.3. Notons cependant que rien ne garantit que ce facteur de réduction soit intéressant dans le cas où les listes sont beaucoup plus grandes ou beaucoup plus petites.

Question 16.a –

```
def indiceMin(P):
    i_min = None
    for i in range(len(P)):
        if P[i] != None:
            if i_min == None or P[i_min] > P[i]:
                i_min = i
    return i_min
```

Question 16.b –

La fonction `bornes` renvoie un couple (x, y) où x est le minimum de `L` et y est son maximum.

En effet, par définition de `indiceMin`, l'entier `i` est l'indice du minimum de `L` et `j` est l'indice du minimum de `M`. Comme `M` est obtenue à partir de `L` en remplaçant chaque élément par son opposé, le minimum de `M` est égal au maximum de `L`. D'où le résultat.

Question 17.a –

```
def copie(L):
    return L[:]
```

Question 17.b –

La fonction `copie` s'exécute en temps $\mathcal{O}$(`n`) où `n = len(L)`

Question 18.a – Pour tout $i \in [\![0, m-2]\!]$:

$$\begin{aligned} x \in I_i &\Leftrightarrow a + i\ell \leqslant x < a + (i+1)\ell \\ &\Leftrightarrow i \leqslant \frac{x-a}{\ell} < i+1 \\ &\Leftrightarrow i \leqslant m\frac{x-a}{b-a} < i+1 \\ &\Leftrightarrow i = \left\lfloor m\frac{x-a}{b-a} \right\rfloor \end{aligned}$$

De plus, pour $i = m-1$:

$$\begin{aligned} x \in I_i &\Leftrightarrow \left[x \in [\, a + i\ell \,;\, a + (i+1)\ell \,[\right] \text{ ou } \left[x = a + (i+1)\ell\right] \\ &\Leftrightarrow \left[i = \left\lfloor m\frac{x-a}{b-a} \right\rfloor\right] \text{ ou } \left[m\frac{x-a}{b-a} = i+1\right] \\ &\Leftrightarrow \left[i = \left\lfloor m\frac{x-a}{b-a} \right\rfloor\right] \text{ ou } \left[m = \left\lfloor m\frac{x-a}{b-a} \right\rfloor\right] \end{aligned}$$

En résumé, pour déterminer l'indice i demandé dans l'énoncé :

On calcule $i_0 = \left\lfloor m\frac{x-a}{b-a} \right\rfloor$, puis $\begin{cases} \text{Si } i_0 \in [\![0, m-1]\!], \text{ alors } i = i_0. \\ \text{Si } i_0 = m, \text{ alors } i = m-1. \end{cases}$

Question 18.b –

```
def partition(L, m):
    a,b = bornes(L)
    P = [[] for _ in range(m)]
    for x in L:
        i = floor((x-a)*m/(b-a))
        if i == m:
            i = m-1
        P[i].append(x)
    return P
```

Question 19 –

```
def triPaquets(L, m):
    # Liste vide
    if L == []:
        return copie(L)
    # Listes dont tous les éléments sont égaux
    a,b = bornes(L)
    if a == b:
        return copie(L)
    # Listes avec au moins deux éléments distincts
    P = partition(L, m)
    T = []
    for i in range(m):
        for _ in range(len(P[i])):
            k = indiceMin(P[i])
            T.append(P[i][k])
            P[i][k] = None
    return T
```