Correction du DS1

Problème I : Partie I

1. Aucune interdiction d'interruption d'une boucle `for` :

```
def admet_point_fixe(L) :
    for i in range(len(L)) :
        if L[i] == i :
            return True
    return False
```

2. Inutile de tester la présence d'un point fixe avant, la fonction renvoie 0 dans ce cas :

```
def nb_points_fixes(L) :
    n = 0
    for i in range(len(L)) :
        if L[i] == i :
            n += 1
    return n
```

3. Version itérative :

```
def itere(L,x,k) :
    r = x
    for _ in range(k) :
        r = L[r]
    return r
```

Version récursive :

```
def itere2(L,x,k) :
    if k == 0 :
        return x
    else :
        return itere2(L,L[x],k-1)
```

4. `L` est supposée admettre un point fixe donc inutile de le vérifier avant (cette fonction ne renvoie rien dans le cas d'une liste sans point fixe) :

```
def point_fixe(L) :
    for i in range(len(L)) :
        if L[i] == i :
            return i
```

5. On commence par vérifier que f admet un point fixe z. Le principal problème est ensuite que l'on ne pourra pas tester toutes les valeurs de k (par exemple pour montrer que z n'est pas un attracteur principal) ; en fait il suffit de tester les entiers $1 \leqslant k \leqslant n$: si $\forall k \leqslant n, f^k(x) \neq z$ alors les n valeurs de $f^k(x)$, $1 \leqslant k \leqslant n$, sont toutes dans $E_n \setminus \{z\}$ qui est un ensemble à $n-1$ éléments ce qui impose qu'il existe $1 \leqslant i < j \leqslant n$ tels que $f^i(x) = f^j(x)$ (la suite $f^k(x)$ prend nécessairement deux fois la même valeur). On aura alors ensuite $f^{i+h}(x) = f^{j+h}(x)$ donc la suite $(f^k(x))_{k\in\mathbb{N}^*}$ est alors périodique à partir du rang i et ne prendra donc jamais la valeur z.

Il faut d'abord tester la présence d'un point fixe et en récupérer la valeur (il est forcément unique dans le cas d'un attracteur principal). On pourrait utiliser la fonction `admet_point_fixe` puis `point_fixe` mais pour gagner du temps, on peut utiliser seulement la fonction `point_fixe` qui renvoie `None` s'il n'y a pas de point fixe :

```
def admet_attracteur_principal(L) :
    fixe = point_fixe(L)
    if fixe == None :
        return False
    for x in L :
        test = (x == fixe)
        for k in range(1,len(L)+1) :
            if itere(L,x,k) == fixe :
                test = True
        if not test :
            return False
    return True
```

6. On se laisse guider :

```
def temps_cv(L) :
    fixe = point_fixe(L)
    temps = {fixe : 0}
    for x in range(len(L)) :
        if x != fixe and L[x] in temps : # ne pas changer la valeur pour le
            point fixe
            temps[x] = 1+temps[L[x]]
        else :
            h = 0
            y = x                    # on mémorise x pour ne pas perdre sa
                valeur
            while y not in temps :  # on cherche la première itération connue
                y = L[y]
                h += 1
            for j in range(h) :      # on remplit toutes les temps de toutes
                les itération rencontrées
                temps[x] = temps[y]+(h-j)
                x = L[x]
    return temps
```

7. Le temps de calcul de la fonction `temps_cv` car chaque élément de `L` n'est examiné qu'une seule fois (et les deux boucles `while` et `for` sont consécutives donc non imbriquées) et les tests d'appartenance avec un dictionnaire sont à coût constant ; reste à faire une recherche de maximum dont on sait qu'il est positif :

```
def temps_cv_max(L) :
    d = temps_cv(L)
    M = 0
    for c in d :
        if d[c] > M :
            M = d[c]
    return M
```

Partie II

8. On teste si `L[i]`$\leqslant$`L[i+1]` ; attention de ne pas dépasser de la liste :

```
def est_croissante(L) :
    for i in range(len(L)-1) :
        if L[i]>L[i+1] :
            return False
    return True
```

9. On fait une recherche dichotomique en s'assurant que `L[g]` $\geqslant$ `g` et `L[d]` $\leqslant$ `d` à chaque étape :

```
def point_fixe_croissant(L) :
    trouve = False
    g,d = 0,len(L)-1
    while not trouve and d-g>0 :
        m = (g+d)//2
        if L[m] == m :
            trouve = True
        elif L[m] > m :
            g = m+1
        else :
            d = m-1
    return m
```

10. La fonction termine bien à cause des ± 1 dans les réaffectations de `g` et `d`.
La recherche d'un point fixe sur une liste de longueur n effectue un premier test puis recommence sur une liste de longueur $\left\lfloor \frac{n}{2} \right\rfloor - 1$ (si `L[m]<m`) ou de longueur $n - \left\lfloor \frac{n}{2} \right\rfloor - 1$ (si `L[m]>m`). Comme $\left\lfloor \frac{n}{2} \right\rfloor - 1 \leqslant \frac{n}{2}$ et $n - \left\lfloor \frac{n}{2} \right\rfloor - 1 \leqslant \frac{n}{2}$, on en déduit que la complexité de la recherche pour une liste de longueur n vérifie $C(n) \leqslant 1 + C\left(\frac{n}{2}\right)$. Si on part de $2^p \leqslant n < 2^{p+1}$ avec $p = \lfloor \log_2(n) \rfloor$, on trouve (par croissance de C) $C(n) \leqslant p+1$ donc $\boxed{C(n) = O(\log(n))}$