EXERCICE 1 (6 points)

Cet exercice porte sur la programmation Python (listes, dictionnaires) et la méthode "diviser pour régner".

Cet exercice est composé de trois parties indépendantes.

Dans cet exercice, on s'intéresse à des algorithmes pour déterminer, s'il existe, l'élément absolument majoritaire d'une liste.

On dit qu'un élément est *absolument majoritaire* s'il apparaît dans strictement plus de la moitié des emplacements de la liste.

Par exemple, la liste `[1, 4, 1, 6, 1, 7, 2, 1, 1]` admet `1` comme élément absolument majoritaire, car il apparaît 5 fois sur 9 éléments. Par ailleurs, la liste `[1, 4, 6, 1, 7, 2, 1, 1]` n'admet pas d'élément absolument majoritaire, car celui qui est le plus fréquent est `1`, mais il n'apparaît que 4 fois sur 8, ce qui ne fait pas plus que la moitié.

1. Déterminer les effectifs possibles d'un élément absolument majoritaire dans une liste de taille 10.

Partie A : Calcul des effectifs de chaque élément sans dictionnaire

On peut déterminer l'éventuel élément absolument majoritaire d'une liste en calculant l'effectif de chacun de ses éléments.

2. Écrire une fonction `effectif` qui prend en paramètres une valeur `val` et une liste `lst` et qui renvoie le nombre d'apparitions de `val` dans `lst`. Il ne faut pas utiliser la méthode `count`.

3. Déterminer le nombre de comparaisons effectuées par l'appel `effectif(1, [1, 4, 1, 6, 1, 7, 2, 1, 1])`.

4. En utilisant la fonction `effectif` précédente, écrire une fonction `majo_abs1` qui prend en paramètre une liste `lst`, et qui renvoie son élément absolument majoritaire s'il existe et renvoie `None` sinon.

5. Déterminer le nombre de comparaisons effectuées par l'appel à `majo_abs1([1, 4, 1, 6, 1, 7, 2, 1, 1])`.

Partie B : Calcul des effectifs de chaque élément dans un dictionnaire

Un autre algorithme consiste à déterminer l'élément absolument majoritaire éventuel d'une liste en calculant l'effectif de tous ses éléments en stockant l'effectif partiel de chaque élément déjà rencontré dans un dictionnaire.

6. Recopier et compléter les lignes 3, 4, 5 et 7 de la fonction `eff_dico` suivante qui prend en paramètre une liste `lst` et qui renvoie un dictionnaire dont les clés sont les éléments de `lst` et les valeurs les effectifs de chacun de ces éléments dans `lst`.

```
1 def eff_dico(lst):
2     dico_sortie = {}
3     for ......... :
4         if ... in dico_sortie:
5             ...
6         else:
7             ...
8     return dico_sortie
```

7. En utilisant la fonction `eff_dico` précédente, écrire une fonction `majo_abs2` qui prend en paramètre une liste `lst`, et qui renvoie son élément absolument majoritaire s'il existe et renvoie `None` sinon.

Partie C : par la méthode "diviser pour régner"

Un dernier algorithme consiste à partager la liste en deux listes. Ensuite, il s'agit de déterminer les éventuels éléments absolument majoritaires de chacune des deux listes. Il suffit ensuite de combiner les résultats sur les deux listes afin d'obtenir, s'il existe, l'élément majoritaire de la liste initiale.

Les questions suivantes vont permettre de concevoir précisément l'algorithme.

On considère `lst` une liste de taille `n`.

8. Déterminer l'élément absolument majoritaire de `lst` si `n = 1`. *C'est le cas de base.*

On suppose que l'on a partagé `lst` en deux listes :

- `lst1 = lst[:n//2]` (`lst1` contient les `n//2` premiers éléments de `lst`)
- `lst2 = lst[n//2:]` (`lst1` contient les autres éléments de `lst`)

9. Si, ni `lst1` ni `lst2` n'admet d'élément absolument majoritaire, expliquer pourquoi `lst` n'admet pas d'élément absolument majoritaire.

10. Si `lst1` admet un élément absolument majoritaire `maj1`, donner un algorithme pour vérifier si `maj1` est l'élément absolument majoritaire de `lst`.

11. Recopier et compléter les lignes 4, 11, 13, 15 et 17 pour la fonction récursive `majo_abs3` qui implémente l'algorithme précédent. *Vous pourrez utiliser la fonction effectif de la question 2.*

```
1 def majo_abs3(lst):
2     n = len(lst)
3     if n == 1:
4         return ...
5     else:
6         lst_g = lst[:n//2]
7         lst_d = lst[n//2:]
8         maj_g = majo_abs3(lst_g)
9         maj_d = majo_abs3(lst_d)
10        if maj_g is not None:
11            eff = ......
12            if eff > n/2:
13                return ...
14        if maj_d is not None:
15            eff = ......
16            if eff > n/2:
17                return ...
```