Exercice 1. Nombres parfaits
Un nombre parfait est un nombre égal à la somme des ses diviseurs stricts. Par exemple 6 est parfait car ses diviseurs stricts (c'est-à-dire différents de lui-même) sont 1, 2 et 3 dont la somme fait bien 6.
De même 28 est parfait car ses diviseurs stricts sont 1, 2, 4, 7 et 14 dont la somme fait 28. Par contre 12 n'est pas parfait car ses diviseurs stricts sont 1, 2, 3, 4 et 6 dont la somme fait 16 et pas 12.
Écrire une fonction `parfait(n)` qui prend comme argument un entier naturel non nul n et renvoie comme résultat le booléen `True` si n est parfait, et le booléen `False` sinon.

Exercice 2. Quelques élémentaires sur les listes

1) Écrire une fonction `moyenne(L)` qui prend en argument une liste (non vide) de flottants ou entiers et renvoie la moyenne des éléments de cette liste. Donner la complexité de fonction fonction en fonction de la longueur de `L`
2) Écrire une fonction `minimum(L)` qui prend en argument une liste (non vide) de flottants ou entiers et renvoie le minimum de cette liste.
3) Écrire une fonction `minimum_position(L)` qui prend en argument une liste (non vide) de flottants ou entiers et renvoie le minimum de cette liste et l'indice de ce minimum (sa première apparition).
4) Écrire une fonction `maximums` qui prend en argument une liste (contenant au moins deux éléments) de flottants ou entiers et renvoie les deux éléments maximums de cette liste. Par exemple la fonction testée avec `[-1,3,2,9,-4]` renvoie `3,9` et avec la liste `[-1,-2,9,9,5]`, elle renvoie `9,9`.
5) Écrire une fonction `est_triee(L)` renvoyant `True` si la liste `L` est triée dans l'ordre croissant et `False` sinon.
6) Écrire une fonction `longueur_triee(L)` renvoyant la longueur et la plus longue sous-liste triée par ordre croissant de la liste `L`. Par exemple, appliquée à la liste `[1,0,2,5,6,3,4,1]`, la fonction renvoie `(4, [0,2,5,6])`.
7) Écrire une fonction `Partitionner(L)` qui prend comme argument une liste d'entiers `L` et renvoie le couple `L1,L2` tel que `L1` est la liste des entiers positifs et `L2` la liste des entiers strictement négatifs de `L`, dans le même ordre.
   Par exemple, l'appel à `Partitionner([5,4,1,-2,9,4,-6,-1,6,-7])` doit renvoyer `([5,4,1,9,4,6],[-2,-6,-1,-7])`.

Exercice 3
Dans cet exercice, on représente une liste de points $M_0, \dots, M_{n-1}$ par la liste de leurs coordonnées:

`L=[[x0,y0],[x1,y1],[x2,y2],...[xn-1,yn-1]]`.

On supposera dans tout ce qui suit que les abscisses sont strictement croissantes: $x_0 < x_1 < \cdots < x_{n-1}$**.**

1) Ecrire une fonction `ordonnee_croissante` qui prend en argument une liste `L` de forme ci-dessus et renvoyant `True` si les ordonnées sont rangées dans l'ordre croissant et `False` sinon. Évaluer la complexité de votre fonction.
2) Ecrire une fonction `ordonnee_positive` qui prend en argument une liste `L` de forme ci-dessus et renvoyant l'abscisse du premier point d'ordonnée strictement positive. Évaluer la complexité de votre fonction.
3) Ecrire une fonction `liste_ordonnees_positives` qui prend en argument une liste `L` de forme ci-dessus et renvoyant la liste des abscisses des points d'ordonnée strictement positive. Évaluer la complexité de votre fonction.
4) Ecrire une fonction `abscisse_maximum` qui prend en argument une liste `L` de forme ci-dessus et renvoyant l'abscisse du point d'ordonnée maximal. Évaluer la complexité de votre fonction.
5) On suppose de plus que la liste des ordonnées donnée en argument est triée par ordre croissant, les premières ordonnées sont négatives et les suivantes sont positives. Ecrire une fonction `ordonnee_positive_dicho` qui prend en argument une liste `L`, qui, à l'aide d'une méthode dichotomique, renvoie l'abscisse du premier point d'ordonnée positive. Évaluer la complexité de votre fonction.

Exercice 4 Écrire une fonction `palindrome(ch)` qui prend comme argument une chaîne de caractères `ch` et renvoie un booléen indiquant si la chaîne est ou pas un palindrome.

Exercice 5. Dictionnaire

1) Écrire une fonction `compte_lettres(ch)` prenant en argument une chaîne de caractères minuscules et qui renvoie un dictionnaire donnant la le nombre d'occurences de chaque lettre utilisée.
   Avec `ch = "bienvenue en pc"` la fonction renverra `{'b':1,'i':1,'e':4,.....}`
2) En déduire une fonction `majoritaire(ch)` prenant en argument une chaîne de caractères minuscules et qui renvoie la/les lettres les plus présentes dans une liste.
3) On veut maintenant regarder la longueur des mots utilisés. Écrire une fonction `longueurs_mots(ch)` prenant en argument une chaîne de caractères et qui renvoie un dictionnaire ayant pour clés les longueurs des différents mots utilisés et pour valeurs leurs nombres d'apparitions.
   Avec `ch = "bienvenue en pc"` la fonction renverra `{9:1,2:2}`

Exercice 6 Écrire une fonction prenant en argument un dictionnaire et testant s'il est bijectif (c'est- à-dire si chaque valeur est associée à une unique clef). Dans le cas où il est bijectif, la fonction renverra le "dictionnaire réciproque" : par exemple, si `D={2:3,4:1,6:4}`, on doit obtenir `{3:2,1:4,4:6}`.

Exercice 7 On définit la suite (u_n) par $\begin{cases} u_{n+1} = \dfrac{u_n}{2} + \dfrac{1}{u_n} \\ u_0 = 1 \end{cases}$.

1) Définir une fonction récursive `suite_rec(n)` prenant en argument n et renvoyant u_n.
2) Prouver la correction totale de votre fonction et déterminer sa complexité.

Exercice 8 On pose pour $n \in \mathbb{N}^*$, $S_n = \sum_{k=1}^{n} \frac{1}{k^2}$.
Définir une fonction récursive `somme_rec(n)` prenant en argument n et renvoyant S_n.

Exercice 9 Dans cet exercice on considère des listes constituées de 0 et de 1 avec d'abord des 0 puis des 1. Et on suppose qu'il y a au moins un 1.
Par exemple `[0,0,0,1,1,1,1]`, `[1,1,1,1]`.

-a- Écrire une fonction récursive `premier_un(L)`, de complexité linéaire, qui prend en argument une telle liste et renvoie la position du premier 1.

-b- Écrire une fonction récursive `premier_un_dicho(L)`, de complexité logarithmique, qui prend en argument une telle liste et renvoie la position du premier 1. On utilisera évidemment une méthode dichotomique.