V - Algorithmes de Recherche

1) - Recherche du maximum d'une liste

On considère une liste de nombres *lst* pas forcément triée.

EXERCICE 1

1. Écrire une fonction *maxi(lst)* qui renvoie le maximum de *lst* sans utiliser la fonction *max*.
2. Écrire une fonction *rang_max*(lst) qui renvoie le rang où le maximum est atteint pour la première fois.
3. Écrire une fonction *rangs_max*(lst) qui renvoie tous les rangs où le maximum est atteint.
4. Écrire une fonction *second_max*(lst) qui renvoie le second maximum.
   Question facultative : on pourra chercher un code avec un seul passage (une seule boucle for).
5. Tester les fonctions avec la liste `lst=[5 1 8 4 8 2 1 3]`.
6. Mêmes questions avec le minimum.

2) - Recherche dichotomique (diviser pour régner)

La recherche dichotomique d'un élément dans un objet consiste à diviser l'objet en deux sous-objets dont les tailles sont les plus proches possibles et à déterminer dans quel sous-objet l'élément est sucseptible de se trouver. On applique alors ce qui précède à ce sous objet et on répète ce processus un nombre de fois suffisant pour répondre à la question posée.

2.1) - Recherche dichotomique d'une solution de $f(x) = 0$

On suppose que f est une fonction continue sur le segment $[a, b]$ avec $f(a)f(b) \leqslant 0$.

Description de la méthode de dichotomie
On pose $c = \frac{a+b}{2}$. Si $f(a)f(c) \leqslant 0$ alors on est sûr de trouver une solution de $f(x) = 0$ dans $[a, c]$ donc on remplace $[a, b]$ par $[a, c]$. Sinon on remplace $[a, b]$ par $[c, b]$.
On répète ce processus jusqu'à ce que la longueur de l'intervalle soit inférieure ou égale à la précision de l'approximation souhaitée.

EXERCICE 2

1. Écrire une fonction python `approx(f,a,b,epsilon)` qui renvoie une approximation d'une solution de $f(x) = 0$ à *epsilon* près sur $[a, b]$ par la méthode de dichotomie.
2. Tester cette fonction sur sur la fonction $f : x \mapsto x \ln x - 2$ sur l'intervalle $[1, 3]$.

2.2) - Recherche dichotomique d'un élément dans une liste triée

On suppose que *lst* est une liste de nombres triée dans l'ordre croissant.
On remarque que la médiane de *lst* se trouve au milieu de celle-ci, on a donc un accès direct à cette médiane.

Description de l'algorithme
On compare l'élément a à la médiane m de lst.
Si $a = m$ l'objet a été trouvé dans la liste.
Sinon si $a < m$ alors a est susceptible de se trouver dans la demi-liste de gauche donc on remplace lst par cette demi-liste.
Sinon on remplace lst par la demi-liste de droite.
On répète ce processus jusqu'à trouver a dans la liste ou bien jusqu'à obtenir une liste vide ce qui signifie que a ne se trouve pas dans lst.
Au lieu de remplacer une liste par une autre on travaillera avec deux curseurs entre lesquels les termes correspondant constitueront la nouvelle liste.

EXERCICE 3

1. Écrire une fonction `rechercheD(a,lst)` qui trie `lst` par la méthode `sort` puis effectue une recherche dichotomique de `a` dans `lst` et renvoie *True* si a se trouve dans *lst* et *False* sinon.
2. Écrire une fonction `rechercheDbis(a,lst)` qui renvoie le rang du premier a dans *lst* et qui ne renvoie rien du tout si a n'est pas dans lst. On procédera comme dans la fonction `rechercheD`.
3. tester ces fonctions sur les exemples $lst = [40, 10, 1, 15, 9, 29, 8, 28, 21, 4]$ et $a = 0$ ou $a = 29$.
4. Existe-t-il une méthode plus rapide si on sait que lst est une liste triée d'entiers régulièrement espacés ?