THÈME I : Algorithme de recherche d'éléments dans une liste

1) Algorithme naïf

Dans le programme appelé "rechercheÉlément.py" disponible sur le GitHub, créez une fonction `rechercheNaive` qui recherche si un élément `elt` est contenu dans une liste L, prise comme argument d'entrée et contenant des nombres entiers.

La fonction `rechercheNaive` renvoie :

- -1 si `elt` n'est pas contenu dans L
- le nombre d'étapes pour trouver l'élément `elt` si `elt` est contenu dans L.

Quelle sera la complexité de cet algorithme dans le pire des cas ?

2) Algorithme de recherche dichotomique

On vous donne ci-dessous l'algorithme de recherche dichotomique dans une liste triée.

```
VARIABLE
t : tableau d'entiers trié
mil : nombre entier
fin : nombre entier
deb : nombre entier
x : nombre entier // x : l'entier recherché
tr : booléen

DEBUT
tr ← FAUX
deb ← 1
fin ← longueur(t)
tant que tr == FAUX et que deb ⩽ fin :
  mil ← partie_entière((deb+fin)/2)
    si t[mil] == x :
      tr = vrai
    sinon :
      si x > t[mil] :
        deb ← mil+1
      sinon :
        fin ← mil−1
      fin si
    fin si
fin tant que

renvoyer la valeur de tr
FIN
```

À l'aide du papier et d'un crayon, appliquez cet algorithme en cherchant l'élément 5 sur la liste [1,2,5,9,10,14,17,24,41].

En déduire le principe de fonctionnement de cet algorithme.
Dans le programme "rechercheÉlément.py", traduire cet algorithme en Python en complétant la fonction `rechercheDicho`.

Aide : On pourra utiliser la fonction `sorted(iterable, key=None, reverse=False)` afin de trier la liste dans un ordre ou l'ordre. Google est votre ami pour comprendre le fonctionnement de `sorted` .

Modifier votre fonction afin de :
- renvoyer -1 si l'élément n'appartient pas à la liste ;
- renvoyer le nombre d'étapes pour trouver l'élément si l'élément est contenu dans la liste.

À l'aide de tests sur des tableaux de plus en plus grands, essayer de trouver la complexité de cet algorithme dans le pire des cas.

Ce deuxième algorithme est-il plus efficace que le premier ? Avec vos propres mots, essayez d'expliquer pourquoi.

Prolongement : Écrivez une fonction qui permettent de trouver de manière efficace si deux entiers appartiennent simultanément à une liste L d'entiers donnée.

3) Comparaison d'algorithmes

La librairie time permet de faire des mesures de vitesse d'exécution.

a. En mettant maxVal à 100000 et en faisant varier nVal, réaliser quelques mesures de temps d'exécution de votre algorithme naïf ainsi que de votre algorithme de recherche par dichotomie.
b. Modifier votre programme de manière à enregistrer vos mesures de temps dans deux variables : `tempsNaif` et `tempsDicho` en fonction de nVal.

   Exemple :
   tempsNaif = [(10, 0,0001) , (100, 0,01), (1000, 0,1), (10000, 1,2)]
   tempsDicho = [(10, 0,0001) , (100, 0,001), (1000, 0,01), (10000, 0,2)]

c. En important matplotlib.pyplot, réaliser un graphique représentant vos mesures de temps en fonction de nVal. On pourra utiliser une échelle logarithmique en ordonnées.