Recherche dans une liste triée. Notions de complexité

Problématique : Comment rechercher un nombre dans une liste triée ?
Evaluer la complexité en temps.

Soit une liste de nombre entiers, par exemple :
ma_liste = [1,4,7,8,9,27,28,30,32,99,105,110 ,115,200,210,220]

On souhaite réaliser une fonction recherche(T,x) qui renvoie l'indice (la position) du nombre x si x est dans la liste T, ou -1 si x n'est pas dans la liste T.
Par exemple, position = recherche(ma_liste, 8) doit renvoyer 3 car 8 est à la position 3 (on part de 0)
Et position = recherche(ma_liste, 10) doit renvoyer -1 car pas de 10 dans la ma_liste

3 méthodes :

1- Méthode naive
2- Méthode dichotomique version itérative
3- Méthode dichotomique version récursive

1- Méthode naive

La solution naive consiste à parcourir linéairement la liste (ou tableau) et de s'arrêter si on a trouvé ou dépassé la valeur recherchée.

```
# 1- Solution Naive
# Parcours linéaire du tableau
def recherche_naive(T,x):
    for i in range(len(T)):
        if T[i] == x:
            return i
        if T[i] > x:    # pas la peine d'aller chercher plus loin
            return -1

    return -1
```

Taper et tester le programme avec la liste triée ci-dessus.

Notion de complexité.

« L'**analyse de la complexité d'un algorithme** consiste en l'étude formelle de la quantité de ressources (par exemple de temps ou d'espace) nécessaire à l'exécution de cet algorithme ».

Vous entendrez souvent parler par exemple de O(n) ou $O(n^2)$ ou encore O(nlog(n)) , ou bien encore complexité exponentielle (2^n) …. A quoi cela correspond t-il ?

La complexité permet de traduire l'efficacité d'un algorithme indépendamment de l'ordinateur sur lequel on va le faire tourner. Cette efficacité est généralement liée au nombre n de données que l'on va traiter et à la façon dont on va le faire.

[1,4,7,8,9,27,28,30,32,99,105,110 ,115,200,210,220] : cette liste contient n éléments , n=

Si dans notre algorithme on parcours la liste 1 fois, on dit que la complexité (en temps) est en O(n)
Cela correspond par exemple à une boucle du type :

For i in range (n) :
.... (instruction)

On ne se préoccupe pas de la vitesse d'exécution des instructions mais on doit uniquement se concentrer sur les boucles. Une boucle simple 1 à n correspond à du **O(n)** : le temps d'exécution de l'algo est proportionnel à n (d'où son autre nom : **complexité linéaire**)

```
For i in range (n) :
    For i in range (n) :
        (instructions)
```

Ici on a deux boucles imbriquées : on parcourt n*n données
La complexité est donc en **O(n^2)** : Le temps d'exécution est proportionnel au carré de n .(**complexité quadratique**)

Quelle est la complexité de l'algorithme de recherche méthode naive ?

Dans notre algorithme on a une boucle for et suivant si l'on trouve ou pas la donnée, ou si elle en début ou fin de liste, le parcours est différent (dans certains cas on parcourt toute la liste, dans d'autres cas que la moitié, ou alors juste le début), comment dans ce cas déterminer la complexité ?
Généralement, on se place dans le cas le plus défavorable.
Ici, dans le pire des cas **on parcourt toute la liste**, donc on est en **O(n)**

Et si ma liste est parcourue 2 fois ou 3 fois ? La complexité reste en O(n) (**O(k*n) = O(n)** si **k=constante**) .
En pratique cela veut dire que si n est grand, le temps d'exécution de l'algorithme va dépendre avant tout de n, et la constante k n'aura finalement que peu d'importance.

2- Méthode dichotomique version itérative

```python
#
# Recherche dichotomique
# version itérative
#
def recherche_dichotomique(tab,val):
    gauche = 0
    droite = len(tab) - 1
    while gauche <= droite:
        milieu = (gauche+droite)//2
        if tab[milieu] == val:  # on a trouvé
            print ("tab[milieu] = ",tab[milieu])
            return milieu
        elif tab[milieu] > val:
            droite = milieu-1
            print ("tab[milieu] = ",tab[milieu])
        else:
            gauche = milieu + 1
            print ("tab[milieu] = ",tab[milieu])

    return -1

ma_liste = [1,4,7,8,9,27,28,30,32,99,105,110 ,115,200,210,220]
val = 115
print ("recherche de ",val)
resultat = recherche_dichotomique(ma_liste,val)
print ("Position = ",resultat)
```

Ecrire la fonction et exécuter la en lui passant comme paramètre une liste TRIEE et une valeur à rechercher.

Comment déterminer la complexité de cet algorithme ?

```
recherche de  115
tab[milieu] =  30
tab[milieu] =  110
tab[milieu] =  200
tab[milieu] =  115
Position =  12
```

Recherche de 115 :

[1, 4, 7, 8, 9, 27, 28, **30**, 32, 99, 105, 110, 115, 200, 210, 220] →

[1, 4, 7, 8, 9, 27, 28, **30**, 32, 99, 105, **110**, 115, 200, 210, 220] →

[1, 4, 7, 8, 9, 27, 28, **30**, 32, 99, 105, **110**, 115, **200**, 210, 220] ←

[1, 4, 7, 8, 9, 27, 28, **30**, 32, 99, 105, **110**, **115**, **200**, 210, 220]

Pour n = 16, le nombre de données parcourues est 4 (on a bouclé 4 fois pour récupérer 30 puis 110 puis 200 et enfin 115)

[1, 4, 7, 8, 9, 27, 28, **30**, 32, 99, 105, 110, 115, 200, 210, 220]

[1, 4, 7, 8, 9, 27, 28, **30**, 32, 99, 105, **110**, 115, 200, 210, 220]

[1, 4, 7, 8, 9, 27, 28, **30**, 32, 99, 105, **110**, 115, **200**, 210, 220]

[1, 4, 7, 8, 9, 27, 28, **30**, 32, 99, 105, **110**, 115, **200**, **210**, 220]

[1, 4, 7, 8, 9, 27, 28, **30**, 32, 99, 105, **110**, 115, **200**, **210**, **220**]

```
recherche de  250
tab[milieu] =  30
tab[milieu] =  110
tab[milieu] =  200
tab[milieu] =  210
tab[milieu] =  220
Position =  -1
```

Dans le pire des cas, on boucle ici 5 fois, la donnée pas trouvée.

A chaque passage dans la boucle, on divise par deux la zone de recherche pour trouver notre valeur (recherche par encadrement). La complexité est en O($\log_2(n)$)
Cela veut dire que le nombre de boucles (ou données parcourues) est proportionnel au logarithme en base 2 de n (voir ce lien pour calcul log en base 2 : https://www.dcode.fr/logarithme)

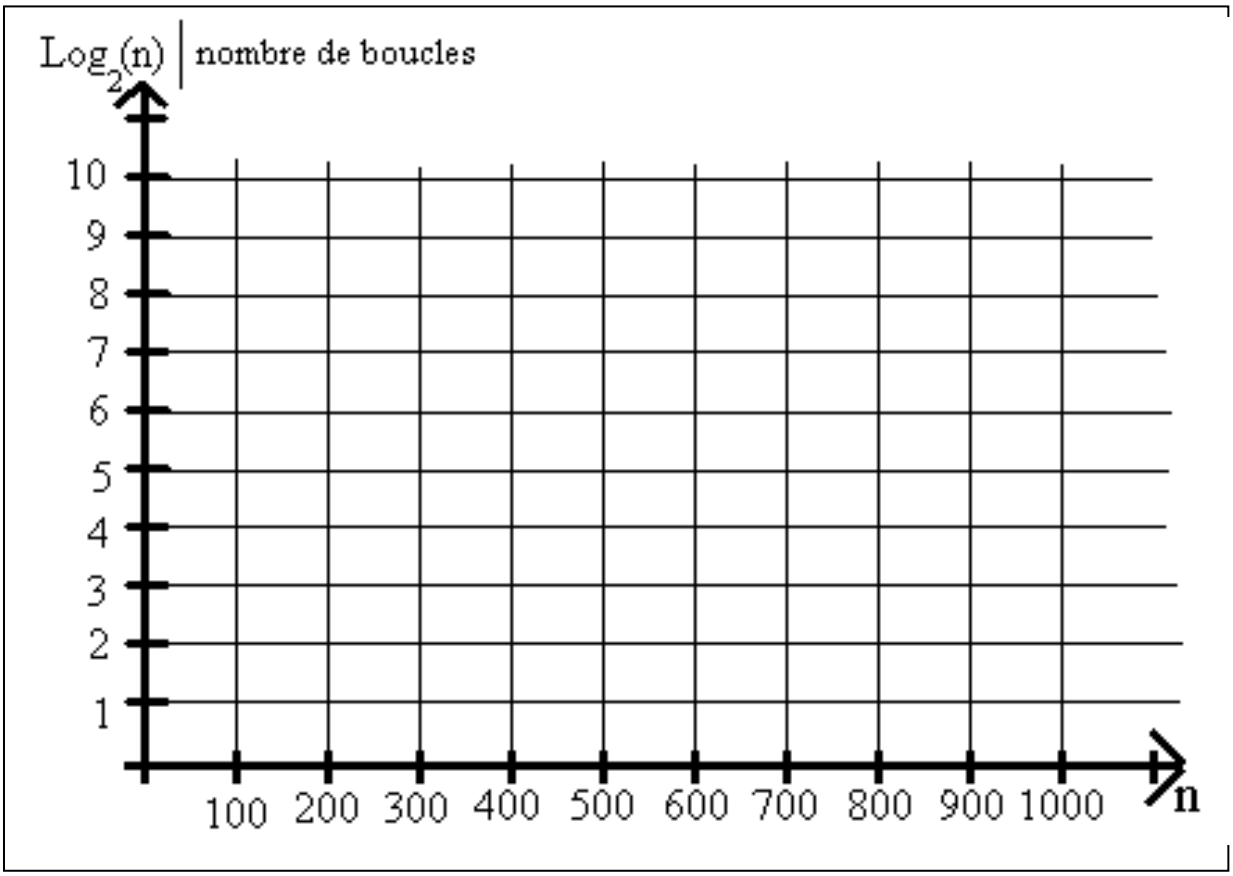
Pour bien comprendre, compléter votre programme avec le code ci-dessous

Compléter votre programme avec le code ci-dessous :

```
29
30  ma_liste=list(range(200))
31  val = 5000
32  print ("recherche de ",val)
33  resultat = recherche_dichotomique(ma_liste,val)
34  print ("Position = ",resultat)
```

Génère une liste de nombres de 0 à 199.

Vous modifierez ensuite la longueur de la liste pour rechercher la valeur val et compléter le graphe ci-dessous afin de mettre en corrélation le log(n) et le nombre de boucles



Combien de fois aurait bouclé la méthode naive ? (peut-on la tracer avec l'échelle du graphique précédent ?)

Que pouvez vous conclure quand à la rapidité de cet algorithme par rapport à la méthode naive ?

3- Méthode dichotomique version récursive

```python
# 2- Solution Recherche dichotomique
# Version Récursive
#
def recherche_dicho1(T,x,gauche,droite):

    milieu = (gauche+droite)//2
    if gauche>droite:
        return -1

    if x == T[milieu]:

        return milieu

    if x > T[milieu] :

        return recherche_dicho1(T,x,milieu+1,droite)  # je cherche entre milieu+1 et droite
    else:

        return recherche_dicho1(T,x,gauche,milieu-1)  # je cherche entre gauche et milieu-1

def recherche_dicho(T,x):
    return recherche_dicho1(T,x,0,len(T)-1)


ma_liste = [1,4,7,8,9,27,28,30,32,99,105,110 ,115,200,210]

val = 115
print ("recherche de ",val)
resultat = recherche_dicho(ma_liste,val)
print ("Position = ",resultat)

```

Taper et tester le programme dichotomique en version récursive.

Fonction intermédiaire pour calculer len(T)

Répondez aux questions suivantes :

Pourquoi la fonction recherche_dicho1(T,x,gauche,droite) est-elle « récursive » ?

Est-ce que la fonction intermédiaire est récursive ?

Si je veux me passer de la fonction intermédiaire, indiquer les paramètres à fournir à la fonction recherche_dicho1 :

resultat = recherche_dicho1( )

```python
if x > T[milieu] :
    print (T[milieu])
    res = recherche_dicho1(T,x,milieu+1,droite)
    print ("retour :",res)
    return res
else:
    print (T[milieu])
    res = recherche_dicho1(T,x,gauche,milieu-1)
    print ("retour",res)
    return res
```

Décomposer (développer) le programme récursif en utilisant la méthode vue en cours en indiquant la valeur des paramètres d'appel et de retour de la fonction (faire légère modif ci-contre pour visualiser valeur de retour) .

Vous pouvez ajouter des print pour afficher les variables et des « a = input() » pour marquer les points d'arrêts que vous jugez utiles …