Chapitre 6 - Recherche par dichotomie

1- ALGORITHME DE RECHERCHE NAÏF :

Lorsque l'on recherche la présence d'une valeur particulière dans une liste, il est nécessaire de parcourir cette liste. Soit la fonction *rechercheNaive()* définie ci-contre :

```
def rechercheNaive(l,val) :
    for i in range(len(l)) :
        if l[i] == val : return i
    return False
```

1- Que renvoie l'exécution suivante : `rechercheNaive([5,9,1,-8,5,1],36)`

2- Que renvoie l'exécution suivante : `rechercheNaive([5,9,1,-8,5,1],1)`

3- Quelle est la classe de complexité de cet algorithme ?

2- COMMENT ALLER PLUS VITE LORSQUE LA LISTE EST TRIEE ?

Lorsque des données sont stockées dans une liste, on prend généralement soin de les ranger triées, par ordre croissant. Il est alors beaucoup plus rapide de retrouver ces valeurs en utilisant le principe de la dichotomie.

Méthodes de dichotomie, méthodes itératives par lesquelles on ramène la résolution, sur un intervalle donné, d'un problème à la résolution d'une suite de problèmes de même nature, sur des sous-intervalles de longueur 2 fois plus petite d'une itération à la suivante.

Par exemple, la fonction *rechercheDichotomique()* définie ci-contre, utilise le principe de la dichotomie pour retrouver (ou pas), l'index d'une valeur présente (ou pas) dans cette liste :

```
def rechercheDichotomique(l,val) :
    iD = 0
    iF = len(l)-1
    while iD <= iF :
        iM = (iD + iF)//2
        if l[iM] == val : return iM
        elif val > l[iM] : iD = iM + 1
        elif val < l[iM] : iF = iM - 1
    return False
```

Si on exécute cette fonction avec la liste `[-75, -47, -35, -20, -19, 14, 45, 49, 71, 90]`, on obtient :

```
>>> rechercheDichotomique([-75, -47, -35, -20, -19, 14, 45, 49, 71, 90],45)

>>> rechercheDichotomique([-75, -47, -35, -20, -19, 14, 45, 49, 71, 90],-30)
```

Pour bien comprendre ce qui se passe précisément, on fait fonctionner l'algorithme « à la main » :

$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$	$i=9$
-75	-47	-35	-20	-19	14	45	49	71	90

```
>>> rechercheDichotomique([-75, -47, -35, -20, -19, 14, 45, 49, 71, 90],45)
```

$\underline{iD}$	$\underline{iF}$	$\underline{iM}$	$\underline{\ell[iM]}$	
				[-75, -47, -35, -20, -19, 14, 45, 49, 71, 90]
				[-75, -47, -35, -20, -19, 14, 45, 49, 71, 90]
				[-75, -47, -35, -20, -19, 14, 45, 49, 71, 90]
				[-75, -47, -35, -20, -19, 14, 45, 49, 71, 90]
				[-75, -47, -35, -20, -19, 14, 45, 49, 71, 90]

```
>>> rechercheDichotomique([-75, -47, -35, -20, -19, 14, 45, 49, 71, 90],-30)
```

$\underline{iD}$	$\underline{iF}$	$\underline{iM}$	$\underline{\ell[iM]}$	
				[-75, -47, -35, -20, -19, 14, 45, 49, 71, 90]
				[-75, -47, -35, -20, -19, 14, 45, 49, 71, 90]
				[-75, -47, -35, -20, -19, 14, 45, 49, 71, 90]
				[-75, -47, -35, -20, -19, 14, 45, 49, 71, 90]
				[-75, -47, -35, -20, -19, 14, 45, 49, 71, 90]

```
rechercheDichotomique([-75, -47, -35, -20, -19, 14, 45, 49, 71, 90],100)
```

iD	iF	iM	$\ell[iM]$	
				[-75, -47, -35, -20, -19, 14, 45, 49, 71, 90]
				[-75, -47, -35, -20, -19, 14, 45, 49, 71, 90]
				[-75, -47, -35, -20, -19, 14, 45, 49, 71, 90]
				[-75, -47, -35, -20, -19, 14, 45, 49, 71, 90]
				[-75, -47, -35, -20, -19, 14, 45, 49, 71, 90]

3- QUELLE EST LA CLASSE DE COMPLEXITE DE CET ALGORITHME ?

La recherche dichotomique est sans commune mesure, extrêmement plus performante qu'une recherche naïve. Pour s'en rendre compte, on intègre dans les scripts des fonctions *rechercheNaive()* et *rechercheDichotomique()* des compteurs d'itérations :

```
def rechercheNaive(l,val) :
    nb = 0
    for i in range(len(l)) :
        nb = nb + 1
        if l[i] == val : return i,nb
    return False,nb
```

```
def rechercheDichotomique(l,val) :
    nb = 0
    iD = 0
    iF = len(l)-1
    while iD <= iF :
        nb = nb + 1
        iM = (iD + iF)//2
        if l[iM] == val : return iM,nb
        elif val > l[iM] : iD = iM + 1
        elif val < l[iM] : iF = iM - 1
    return False,nb
```

```
# Programme principal
N = 10
l = [randint(-100*N , 100*N) for i in range(N)]
print("liste générée")
l = sorted(l)
print("liste triée")
if N<10 : print(l)
val = 50*N
retourNaif = rechercheNaive(l,val)
print("retourNaif : ",retourNaif)
retourDicho = rechercheDichotomique(l,val)
print("retourDicho : ",retourDicho)
```

sorted() : fonction native de python qui retourne la liste mise en argument triée

On teste ces fonctions sur des listes de tailles variables, dont les éléments sont des nombres aléatoires. Le programme principal de test est donné ci-contre.

On réalise des exécutions pour des tailles de liste N = 10 , N = 100, N = 1000 , N = 10 000 , N = 100 000 , N = 1 000 000 , N = 10 000 000 . Les résultats sont consignés dans le tableau ci-dessous :

nombre itérations	$N = 10$	$N = 100$	1000	10 000	100 000	1 000 000	10 000 000
Recherche naïve							
Recherche dichotomique							
Calcul de $log_2(N) = \frac{\ln(N)}{\ln(2)}$							

4- COMPLEXITE DE CET ALGORITHME :

Point Cours :

On dit que l'algorithme a une complexité de classe $\mathcal{O}(log_2(n))$ ou aussi que la complexité de cet algorithme est *logarithmique*. Les temps de calcul seront « *à peu près* » proportionnels au nombre $log_2(n)$:

- Pour une taille de liste de $n = 2^{10} = 1024$, le nombre de divisions par 2 pour arriver à l'unité est égal à 10. On a $log_2(1024) = log_2(2^{10}) = 10$
- Pour une taille de liste de $n = 2^{20} = 1\,048\,576$, le nombre de divisions par 2 pour arriver à l'unité est égal à 20. On a $log_2(1\,048\,576) = log_2(2^{20}) = 20$

Question :

1- Calculer le nombre de divisions par 2 qui permet d'arriver à l'unité pour le nombre $n = 1\,073\,741\,824$.

2- Une liste a une taille $n = 10\,000\,500$. Combien d'itérations sont-elles nécessaires au maximum pour pouvoir retrouver l'index d'un de ses éléments. ?

5- APPLICATION :

⇨ Télécharger le fichier *dictionnaire.zip* proposé sur *nsibranly.fr*. Décompresser et copier les 2 fichiers qu'il contient, dans votre répertoire de travail sur la recherche dichotomique.

⇨ Ouvrir le fichier *mots.txt* avec le bloc-notes (double-clic sur le nom). Il contient des centaines de milliers de lignes qui indiquent chacune un mot de la langue française.

⇨ Editer sur Pyzo le fichier *dictionnaire.py* . Analyser le code qu'il contient …

```
>>> (executing file "dictionnaire.py")
aboyant
amaigrissant
atermoyai
bisasse
calamistrasses
chiee
confessat
crepait
decongelassiez
demis
```

Il permet de lire le fichier *mots.txt* et de copier toutes les données contenues dans une liste simple. Pour l'instant le script affiche un extrait de quelques éléments de cette liste, dans la console :

Contrairement aux codes présentés jusqu'à présent, un « *docString* » a été inséré pour introduire les fonctions. Ces commentaires permettent de documenter son code. Le *docString* d'une fonction peut être récupéré facilement dans la console, en exécutant par exemple `>>> print(motsFrancais.__doc__)` pour la documentation de la fonction *motsFrancais()* .

Exercice : On souhaite compléter ce code, en créant une fonction *dictionnaire()* dont le code incomplet est donné ci-dessous :

```
def dictionnaire(l : list) -> None :
    """
        Paramètres : liste dont les éléments sont les mots fraçais non accentués
        Sortie : aucune variable renvoyée.
    """
```

En exécutant le programme principal ci-contre, la fonction *dictionnaire()* permet de rechercher la présence d'un mot particulier dans le fichier *mots.txt* . Ce mot est saisie par l'utilisateur (fonction *input()*). On donne ci-dessous un exemple d'exécution :

```
# Programme principal
l = motsFrancais()
dictionnaire(l)
```

```
>>> (executing file "dictionnaireCorrige.py")
nom recherché (return pour arrêter): anticonstitutionnellement
recherche naïve : index 14508 en 14509 itérations
recherche dichotomique : index 14508 en 19 itérations

nom recherché (return pour arrêter) : infformattique
recherche naïve : index False en 336531 itérations
recherche dichotomique : index False en 19 itérations

nom recherché (return pour arrêter) :

>>>
```

Questions :

1- Compléter le code du fichier *dictionnaire.py* en y écrivant celui de la fonction *dictionnaire*().
2- Exécuter dans la console l'instruction :

```
>>> print(dictionnaire.__doc__)
```