TRAVAUX PRATIQUES 4

ALGORITHMES DICHOTOMIQUES

Voici les instructions éventuellement utiles dans ce TP :

`len(L)`	donne la longueur de `L`
`L=[]`	`L` est la liste vide
`L[i]`	donne la valeur de l'élément de la liste d'indice `i`
`L+M`	concatène (juxtapose) les listes `L` et `M`
`L.append(a)`	ajoute `a` à la fin de la liste
`L.insert(i,a)`	ajoute l'élément `a` à la i^e position
`del L[i]`	supprime et retourne le i^e élément
`for k in range(i,j)`	boucle **pour** `k` allant **de** `i` (inclus) **à** `j` (exclus)
`while`	boucle **tant que**
`random.randrange(a,b)`	renvoie un nombre entier `n` aléatoire tel que $a \leqslant n < b$
`random.randint(a,b)`	renvoie un nombre entier `n` aléatoire tel que $a \leqslant n \leqslant b$
`L.sort()`	renvoie la liste `L` triée dans l'ordre croissant

1. **Recherche dichotomique dans une liste triée**

   (a) Que renvoie les instructions suivantes ?

```
import random as rd
L=[]
n=1000
for k in range(n):
    L.append(rd.randint(0,1000))
L.sort()
```

   (b) Expliquer le rôle et le fonctionnement de la fonction suivante :

```
def fonction(L,x):
    for k in range(len(L)):
        if L[k]==x:
            return True,k
    return False
```

   (c) Si on décide de rechercher un mot dans un dictionnaire qui comporte 40 000 mots avec la méthode précédente, au pire des cas il faudra 40 000 tours de boucle !

   Or, le dictionnaire est trié par ordre alphabétique ; une autre méthode consiste à comparer le mot à chercher avec celui se trouvant au milieu du dictionnaire. Si le mot à chercher est antérieur dans l'ordre alphabétique, on sait qu'on devra le chercher dans la première moitié du dictionnaire. Sinon, on le cherche dans la deuxième moitié.

   Et on recommence...

   Voici ce que cela donne en terme de nombre d'opérations à effectuer, en choisissant le pire cas : celui où le mot est absent du dictionnaire.

   - ⋆ Au départ, on cherche le mot parmi 40 000.
   - ⋆ Après le test n°1, on ne le cherche plus que parmi 20 000.
   - ⋆ Après le test n°2, on ne le cherche plus que parmi 10 000.
   - ⋆ Après le test n°3, on ne le cherche plus que parmi 5 000.

⋆ et ainsi de suite ...
⋆ Après le test n°15, on ne le cherche plus que parmi 2.
⋆ Après le test n°16, on ne le cherche plus que parmi 1.

Et là, on sait que le mot n'existe pas : on a obtenu notre réponse en 16 opérations contre 40 000 précédemment.

Pour un tableau comportant n éléments, le programme s'arrête lorsque $\frac{n}{2^k} = 1$, où k est le nombre d'étapes.

On obtient $k = \frac{\ln n}{\ln 2}$; et la complexité du programme est de l'ordre de $\ln n$. (On écrit $\log(n)$ en informatique.)



la recherche dichotomique ne peut s'effectuer que sur des éléments préalablement triés.

Ecrire une fonction `recherche_dichotomie(L,x)` qui prend en argument une liste triée et un élément à chercher par dichotomie dans la liste. Cette fonction renvoie `True` et l'indice de `x` si celui-ci est dans la liste, et `False` sinon.

2. **Résolution dichotomique de l'équation** $f(x) = 0$

L'objectif de cette question est de programmer une méthodes numériques qui permet la résolution de l'équation $f(x) = 0$.

Pour illustrer cette méthode, on considère la fonction f définie sur $\mathbb{R}$ par

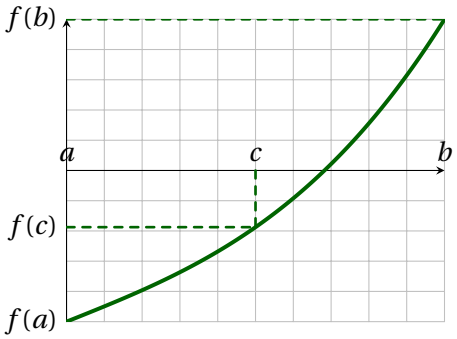
$$f(x) = x^3 + x - 1$$

(a) Déterminer $f'(x)$, $\forall x \in \mathbb{R}$.

(b) Etudier les variations de la fonction f sur $\mathbb{R}$.

(c) Démontrer que l'équation $f(x) = 0$ admet une unique solution et localiser celle-ci entre deux entiers consécutifs a et b.

(d) On pose $c = \frac{a+b}{2}$.
Calculer $f(c)$; que peut-on en déduire ?



(e) Créer une fonction python qui permet de définir la fonction `f`.

(f) Ecrire une fonction `Dichotomie(f,p,a,b)`, prenant en argument la fonction f, un entier p et les bornes a et b de l'intervalle sur lequel on applique la méthode de dichotomie. Cette fonction doit retourner une valeur approchée à 10^{-p} près de la solution de l'équation $f(x) = 0$.