TP 12 – Recherche par dichotomie

La **dichotomie** est un principe général se rapprochant du fameux « diviser pour mieux régner ». Nous allons voir plusieurs applications.

I Recherche par dichotomie dans une liste croissante

Lorsqu'une une liste `L` est triée dans l'ordre croissant, il existe une méthode plus efficace (nécessitant moins d'étapes) que la recherche séquentielle pour déterminer si un élément `x` appartient à la liste `L`. L'algorithme est le suivant.

- On regarde l'élément au "milieu" de la liste. Si la liste est de longueur impaire, alors le milieu de la liste est effectivement un élément de la liste. Si la liste est de longueur paire, alors le milieu de la liste tombe entre deux valeurs, on prend la valeur qui est à gauche de ce milieu (choix arbitraire).
  - Si cet élément vaut `x`, c'est fini, et on renvoie `True`.
  - Si cet élément est plus grand strictement que `x`, on cherche dans la première moitié de la liste.
  - Si cet élément est plus petit strictement que `x`, on cherche dans la seconde moitié de la liste.
- On continue jusqu'à avoir d'un seul élément.
  - Si ce dernier élément vaut `x`, on renvoie `True`.
  - Sinon, cela veut dire que `x` n'est pas dans la liste donc on renvoie `False`.

I.1 Des exemples à la main

• Rechercher si l'élément `x=9` est dans la liste `L = [1, 1, 2, 3, 5, 6, 8, 8, 9, 9, 10, 12, 13]`.

1. On repère le milieu de la liste.

1	1	2	3	5	6	8	8	9	9	10	12	13

2. On le compare à `x` et on garde la "bonne" moitié de la liste.

							8	9	9	10	12	13

3. On repère le milieu de la nouvelle liste.

							8	9	9	10	12	13

4. On a trouvé l'élément `x` dans la liste, on s'arrête.

deb	fin	m	L[m]	Conséquence
0	12	6	8	On garde la seconde moitié
7	12	9	9	On a trouvé x !

• Rechercher si l'élément `x=10` est dans la liste `L = [1, 8, 9, 9, 10, 12, 15, 17, 17, 20, 23]`.

1. On repère le milieu de la liste.

1	8	9	9	10	12	15	17	17	20	23

2. On le compare à `x` et on garde la "bonne" moitié de la liste.

1	8	9	9	10						

3. On repère le nouveau milieu.

1	8	9	9	10						

4. On garde la bonne moitié.

			9	10						

5. On repère le nouveau milieu.

			9	10						

6. On garde la bonne moitié.

				10						

7. On a trouvé l'élément `x` dans la liste, on s'arrête.

`deb`	`fin`	`m`	`L[m]`	Conséquence
`0`	`10`	`5`	`12`	On garde la première moitié
`0`	`4`	`2`	`9`	On garde la seconde moitié
`3`	`4`	`3`	`9`	On garde la seconde moitié
`4`	`4`	`4`	`10`	On a trouvé `10`!

• Rechercher si l'élément `x=18` est dans la liste `L = [1, 8, 9, 9, 10, 12, 15, 17, 17, 20, 23]`.

1. On repère le milieu de la liste.

1	8	9	9	10	12	15	17	17	20	23

2. On le compare à `x` et on garde la "bonne" moitié de la liste.

						15	17	17	20	23

3. On repère le nouveau milieu.

						15	17	17	20	23

4. On garde la bonne moitié.

						15	17			

5. On repère le nouveau milieu.

						15	17			

6. On garde la bonne moitié.

							17			

7. Il ne reste qu'un élément, et ce n'est pas `x` donc finalement `x` n'est pas dans la liste.

`deb`	`fin`	`m`	`L[m]`	Conséquence
`0`	`10`	`5`	`12`	On garde la seconde moitié
`6`	`10`	`8`	`17`	On garde la première moitié
`6`	`7`	`6`	`15`	On garde la seconde moitié
`7`	`7`	`7`	`17`	On n'a pas trouvé `18` !

I.2 Pré-requis : Savoir trier une liste par ordre croissant

1. Écrire une fonction `maximum` qui prend en argument une liste `L` et qui renvoie la plus grande valeur de la liste ainsi que la position de cette plus grande valeur.

Entrée [1]:
```
def maximum(L):
    max = L[0]
    imax = 0
    for k in range(len(L)):
        if L[k] > max:
            max = L[k]
            imax = k
    return(max, imax)
```

Entrée [2]:
```
L=[1,5,20,7]
maximum(L)
```

Out [2]:
```
(20, 2)
```

2. Écrire une fonction `tricroissant` qui prend en argument une liste `L` et qui renvoie cette même liste rangée dans l'ordre croissante. On pourra utiliser l'algorithme suivant.

   - On créé une nouvelle liste, appelée par exemple `M`, de la même taille que `L`, qui ne contient initialement que des zéro.
   - On récupère le maximum de la liste `L` ainsi que sa position.
     - On insère ce maximum à la dernière place de la liste `M`.
     - On le supprime de la liste `L` (attention, la longueur de la liste `L` diminue donc).
   - On récupère le nouveau maximum de la liste `L` ainsi que sa position (qui est donc le «second maximum» de la liste `L` de départ).
     - On insère ce nouveau maximum à l'avant-dernière place de la liste `M`.
     - On le supprime de la liste `L`.
   - Et on continue ainsi de suite jusqu'à avoir «vider» la liste `L` et avoir rempli complètement la liste `M`.

Entrée [3]:
```
def tricroissant(L):
    M = [0]*len(L)
    n = len(M)
    for k in range(1,n+1):
        (max, imax) = maximum(L)
        M[n-k] = max
        del L[imax]
    return(M)
```

Entrée [4]:
```
L=[1,5,20,7]
tricroissant(L)
```

Out [4]:
```
[1, 5, 7, 20]
```

I.3 L'algorithme

On décrit l'algorithme de recherche par dichotomie de la manière suivante.

Entrée [5]:
```
Données : L (liste triée dans l'ordre croissant), x (nombre)
Sortie : Vrai si x appartient à L, Faux sinon

d=0 #premier indice dans L
f = len(L) - 1 #dernier indice dans L

Tant que d <= f
    m = (d+f)//2 # position de l'élément "central"
    Si L[m] est égal à x : # on compare l'élément central à x
        C'est fini, on renvoie Vrai
    Sinon , si L[m] > x
        On actualise d ou f pour chercher dans la première moitié
    Sinon
        On actualise d ou f pour chercher dans la deuxième moitié

Si on arrive ici, c'est que x n'est pas dans la liste.
On renvoie Faux
```

① Écrire une fonction `dichotomie` qui prend en argument une liste `L` et un élément `x` et qui renvoie `True` si `x` appartient à la liste `L` et qui renvoie `False` sinon.

Entrée [6]:
```
def dichotomie(L,x):
    d = 0 # indice de début
    f = len(L) -1 # indice de fin
    while f >= d:
        m = (d+f)//2
        if x == L[m]:
            return True
        elif x < L[m]:
            f = m-1 # on regarde la 1ere moitié
        else :
            d = m+1 # on regarde la 2nd moitié
    return False
```

Tester cette fonction.

Entrée [7]:
```
dichotomie([1,2,3,4], 3)
```

Out [7]:
```
True
```

Entrée [8]:
```
dichotomie([1,2,3,4], 5)
```

Out [8]:
```
False
```