

Exercice 1 :

On donne l'algorithme de recherche dichotomique suivant :

```
def recherche_dicho(L,elt):
    index_debut=-1
    index_fin=len(L)
    while index_fin-1>index_debut:
        index_milieu=(index_fin+index_debut)//2
        if L[index_milieu]==elt:
            return (print("l'élément {0} est à
l'indice{1}".format(elt,index_milieu)))
        elif elt>L[index_milieu]:
            index_debut=index_milieu
        else :
            index_fin=index_milieu
    print("l'élément n'est pas dans la liste")
```

On cherche dans la liste [2,4,6] la valeur 6 puis la valeur 7 à l'aide de la fonction ci-dessus. Expliciter les différentes valeurs de *indice_debut*, *indice_milieu*, *indice_fin*

Elt=6	Indice_debut	Indice_milieu	Indice_fin
Début 1 ^e passage	-1	1	3
A la fin du 1 ^e passage	1	1	3
Début 2 ^e passage	1	2	3
		Fin	
Elt=7	Indice_debut	Indice_milieu	Indice_fin
Début 1 ^e passage	-1	1	3
A la fin du 1 ^e passage	1	1	3
Début 2 ^e passage	1	2	3
Fin 2 ^e passage	2	2	3
Début de 3 ^e passage	Inégalité stricte non respecté Elément non trouvé		

Exercice 2 :

On donne l'algorithme de recherche dichotomique suivant :

```
def recherche_dicho(L,elt):
    index_debut=0
    index_fin=len(L)-1
    while index_fin>=index_debut:
        index_milieu=(index_fin+index_debut)//2
        if L[index_milieu]==elt:
            return (print("l'élément {0} est à
l'indice{1}".format(elt,index_milieu)))
        elif elt>L[index_milieu]:
            index_debut=index_milieu+1
        else :
            index_fin=index_milieu-1
    print("l'élément n'est pas dans la liste")
```

On prend $L = [12,15,16,18,21,24,25,27,31,33,35,36,38]$

Donner l'ordre de parcours des différentes valeurs d'index-milieu dans une recherche dichotomique des valeurs 11 et 39.

12 ⁰	15 ¹	16 ²	18 ³	21 ⁴	24 ⁵	25 ⁶	27 ⁷	31 ⁸	33 ⁹	35 ¹⁰	36 ¹¹	38 ¹²
						25						
		16							33			
12											36	
												38

Ici on retrouve un nombre d'itérations maximal de l'ordre $\log_2(13) \approx 4$

Exercice 3 :

On reprend le code de l'exercice 2 et on considère la liste [2,4] .

- 1) Quel est le problème si on cherche la valeur 4 lorsqu'on remplace $\text{index_fin} \geq \text{index_debut}$ par $\text{index_fin} > \text{index_debut}$?
- 2) Quel est le problème si on cherche la valeur 4 et lorsqu'on remplace $\text{index_debut} = \text{index_milieu} + 1$ par $\text{index_debut} = \text{index_milieu}$?
- 3) Quel est le problème si on cherche la valeur 3 et lorsqu'on remplace $\text{index_fin} = \text{index_milieu} - 1$ par $\text{index_fin} = \text{index_milieu}$?

1) Si on cherche 4, alors on a un problème avec une inégalité stricte :

A l'issue du 1^e passage, $\text{index_min} = \text{index_max}$ et la boucle est stoppée. On ne peut donc pas tester les extrémités.

2^0	4^1	$d=0, f=1, m=0$
2^0		$d=1, f=1$

2) Si on cherche 4 avec $\text{index_deb} = \text{index_milieu}$:

A la fin du 1^e passage, on ne change rien et la boucle ne se termine pas (boucle infinie)

2^0	4^1	$d=0, f=1, m=0$
2^0		$d=0, f=1$

3) Si on cherche 3 avec $\text{index_fin} = \text{index_milieu}$:

A la fin du 2^e passage, on ne change rien et la boucle ne se termine pas

2^0	4^1	$d=0, f=1, m=0$
2^0		$d=1, f=1, m=1$
	4^1	$d=1, f=1, m=1$

Exercice 4 :

Simuler un jeu où un nombre entier n entre 1 et 1000 est choisi au hasard par la machine avec la fonction `randint`, du module `random` de la bibliothèque `numpy`. Puis proposer un programme permettant de deviner ce nombre par dichotomie (la fonction proposée génère une liste `[i for i in range(1, 1001)]`, puis cherche n).

```
from numpy import random as rd
def devine(n):
    liste=[i for i in range(1,1001)]
    index_debut=0
    index_fin=len(liste)-1
    compteur=0
    while index_fin >= index_debut:
        compteur=compteur+1
        milieu=(index_debut+index_fin)//2
        if liste[milieu]==n:
            return (print("la valeur est {0} trouvée en {1}
fois".format(n,compteur)))
        elif liste[milieu]<n:
            index_debut=milieu+1
        else :
            index_fin=milieu-1
    print("la valeur n'est pas comprise entre 1 et 1000")
n=rd.randint(1,1001)
```

On trouve le résultat en 8 fois ce qui est assez estimé par $\log_2(1000)$

Exercice 5 :

On donne la fonction suivante qui doit renvoyer une solution approchée sur un intervalle $[a; b]$ de l'équation $f(x) = 0$. La fonction f est supposée continue et monotone et a solution unique sur $[a; b]$

```
def dichotomie(f, a, b, p):
    while b-a > p:
        m = (a+b)/2
        if f(a)*f(m) < 0:
            b = m
        else :
            a = m
    return (a+b)/2
```

- 1) Tester ce programme avec $f(x) = x$, $a = -3$, $b = 3$, $p = 0,001$.

2) Corriger ce programme

Avec les valeurs introduites, il arrive la situation $f(m) = 0$ ce qui conduit poser $a = m$ (ce qui nous fait sauter la racine). Le programme continue dans l'intervalle $[1,5; 3]$, ce qui conduit à $a \rightarrow b$. Il faut modifier l'inégalité stricte

Exercice 6 :

Ecrire une fonction *insertion* qui prend en arguments une liste L de nombres triés et un nombre x . Cette fonction renvoie une nouvelle liste en utilisant le mode suivant :

- Si $x \leq L[0]$ alors l'élément est placé en début de liste
- Si $x \geq L[-1]$ alors l'élément est placé en fin de liste
- Si x est dans la liste alors la dichotomie est utilisée pour insérer x dans L à la bonne position (juste après la valeur déjà présente).
- Si x n'est pas repéré dans la liste à l'issue de la recherche dichotomique, alors il est placé à la bonne position (entre deux éléments l'encadrant).

```
def insertion3(L,x):
    deb=0
    fin=len(L)-1
    if x<=L[0]:
        return [x]+L
    elif x>=L[-1]:
        return L+[x]
    else :
        while fin>=deb:
            m=(deb+fin)//2
            if L[m]==x:
                return L[:m+1]+[x]+L[m+1:]
            elif L[m]<x:
                deb=m+1
            else :
                fin=m-1
        return L[:deb]+[x]+L[deb:]
```

Exercice 7 :

On considère la fonction exponentiation suivante :

```
def expo_dicho(x,n):
    resultat=1
    while n!=0:
        if n%2==1:
```

```
        resultat=resultat*x
    x=x*x
    n=n//2
    return resultat
```

Décrire les valeurs successives des variables pour obtenir *expo_dicho(2,9)*

$résultat = 1$	$x = 2$	$n = 9$
$résultat = 1 * 2$	$x = 2^2$	$n = 4$
$résultat = 2$	$x = 2^4$	$n = 2$
$résultat = 2$	$x = 2^8$	$n = 1$
$résultat = 2 * 2^8 = 2^9$	$x = 2^{16}$	$n = 0$

Exercice 8 :

- 1) Ecrire une fonction qui calcule un encadrement entre deux entiers consécutifs du logarithme en base 2 de n avec $n > 0$.

Donc si $p \leq \log_2(n) < p + 1$

Alors $2^p \leq n < 2^{p+1}$

Si on divise k fois par 2 : Alors $2^{p-k} \leq \frac{n}{2^k} < 2^{p-k+1}$

Donc si $p = k$ alors $\frac{n}{2^k} < 2$: donc pour trouver p on divise n par 2 tant que $n \geq 2$

```
def log2(n):
    p=0
    while n>=2:
        n=n//2
        p=p+1
    return p, p+1
```

- 2) Ecrire sur le même modèle une fonction qui calcule un encadrement entre deux entiers consécutifs du logarithme en base 10 de n avec $n > 0$.

Donc si $p \leq \log_{10}(n) < p + 1$

Alors $10^p \leq n < 10^{p+1}$

Si on divise k fois par 10 : Alors $10^{p-k} \leq \frac{n}{10^k} < 10^{p-k+1}$

Donc si $p = k$ alors $\frac{n}{10^k} < 10$: donc pour trouver p on divise n par 10 tant que $n \geq 10$

```
def log10(n):
    p=0
    while n>=10:
        n=n//10
        p=p+1
    return p, p+1
```