Exercice 1.

```
def dichotomie(Y,a,b):
    if Y[0]*Y[len(Y)-1] >0:
        return "Une dichotomie ne s'applique pas"
    debut = 0
    fin = len(Y)-1
    while fin - debut > 1:
        milieu = (debut+fin)//2     # Indice du milieu
        if Y[debut] * Y[milieu] <= 0:
            fin = milieu          # Dichotomie à gauche
        else:
            debut = milieu # Dichotomie à droite
    dx = (b-a)/(len(Y)-1)
    return a + dx * debut , a + dx * fin   # Encadrement
```

Exemple :

```
from math import sin
N = 100000
a, b = 2, 4
X = [ a+k*(b-a)/N for k in range(N+1) ]
Y = [ sin(x) for x in X ]
print(dichotomie(Y,a,b))
```

ce qui produit :

```
>>> (executing cell "TP 9" (line 2 of "<tmp 1>"))
(3.1415800000000003, 3.1416000000000004)
```

En effet, la solution est $\pi \approx 3,1415926$.

Exercice 2.

```
def recherche(L,e):
   for x in L:
      if x == e:
         return True    # Trouvé !
   return False          # Pas trouvé dans la liste
```

Exercice 3.

a) A l'aide d'un slicing.

```
def dich_search_s(L,e):
    while len(L)>1:    # tant que L contient >1 élément
        iMed = len(L)//2   # indice de l'élément médian
        if e == L[iMed]:   # Cas de succés
            return True
        elif e > L[iMed]:
            L=L[iMed:]  # dichotomie à droite
        else:
            L= L[:iMed]     # dichotomie à gauche
        if (len(L)==1 and e == L[0]):    # reste 1 élément
            return True
    return False    # Cas d'échec
```

b) Sans slicing.

```
def dich_search(L,e):
    Imin, Imax = 0, len(L)-1   # Imin, Imax : délimiteurs
    while Imax - Imin >= 0:
        Imed = (Imin+Imax)//2
        if e == L[Imed]:   # Cas de succès
            return True
        elif e > L[Imed]:
            Imin = Imed + 1  # Dichotomie à droite
        else:
            Imax = Imed - 1  # Dichotomie à gauche
    return False  # Cas d'échec
```

c) Test du temps d'exécution des 4 algorithmes de recherche sur une liste aléatoire ordonnée de 100 000 nombres :

```
from random import random
L = [random() for k in range(100000)]   # Liste aléatoire
L.sort()    # Tri de la liste

from time import time
for f in (recherche,  dich_search_s, dich_search):
   a = time()
   f(L,0.5)
   b = time()
   print("avec",f,":",b-a, 'secondes')  # Temps d'éxécution
```

```
>>> (executing lines 27 to 91 of "<tmp 1>")
avec <recherche> : 0.0076819999999990785 secondes
avec <recherche_c> : 0.008221999999999952 secondes
avec <dich_search_s> : 0.0008760000000016532 secondes
avec <dich_search> : 1.2999999999152578e-05 secondes
```

C'est clairement la fonction `dich\_search` qui est la plus rapide. Avec 10^6 éléments, l'écart se creuse :

```
>>> (executing lines 27 to 91 of "<tmp 1>")
avec <recherche> : 0.1448229999999988 secondes
avec <recherche_c> : 0.11208799999999997 secondes
avec <dich_search_s> : 0.030469000000000008 secondes
avec <dich_search> : 1.5999999998683734e-05 secondes
```