Algorithmes de recherche dichotomique

On a une liste de valeurs ordonnées.
On a déjà vu une méthode pour déterminer si un élément est présent dans une liste, en comparant tous les éléments un à un.
On va se servir du renseignement supplémentaire : la liste est **triée**.

1 Un jeu

L'ordinateur choisit un nombre entier aléatoire entre 0 et 100.
Le jeu consiste à trouver ce nombre en un minimum d'essai, l'ordinateur précisant *trop grand*, *trop petit* ou *trouvé*.

```
from random import randint
def unJeu():
    n=randint(0,100)
    essai=0
    trouve=False
    while trouve==False:
        rep=int(input('Donne un nombre entre 0 et 100'))
        essai+=1
        if rep<n:
                print('Trop petit !')
        elif rep>n:
                print('Trop grand !')
        else:
                trouve=True
                print(essai,' tentatives,  bravo !')
```

La méthode triviale est de proposer les entiers les uns après les autres.
Au mieux, on a le résultat en un essai. Au pire, on l'obtient en 101 essais.
En moyenne, on peut espérer 51 essais. Pas très efficace.
.
Une stratégie classique plus intéressante : **diviser pour mieux reigner**.
Prenons l'élément central.
Si l'élément central n'est pas celui cherché alors on sait si celui-ci est dans la moitié supérieure ou l'autre.
C'est **l'algorithme de dichotomie**.
En un essai, on divise par deux la quantité d'informations à traiter.
Avec cette méthode, au pire des cas on trouve le nombre cherché en 7 essais : les longueurs des listes à tester sont 50 puis 25 puis 13 puis 7 puis 4 puis 2 puis 1.
Un petit calcul avec votre calculatrice : $ln(101)/ln(2)$ et on obtient ...

2 Cas général

On considère une liste L de n nombres triée et on veut déterminer si une valeur m appartient à L.

L'algorithme de dichotomie s'écrit :

```
debut=0
fin=longueur de la liste L
Tant que debut<=fin faire
        m=(debut+fin)//2
        si L[m]==v alors
                retourner m et "trouve"
        sinon si L[m]<v alors
                debut=m+1
        sinon
                fin=m-1
Fin Tant que
retourner "pas trouve"
```

m donne la position centrale en faisant la moyenne de les positions du premier et dernier élément de la liste L : on récupère un quotient entier pour avoir effectivement un entier compris entre 0 et n-1.

Si v n'est pas à la position m,

soit L[m]<v et dans ce cas v ne peut être que dans L[m+1 :fin]

soit L[m]>v et dans ce cas v ne peut être que dans L[debut :m-1]

Dans la boucle *Tant que*, on a comme invariant :

0<=debut et fin<=n-1
v est à trouver dans L[debut :fin]

En Python :

```
def recherche_dicho(L,v):
        debut=0
        fin=len(L)-1
        while debut<=fin:
                m=(debut+fin)//2
                if L[m]==v:
                        return m
                elif L[m]<v:
                        debut=m
                else:
                        fin=m
        return False
```

3 Coût

Dans le meilleur des cas, v est L[m] avec m=(0+n-1)//2. On a donc un seul passage dans la boucle Tant que avec le calcul donnant m et une comparaison.

Dans le pire des cas, v n'est pas dans L[m] et on effectue à chaque passage dans la boucle le calcul donnant m et deux comparaisons.

Après chaque passage, on a divisé par deux la longueur de liste dans laquelle on recherche v.

Donc n/2 puis n/2/2 soit n/4 ... jusqu'à avoir n/2/2/.../2 <= 1.

Si k est le nombre d'étapes dans le pire des cas, on a donc $\frac{n}{2^k} \leqslant 1$ soit $n \leqslant 2^k$.

En mathématiques, en utilisant la fonction ln, on obtient $\frac{\ln n}{\ln 2} \leqslant k$.

Par exemple, pour n=128, on a $k \geqslant 7$ donc en 7 étapes maximum on détermine si v est dans la liste L.

Pour $n = 1024$, on a 10 étapes. Pour $n = 2^k$, on a k étapes. On dit que le coût est **logarithmique**.

4 En mathématiques

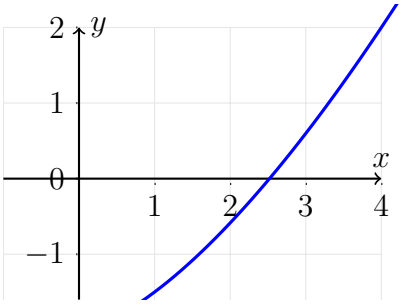
Beaucoup de problèmes se ramène à résoudre une équation $f(x) = 0$ où f est une fonction définie sur un intervalle I. Dans certaines conditions, on peut appliquer un algorithme dichotomique pour déterminer une valeur approchée d'une solution de cette équation avec une certaine précision :

La fonction f est définie continue strictement croissante sur un intervalle $[a;b]$ avec $f(a) < 0 < f(b)$.

ou

La fonction f est définie continue strictement décroissante sur un intervalle $[a;b]$ avec $f(a) > 0 > f(b)$.

Plaçons dans le premier cas avec $f(x) = \frac{x\sqrt{x} - 4}{2}$ sur $I = [1;3]$:



Le programme suivant fournit une approximation à une précision e près :

```
from math import sqrt
def f(x):
        return (x*sqrt(x)-4)/2
def solution(e):
        debut=1
        fin=3
        while fin-debut>e:
                m=(debut+fin)/2
                if f(m)>0:
                        fin=m
                else:
                        debut=m
        return m
```