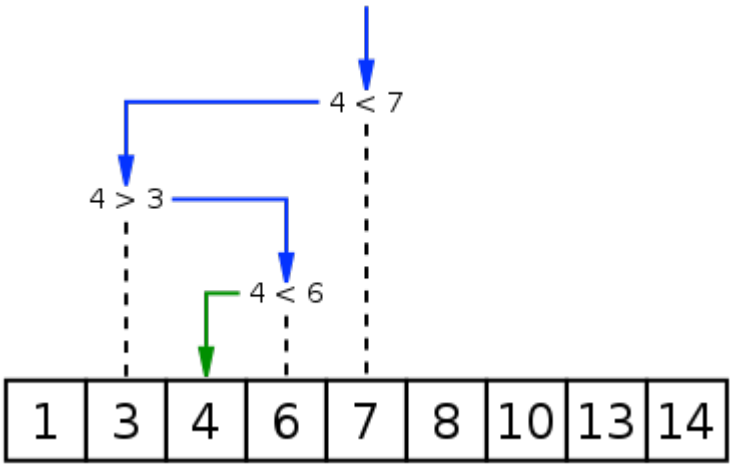
Les tableaux

Algorithme de recherche par DICHOTOMIE




```
def recherche_dichotomique(x,a):
    g,d=0,len(a)-1
    while g<=d:
        m=(g+d)//2
        if a[m] == x:
            return "OUI"
        if a[m]<x:
            g=m+1
        else:
            d=m-1
    return "NON"
```

Les tableaux – Algorithme de recherche par DICHOTOMIE

1. Les tableaux – Compléments sur les listes

1.1. Construction d'un tableau

De manière simple, un tableau n'est rien d'autre qu'une suite ***de valeurs stockées dans des cases mémoire contiguës***. Un tableau est une liste ordonnée de ***nombres uniquement*** (flottants, entiers, complexes voire booléens codés en binaire) :

3	7	42	1	4	8	12

Dans python, un tel tableau est défini avec ***crochets et virgules***, se sont en fait des ***listes de nombres.***

```
>>> a=[3,7,42,1,4,8,12]
>>> a[4]
4
>>> a[-1]
12
>>> len(a)
7
```

Il existe d'autres procédés pour construire un tableau que d'énumérer explicitement ses éléments. On peut par exemple construire un tableau en ***concaténant un certain nombre de fois un tableau donné***. Ainsi, l'expression suivante construit un tableau de taille 100:

```
>>> 50*[0,1]
[0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0
, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
1]
```

On peut utiliser aussi une construction ***par compréhension*** de la forme `[e(i) for i in range(n)]`, où `e` est une expression quelconque, qui construit le tableau `[e(0), e(1),....e(n-1)]`. Ainsi, l'expression suivante construit le même tableau que précédemment:

```
>>> [i%2 for i in range(100)]
[0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1]
```

1.2. Bibliothèque Numpy

Les tableaux sont aussi accessibles à partir de la librairie ***Numpy*** qui doit être préalablement chargée.
✔ Un tableau dans Numpy s'appelle « `array` ».
✔ Il est possible de construire un tableau de manière explicite. Comme les listes, il s'agit d'un nouveau type de variable.

On verra plus tard l'utilisation de cette bibliothèque, mais voici quelques exemples :

Les tableaux – Algorithme de recherche par DICHOTOMIE

Exemple : construction tableau avec `numpy`

```
from numpy import *
a = array([1,2,3,4,5,1,9])
print(a)
print(type(a))

print()

t=[1,2,3,4,5,1,9]
print(t)
print(type(t))
```

```
>>>
[1 2 3 4 5 1 9]
<class 'numpy.ndarray'>

[1, 2, 3, 4, 5, 1, 9]
<class 'list'>
>>>
```

Les tableaux créés à partir de listes sont suite ordonnée de nombres séparés par des virgules, les tableaux array sont juste une suite de nombres.

PS: on verra plus tard dans le semestre l'utilisation des tableaux `array`.

Ne pas utiliser array pour le moment

1.3. Parcours de tous les éléments d'un tableau type liste (rappels)

La méthode utilisée pour parcourir les éléments d'un tableau consiste à utiliser une boucle `FOR`. La construction `for i in range(n)` permet d'affecter successivement à la variable `i` les valeurs `0,1...,n-1`. Ainsi on peut écrire une fonction somme ainsi:

```
def somme(a):
    s=0
    for i in range (len(a)):
        s=s+a[i]
    return s

a=eval(input("entrez un tableau de valeurs: "))
print(somme(a))
```

```
>>>
entrez un tableau de valeurs: [3,7,14,45]
69
>>>
```

On peut faire encore plus simple dans python avec `for x in a :`

```
def somme(a):
    s=0
    for x in a:
        s=s+x
    return s

a=eval(input("entrez un tableau de valeurs: "))
print(somme(a))
```

```
>>>
entrez un tableau de valeurs: [3,7,14,45]
69
>>>
```

1.4. Recherche dans un tableau - par balayage

Supposons que l'on veuille ***déterminer si un tableau contient une certaine valeur***. On ne va pas nécessairement examiner tous les éléments du tableau, car on ***souhaite interrompre le parcours dès que l'élément est trouvé.***

Les tableaux – Algorithme de recherche par DICHOTOMIE

Voici une première solution avec une boucle `for`:

```
def appartient(x,a):
    for y in a:
        if y==x:
            return True
    return False

a=eval(input("entrez un tableau de valeurs: "))
x=eval(input("entrez la valeur recherchée: "))
print(appartient(x,a))
```

```
>>>
entrez un tableau de valeurs: [12,7,8,2,15]
entrez la valeur recherchée: 8
True
>>>
```

Ecrivons maintenant une fonction de recherche légèrement différente, qui ***renvoie l'indice où la valeur x apparait*** dans le tableau. On utilise `enumerate`.

```
def indice(x,a):
    for i,y in enumerate(a):
        if y==x:
            return i
    return None

a=eval(input("entrez un tableau de valeurs: "))
x=eval(input("entrez la valeur recherchée: "))
print(indice(x,a))
```

```
>>>
entrez un tableau de valeurs: [12,5,78,14]
entrez la valeur recherchée: 14
3
>>>
```

2. Recherche dans un tableau par dichotomie

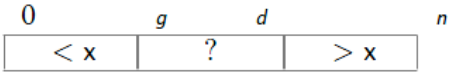
On note que dans les fonctions précédentes, on parcours au pire tout le tableau en effectuant n comparaisons. La recherche peut être plus efficace avec un ***tableau trié.***

Idée: on coupe le tableau en 2 par le milieu et on détermine si la valeur de x doit être recherchée dans moitié G ou D. Il suffit pour cela de la comparer avec la valeur centrale. Puis on repète le processus sur la portion sélectionnée.

Supposons que l'on cherche la valeur 9 dans le tableau `[1,3,5,6,9,12,14].` La recherche s'effectue ainsi:

on cherche dans a[0:7]	1	3	5	6	9	12	14
on compare x=9 avec a[3]=6	1	3	5	**6**	9	12	14
on cherche dans a[4:7]					9	12	14
on compare x=9 avec a[5]=12					9	**12**	14
on cherche dans a[4:4]					9		
on compare x=9 avec a[4]=9					**9**		

Pour écrire l'algorithme, on délimite la portion du tableau a dans laquelle la valeur x doit être recherchée à l'aide de 2 indices g et d. On maintient l'invariant suivant: les valeurs strictement à G de g sont inférieures à x, celles à D de d sont supérieures à x, ce qui s'illustre ainsi:



Les tableaux – Algorithme de recherche par DICHOTOMIE

Voici le programme python:

```
def recherche_dichotomique(x,a):
    g,d=0,len(a)-1
    while g<=d:
        m=(g+d)//2
        if a[m] == x:
            return "OUI"
        if a[m]<x:
            g=m+1
        else:
            d=m-1
    return "NON"

#main
tableau=eval(input("entrez un tableau trié: "))
valeur=eval(input("entrez le chiffre à rechercher: "))

print("le chiffre",valeur,"est dans le tableau ? ",recherche_dichotomique(valeur,tableau))
```

initialise `g=0` et `d=longueur de a-1`

tant que portion considérée contient au moins 1 élément

on calcule l'indice de l'élément central (div. entière)

si `a[m]` est l'élément recherché, on termine

sinon, on détermine si recherche doit être poursuivie à G ou D

si on sort boucle `while`: élément pas dans tableau, renvoie none

3. Recherche d'une séquence de valeur dans un tableau

Un problème classique en informatique consiste à ***rechercher***, non pas une seule valeur, mais ***une séquence de valeurs dans un tableau***. Cela revient à ***chercher l'occurrence d'un tableau dans un autre***.

On souhaite donc écrire une fonction `recherche_mot` qui, étant donnés 2 tableaux `m` et `t`, détermine la position de la 1ère occurrence de `m` dans `t` (1er indice), si elle existe, et qui renvoie None sinon. Ainsi, pour les tableaux `m=[1,2,3]` et `t=[2,1,4,1,2,6,1,2,3,7]`, ***la fonction renvoie 6:***

`[2, 1, 4, 1, 2, 6, 1, 2, 3, 7]`

Voici le programme python:

```
def recherche_mot(m,t):
    #renvoie la position de la 1ère occurence du mot m dans un texte t, et renvoie None sinon

    for i in range(1+len(t)-len(m)):
        j=0
        while j < len(m) and m[j] == t[i+j]:
            j=j+1
        if j==len(m):
            return i
    return None

#main

t=[2,1,4,1,2,6,1,2,3,7]
m=[1,2,3]

print("la position de la 1ère occurence du mot",m," dans le texte ",t,"est: ",recherche_mot(m,t))
```

Cela nous donne en compilant :

```
>>>
la position de la 1ère occurence du mot [1, 2, 3]  dans le texte  [2, 1, 4, 1, 2, 6, 1, 2, 3, 7] est:  6
>>>
```

Les tableaux – Algorithme de recherche par DICHOTOMIE

Explications sur l'algorithme

1) On effectue la recherche avec une boucle for, qui va considérer toutes les positions possibles pour le mot m, c'est à dire tous les indices `i` entre `0` et `len(t)-len(m)`, au sens large. Soit pour notre cas : ***8 essais possibles.***

2) On teste si le mot m apparaît à la position i avec une seconde boucle, qui compare les caractères de m et de t un à un. On utilise une variable `j` pour cela et on s'arrête soit lorsque `j` atteint `len(m),` soit lorsque les caractères diffèrent.

3) Une fois sorti de la boucle `while`, on a reconnu le mot m à la position `i` si et seulement si `j==len(m)`, auquel cas on renvoie `i`.

4) Sinon, on passe à la valeur suivante de `i`. Si on parvient à la fin de la boucle for principale, c'est qu'il n'y a pas d'occurrence de `i` dans `t`, on renvoie `None`.

Autre solution pour la recherche d'une suite de nombres :

```
def recherche_mot(m,t):
    for i in range(len(t)-len(m)):
        n=t[i:(i+len(m))]
        if n==m:
            return i
    return None

#main

t=[2,1,4,1,2,6,1,2,3,7]
m=[1,2,3]

print("la position de la 1ere occurence du mot",m," dans le texte ",t,"est: ",recherche_mot(m,t))
```

SAVOIR-FAIRE Concevoir un algorithme répondant à un problème précisément posé

1. Identifier la structure adaptée pour représenter les données du problème (par exemple un tableau).
2. Déterminer si le problème peut se ramener à un des algorithmes usuels sur cette structure (par exemple le parcours du tableau).
3. Apporter les modifications nécessaires à cet algorithme pour répondre au problème.

Les tableaux – Algorithme de recherche par DICHOTOMIE

Exercice : Concevoir une fonction permettant de vérifier qu'une suite, saisie par l'utilisateur sous forme de tableau, est croissante.

1. *On peut représenter les termes de la suite comme un tableau de flottants* `u`.
2. *Si la suite est effectivement croissante il faudra le vérifier à chaque rang, mais si elle ne l'est pas on pourra interrompre le parcours du tableau dès qu'on aura trouvé deux valeurs en ordre décroissant. L'algorithme que l'on va écrire est donc à rapprocher d'une recherche par balayage.*
3. *Il y a principalement deux différences avec la recherche par balayage. D'une part, on ne va pas comparer un élément avec une valeur fixée, mais avec l'élément suivant. D'autre part, on veut savoir si la suite est croissante, et donc on renverra* `True` *dans le cas où on a parcouru tout le tableau sans trouver de valeurs en ordre décroissant.*

```
def croissante(u):
    for i in range(len(u)-1):
        if u[i] > u[i+1]:
            return False
    return True

u=eval(input("entrez une suite d'entiers sous forme de tableau: "))
print("la suite est croissante: ",croissante(u))
```

```
>>>
entrez une suite d'entiers sous forme de tableau: [12,2,15,16]
la suite est croissante:  False

>>>
entrez une suite d'entiers sous forme de tableau: [1,5,9,15,78]
la suite est croissante:  True
```

ATTENTION Les tableaux Python

Python propose plusieurs structures de tableaux, chacune prévue pour un usage spécifique. Dans ce chapitre, nous avons utilisé les « listes », dont on verra au chapitre qu'elles sont en fait un peu plus que de simples tableaux.

Il existe également une bibliothèque appelée `array` qui, comme son nom l'indique, correspond effectivement à une structure de tableaux. Mais il s'agit uniquement d'une représentation plus compacte pour des tableaux très spécifiques, dont les éléments sont tous d'un même type numérique simple (caractère, entier ou flottant). En particulier, nous ne pourrions pas utiliser la bibliothèque `array` pour représenter des matrices.

Enfin la bibliothèque `numpy` propose aussi un type *ndarray* qui permet de représenter des tableaux de dimension arbitraire dont tous les éléments sont d'un même type, principalement utilisés pour le calcul matriciel. Attention à ne pas confondre ce type avec le précédent, même si ses valeurs s'affichent parfois aussi sous la forme `array(...)`.

Exercice 64 : Dichotomie

Réalisez un programme permettant de rechercher par dichotomie, dans le tableau suivant donné: [12,18,47,68,79], si les nombres 68 et 44 sont présents. Le résultat affiché devra être:

le chiffre 68 est présent dans le tableau [12,18,47,68,79]
NON, le chiffre 44 n'est pas présent dans le tableau [12,18,47,68,79]

Exercice 65 : Décomposition d'un nombre en produits de facteurs

Tout entier naturel peut s'écrire comme produits de nombres premiers sous la forme :

$$\prod_{k=1}^{m} p_k^{a_k} \qquad \forall k \in [1,m]$$

$1 \leq a_k$ et p_k nombre premier

On demande d'écrire un programme qui effectue cette décomposition pour un entier n donné. Le principe de le programme est de tester la divisibilité de n successivement par les nombres premiers pris dans l'ordre croissant et à répéter l'opération avec le dividende jusqu'à obtenir 1.

par exemple : $87\,464 = 2^3 * 13 * 29^2$

1) Ecrire une fonction qui construit la liste *premier* dans laquelle sont marqués les nombres premiers inférieurs à n
2) Écrire le programme qui construit la liste décrivant la décomposition d'un entier n en produits de facteurs, pour chaque diviseur successif de ce nombre on donnera la puissance avec laquelle il le divise.

Exercice 66 Interpolation dans un tableau incomplet

On dispose d'un tableau incomplet dans lequel sont stockées des valeurs de Perlimpimpinate de Potassium anhydre relevées au cours d'une série d'expériences. Les valeurs inconnues sont notées par une valeur négative.

Exemple :

1	-10	-10	7	-10	12	-10	-10	-10	28

Ecrire une fonction qui complète par interpolation linéaire un tel tableau.
Pour cela, on peut procéder en calculant la différence entre les 2 plus proches valeurs connues encadrant une valeur inconnue et en répartissant cette distance sur les valeurs inconnues dans cet intervalle.
On envisagera une version simplifiée du problème dans laquelle le tableau ne commence et ne se termine jamais par des valeurs négatives.

Exemple : L'interpolation du tableau précédent donnerait :

1	3	5	7	9.5	12	16	20	24	28

Exercice 63 : Période d'un nombre rationnel

Un nombre rationnel est donné par son numérateur et son dénominateur (entiers). Tout nombre rationnel admet un développement décimal périodique.

Exemples: *la période de 74/14 est : 285714* *La période de 73 / 6 est 6*

```
74  | 14
 40 | 5,285714
 120
  80
 100
  20
  60
   4
```

```
73  | 6
    | 12,16
 10
 40
  4
```

On souhaite un programme qui donne ce développement pour un nombre rationnel donné :

1) Ecrire une fonction *pgcd* en mettant en œuvre l'algorithme d'Euclide :
   Soit a et b deux entiers naturels, si b est non nul, on peut effectuer la division euclidienne de a par b. Il existe un couple unique d'entiers (q,r) tels que :
   $a = bq + r$ *et* $0 \leq r < b$
   si $r = 0$ *alors b divise a et* $PGCD(a,b) = b$
   si $r > 0$ *alors tout diviseur commun à a et b est diviseur de r et réciproquement tout diviseur commun à b et r est diviseur de a. On a donc* $PGCD(a,b) = PGCD(b,r)$.
2) Rendre la fraction irréductible en divisant le numérateur et le dénominateur par leur PGCD.
3) Ecrire la fonction qui calcule la période d'un nombre rationnel donné par son numérateur et son dénominateur.
   Pour connaître la période, il suffit d'arrêter les divisions des restes successifs par les quotients successifs lorsqu'on retrouve un reste déjà obtenu.

Indications:

✔ le numérateur et dénominateur seront saisis par appel utilisateur
✔ on pourra faire un affichage intermédiaire du pgcd de ces nombres
✔ on pourra faire un affichage intermédiaire du nouveau `num`, `deno` et `reste` (fraction irréductible)

✔ la période sera affichée dans une liste.

```
>>>
Entrez le numérateur : 74
Entrez le denominateur : 14
Le Pgcd de a et b est:  2
La fraction irréductible est donc:  37 / 7
La période du nombre rationnel est:  [2, 8, 5, 7, 1, 4]
>>>
```