On présente dans ce chapitre deux algorithmes de recherche d'un élément dans une liste ou un tableau.

1 Recherche séquentielle

1.1 Algorithme et implémentation

Bloc-Note 1 *Recherche séquentielle*

L'algorithme de **recherche séquentielle** d'un élément `e` dans une liste (ou tableau) `L` consiste en un simple balayage de `L` de son début jusqu'à sa fin en comparant chaque élément lu avec l'élément recherché.
On peut sortir prématurément de la boucle de balayage dès que l'élément `e` a été trouvé.

Exemple 1

1. Implémenter la recherche séquentielle dans la fonction `recherche_seq` pour qu'elle retourne `True` si `e` appartient une liste `L` ou `False`.

```
def recherche_seq(L, e):
    trouve = False
    for element in L:
        if ..... :
            .....
    return trouve
```

2. Modifier la fonction précédente pour ne plus utiliser la variable `trouve` et retourner `True` dès que l'élément `e` a été trouvé ou `False` sinon.

```
def recherche_seq2(L, e):
    ..................
    ..................
    ..................
    ..................
    ..................
    ..................
```

3. Compléter cette autre version, sans boucle `for`, de la fonction de recherche séquentielle de l'élément `e` dans la liste `L`.

```
def recherche_seq3(L, e):
    i = 0
    n = len(L)
    while i < n and ........ :
        ......................
    return ...................
```

Peut-on placer `i < n` après le `and` dans le test de la boucle `while`?

.....

.....

1.2 Applications

Exemple 2 *Recherche d'extremum*

1. Écrire une fonction `recherche_maximum` qui s'applique à une liste de nombres (supposée non vide) et retourne son maximum.

```
def recherche_maximum(L):
    maxi = L[0]       #initialisation du maximum avec la première valeur
    ......................
    ......................
    ......................
    ......................
    return maxi
```

2. Écrire une fonction `pos_maximum` qui s'applique à une liste de nombres et retourne la liste des indices où son maximum est atteint.

```
def pos_maximum(L):
    maxi, pos = L[0], [0]  #initialisations
    ......................
    ......................
    ......................
    ......................
    ......................
    ......................
    ......................
```

3. Reprendre la fonction `recherche_maximum` du **1.** mais cette fois elle doit s'appliquer à une liste de listes à deux dimensions comme `M = [[1, 4, 3], [18, -4, -1]]`.

```
def recherche_maximum2(M):
    maxi = ..............  #initialisation
    ......................
    ......................
    ......................
    ......................
    ......................
    ......................
    return maxi
```

Exemple 3 *Histogramme*

Compléter la fonction `histogramme` qui s'applique à une liste `L` de notes qui sont des entiers entre 0 et 20 et qui retourne une liste indiquant le nombre d'occurrences de chaque note.

```
def histogramme(L):
    histo = [0] * 21
    ......................
    ......................
    return histo
```

2 Recherche dichotomique

2.1 Juste prix et tas de graine

Exemple 4

Le jeu du *juste prix* consiste à trouver le prix tenu secret d'un article sachant qu'il est compris entre une borne inférieure `prixInf` et une borne supérieure `prixSup`.
Compléter le code de la fonction `juste_prix` pour qu'elle retrouve le `prix` secret. On donne ci-contre un exemple d'exécution.

```
In [17]: juste_prix(43,1, 100)
prixInf : 1 prixSup : 100 proposition : 50
C'est moins.
prixInf : 1 prixSup : 49 proposition : 25
C'est plus.
prixInf : 26 prixSup : 49 proposition : 37
C'est plus.
Juste prix 43 trouvé en 4 essais.
```

```
def juste_prix(prix, prixInf, prixSup):
    proposition = ..................
    essai = 1
    while proposition != prix:
        print("Proposition :", proposition, "prixInf :",prixInf,
              "prixSup :", prixSup)
        if proposition > prix:
            print("C'est moins.")
            ....... = ..................
        else:
            print("C'est plus.")
            ....... = ..................
        proposition = ..................
        essai += 1
    print("Juste prix", prix, "trouvé en", essai, "essais.")
```

Exemple 5 *Tas de Graine*

Traiter l'exercice *Tas de Graine* du Castor Informatique 2017 :
https://concours.castor-informatique.fr/index.php?team=castor2017

2.2 Algorithme et implémentation

Bloc-Note 2 *Recherche dichotomique*

On considère une liste d'éléments comparables entre eux, qui est triée dans l'ordre croissant.
Un bon exemple est une liste de mots ou de noms dans l'ordre alphabétique, comme un dictionnaire ou un annuaire.
En pratique, pour rechercher un élément dans cette liste, nous utilisons un algorithme de **recherche dichotomique** plutôt qu'une **recherche séquentielle** :

- ☞ on compare l'élément recherché avec l'élément au milieu de la liste et s'ils sont égaux, on peut stopper la recherche ;

- ☞ sinon deux cas se présentent (en grec *dikha* signifie deux et *tomia* couper) :
  - l'élément recherché est plus petit que l'élément au milieu et on poursuit la **recherche dichotomique** dans la première moitié de la liste qui est la nouvelle zone de recherche ;
  - l'élément recherché est plus grand que l'élément au milieu et on poursuit la **recherche dichotomique** dans la seconde de la liste qui est la nouvelle zone de recherche ;
- ☞ on répète en boucle ces instructions jusqu'à ce que la zone de recherche ne contienne plus d'éléments ...

Dans une **recherche dichotomique** , la zone de recherche est au départ toute la liste, puis elle se réduit de moitié à chaque étape. La recherche se termine nécessairement : quand l'élément est trouvé ou quand la zone de recherche ne contient plus aucun élément.
⚠ *Une recherche dichotomique n'est possible que sur une liste ordonnée.*

Exemple 6 *Une implémentation classique*

Compléter le code de la fonction `recherche_dicho` ci-dessous qui implémente la recherche dichotomique pour retourner `True` si l'élément `e` appartient à la liste `L` et `False` sinon.

```
def recherche_dicho(L, e):
    debut = 0
    fin = len(L) - 1
    while debut <= fin:
        milieu = ..............
        if L[milieu] == e:
            return True
        elif L[milieu] < e:
            ...................
        else:
            ...................
    return False
```

Exemple 7 *Une autre implémentation*

Dans l'algorithme de recherche dichotomique, après division en deux de la zone de recherche, l'algorithme s'appelle lui-même sur l'une des deux moitiés. C'est un algorithme de type *Diviser pour régner* qui peut se programmer récursivement comme nous le verrons dans le chapitre sur la récursivité.
Si on note `n` la taille de la liste, une autre implémentation, non récursive, est la suivante :

- ☞ on commence la recherche au début de la liste et on avance avec un pas `p = n // 2` jusqu'au premier élément supérieur à l'élément cherché ;
- ☞ on repart de l'élément précédent le point d'arrêt et on avance désormais avec un pas `p = p // 2` ;
- ☞ on répète en boucle ces instructions jusqu'à ce que le pas atteigne 1.

A la fin de de la boucle, on détermine si l'élément sur lequel on s'est arrêté est l'élément recherché.
Compléter le code de la fonction `recherche_dicho2` qui implémente cet algorithme.

```
def recherche_dicho2(L, e):
    x, n, pas = 0, len(L), n // 2
    while pas >= 1:
        while x + pas < n and ..................:
            x = ..............
        pas = ................
    return ............
```

2.3 Comparaison des recherches séquentielles et dichotomiques

Exemple 8

On se donne une liste L de taille n dans laquelle on recherche un élément.

1. Combien de comparaisons faut-il effectuer avec une recherche séquentielle si l'élément recherché est en dernière position dans la liste ou n'appartient pas à la liste ?

   .....

   .....

2. Combien de comparaisons sont effectués avec une recherche dichotomique si l'élément recherché est trouvé au dernier tour de boucle ou n'appartient pas à la liste ?

   .....

   .....

   .....

   .....

3. On a représenté ci-dessous le temps d'exécution de fonctions de recherches dichotomiques ou séquentielles en fonction de la taille de la liste d'entrée.

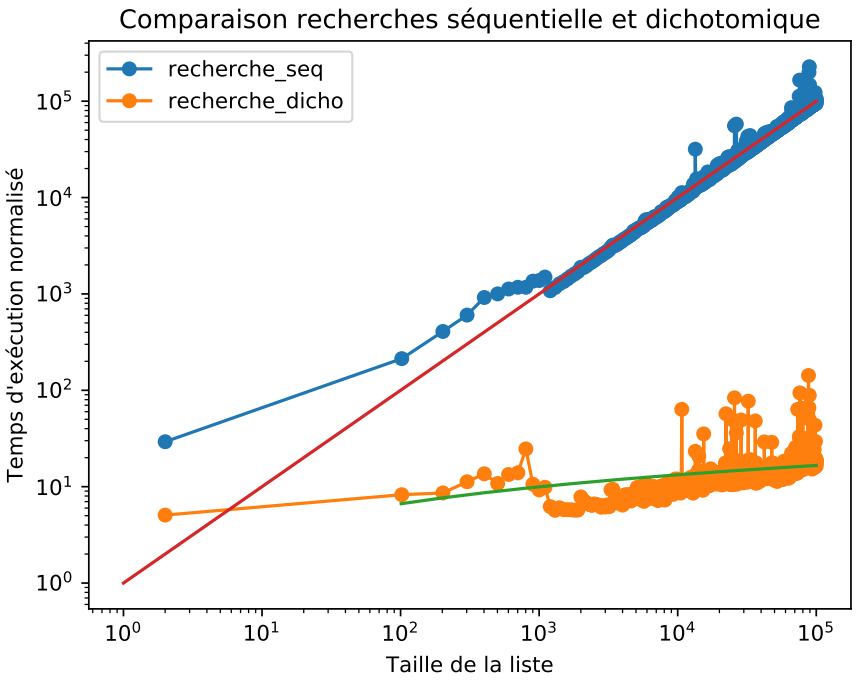
   Quels types de fonctions usuelles peuvent modéliser ces courbes ?

   .....

   .....

   .....

   .....



2.4 Application

Exemple 9 *Recherche de point d'insertion*

1. Compléter la fonction `recherche_dicho_gauche(L, x)` qui détermine l'indice du premier élément de la liste de nombres `L` triée dans l'ordre croissant qui est *supérieur ou égal* au nombre `x`. Si on veut insérer un élément dans `L`, le point d'insertion renvoyé sera à gauche de la première occurrence de `x`.

   Le module `bisect` fournit une fonction équivalente appelée `bisect_left`.

```
def dicho_gauche(L, x):
    debut = 0
    fin = len(L) - 1
    if L[fin] < x:
        return fin + 1
    ..............................
    ..............................
    ..............................
    ..............................
    ..............................
    ..............................
    ..............................
    ..............................
    ..............................
```

2. Écrire une fonction `dicho_droite(L, x)` qui détermine l'indice du premier élément de la liste de nombres `L` triée dans l'ordre croissant qui est *strictement supérieur* au nombre `x`. Si on veut insérer un élément dans `L`, le point d'insertion renvoyé sera à droite de la dernière occurrence de `x`.

   Le module `bisect` fournit une fonction équivalente appelée `bisect_right`.

```
def dicho_droite(L, x):
    ..........................
    ..........................
    ..........................
    ..........................
    ..........................
    ..........................
    ..........................
    ..........................
    ..........................
    ..........................
    ..........................
    ..........................
```

3. En déduire une fonction `dicho_comptage(L, x)` qui détermine le nombre d'éléments de la liste `L` triée dans l'ordre croissant qui sont égaux au nombre `x`.

```
def dicho_comptage(L, x):
    ..........................
    ..........................
```