

# Algorithmes et Pensée Computationnelle

## Algorithmes de recherche

Le but de cette séance est de se familiariser avec les algorithmes de recherche. Dans la série d'exercices, nous manipulerons des listes et collections en Java et Python. Nous reviendrons sur la notion de récursivité et découvrirons les arbres de recherche. Au terme de cette séance, l'étudiant sera en mesure d'effectuer des recherches de façon efficiente sur un ensemble de données.

Le code présenté dans les énoncés se trouve sur Moodle, dans le dossier **Ressources**.

## 1 Recherche séquentielle (ou recherche linéaire)

### 1.1 Définition

Une recherche **séquentielle** (ou **linéaire**) est une méthode permettant de trouver un élément dans un ensemble de données (liste, tableau ou dictionnaire). Elle vérifie un à un chaque élément de gauche à droite jusqu'à ce qu'une correspondance soit trouvée ou que toute la liste ait été parcouru.

Si l'élément recherché est trouvé, l'algorithme **renvoie l'index**, c'est-à-dire la position, de l'élément dans l'ensemble de données.

Sa complexité dans le pire des cas est de **O(n)** correspondant à la longueur de l'ensemble de données, et dans le meilleur des cas de **O(1)**, lorsque l'élément se trouve en première position. Dans la pratique, l'algorithme de recherche séquentielle n'est pas couramment utilisé eu égard de sa complexité élevée et des alternatives de recherche plus efficaces comme la recherche binaire.

### 1.2 Exercices

#### Question 1: (🕒 5 minutes) Recherche séquentielle - 1 (Python)

À partir des éléments ci-dessous, écrivez une fonction qui cherche **x** dans la liste **L**.

La fonction doit retourner l'index de l'élément correspondant de la liste si **x** est dans la liste et "-1" si **x** n'est pas dans la liste (avec **x = 100**).

**Python :**

```
1 def recherche_sequentielle(liste, x):
2     #complétez ici
3
4     L = [3,55,6,8,3,5,56,33,6,5,3,2,99,53,532,75,21,963,100,445,56,56,24]
5     x = 100
6
7     resultat = recherche_sequentielle(L, x)
8     print(resultat)
```

#### 💡 Conseil

Définissez votre fonction de recherche linéaire en utilisant une boucle **for** ou une boucle **while**.

Attention : la fonction doit retourner l'index de la valeur et non pas la valeur. Pour cela, pensez à utiliser **range(len(list))** avec la boucle **for** et une incrémentation **"i = i+1"** avec la boucle **while**.

Utilisez la fonction **print()** pour afficher l'index lorsque vous l'aurez trouvé et un autre message le cas échéant.

## >\_ Solution

### Python :

```
1 #Définition de la fonction
2 def recherche_sequentielle(liste, x):
3     for i in range(len(liste)): # i représente l'index
4         if liste[i] == x:
5             print("X est présent dans la liste à l'index :", i)
6             return i
7
8
9     print("X n'est pas présent dans la liste")
10    return -1
11
12 # Déclaration de la liste et de la variable x
13 L = [3, 55, 6, 8, 3, 5, 56, 33, 6, 5, 3, 2, 99, 53, 532, 75, 21, 963, 100, 445, 56, 45, 12, 56, 24]
14 e = 100
15
16 # Exécution de l'algorithme
17 resultat = recherche_sequentielle(L,e)
18 print(resultat)
```

### Question 2: (🕒 10 minutes) Recherche séquentielle - 2 (Python)

Soit une liste d'entiers non triée **L** ainsi qu'un entier **e**. Écrivez un programme qui retourne l'élément de la liste **L** dont la valeur est la plus proche de **e** en utilisant une recherche séquentielle.

Exemple :

**L** = [16, 2, 25, 8, 12, 31, 2, 56, 58, 63]

**e** = 50

Résultat attendu : 56

Au cas où deux éléments se trouveraient à équidistance de **e**, renvoyez l'élément le plus petit.

### Python :

```
1 #Definition de la fonction ayant pour argument une liste et un nombre
2 def plus_proche_sequentielle(list,n):
3     diff = -1 #Initialisation de la variable (-1 car les différence calculée après seront toujours positives)
4     resultat = None #Initialisation de la variable pour le résultat
5
6     #Complétez ici
7
8     #Déclaration de la liste et de la variable e
9     L = [16, 2, 25, 8, 12, 31, 2, 56, 58, 63]
10    e = 50
11
12    #Exécution de la fonction
13    resultat = plus_proche_sequentielle(L,e)
14    print(resultat)
```

### Conseil

Complétez la fonction `plus_proche_sequentielle` et exécutez le code.

Veillez à utiliser les valeurs absolues pour comparer les différences, la plus petite pouvant être positive ou négative. En Python, la fonction `abs()` retourne la valeur absolue. Exemple : `abs(3-10)` retourne 7.

Étant donné que la liste est **non triée**, l'algorithme doit obligatoirement la parcourir intégralement.

L'algorithme doit calculer la différence entre `e` et chaque élément de la liste `L` en gardant toujours la plus petite différence trouvée. À la fin, il retourne l'élément de la liste correspondant à la plus petite différence.

### >\_ Solution

#### Python :

```
1 #Definition de la fonction ayant pour argument une liste et un nombre
2 def plus_proche_sequentielle(liste,n):
3     diff = -1 #Initialisation de la variable ( -1 car les différences calculées après seront toujours positives)
4     resultat = None #Initialisation de la variable pour le résultat
5
6     #Solution
7     for i in liste:
8         if diff == -1 or abs(i-n) < diff:
9             diff = abs(i-n) #new diff #
10            resultat = i
11
12     return resultat
13
14 #Déclaration de la liste L et de la variable e
15 L = [16, 2, 25, 8, 12, 31, 2, 56, 58, 63]
16 e = 50
17
18 #Exécution de la fonction
19 resultat = plus_proche_sequentielle(L,e)
20 print(resultat)
```

### Question 3: 5 minutes) Recherche séquentielle - 3 (Python)

Considérez une **liste d'entiers triés** `L` ainsi qu'un entier `e`. Écrivez un programme qui retourne l'index de l'élément `e` de la liste `L` en utilisant une recherche séquentielle. Si `e` n'est pas dans `L`, retournez -1.

### >\_Exemple

```
L = [1231321,3213125,3284016,4729273,5492710]
e = 3284016
Résultat attendu : 2
```

```
1 def recherche_sequentielle(L,e):
2     for i in L: #Ici, i correspond à la valeur et non l'index.
3         #complétez ici
4
5     L = [1231321,3213125,3284016,4729273,5492710]
6     e = 3284016
7     resultat = recherche_sequentielle(L,e)
8     print(resultat)
```

### Conseil

Une liste triée permet une recherche plus efficace à l'aide d'un algorithme plus simple. Retournez l'index de la valeur dans la liste. Pensez à utiliser la fonction `index()` qui retourne l'index d'un élément au sein d'une liste en Python.

Exemple et syntaxe : `lst.index(i)` va indiquer la position de l'élément `i` dans la liste `lst`.

### >\_ Solution

#### Python :

```
1 def recherche_sequentielle(L,e):
2     for i in L:
3         #Solution
4         if i == e:
5             return L.index(i) #L'algorithme prend fin aussitot que la valeur recherchée est trouvée.
6     return -1 # Si la valeur n'a pas été trouvée, la fonction retourne -1
7
8 L = [1231321,3213125,3284016,4729273,5492710]
9 e = 3284016
10 resultat = recherche_sequentielle(L,e)
11 print(resultat)
```

## 2 Recherche binaire

### 2.1 Définition

Le but de la recherche binaire est de trouver l'élément recherché de façon optimale. Pour cela, il est nécessaire d'utiliser **une liste d'éléments triés**.

La complexité de l'algorithme de recherche binaire est  $O(\log n)$ . Cependant, il ne faut pas oublier le coût lié à l'obtention d'une liste triée à partir d'une liste non triée.

L'algorithme de recherche binaire divise l'intervalle de recherche par deux à chaque itération jusqu'à ce qu'il trouve l'élément  $x$  ou que l'intervalle soit vide.

Ainsi, si  $x$  est plus petit que l'élément du milieu, l'algorithme va choisir la moitié de gauche comme intervalle de recherche et ainsi de suite. Si  $x$  est plus grand que l'élément du milieu la recherche va se faire dans la moitié droite de l'intervalle.

La recherche binaire se base sur les comparaisons d'ordre, alors que la recherche séquentielle se base les comparaisons d'égalité pour trouver l'élément recherché.

### 2.2 La récursivité

Une fonction récursive est une fonction qui s'appelle elle-même pendant son exécution. Vous trouverez ci-dessous un exemple de fonction récursive utilisée pour effectuer un calcul factoriel.

(Rappel :  $4! = 4 \times 3 \times 2 \times 1 = 24$ )

```
1 def factoriel(n):
2     if n == 1:
3         return n
4     else:
5         return n * factoriel(n - 1)
6
7 #Exécution de la fonction
8 factorielle(4)
9
10 #La fonction retourne 24
```

Les fonctions récursives sont courantes en informatique car elles permettent aux programmeurs d'écrire des programmes efficaces en utilisant une quantité minimale de code. Leur principal inconvénient est le fait qu'elles peuvent provoquer des exécutions infinies et d'autres résultats inattendus si elles ne sont pas écrites correctement. Si la fonction n'inclut pas les cas permettant d'arrêter la récursivité, celle-ci se répétera à l'infini, provoquant le plantage du programme ou, pire encore, l'arrêt de tout le système informatique.

### 2.3 Exercices

**Question 4:** (🕒 20 minutes) Récursivité et itération (Python)

À partir de la liste d'éléments **triés** ci-dessous, écrivez premièrement **une fonction récursive** puis **une fonction itérative** qui cherche  $x$  dans la liste. La fonction doit retourner l'**index** de l'élément correspondant de la liste si  $x$  est dans la liste et  $-1$  dans le cas contraire.

Ici,  $x = 5$ .

**Question 4.1 - Version récursive**

```
1 #Version récursive
2
3 def recherche_binaire_recursive(liste,s,r,x):
4     #complétez ici
5
6 L=[1,3,4,5,7,8,9,15]
```

```

7 s = 0
8 r = len(L)
9 x = 5
10 recherche_binaire_recursive(L,s, r, x)

```

#### Question 4.2 - Version itérative

```

1 #Version itérative
2
3 def recherche_binaire_iterative(liste,s,r,x):
4     #complétez ici
5
6     L = [1,3,4,5,7,8,9,15]
7     s = 0
8     r = len(L)-1
9     x = 7
10    recherche_binaire_iterative(L,s, r, x)

```

#### Conseil

Complétez la fonction récursive `recherche_binaire_recursive` et la fonction itérative `recherche_binaire_iterative`.

Détails sur les arguments de la fonction :

**L** : La liste dans laquelle nous effectuons la recherche.

**s** : L'élément de départ de la partie de la liste à fouiller (0 au départ).

**r** : Le dernier élément de la partie de la liste à fouiller (len(liste) au départ).

**x** : La valeur recherchée.

Ainsi, à chaque itération, vos fonctions vont modifier les valeurs de base données en argument pour resserrer l'intervalle jusqu'à trouver la valeur recherchée.

Pour définir le milieu d'un intervalle qui contient un nombre pair ou impair d'éléments, divisez l'ensemble en 2 et utilisez la fonction `int()` pour convertir le résultat en entier.

Exemple :

`liste1 = [1,2,3,4,5]`

`s = 0`

`r = len(liste1)`

Calcul du milieu de l'intervalle :

$(s+r)/2 = (0+5)/2 = 2.5$  #Pas de correspondance

`int((s+r)/2) = int((0+5)/2) = int(2.5) = 2`

Arrondi vers le bas

Pour la version itérative, il est conseillé d'utiliser une boucle `while`.

## >\_ Solution

### 4.1 - Version récursive

```
1
2 def recherche_binaire_recursive(liste,s,r,x): # s représente le premier index et r le dernier index de la liste
3     #SOLUTION
4     mid = int((s+r)/2) #int arrondi vers le bas
5     print(mid)
6     if liste[mid] < x:
7         return recherche_binaire_recursive(liste, mid, r, x)
8     elif liste[mid] > x:
9         return recherche_binaire_recursive(liste, s, mid, x)
10    elif liste[mid] == x:
11        print("X dans liste à l'index: ", mid)
12        return mid
13    else:
14        print("X absent de liste")
15        return -1
16
17 L=[1,3,4,5,7,8,9,15]
18 s = 0
19 r = len(L) #8 --> première moitié = 4 --> "7" in L --> 7>5 --> deuxième moitié = (0+4)/2 = 2, etc.
20 x = 5
21 recherche_binaire_recursive(L,s, r, x)
```

## >\_ Solution

### 4.2 - Version itérative

```
1
2 def recherche_binaire_iterative(liste,s,r,x):
3     #SOLUTION
4     while s <= r:
5         mid = int(s + (r-s)/2)
6         print(mid)
7         if liste[mid] == x:
8             print("X dans liste à l'index: ", mid)
9             return mid
10        elif liste[mid] < x:
11            s = mid+1
12        else:
13            r = mid-1
14        print("X absent de liste")
15        return -1
16
17 L = [1,3,4,5,7,8,9,15]
18 s = 0
19 r = len(L)
20 x = 15
21 recherche_binaire_iterative(L,s, r, x)
```

**Question 5:** (🕒 15 minutes) Recherche binaire - plus proche élément (Python)

Soit une liste d'entiers **triés** **L** ainsi qu'un entier **e**. Écrivez un programme retournant la valeur dans **L** la plus proche de **e** en utilisant une recherche binaire (binary search).

```
1 def plus_proche_binaire(liste,n):
2     #complétez ici
3
4
5     L = [1, 2, 5, 8, 12, 16, 24, 56, 58, 63]
6     e = 41
7     print(plus_proche_binaire(L,e))
8     # Résultat attendu : 56\
```

### 💡 Conseil

Pensez à définir des variables `min` et `max` délimitant l'intervalle de recherche et une variable booléenne `found` initialisée `false` et qui devient `true` lorsque l'algorithme a trouvé la valeur la plus proche de `e`.

### >\_ Solution

#### Python :

```
1 def plus_proche_binaire(liste, n):
2     # SOLUTION
3
4     min = 0
5     max = len(liste)
6     found = False
7     while min <= max and not found: # 0<10 and true puis 6<10 and true, etc.
8         mid = (max + min) // 2 # mid = 5 --> 16 in list
9         print(mid)
10        if n > liste[mid]: # 41>16
11            min = mid + 1 # min = 5+1=6
12        elif n < liste[mid]:
13            max = mid - 1
14        else:
15            found = True
16        if found:
17            return n
18        else:
19            if abs(liste[mid-1]-n) < abs(liste[mid]-n):
20                return liste[mid-1]
21            elif abs(liste[mid+1]-n) < abs(liste[mid]-n):
22                return liste[mid+1]
23            else:
24                return liste[mid]
25
26
27 L = [1, 2, 5, 8, 12, 16, 24, 56, 58, 63]
28 e = 41
29 print(plus_proche_binaire(L, e))
```

### Question 6: (🕒 10 minutes) Recherche binaire (Python)

Considérez une liste d'entiers triés `L` ainsi qu'un entier `e`. Écrivez un programme qui retourne l'index de l'élément `e` de la liste `L` en utilisant une recherche binaire. Si `e` n'est pas dans `L`, retournez `-1`.

### >\_ Exemple

`L = [1231321,3213125,3284016,4729273,5492710]`

`e = 3284016`

Résultat attendu : 2

```
1 def recherche_binaire(liste,e):
2     first = 0
3     last = len(L)-1
4
5     #complétez ici
6
7
8 L = [1231321,3213125,3284016,4729273,5492710]
9 e = 3284016
10 recherche_binaire(L,e)
```



### Conseil

Inspirez-vous des exercices et des conseils précédents.

### >\_ Solution

#### Python :

```
1 def recherche_binaire(liste,e):
2     first = 0 #correspond à l'index du premier élément de la liste
3     last = len(liste) #correspond à l'index du dernier élément de la liste
4     #SOLUTION
5     while first <= last:
6         mid = int((first+last)/2) #l'élément du milieu. La fonction int() permet d'obtenir un entier dans le cas ou
7         # "first + last" est un nombre impair
8         print(mid)
9         if liste[mid] == e:
10            return mid #Si la condition est juste, la fonction retourne l'index de la valeur recherchée dans la liste
11        else:
12            if liste[mid] > e:
13                last = mid - 1
14            else:
15                first = mid + 1
16        return -1 # Si la condition de la ligne 8 n'est jamais remplie, la fonction retourne -1
17
18 L = [1231321,3213125,3284016,4729273,5492710]
19 e = 3284016
20 recherche_binaire(L,e)
```

### Question 7: ( 20 minutes) Recherche matricielle (Python) Optionnel

#### Matrice en Python

Considérez une matrice ordonnée **m** et un élément **l**.

Pour rappel, une matrice ordonnée répond aux critères suivants :

$[i][j] \leq m[i+1][j]$  (une ligne va du plus petit au plus grand)

$[i][j] \leq m[i][j+1]$  (une colonne va du plus petit au plus grand)

Écrivez un algorithme qui retourne la position de l'élément **l** dans **m**. Si **l** n'est pas présent dans **m** alors il faut retourner  $(-1, -1)$

### >\_ Exemples

**Exemple 1 :** si  $m = [[1,2,3,4],[4,5,7,8],[5,6,8,10],[6,7,9,11]]$  et que  $l=7$ . Nous souhaitons avoir la réponse (1,2) OU (3,1) (l'une des deux, pas besoin de retourner les deux résultats).

**Exemple 2 :** si  $m = [[1,2],[3,4]]$  et que  $l=7$ . Nous souhaitons avoir la réponse  $(-1,-1)$  car 7 n'est pas dans la matrice **m**.

```
1 def recherche_matricielle(m,l):
2     #complétez ici
3
4
5     m=[[1,2,3,4],[4,5,7,8],[5,6,8,10],[6,7,9,11]]
6     l=7
7     recherche_matricielle(m,l)
```

### Conseil

Pour cet exercice, il est nécessaire d'utiliser des boucles **for** imbriquées, c'est-à-dire : une boucle **for** dans une autre boucle **for**. Cela permet de parcourir tous les éléments d'une liste (ou d'un tableau) à deux dimensions (dans notre cas une matrice).

## >\_ Solution

### Python :

```
1 def recherche_matricielle(m,l):
2     #complétez ici
3     for i in range(len(m)): # "Pour chaque ligne de la matrice" ou "Pour chaque liste de la liste"
4         for j in range(len(m[i])): # "Pour chaque colonne de la matrice" ou "Pour chaque élément de la liste"
5             if m[i][j] == l:
6                 return (i,j)#
7     return (-1,-1)
8
9 m=[[1,2,3,4],[4,5,7,8],[5,6,8,10],[6,7,9,11]]
10 l=7
11 recherche_matricielle(m,l)
```

## 3 Arbre de recherche binaire

### 3.1 Définition

Un arbre de recherche binaire est une structure de données au même titre que les listes, tuples, dictionnaires, etc. Leur particularité est qu'au lieu d'ordonner les éléments les uns à la suite des autres, les arbres de recherche binaire enregistrent les valeurs de manière relationnelle.

En effet, l'arbre est divisé en branches qui peuvent elles-même contenir deux branches (enfants) et ainsi de suite. Lorsqu'une branche n'a pas de branches subséquentes (enfants), on parle de feuille.

Les branches peuvent donc contenir de 0 à 2 autres branches. On parle alors de branche de gauche et de celle de droite. La branche de gauche est forcément plus petite que la branche parente et celle de droite est forcément plus grande.

De cette façon, on garantit que l'arbre est toujours ordonné même en rajoutant ou en retirant des éléments

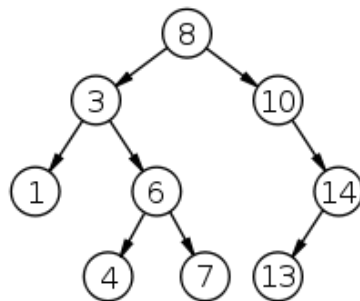


FIGURE 1 – Exemple d'arbre binaire

### 3.2 Exercices

**Question 8:** (🕒 15 minutes) Recherche dans un arbre - Python

Dans l'exercice suivant, nous vous donnons un arbre binaire avec les mêmes valeurs que le schéma précédent et dont la racine est la variable `root`.

Écrivez une fonction qui retourne `True` si la valeur `value` est dans l'arbre et `False` dans le cas contraire. Vous pouvez utiliser la variable `value` pour afficher la valeur d'un nœud (`node`) et les variables `left` et `right` sur les `nodes` pour accéder aux branches enfants, utilisez `node.value`, `node.left` et `node.right`.

Complétez la fonction `recherche_arbre`.

```
1 import sys
2 import traceback
3
4 def recherche_arbre(node, value):
5     # Complétez ici
6
7
8     #Attention à inclure les lignes de codes additionnelles présentes dans le fichier "question8.py" sur Moodle
9
```

### Conseil

Indice : Utilisez une fonction récursive.

Pour cette question, vous devez télécharger le fichier **"question8.py"** sur moodle et copier tout son contenu dans votre fichier python. Celui-ci contient les classes permettant de définir les arbres selon le concept vu en cours.

Il contient également un code de "vérification". Celui-ci va automatiquement tester votre fonction avec différent paramètres et vérifier si les résultats obtenus correspondent au résultats attendus.

### >\_ Solution

#### Python :

```
1 def recherche_arbre(node, value):
2     #Solution
3     if node == None:
4         return False
5
6     if node.value == value:
7         return True
8     if node.value > value:
9         return recherche_arbre(node.left,value)
10    else:
11        return recherche_arbre(node.right,value)
```