

# Algorithmes et Pensée Computationnelle

## Algorithmes de recherche - Exercices avancés

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)

**Question 1:** (🕒 10 minutes) Recherche séquentielle - 3 (Python)

Cet exercice ne devrait pas être dans les avancés. Il est beaucoup trop simple

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 = [123,321,328,472,549]
e = 328
Résultat attendu : 2
```

```
1 def recherche_sequentielle(L,e):
2     for elem in L: #Ici, i correspond à la valeur et non l'index.
3         #complétez ici
4
5     L = [123,321,328,472,549]
6     e = 328
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 elem in L:
3         #Solution
4         if elem == e:
5             return L.index(elem) #L'algorithme prend fin aussitôt que la valeur recherchée est trouvée.
6     return -1 # Si la valeur n'a pas été trouvée, la fonction retourne -1
7
8 L = [123,321,328,472,549]
9 e = 328
10 resultat = recherche_sequentielle(L,e)
11 print(resultat)
```

## 2 Recherche binaire

### Question 2: (🕒 10 minutes) Recherche binaire (Python)

Soient une liste d'entiers triés  $L$  et 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

$L = [123, 321, 328, 472, 549]$

$e = 328$

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 conseils des exercices précédents (exercices basiques).

#### >\_ 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 = [123,321,328,472,549]
19 e = 328
20 recherche_binaire(L,e)
```

### Question 3: (🕒 15 minutes) Recherche binaire - plus proche élément (Python)

Soient une liste d'entiers **triés**  $L$  et 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 à `false` et qui devient `true` lorsque l'algorithme trouve la valeur la plus proche de `e`.

### >\_ Solution

#### Python :

```

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

```

### Question 4: (🕒 20 minutes) Recherche dans une matrice (Python)

#### Matrice en Python

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

Pour rappel, une matrice ordonnée répond aux critères suivants :

`[i][j] <= m[i+1][j]` (une ligne va du plus petit au plus grand)

`[i][j] <= 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 ligne in range(len(m)): # "Pour chaque ligne de la matrice" ou "Pour chaque liste de la liste"
4         for colonne in range(len(m[ligne])): # "Pour chaque colonne de la matrice" ou "Pour chaque élément de la
           liste"
5             if m[ligne][colonne] == l:
6                 return (ligne,colonne)#
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 print(recherche_matricielle(m,l))
```