Recherche dans une liste

① MÉMO

Recherche séquentielle

Principe Parcourir la liste du début à la fin, comparer chaque élément à la valeur cherchée.
Résultat Renvoie l'indice de la première occurrence, ou -1 si l'élément est absent.
Complexité $O(n)$ dans le pire cas.
Avantage Fonctionne sur n'importe quelle liste, triée ou non.

Recherche dichotomique

Précondition La liste doit être **triée**.
Principe Comparer la valeur cherchée à l'élément du milieu, puis réduire l'intervalle de moitié.
Complexité $O(\log_2 n)$: pour 10^6 éléments, au plus 20 comparaisons.
Invariant Si la valeur existe, elle se trouve dans l'intervalle `[gauche, droite]`.

Pièges fréquents

- Placer le `return -1` **dans** la boucle (dans un `else`) : la fonction s'arrête dès le premier élément qui ne correspond pas.
- Appliquer la dichotomie sur une liste non triée : le résultat est faux sans message d'erreur.
- Confondre `milieu = (gauche + droite) // 2` (correct) avec `milieu = (gauche + droite) / 2` (donne un flottant).

② EXEMPLES

Programme 1 · Recherche séquentielle

```
def recherche(L, x):
    for i in range(len(L)):
        if L[i] == x:
            return i     # trouvé : indice de la première occurrence
    return -1            # absent

print(recherche([4, 8, 2, 6, 3, 8], 2))   # 2
print(recherche([4, 8, 2, 6, 3, 8], 5))   # -1
```

Programme 2 · Recherche dichotomique

```
def dichotomie(L, x):
    gauche = 0
    droite = len(L) - 1
    while gauche <= droite:
        milieu = (gauche + droite) // 2
```

```
        if L[milieu] == x:
            return milieu
        elif L[milieu] < x:
            gauche = milieu + 1
        else:
            droite = milieu - 1
    return -1
```

Trace : recherche de **4** dans $L = [1, 3, 5, 7, 9, 11, 13]$.

Étape	gauche	droite	milieu	Comparaison	Conséquence
1	0	6	3	$L[3] = 7 > 4$	on cherche à gauche
2	0	2	1	$L[1] = 3 < 4$	on cherche à droite
3	2	2	2	$L[2] = 5 > 4$	on cherche à gauche
4	2	1		gauche > droite	**4 est absent**

③ EXERCICES

Exercice 1 *Toutes les occurrences* **(2 points)** Écrire une fonction `toutes_occurrences(L, x)` qui renvoie la liste des indices de toutes les occurrences de `x` dans `L`. Quelle est sa complexité ?

Exercice 2 *Nombre d'occurrences* **(2 points)** Écrire une fonction `nb_occurrences(L, x)` qui renvoie le nombre d'apparitions de `x` dans `L`, sans utiliser `.count()`.

Exercice 3 *Trace de la dichotomie* **(3 points)** On considère `L = [2, 5, 8, 12, 16, 23, 38, 42, 55, 67]`.

Donner la trace complète (`gauche`, `droite`, `milieu` à chaque étape) pour :

1. la recherche de `23` ;
2. la recherche de `10` (absent).

Exercice 4 *Comparaison des méthodes* **(4 points)**

1. Écrire une fonction `recherche_compteur(L, x)` qui effectue une recherche séquentielle et renvoie le tuple `(indice, nb_comparaisons)`.
2. Écrire une fonction `dichotomie_compteur(L, x)` qui effectue une recherche dichotomique et renvoie le tuple `(indice, nb_comparaisons)`.
3. Pour $L = [0, 1, 2, \ldots, 999]$ et $x = -1$ (absent), combien de comparaisons effectue chaque méthode ? Vérifier avec le code.

SOLUTIONS DES EXERCICES

Corrigé de l'exercice 1.

```python
def toutes_occurrences(L, x):
    indices = []
    for i in range(len(L)):
        if L[i] == x:
            indices.append(i)
    return indices
```

Vérification : `toutes_occurrences([3, 1, 4, 1, 5, 1], 1)` = `[1, 3, 5]`.

La complexité est $O(n)$: on parcourt toujours toute la liste car on cherche *toutes* les occurrences.

Corrigé de l'exercice 2.

```python
def nb_occurrences(L, x):
    compteur = 0
    for elem in L:
        if elem == x:
            compteur += 1
    return compteur
```

Vérification : `nb_occurrences([3, 1, 4, 1, 5, 1], 1)` renvoie `3`.

Corrigé de l'exercice 3. **Recherche de 23 :**

1. `gauche=0`, `droite=9`, `milieu=4`, `L[4]=16 < 23`, donc `gauche=5`.
2. `gauche=5`, `droite=9`, `milieu=7`, `L[7]=42 > 23`, donc `droite=6`.
3. `gauche=5`, `droite=6`, `milieu=5`, `L[5]=23`, trouvé à l'indice 5.

Trois comparaisons suffisent.

Recherche de 10 :

1. `gauche=0`, `droite=9`, `milieu=4`, `L[4]=16 > 10`, donc `droite=3`.
2. `gauche=0`, `droite=3`, `milieu=1`, `L[1]=5 < 10`, donc `gauche=2`.
3. `gauche=2`, `droite=3`, `milieu=2`, `L[2]=8 < 10`, donc `gauche=3`.
4. `gauche=3`, `droite=3`, `milieu=3`, `L[3]=12 > 10`, donc `droite=2`.
5. `gauche=3 > droite=2` : 10 est absent.

Corrigé de l'exercice 4.

```python
def recherche_compteur(L, x):
    for i in range(len(L)):
        if L[i] == x:
            return i, i + 1
    return -1, len(L)

def dichotomie_compteur(L, x):
    gauche, droite, nb = 0, len(L) - 1, 0
    while gauche <= droite:
        nb += 1
        milieu = (gauche + droite) // 2
        if L[milieu] == x:
            return milieu, nb
```

```
        elif L[milieu] < x:
            gauche = milieu + 1
        else:
            droite = milieu - 1
    return -1, nb
```

Résultat pour $n = 1000$**,** $x = -1$ **:** séquentielle = 1000 comparaisons, dichotomie = 10 comparaisons ($\lceil \log_2(1000) \rceil = 10$).