

## Recherche dans une liste

Recherche séquentielle ou dichotomique, notions de correction et de performances.

### Présentation du problème

On dispose d'une table de données. Par exemple :

Id	Nom Prénom	Naissance	Adresse
11	Dupont Élia	12/04/1997	12 rue des Boulets
25	Leroy Michel	29/02/1996	3 avenue de la République
21	Aubert Roland	31/12/2000	11 rue Pirandello
...	...	...	...

Supposons que l'on cherche un individu avec un certain identifiant **Id**. Cela revient donc à chercher un nombre dans un tableau [11, 25, 21, ...].

On désire donc écrire une fonction `recherche(tab, needle)` :

- `tab` est le tableau de nombres et `needle` le nombre cherché,
- la fonction renvoie l'indice de la première apparition de `needle` dans `tab`.
- si `needle` n'est pas dans `tab`, il faut faire un choix : la fonction renvoie `-1` ? ou `None` ? ou `False` ? ou lève une erreur ?

Python dispose d'une méthode de recherche qui fait partie du type `list`.

```
>>> tableau = [11, 25, 21, 69, 41, 26, 14]
>>> tableau.index(69)
3
>>> tableau.index(3)
ValueError: 3 is not in list
```

Python fait le choix de lever une erreur si l'aiguille est absente. Javascript fait le choix de renvoyer `-1`.

Mieux vaut renvoyer `-1` que `None` ou `False` car mieux vaut qu'une fonction renvoie toujours le même type. On attend de la fonction renvoie un indice, donc un entier. `-1` est un entier, pas `None` ou `False`.

### Recherche séquentielle

	ENTRÉES: liste T et aiguille
	DÉBUT
Ci-contre un exemple d'algorithme de recherche séquentielle.	soit n la taille de T
	POUR i ALLANT DE 0 À n-1
	SI T[i] est égal à aiguille ALORS
On y fait le choix de renvoyer <code>-1</code> si l'aiguille n'est pas dans la le tableau.	RENVOYER i
	FIN
	FIN
	RENVOYER -1
	FIN

### Performances du tri séquentiel

Si on a de la chance, on trouve l'aiguille dans la première case du tableau. Mais si on n'a pas de chance, il faut aller jusqu'à la fin du tableau pour trouver l'aiguille ou bien pour savoir que l'aiguille n'est pas dans le tableau.

On exprime généralement les performances pour les cas désavantageux.

Dans le pire des cas, le tri séquentiel exécute `n` boucles, `n` étant la taille de T. Si la taille du tableau est multiplié par 2, le temps d'exécution, au pire, est aussi multiplié par 2. Dans ce cas on dit que l'algorithme est d'**Ordre n**.

**Tri dichotomique**

```

1 ENTRÉES : liste T triée croissante, needle
2 DÉBUT
3     SI T vide ALORS RENVOYER -1
4     n = taille de T ; a = 0 et b = n-1
5     TANT QUE a < b FAIRE
6         # on cherche dans les indices [a;b]
7         m = indice milieu [a;b]
8         # (arrondi par défaut)
9         SI T[m] >= needle ALORS
10            chercher dans [a;m]
11        SINON
12            chercher dans [m+1;b]
13    FIN
14    FIN
15    SI T[a] est needle ALORS
16        RENVOYER a
17    SINON
18        RENVOYER -1
19    FIN SI
20 FIN

```

#	a	b	m	T[m]
4				
9				
9				
9				
9				
13				

Complétez l'exemple dans le cas où :

needle = 9

T = [2, 5, 7, 9, 9, 21, 38, 39, 40, 41, 41, 52, 57, 71, 78, 78, 83, 96]

**Performance :** La taille du tableau exemple est n =

L'algorithme arrive au résultat en ..... tours de boucle. On peut observer que  $2^4 \approx 18$  et c'est pourquoi on dit que cet algorithme est en  $\log_2(n)$ .

**Terminaison :** a ne fait que  $\nearrow$  et  $b \searrow$  et l'écart  $|b - a|$  diminue d'au moins 1 à chaque répétition du **while**. On finit donc toujours par avoir  $a \geq b$  et **while** termine. L'algo termine donc.

**Correction :** Si needle est dans le tableau, sa première apparition est toujours dans  $T_{[a;b]}$ . Comme à la fin  $[a;b] = [a;a]$  alors needle fini par être dans  $T_{[a;a]} = T[a]$ . La fonction est donc correcte.

À noter que l'algorithme poserait problème dans l'éventualité d'un tableau vide. C'est pourquoi on prévoit une clause de garde en ligne 3 qui permet de traiter ce cas à part.

**Performances du tri dichotomique**

Quand la taille n du tableau T est multipliée par 2, la boucle n'est exécuté qu'une fois de plus. On dit que le tri dichotomique est d'ordre  $\log_2(n)$ .

Voici un comparatif pour être plus concret. On prend comme temps de référence 1 pour un tableau de taille 10.

n	10 = référence	100	1000	10000	100000
séquentiel	temps de référence = 1	10	100	1000	10000
dichotomique	temps de référence = 1	7	10	14	17

L'important n'est pas la durée mais la façon dont la durée augmente avec la taille du tableau.

La recherche dichotomique est beaucoup plus rapide **mais elle nécessite de trier le tableau**. Si le tableau n'est pas trié, il faut tenir compte de ce coût supplémentaire.