

November 19, 2025

```
[2]: def dichot(l,x):#l trie par ordre croissant
      g,d = 0, len(l)
      while d>g:
          m = (g+d)//2
          if l[m] == x:
              return m
          elif l[m] > x: #chercher à gauche de m
              d = m
          else:#chercher à droite
              g=m+1
      return -1
```

```
[13]: l = [10,20,30,40,50]
      dichot(l,40),dichot(l,50),dichot(l,10),\
      dichot(l,25), dichot(l,20), dichot(l,70)
```

```
[13]: (3, 4, 0, -1, 1, -1)
```

```
[11]: l = [10,20,30,40,50,60]
      dichot(l,40),dichot(l,60),dichot(l,10), dichot(l,25), dichot(l,20)
```

```
[11]: (3, 5, 0, -1, 1)
```

La quantité  $d - g$  est positive, entière, strictement décroissante.

C'est un *variant* de boucle : l'algorithme termine

## 0.1 Etude de complexité

A chaque étape la longueur de l'intervalle de recherche est divisée par 2 au moins.

```

      milieu
      |
x x x x x x
zone de recherche x x x (gauche) ou x x
```

```

      milieu
      |
x x x x x x x
```

zone de recherche x x x (gauche) ou x x x (droite)

La taille  $d - g$  du nouvel intervalle de recherche est au moins deux fois plus petite que le précédent.

On note  $d_i, g_i$  les indices **A LA FIN DU TOUR** de boucle numéro  $i$ , et  $d_0 - g_0$  représente la taille initiale (avant le passage dans la boucle)

$$d_1 - g_1 \leq \frac{d_0 - g_0}{2}$$

$$\text{Par récurrence immédiate } d_n - g_n \leq \frac{d_0 - g_0}{2^n}$$

Rappel:

$$2^{\lfloor \log_2 k \rfloor} \leq k < 2^{\lfloor \log_2 k \rfloor + 1}$$

En  $n = \lfloor \log_2(d_0 - g_0) \rfloor + 1$  étapes on aura

$$0 \leq d_n - g_n \leq \frac{d_0 - g_0}{2^{\lfloor \log_2(d_0 - g_0) \rfloor + 1}} < 1$$

donc  $d_n - g_n = 0$  et on sort de la boucle.

Il y a donc au plus  $\log_2(|\ell|) + 1$  passages dans la boucle, chacune étant  $O(1)$  : donc la complexité est logarithmique ( $O(\log_2 |\ell|)$ )

```
[15]: texte = "toto et gogo"
      print(texte[1], len(texte))
```

o 12

```
[19]: texte[1]='0' #on ne peut modifier un string
```

```
-----
TypeError                                Traceback (most recent call last)
Cell In[19], line 1
----> 1 texte[1]='0'

TypeError: 'str' object does not support item assignment
```

```
[17]: def start (t, m, i):
      j = 0 #position dans le mot
      while j < len(m):
          if m[j] != t[i+j]:
              return False
          j+=1 #on passe au char suivant
      return True
```

```
[18]: t = "toto et gogo"
      m1, m2, m3 = "toto", "gogo", "gogox"
      start(t,m1,0), start(t,m2,1), start(t,m2,8)
```

[18]: (True, False, True)

## 0.2 Complexité

Au plus  $|m|$  passages dans la boucle, chacun en  $O(1)$ .

Donc complexité au pire en  $O(|m|)$ .

Dans le meilleur cas, la première lettre du mot ne coïncide pas avec la lettre  $i$  du texte. La complexité est alors en  $O(1)$ .

```
[12]: def recherche_naive(l,x):  
      """  
      Aucune hypothèse n'est faite sur l  
      """  
      for i in range(len(l)):  
          if l[i]==x:  
              return x  
      return -1
```

### Complexité :

Si  $n$  est la longueur de la liste et si  $x \notin \ell$ , alors il y a  $n$  passages dans la boucle, chacun en  $O(1)$ .

Au total, la complexité au pire est un  $O(n)$  (linéaire).

Si  $x = \ell_0$ , alors la complexité est un  $O(1)$

[ ]:

[ ]: