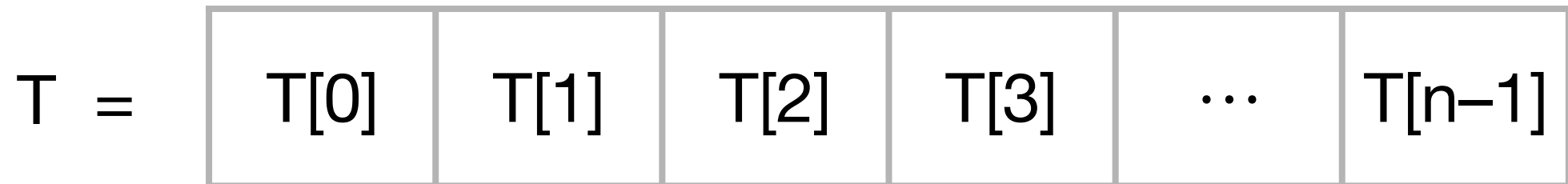


# Introduction à l'informatique CM4

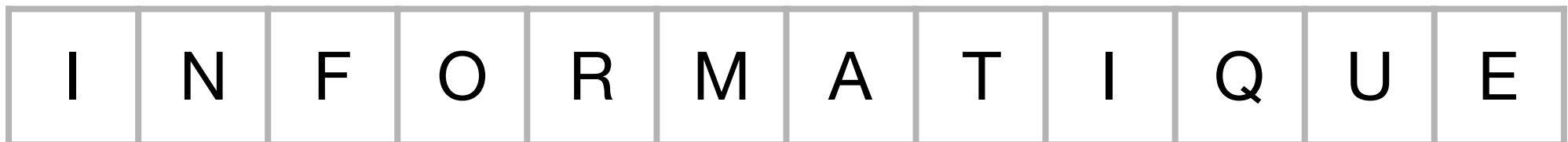
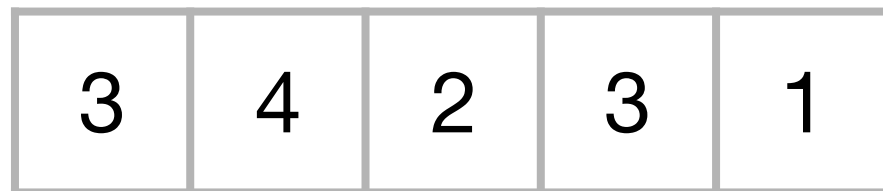
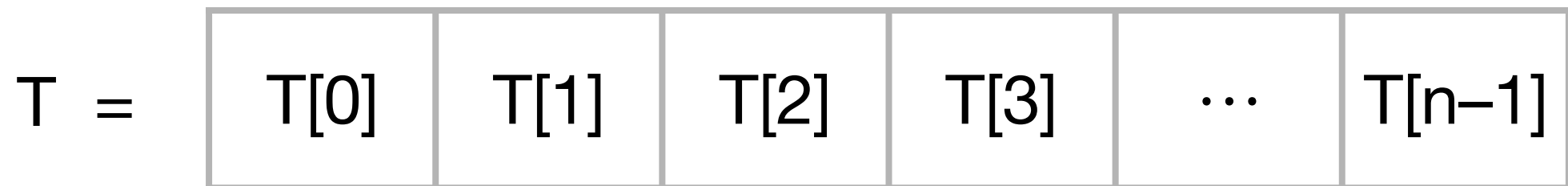
Antonio E. Porreca

<https://aeporreca.org/introinfo>

# Structures de données : les tableaux



# Structures de données : les tableaux



# Parcourir un tableau

```
procedure parcours(T)
  n := longueur(T)
  i := 0
  tant que i < n faire
    écrire T[i]
  fin tant que
fin procedure
```

# Avec la boucle « pour »

```
procedure parcours(T)
  n := longueur(T)
  i := 0
  tant que i < n faire
    écrire T[i]
  fin tant que
fin procedure
```

=

```
procedure parcours(T)
  n := longueur(T)
  pour i := 0 à n - 1 faire
    écrire T[i]
  fin pour
fin procedure
```

# Recherche dans un tableau

```
fonction chercher(x, T)
  n := longueur(T)
  i := 0
  tant que i < n faire
    si T[i] = x alors
      retourner i
    fin si
    i := i + 1
  fin tant que
  retourner -1
fin fonction
```

# Recherche dans un tableau

```
fonction chercher(x, T)
  n := longueur(T)
  i := 0
  tant que i < n faire
    si T[i] = x alors
      retourner i
    fin si
    i := i + 1
  fin tant que
  retourner -1
fin fonction
```

=

```
fonction chercher(x, T)
  n := longueur(T)
  pour i := 0 à n - 1 faire
    si T[i] = x alors
      retourner i
    fin si
  fin pour
  retourner -1
fin fonction
```

# Recherche dans un tableau

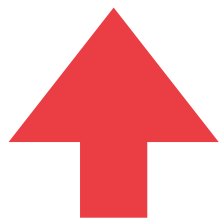
```
fonction chercher(x, T)
  n := longueur(T)
  pour i := 0 à n - 1 faire
    si T[i] = x alors
      retourner i
    fin si
  fin pour
  retourner -1
fin fonction
```



# Recherche linéaire

Recherche de 33

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

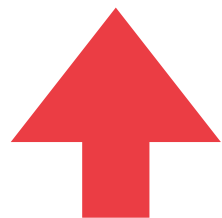


i

# Recherche linéaire

Recherche de 33

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11



i

# Recherche linéaire

Recherche de 33

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

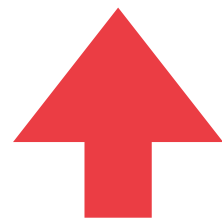


i

# Recherche linéaire

Recherche de 33

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11



i

# Recherche linéaire

Recherche de 33

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

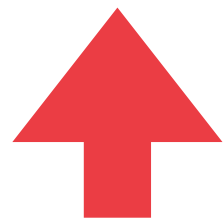


i

# Recherche linéaire

Recherche de 33

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11



i

# Recherche linéaire

Recherche de 33

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

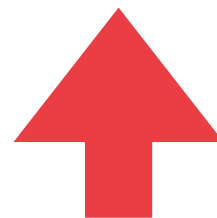


i

# Recherche linéaire

Recherche de 33

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11



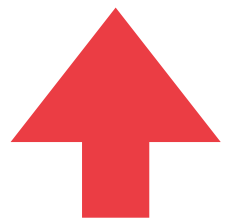
i



# Recherche linéaire

Recherche de 3

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11



i

# Recherche linéaire

Recherche de 3

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11



i

# Recherche linéaire

Recherche de 3

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

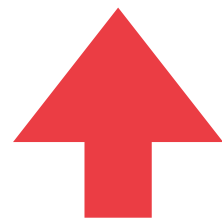


i

# Recherche linéaire

Recherche de 3

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11



i

# Recherche linéaire

Recherche de 3

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

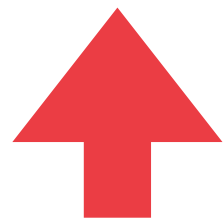


i

# Recherche linéaire

Recherche de 3

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11



i

# Recherche linéaire

Recherche de 3

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

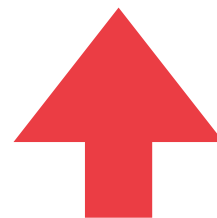


i

# Recherche linéaire

Recherche de 3

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11



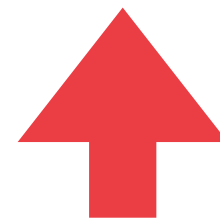
i



# Recherche linéaire

Recherche de 3

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

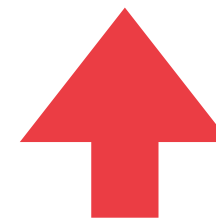


i

# Recherche linéaire

Recherche de 3

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

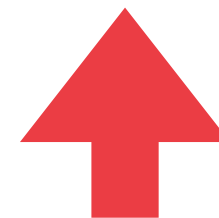


i

# Recherche linéaire

Recherche de 3

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

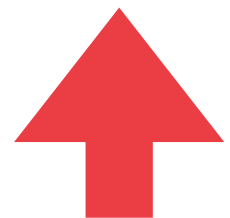


i

# Recherche linéaire

Recherche de 3

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

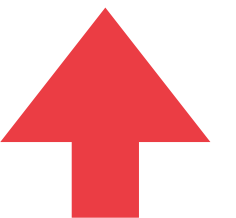


i

# Recherche linéaire

Recherche de 3

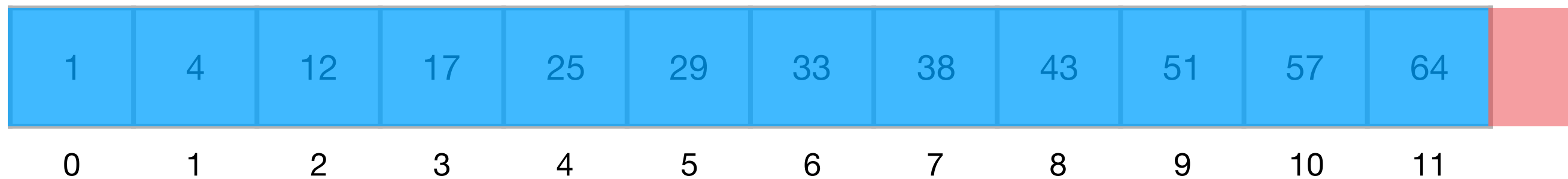
1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11



i

# Recherche linéaire

Recherche de 3



i

# Recherche linéaire

Recherche de 3

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11



i

# Recherche dans un tableau

```
fonction chercher(x, T)
  n := longueur(T)
  pour i := 0 à n – 1 faire
    si T[i] = x alors
      retourner i
    fin si
  fin pour
  retourner –1
fin fonction
```

Terminaison ?

Correction ?

Efficacité ?



# Terminaison

```
fonction chercher(x, T)
  n := longueur(T)
  pour i := 0 à n – 1 faire
    si T[i] = x alors
      retourner i
    fin si
  fin pour
  retourner –1
fin fonction
```

- Au début, on a  $i = 0$
- Il reste toujours  $n - i$  positions à examiner
- $i$  est incrémenté à chaque itération
- Tôt ou tard soit on trouve  $x$ , soit on arrive à  $i = n$ , et l'algorithme termine

# Correction

```
fonction chercher(x, T)
  n := longueur(T)
  pour i := 0 à n – 1 faire
    si T[i] = x alors
      retourner i
    fin si
  fin pour
  retourner –1
fin fonction
```

- **Invariant de boucle** : si x est dans le tableau, alors il se trouve dans le sous-tableau  $T[i, \dots, n - 1]$ 
  - C'est vrai au début de l'algorithme
  - Ça reste vrai à chaque itération de la boucle, parce qu'on vérifie toujours si  $T[i] = x$
- Si on sort de la boucle avec  $i = n$ , alors si x est dans le tableau, il est dans le sous-tableau vide  $T[n, n - 1]$ , c'est à dire qu'il n'est pas là

# Comptage des opérations

```
fonction chercher(x, T)
  n := longueur(T)
  pour i := 0 à n – 1 faire
    si T[i] = x alors
      retourner i
    fin si
  fin pour
  retourner –1
fin fonction
```


# Comptage des opérations

#opérations

👉 **fonction** chercher(x, T)  
    n := longueur(T)  
    **pour** i := 0 à n – 1 **faire**  
        **si** T[i] = x **alors**  
            **retourner** i  
        **fin si**  
    **fin pour**  
    **retourner** –1  
**fin fonction**

# Comptage des opérations

#opérations

 **fonction** chercher(x, T)  
    n := longueur(T)  
    **pour** i := 0 à n – 1 **faire**  
        **si** T[i] = x **alors**  
            **retourner** i  
        **fin si**  
    **fin pour**  
    **retourner** –1  
**fin fonction**

1

# Comptage des opérations

#opérations

**fonction** chercher(x, T)

n := longueur(T)

1



**pour** i := 0 à n – 1 **faire**

1

**si** T[i] = x **alors**

**retourner** i

**fin si**

**fin pour**

**retourner** –1

**fin fonction**

# Comptage des opérations

#opérations

**fonction** chercher(x, T)

n := longueur(T)

1

**pour** i := 0 à n – 1 **faire**

1



si T[i] = x **alors**

1

retourner i

**fin si**

**fin pour**

retourner –1

**fin fonction**

# Comptage des opérations

#opérations

```
fonction chercher(x, T)
  n := longueur(T)
  pour i := 0 à n - 1 faire
    si T[i] = x alors
      retourner i
    fin si
  fin pour
  retourner -1
fin fonction
```



1  
1  
1  
1



# Comptage des opérations

#opérations

```
fonction chercher(x, T)
  n := longueur(T)
  pour i := 0 à n – 1 faire
    si T[i] = x alors
      retourner i
    fin si
  fin pour
  retourner –1
fin fonction
```



1  
1  
1  
1

# Comptage des opérations

#opérations

```
fonction chercher(x, T)
  n := longueur(T)
  pour i := 0 à n – 1 faire
    si T[i] = x alors
      retourner i
    fin si
  fin pour
  retourner –1
fin fonction
```

1

1

1

1



**fin pour**

**retourner –1**

# Comptage des opérations

#opérations

**fonction** chercher(x, T)

  n := longueur(T)

1

**pour** i := 0 à n – 1 **faire**

1

**si** T[i] = x **alors**

1

**retourner** i

1

**fin si**

**fin pour**



**retourner** –1

1

**fin fonction**

# Comptage des opérations

#opérations

**fonction** chercher(x, T)

  n := longueur(T)

1

**pour** i := 0 à n – 1 **faire**

1

**si** T[i] = x **alors**

1

**retourner** i

1

**fin si**

**fin pour**

**retourner** –1

1

👉 **fin fonction**

# Comptage des opérations

#opérations

<b>fonction</b> chercher(x, T)	
n := longueur(T)	1
<b>pour</b> i := 0 à n – 1 <b>faire</b>	1
<b>si</b> T[i] = x <b>alors</b>	1
<b>retourner</b> i	1
<b>fin si</b>	
<b>fin pour</b>	
<b>retourner</b> –1	1
<b>fin fonction</b>	

# Comptage des opérations

#opérations

<b>fonction</b> chercher(x, T)	
n := longueur(T)	1
<b>pour</b> i := 0 à n – 1 <b>faire</b>	1
<b>si</b> T[i] = x <b>alors</b>	1
<b>retourner</b> i	1
<b>fin si</b>	
<b>fin pour</b>	
<b>retourner</b> –1	1
<b>fin fonction</b>	

si on a  $T[k] = x$

# Comptage des opérations

#opérations

<b>fonction</b> chercher(x, T)		
n := longueur(T)	1	
<b>pour</b> i := 0 à n – 1 <b>faire</b>	1	} k + 1 fois
<b>si</b> T[i] = x <b>alors</b>	1	
<b>retourner</b> i	1	
<b>fin si</b>	1	
<b>fin pour</b>		
<b>retourner</b> –1	1	
<b>fin fonction</b>		

si on a  $T[k] = x$

# Comptage des opérations

#opérations

```
fonction chercher(x, T)
  n := longueur(T)
  pour i := 0 à n - 1 faire
    si T[i] = x alors
      retourner i
    fin si
  fin pour
  retourner -1
fin fonction
```

1

1

1

1

}

k + 1 fois

1

si on a  $T[k] = x$

$2(k + 1) + 2$   
opérations



# Comptage des opérations

#opérations

<b>fonction</b> chercher(x, T)		
n := longueur(T)	1	
<b>pour</b> i := 0 à n – 1 <b>faire</b>	1	} n fois
<b>si</b> T[i] = x <b>alors</b>	1	
<b>retourner</b> i	1	
<b>fin si</b>	1	
<b>fin pour</b>		
<b>retourner</b> –1	1	
<b>fin fonction</b>		

si x n'est pas là

# Comptage des opérations

#opérations

<b>fonction</b> chercher(x, T)		
n := longueur(T)	1	
<b>pour</b> i := 0 à n – 1 <b>faire</b>	1	} n fois
<b>si</b> T[i] = x <b>alors</b>	1	
<b>retourner</b> i	1	
<b>fin si</b>	1	
<b>fin pour</b>		
<b>retourner</b> –1	1	
<b>fin fonction</b>		

si x n'est pas là

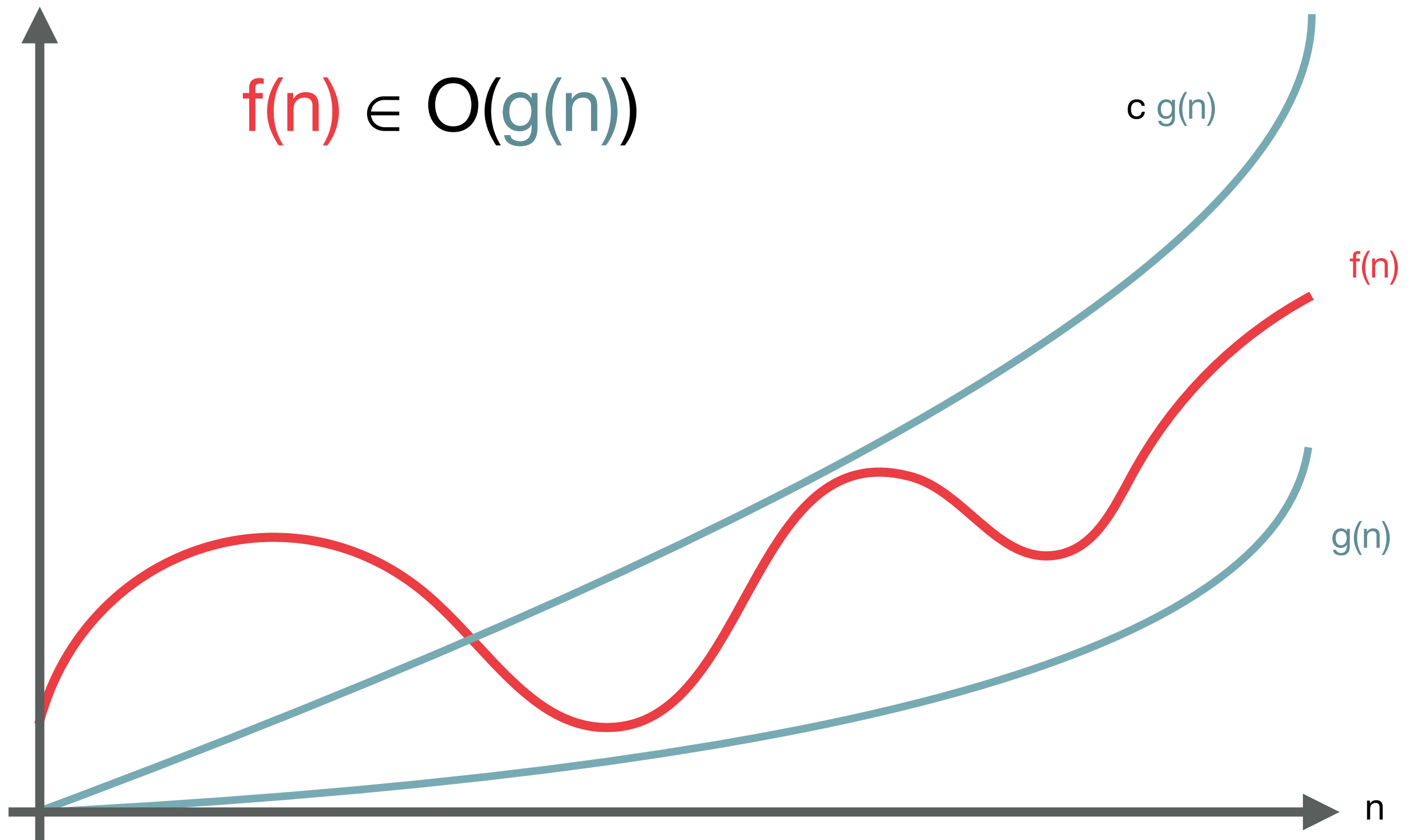
$2n + 2$   
opérations

# Efficacité

```
fonction chercher(x, T)
  n := longueur(T)
  pour i := 0 à n - 1 faire
    si T[i] = x alors
      retourner i
    fin si
  fin pour
  retourner -1
fin fonction
```

- Si on a de la chance, on a  $T[0] = x$  et on termine tout de suite en **4 opérations**
- Si  $T[k] = x$  on fait  $2(k + 1) + 2 = \mathbf{2k + 5}$  **opérations**
- Si  $x$  n'est pas là on fait  **$2n + 2$**  opérations

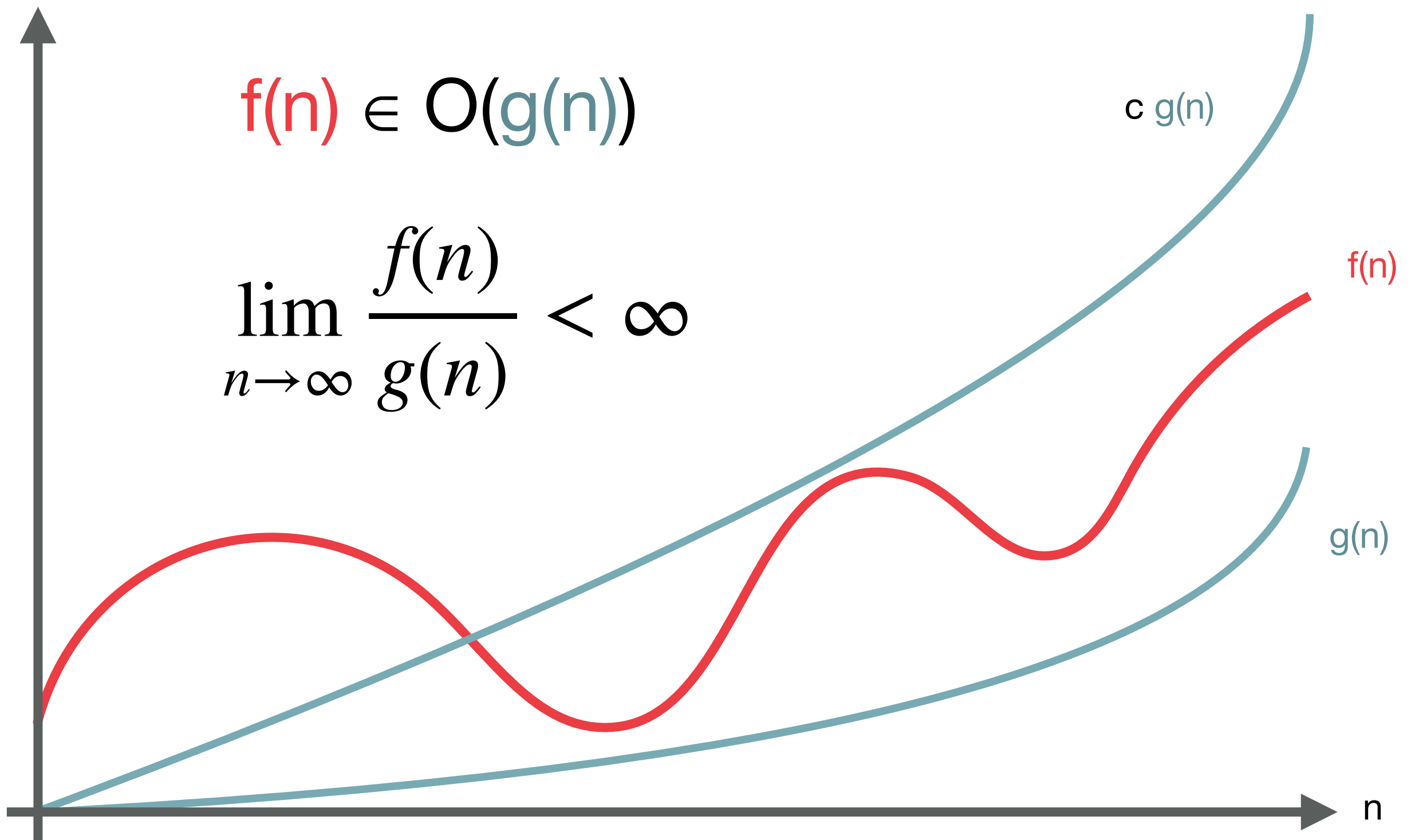
# Notation « grand O »



# Notation « grand O »

$$f(n) \in O(g(n))$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$$



# Ordres de grandeur

- $n \in O(n)$
- $n + 5 \in O(n)$
- $2n + 5 \in O(n)$
- $n^2 \notin O(n)$

**Recherche dans  
un annuaire  
ou un dictionnaire ?**

# Recherche dichotomique dans un tableau d'entiers trié

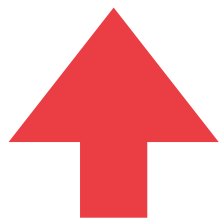
```
fonction chercher(x, T)
  n := longueur(T)
  i := 0
  j := n - 1
  tant que i < j faire
    m := (i + j) ÷ 2
    si T[m] = x alors
      retourner m
    sinon si x < T[m] alors
      j := m - 1
    sinon
      i := m + 1
    fin si
  fin tant que
  retourner -1
fin fonction
```



# Recherche dichotomique

Recherche de 33

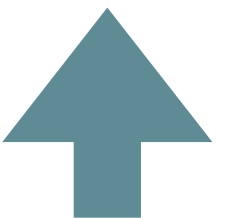
1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11



i



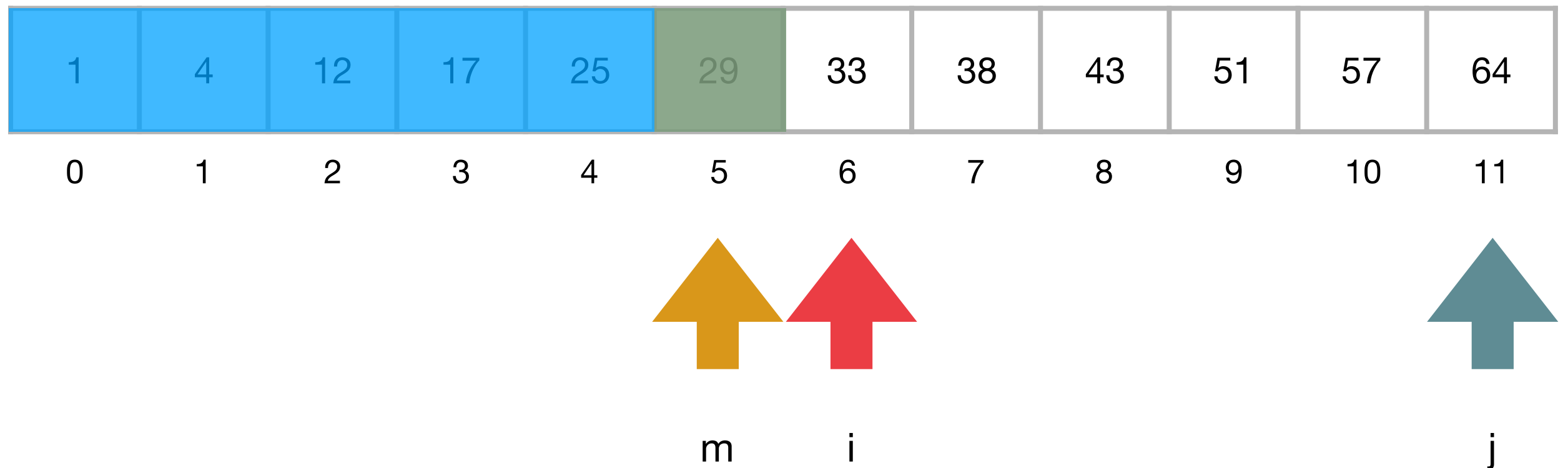
m



j

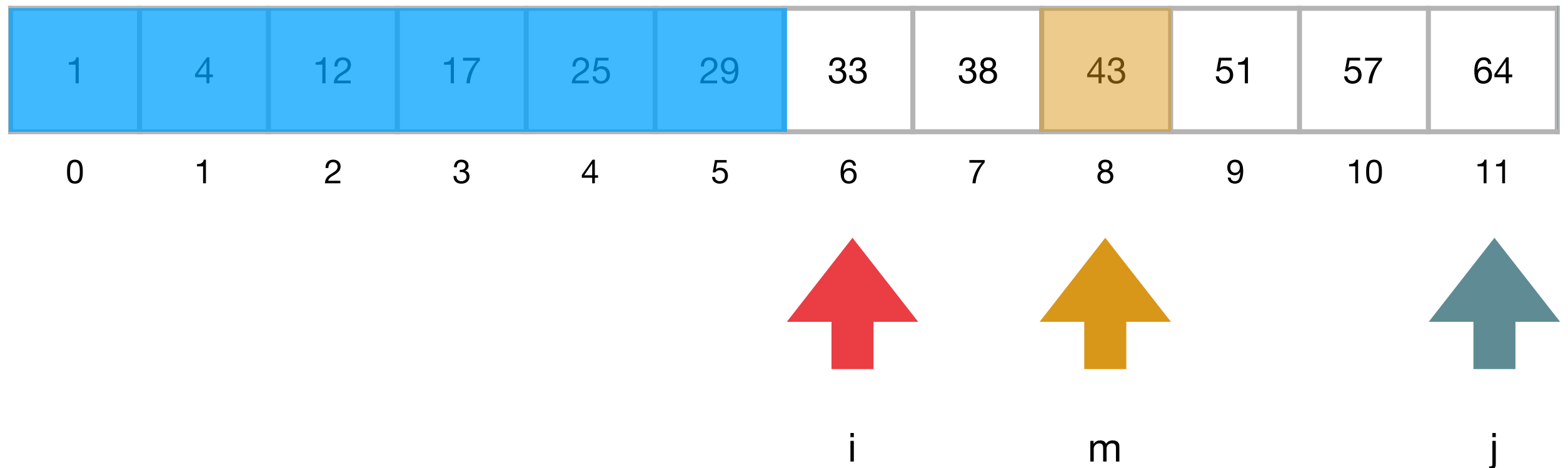
# Recherche dichotomique

Recherche de 33



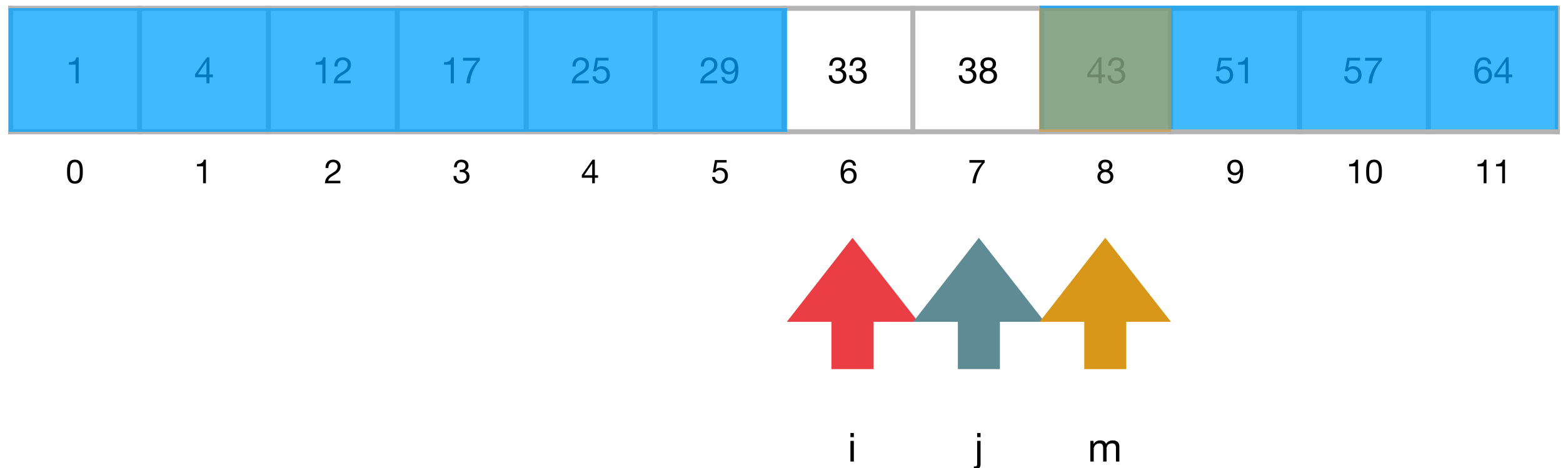
# Recherche dichotomique

Recherche de 33



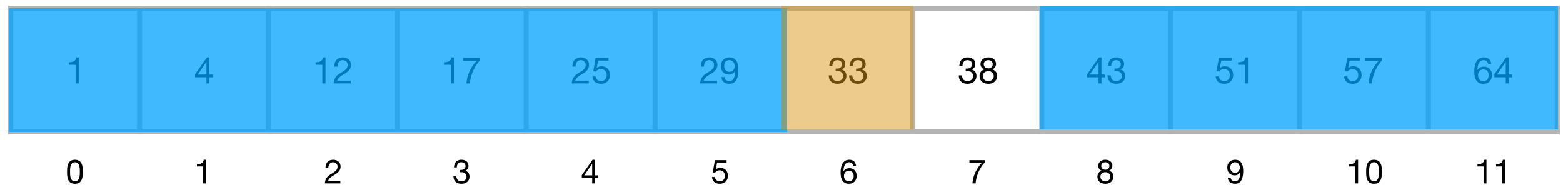
# Recherche dichotomique

Recherche de 33



# Recherche dichotomique

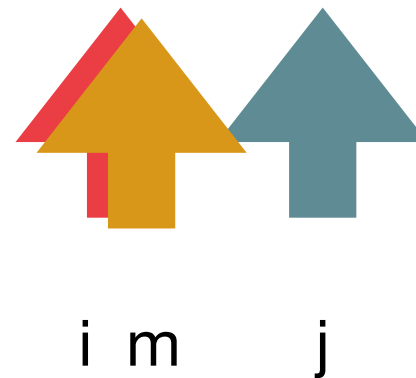
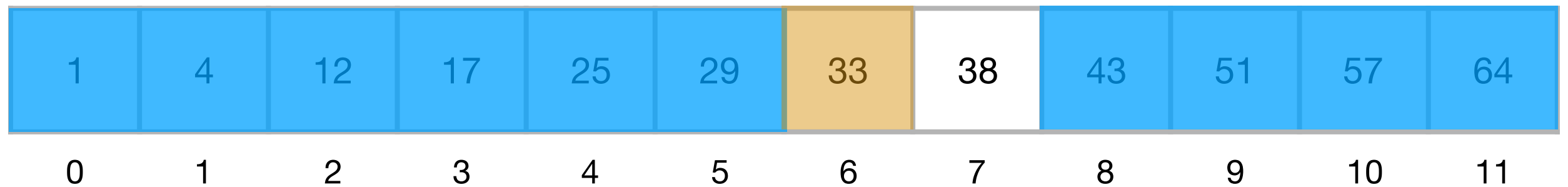
Recherche de 33



i m j

# Recherche dichotomique

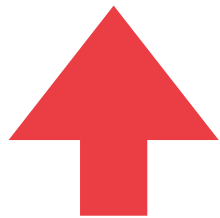
Recherche de 33



# Recherche dichotomique

Recherche de 16

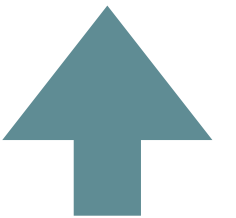
1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11



i



m

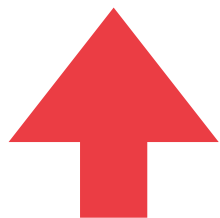


j

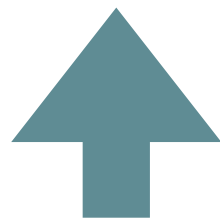
# Recherche dichotomique

Recherche de 16

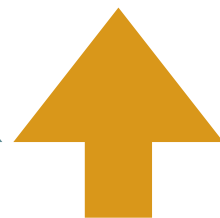
1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11



i



j



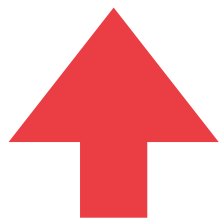
m



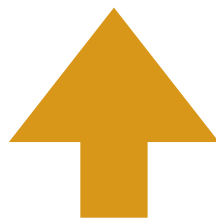
# Recherche dichotomique

Recherche de 16

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11



i



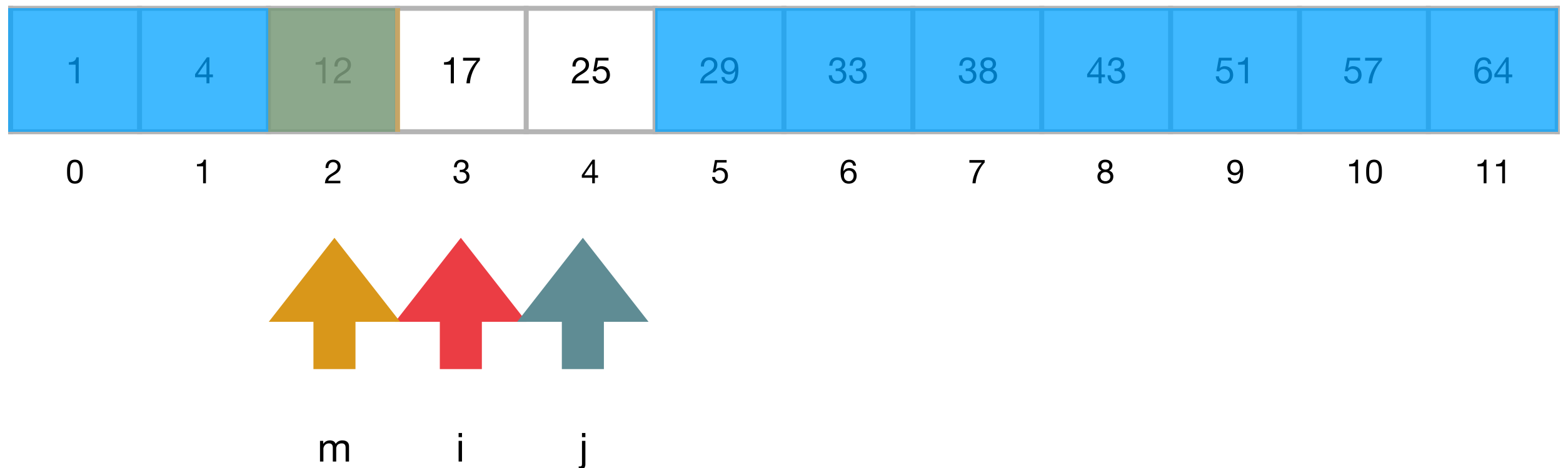
m



j

# Recherche dichotomique

Recherche de 16



# Recherche dichotomique

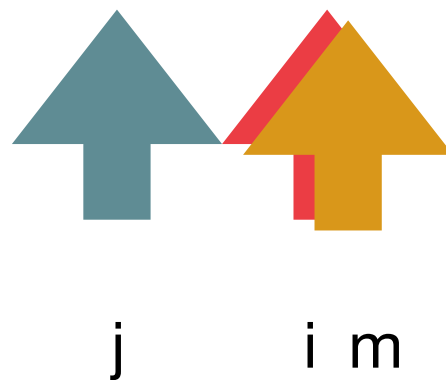
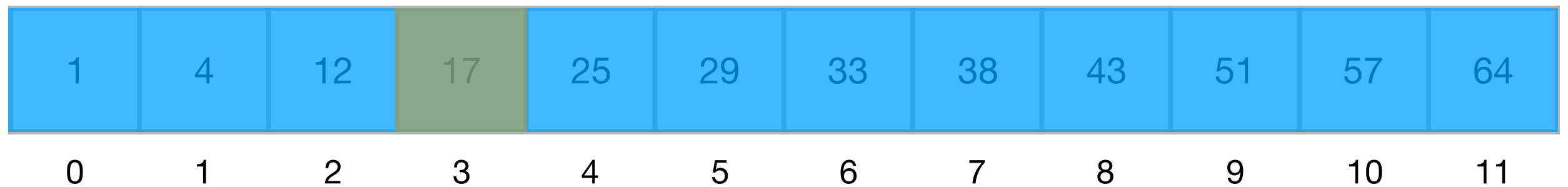
Recherche de 16



i m j

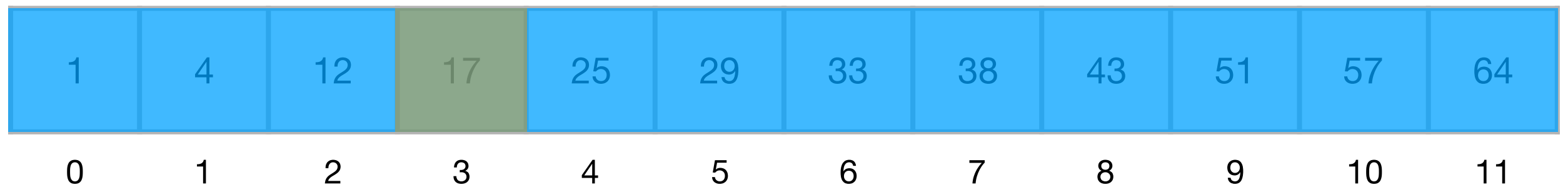
# Recherche dichotomique

Recherche de 16



# Recherche dichotomique

Recherche de 16



# Recherche dichotomique dans un tableau d'entiers trié

```
fonction chercher(x, T)
  n := longueur(T)
  i := 0
  j := n - 1
  tant que i < j faire
    m := (i + j) ÷ 2
    si T[m] = x alors
      retourner m
    sinon si x < T[m] alors
      j := m - 1
    sinon
      i := m + 1
    fin si
  fin tant que
  retourner -1
fin fonction
```

**Terminaison ?**

**Correction ?**

**Efficacité ?**

# Terminaison

```
fonction chercher(x, T)
  n := longueur(T)
  i := 0
  j := n - 1
  tant que i < j faire
    m := (i + j) ÷ 2
    si T[m] = x alors
      retourner m
    sinon si x < T[m] alors
      j := m - 1
    sinon
      i := m + 1
    fin si
  fin tant que
  retourner -1
fin fonction
```

- Il reste toujours  $j - i + 1$  éléments à examiner
- À chaque itération, on élimine approx. la moitié des éléments qui restent
- Tôt ou tard on trouve x, ou on reste sans éléments, et l'algorithme termine

# Correction

```
fonction chercher(x, T)
  n := longueur(T)
  i := 0
  j := n - 1
  tant que i < j faire
    m := (i + j) ÷ 2
    si T[m] = x alors
      retourner m
    sinon si x < T[m] alors
      j := m - 1
    sinon
      i := m + 1
    fin si
  fin tant que
  retourner -1
fin fonction
```

- **Invariant de boucle** : si x est dans le tableau, il se trouve dans le sous-tableau  $T[i, \dots, j]$ 
  - C'est vrai au début de l'algorithme
  - Ça reste vrai à chaque itération de la boucle, parce qu'on vérifie toujours si  $T[m] = x$  ou  $T[m] > x$  ou  $T[m] < x$
- Si on sort de la boucle avec  $i \geq j$ , alors si x est dans le tableau, il est dans le sous-tableau vide  $T[i, j]$ , c'est à dire qu'il n'est pas là



# Efficacité

```
fonction chercher(x, T)
  n := longueur(T)
  i := 0
  j := n - 1
  tant que i < j faire
    m := (i + j) ÷ 2
    si T[m] = x alors
      retourner m
    sinon si x < T[m] alors
      j := m - 1
    sinon
      i := m + 1
    fin si
  fin tant que
  retourner -1
fin fonction
```

- Dans le pire des cas, x n'est pas là
- Comme on élimine à chaque itération la moitié du tableau, on exécute la boucle  $\log_2 n$  fois au maximum
- Ça fait  $O(\log_2 n)$  opérations