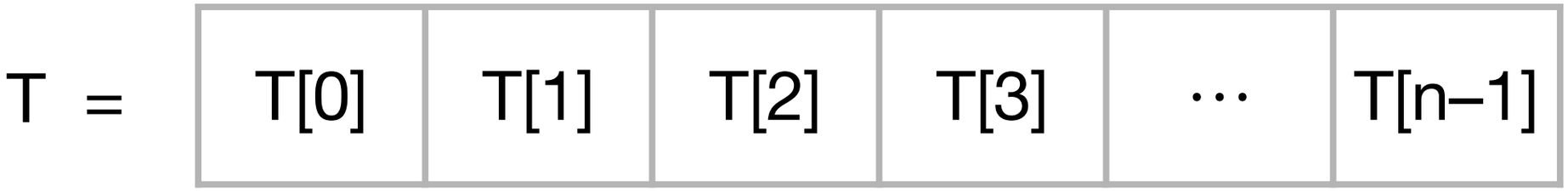
# Introduction à l’informatique CM3

Antonio E. Porreca
aeporreca.org/introinfo

# Les tableaux

# Structures de données : les tableaux



# Structures de données : les tableaux

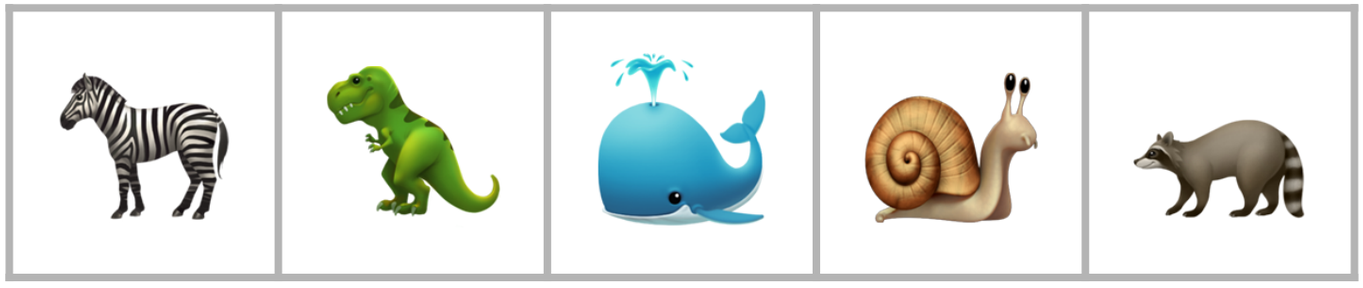
T =

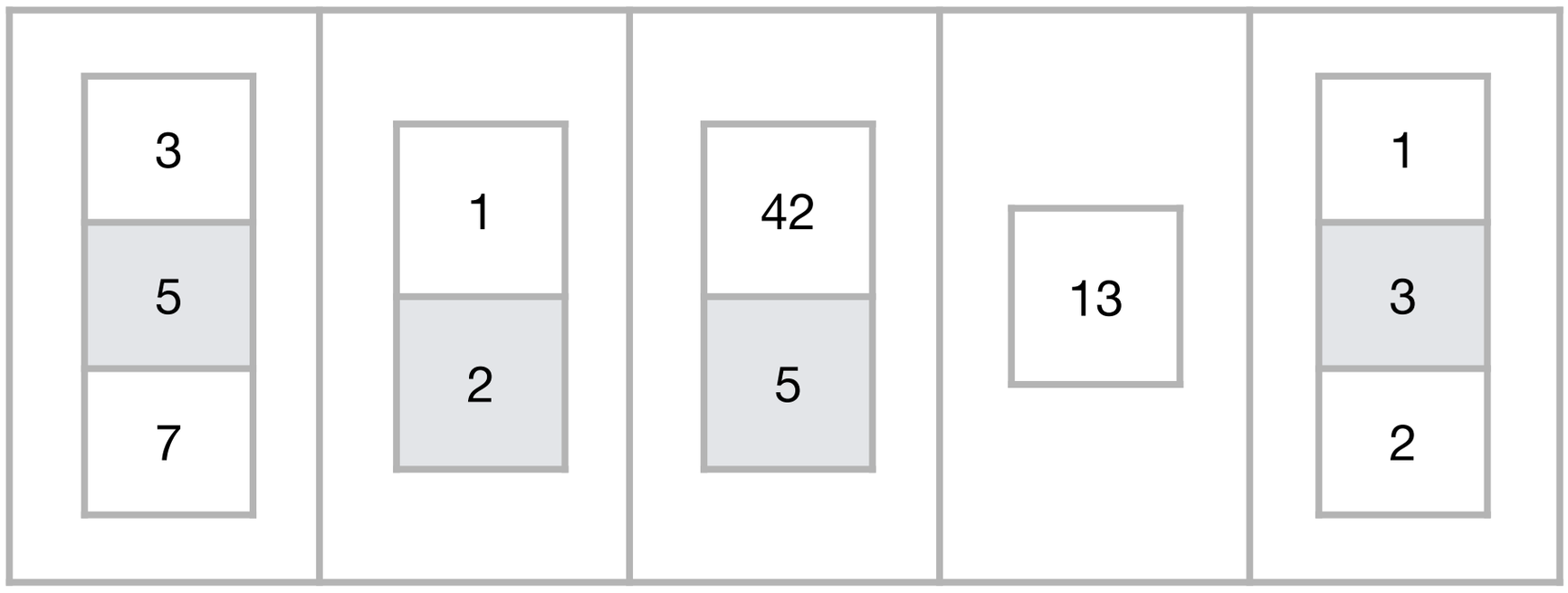
| T[0] | T[1] | T[2] | T[3] | ⋯ | T[n–1] |
|---|---|---|---|---|---|

| 3 | 4 | 2 | 3 | 1 |
|---|---|---|---|---|

| I | N | F | O | R | M | A | T | I | Q | U | E |
|---|---|---|---|---|---|---|---|---|---|---|---|

# Tableaux





# Tableaux (listes) en Python

# Tableaux (listes) en Python

```
>>> A = [5, 1, 4, 2, 3]
```

# Tableaux (listes) en Python

```
>>> A = [5, 1, 4, 2, 3]
>>> A
```

# Tableaux (listes) en Python

```
>>> A = [5, 1, 4, 2, 3]
>>> A
[5, 1, 4, 2, 3]
```

# Tableaux (listes) en Python

```
>>> A = [5, 1, 4, 2, 3]
>>> A
[5, 1, 4, 2, 3]
>>> A[0]
```

# Tableaux (listes) en Python

```
>>> A = [5, 1, 4, 2, 3]
>>> A
[5, 1, 4, 2, 3]
>>> A[0]
5
```

# Tableaux (listes) en Python

```
>>> A = [5, 1, 4, 2, 3]
>>> A
[5, 1, 4, 2, 3]
>>> A[0]
5
>>> A[1]
```

# Tableaux (listes) en Python

```
>>> A = [5, 1, 4, 2, 3]
>>> A
[5, 1, 4, 2, 3]
>>> A[0]
5
>>> A[1]
1
```

# Tableaux (listes) en Python

```
>>> A = [5, 1, 4, 2, 3]
>>> A
[5, 1, 4, 2, 3]
>>> A[0]
5
>>> A[1]
1
>>> A[4]
```

# Tableaux (listes) en Python

```
>>> A = [5, 1, 4, 2, 3]
>>> A
[5, 1, 4, 2, 3]
>>> A[0]
5
>>> A[1]
1
>>> A[4]
3
```

# Tableaux (listes) en Python

```
>>> A = [5, 1, 4, 2, 3]
>>> A
[5, 1, 4, 2, 3]
>>> A[0]
5
>>> A[1]
1
>>> A[4]
3
>>> A[5]
```

# Tableaux (listes) en Python

```
>>> A = [5, 1, 4, 2, 3]
>>> A
[5, 1, 4, 2, 3]
>>> A[0]
5
>>> A[1]
1
>>> A[4]
3
>>> A[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

# Parcourir un tableau

```
def parcours(A):
    n = len(A)
    i = 0
    while i < n:
        print(A[i])
        i = i + 1
```

# Avec la boucle « for »

```
def parcours(A):
    n = len(A)
    i = 0
    while i < n:
        print(A[i])
        i = i + 1
```

# Avec la boucle « for »

```
def parcours(A):
    n = len(A)
    i = 0
    while i < n:
        print(A[i])
        i = i + 1
```

=

```
def parcours(A):
    n = len(A)
    for i in range(n):
        print(A[i])
```

# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    i = 0
    while i < n:
        if A[i] == x:
            return i
        i = i + 1
    return -1
```

# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    i = 0
    while i < n:
        if A[i] == x:
            return i
        i = i + 1
    return -1
```

# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    i = 0
    while i < n:
        if A[i] == x:
            return i
        i = i + 1
    return -1
```

=

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

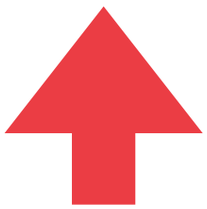
# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

# 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

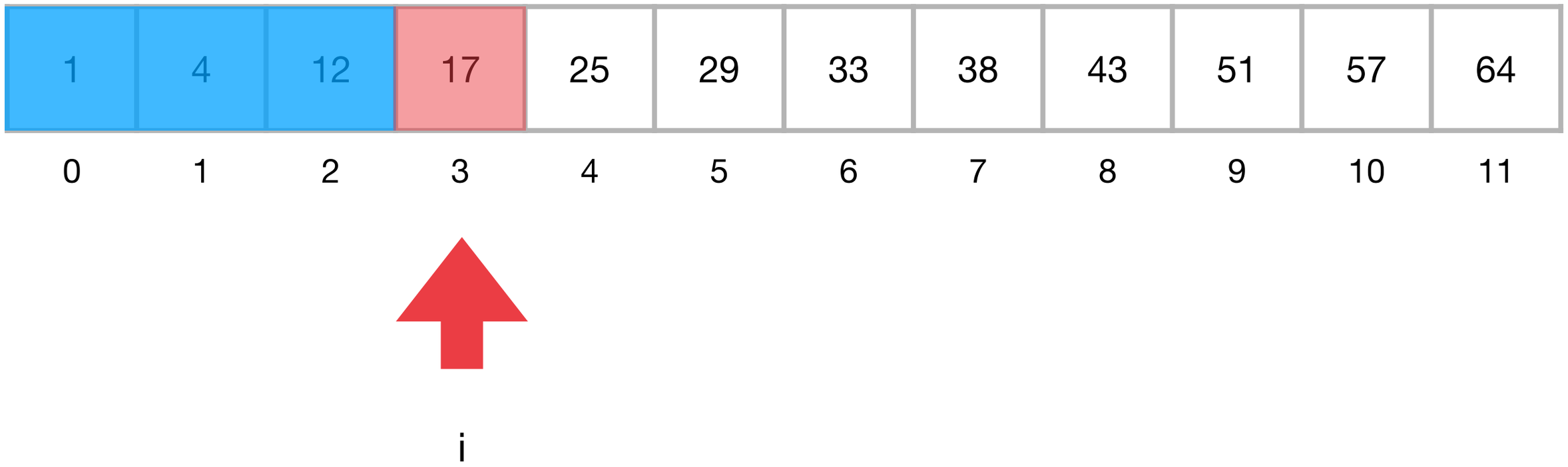
Recherche de 33

| 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



# 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

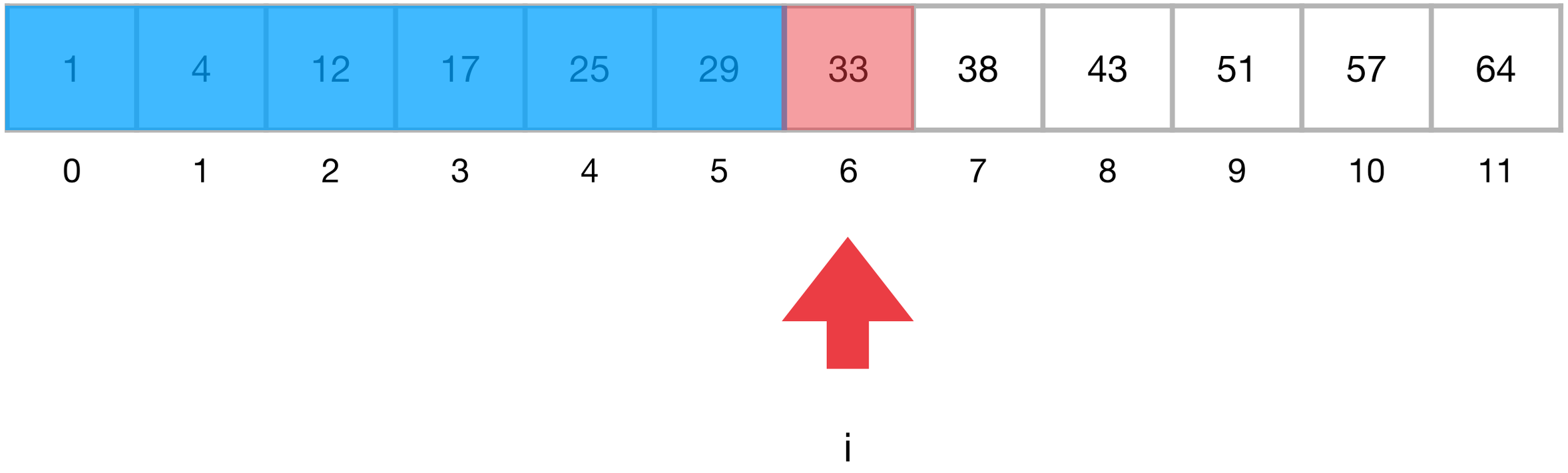
Recherche de 33

| 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

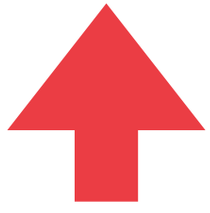
Recherche de 33



# 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 |

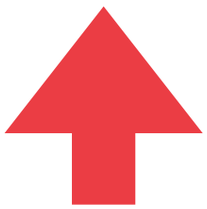


i

# 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

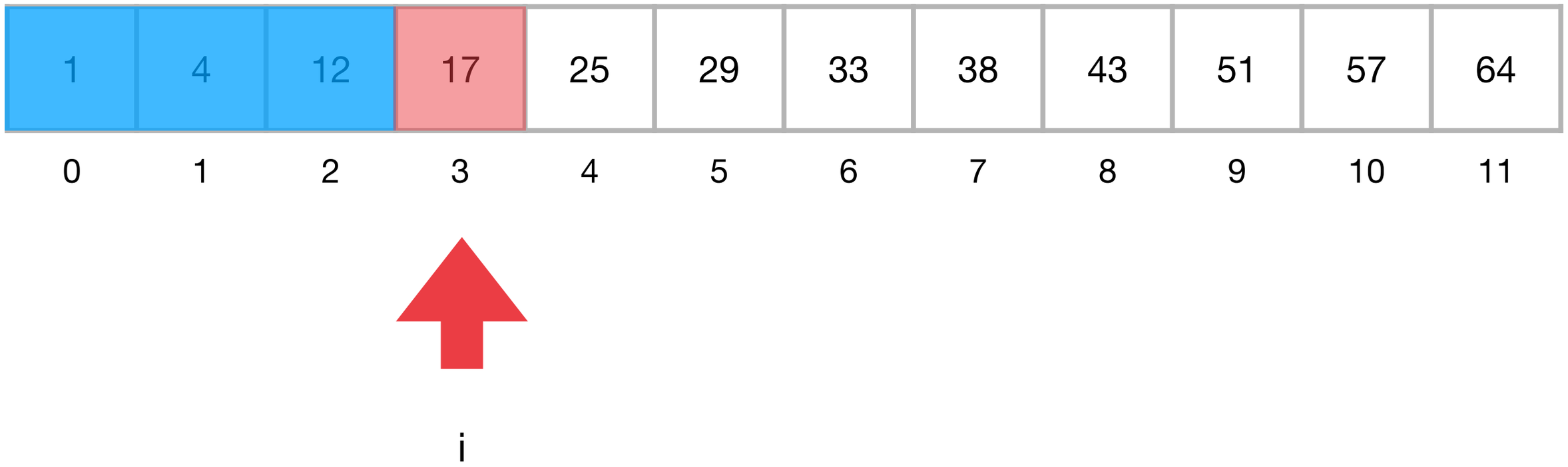
Recherche de 3

| 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

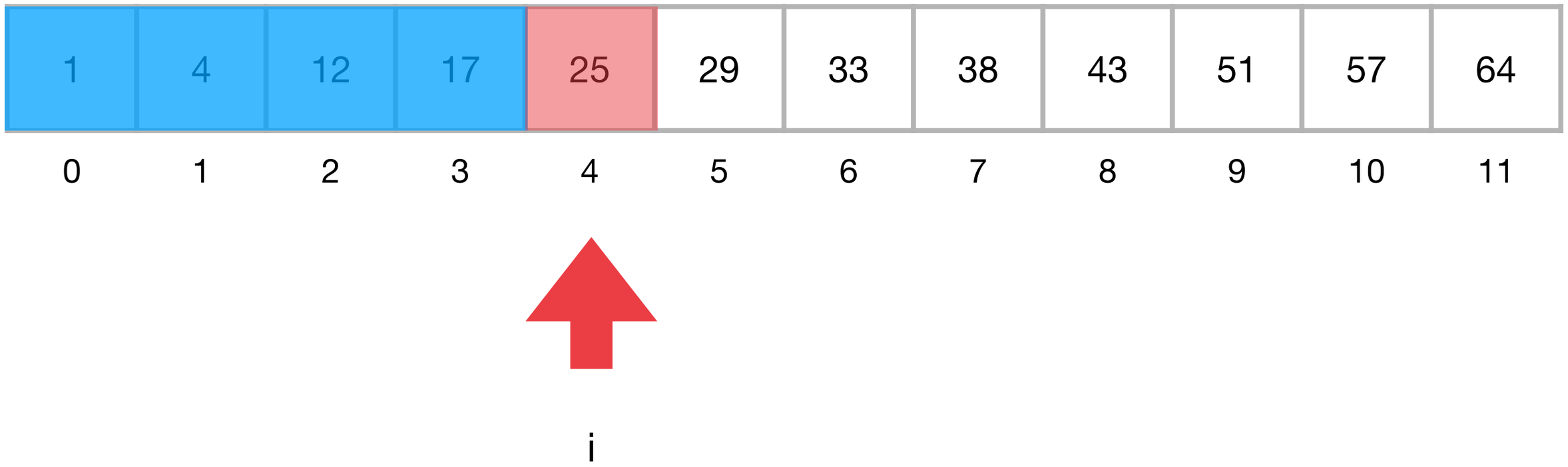
# Recherche linéaire

Recherche de 3



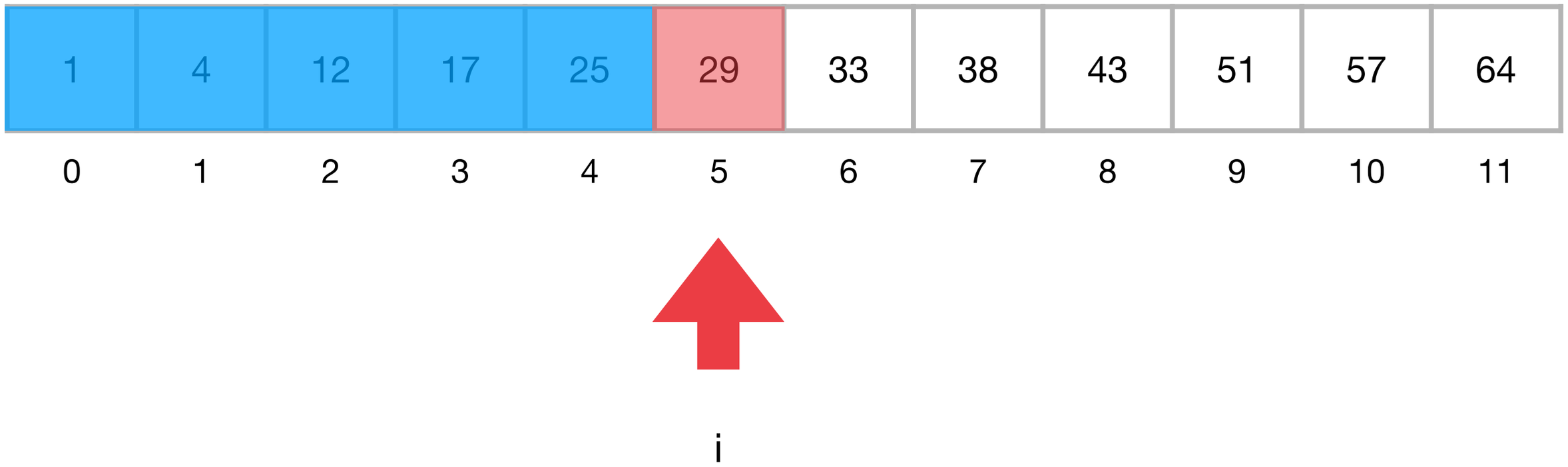
# Recherche linéaire

Recherche de 3



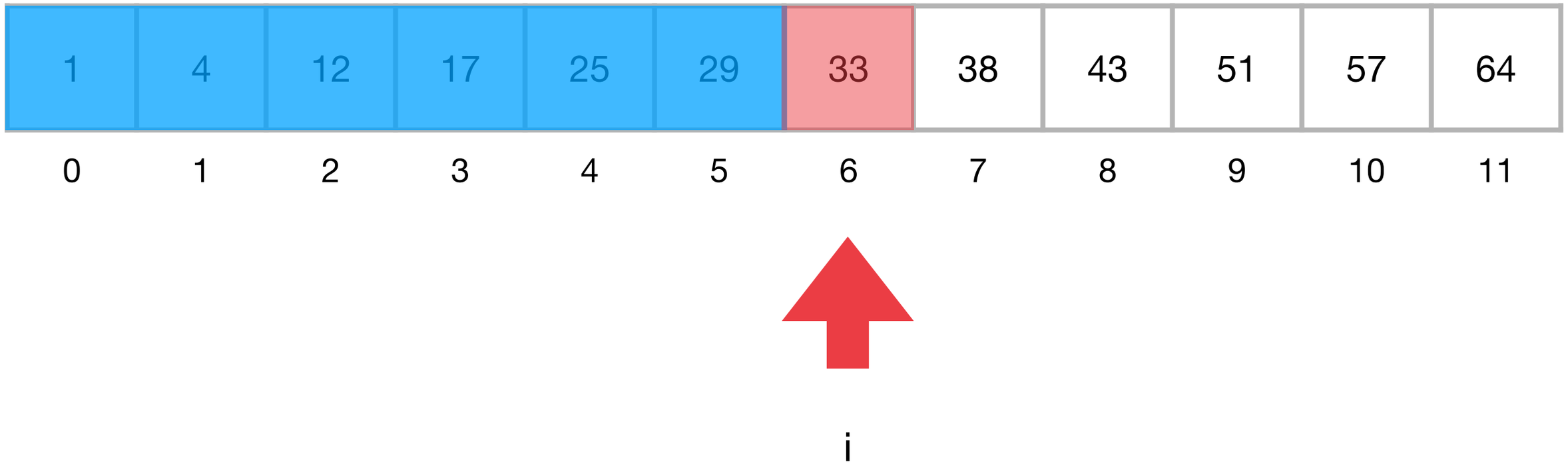
# Recherche linéaire

Recherche de 3



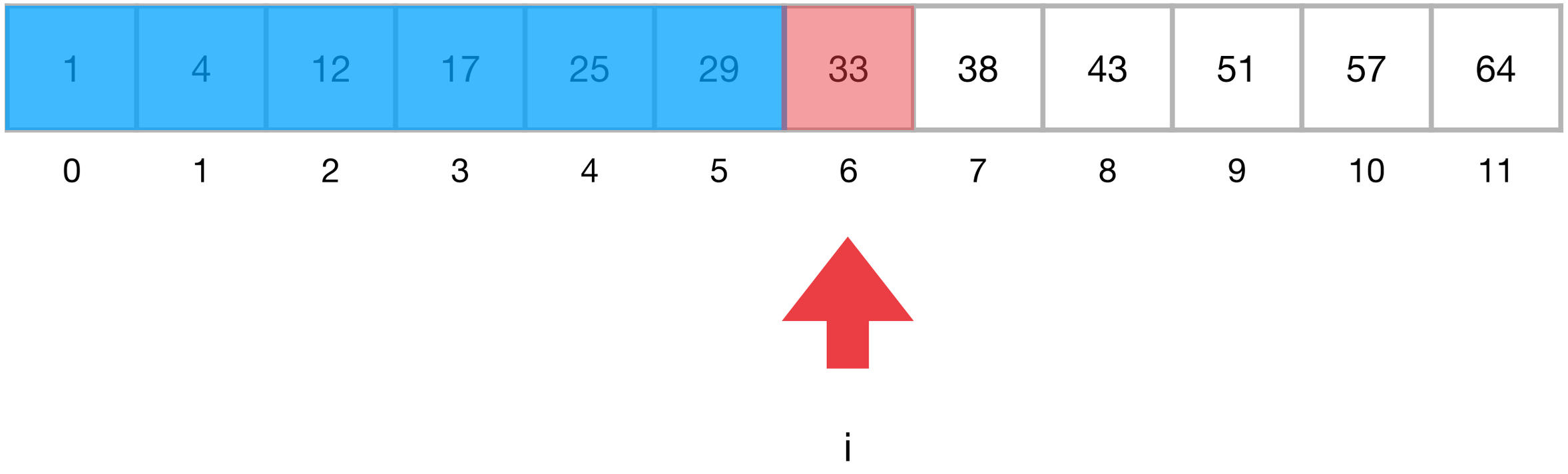
# Recherche linéaire

Recherche de 3



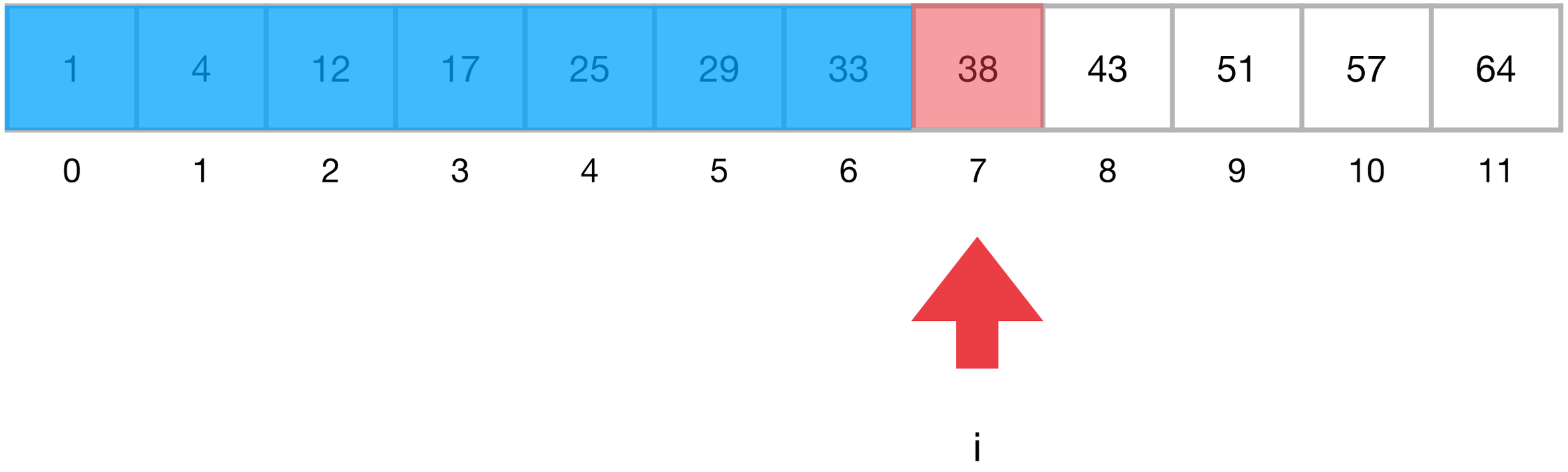
# Recherche linéaire

Recherche de 3



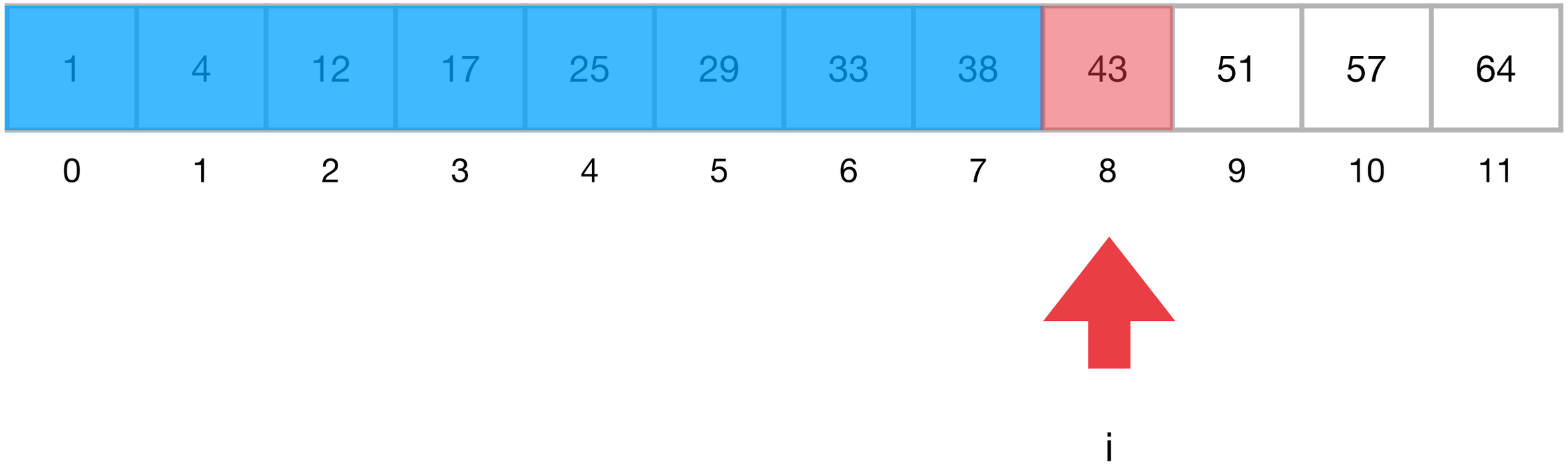
# Recherche linéaire

Recherche de 3



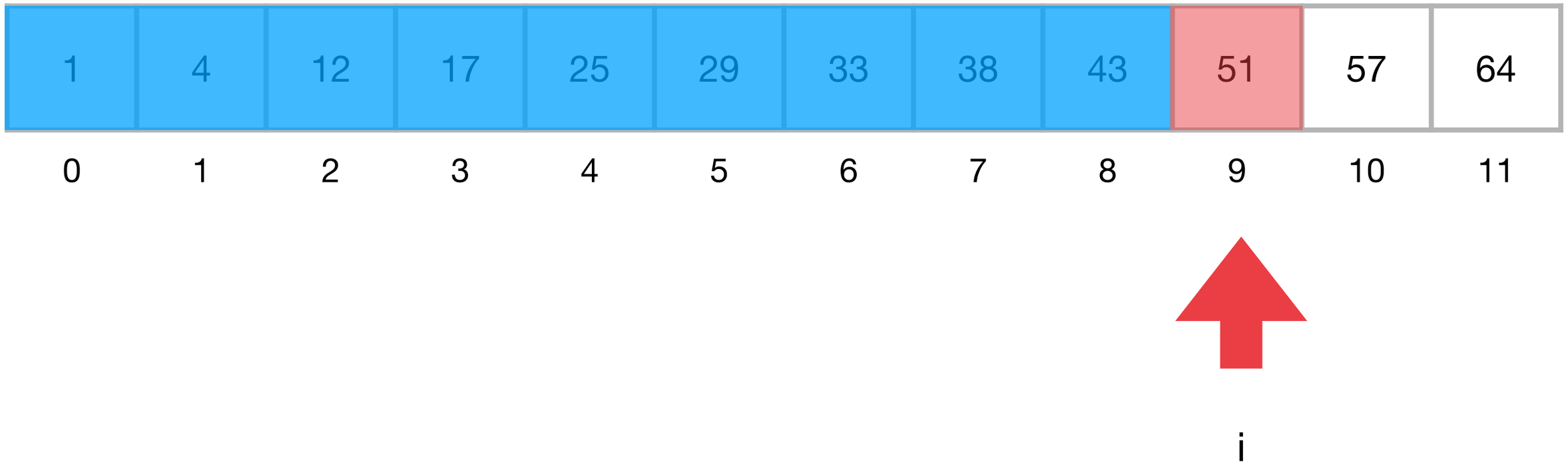
# Recherche linéaire

Recherche de 3



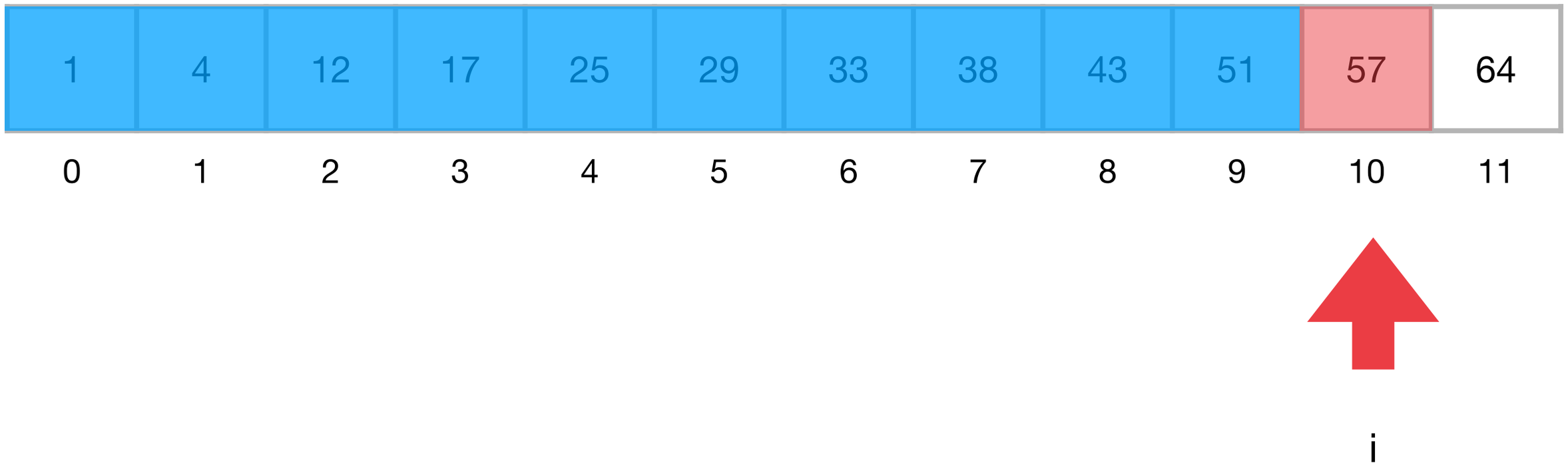
# Recherche linéaire

Recherche de 3



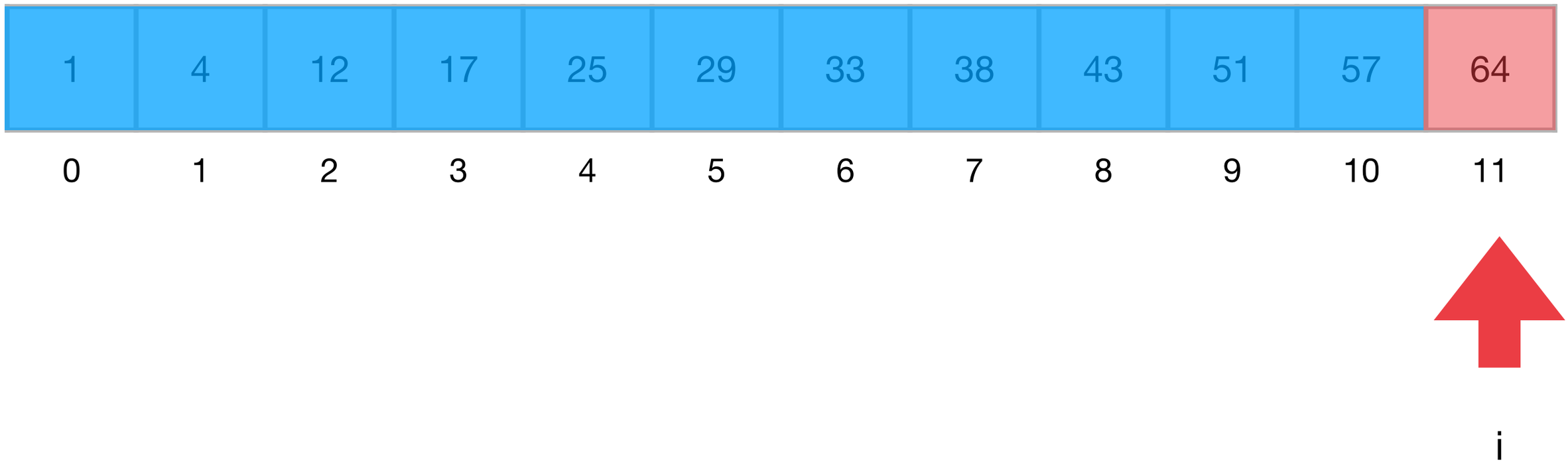
# Recherche linéaire

Recherche de 3



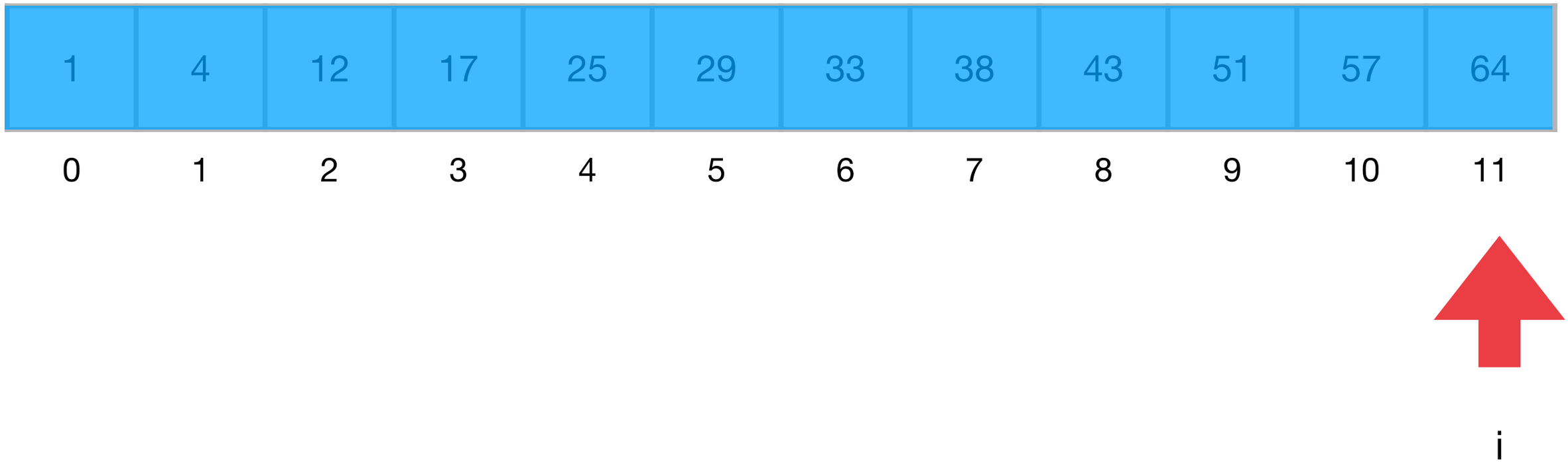
# Recherche linéaire

Recherche de 3



# Recherche linéaire

Recherche de 3



# 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

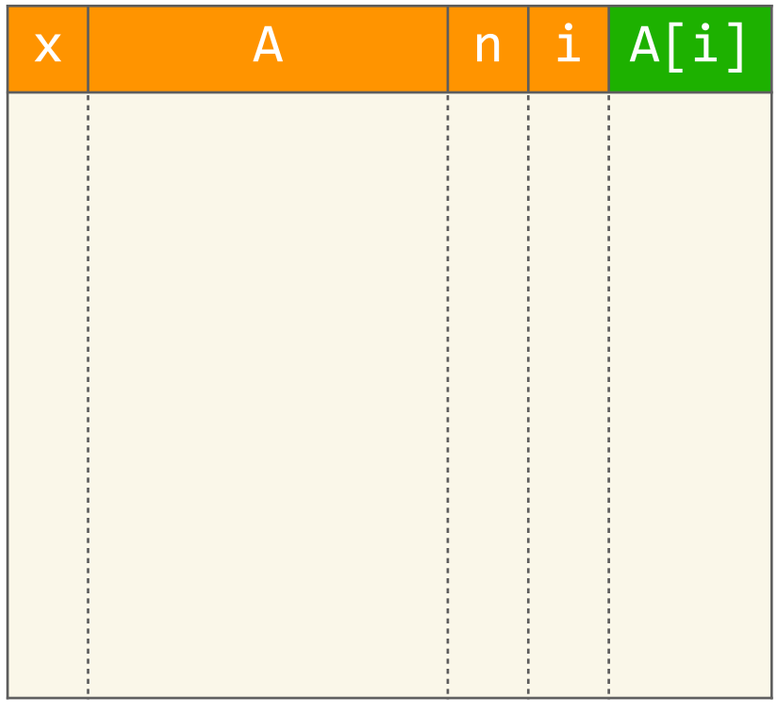
```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

```
>>> chercher(3, [6,2,1,3,5])
```

# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

```
>>> chercher(3, [6,2,1,3,5])
```



# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

```
>>> chercher(3, [6,2,1,3,5])
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 3 | [6,2,1,3,5] | | | |

# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

```
>>> chercher(3, [6,2,1,3,5])
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 3 | [6,2,1,3,5] | 5 | | |

# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

```
>>> chercher(3, [6,2,1,3,5])
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 3 | [6,2,1,3,5] | 5 | 0 | |

# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

```
>>> chercher(3, [6,2,1,3,5])
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 3 | [6,2,1,3,5] | 5 | 0 | 6 |

# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

```
>>> chercher(3, [6,2,1,3,5])
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 3 | [6,2,1,3,5] | 5 | 0 | 6 |

# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

```
>>> chercher(3, [6,2,1,3,5])
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 3 | [6,2,1,3,5] | 5 | 0 | 6 |
| | | | 1 | |

# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

```
>>> chercher(3, [6,2,1,3,5])
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 3 | [6,2,1,3,5] | 5 | 0 | 6 |
| | | | 1 | 2 |

# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

```
>>> chercher(3, [6,2,1,3,5])
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 3 | [6,2,1,3,5] | 5 | 0 | 6 |
| | | | 1 | 2 |

# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

👉

```
>>> chercher(3, [6,2,1,3,5])
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 3 | [6,2,1,3,5] | 5 | 0 | 6 |
| | | | 1 | 2 |
| | | | 2 | |

# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

👉

```
>>> chercher(3, [6,2,1,3,5])
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 3 | [6,2,1,3,5] | 5 | 0 | 6 |
| | | | 1 | 2 |
| | | | 2 | 1 |

# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

```
>>> chercher(3, [6,2,1,3,5])
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 3 | [6,2,1,3,5] | 5 | 0 | 6 |
| | | | 1 | 2 |
| | | | 2 | 1 |

# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

```
>>> chercher(3, [6,2,1,3,5])
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 3 | [6,2,1,3,5] | 5 | 0 | 6 |
| | | | 1 | 2 |
| | | | 2 | 1 |
| | | | 3 | |

# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

👉

```
>>> chercher(3, [6,2,1,3,5])
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 3 | [6,2,1,3,5] | 5 | 0 | 6 |
| | | | 1 | 2 |
| | | | 2 | 1 |
| | | | 3 | 3 |

# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

```
>>> chercher(3, [6,2,1,3,5])
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 3 | [6,2,1,3,5] | 5 | 0 | 6 |
| | | | 1 | 2 |
| | | | 2 | 1 |
| | | | 3 | 3 |

# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1

>>> chercher(3, [6,2,1,3,5])
3
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 3 | [6,2,1,3,5] | 5 | 0 | 6 |
| | | | 1 | 2 |
| | | | 2 | 1 |
| | | | 3 | 3 |

# Si l'élément cherché n'est pas là…

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

```
>>> chercher(7, [6,2,1,3,5])
```

# Si l'élément cherché n'est pas là…

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

```
>>> chercher(7, [6,2,1,3,5])
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 7 | [6,2,1,3,5] | 5 | 0 | 6 |
| | | | 1 | 2 |
| | | | 2 | 1 |
| | | | 3 | 3 |

# Si l'élément cherché n'est pas là…

```
def chercher(x, A):
    n = len(A)
👉  for i in range(n):
        if A[i] == x:
            return i
    return -1

>>> chercher(7, [6,2,1,3,5])
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 7 | [6,2,1,3,5] | 5 | 0 | 6 |
| | | | 1 | 2 |
| | | | 2 | 1 |
| | | | 3 | 3 |

# Si l'élément cherché n'est pas là…

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

```
>>> chercher(7, [6,2,1,3,5])
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 7 | [6,2,1,3,5] | 5 | 0 | 6 |
| | | | 1 | 2 |
| | | | 2 | 1 |
| | | | 3 | 3 |
| | | | 4 | |

# Si l'élément cherché n'est pas là…

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

```
>>> chercher(7, [6,2,1,3,5])
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 7 | [6,2,1,3,5] | 5 | 0 | 6 |
| | | | 1 | 2 |
| | | | 2 | 1 |
| | | | 3 | 3 |
| | | | 4 | 5 |

# Si l'élément cherché n'est pas là…

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

```
>>> chercher(7, [6,2,1,3,5])
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 7 | [6,2,1,3,5] | 5 | 0 | 6 |
| | | | 1 | 2 |
| | | | 2 | 1 |
| | | | 3 | 3 |
| | | | 4 | 5 |

# Si l'élément cherché n'est pas là…

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

```
>>> chercher(7, [6,2,1,3,5])
```

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 7 | [6,2,1,3,5] | 5 | 0 | 6 |
| | | | 1 | 2 |
| | | | 2 | 1 |
| | | | 3 | 3 |
| | | | 4 | 5 |

# Si l'élément cherché n'est pas là…

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1

>>> chercher(7, [6,2,1,3,5])
-1
```

👉

| x | A | n | i | A[i] |
|---|---|---|---|---|
| 7 | [6,2,1,3,5] | 5 | 0 | 6 |
| | | | 1 | 2 |
| | | | 2 | 1 |
| | | | 3 | 3 |
| | | | 4 | 5 |

# Recherche dans un tableau

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

Terminaison ?

Correction ?

Efficacité ?

# Terminaison

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

# Terminaison

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

- La boucle **`for`** est exécutée un maximum de $n$ fois, pour $i = 0,1,\ldots,n-1$

# Terminaison

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

- La boucle **`for`** est exécutée un maximum de $n$ fois, pour $i = 0,1,\ldots,n-1$
- Si $x$ n'est pas là, on termine avec « **`return -1`** »

# Terminaison

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

- La boucle **`for`** est exécutée un maximum de $n$ fois, pour $i = 0,1,\ldots,n-1$
- Si $x$ n'est pas là, on termine avec « **`return -1`** »
- Sinon, on trouve $x$ quelque part, et on termine avec « **`return i`** »

# Terminaison

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

- La boucle **`for`** est exécutée un maximum de $n$ fois, pour $i = 0,1,\ldots,n-1$
- Si $x$ n'est pas là, on termine avec « **`return -1`** »
- Sinon, on trouve $x$ quelque part, et on termine avec « **`return i`** »
- Dans les deux cas, on s'arrête, donc ça termine toujours

# Correction

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

# Correction

- Si $x$ est dans le tableau, alors il se trouve dans le sous-tableau $A[i, \ldots, n-1]$

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

# Correction

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

- Si $x$ est dans le tableau, alors il se trouve dans le sous-tableau $A[i, \ldots, n-1]$
  - C'est vrai au début : $A[0, \ldots, n-1]$

# Correction

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

- Si $x$ est dans le tableau, alors il se trouve dans le sous-tableau $A[i, \ldots, n-1]$
  - C'est vrai au début : $A[0, \ldots, n-1]$
  - Ça reste vrai à chaque itération de la boucle, parce qu'on s'arrête si on découvre $A[i] = x$

# Correction

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

- Si $x$ est dans le tableau, alors il se trouve dans le sous-tableau $A[i, \ldots, n-1]$
  - C'est vrai au début : $A[0, \ldots, n-1]$
  - Ça reste vrai à chaque itération de la boucle, parce qu'on s'arrête si on découvre $A[i] = x$
- Si on s'arrête avec « `return i` », alors c'est le bon résultat

# Correction

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

- Si $x$ est dans le tableau, alors il se trouve dans le sous-tableau $A[i, \ldots, n-1]$
  - C'est vrai au début : $A[0, \ldots, n-1]$
    - Ça reste vrai à chaque itération de la boucle, parce qu'on s'arrête si on découvre $A[i] = x$
- Si on s'arrête avec « `return i` », alors c'est le bon résultat
- Si on s'arrête avec « `return -1` », c'est parce que $x$ n'est pas là

# Comptage des opérations

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

# Comptage des opérations si $A[k] = x$ est la 1ère occurrence

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

# Comptage des opérations si $A[k] = x$ est la 1ère occurrence

```
def chercher(x, A):
    n = len(A)                          1 op
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

# Comptage des opérations si $A[k] = x$ est la 1ère occurrence

```
def chercher(x, A):
    n = len(A)                      1 op
    for i in range(n):              1 op
        if A[i] == x:
            return i
    return -1
```

# Comptage des opérations si $A[k] = x$ est la 1ère occurrence

```
def chercher(x, A):
    n = len(A)                      1 op
    for i in range(n):              1 op
        if A[i] == x:               1 op
            return i
    return -1
```

# Comptage des opérations si $A[k] = x$ est la 1ère occurrence

```
def chercher(x, A):
    n = len(A)                  1 op
    for i in range(n):          1 op
        if A[i] == x:           1 op
            return i            1 op
    return -1
```

# Comptage des opérations si $A[k] = x$ est la 1ère occurrence

```
def chercher(x, A):
    n = len(A)                      1 op
    for i in range(n):              1 op
        if A[i] == x:               1 op
            return i                1 op
    return -1                       1 op
```

# Comptage des opérations si $A[k] = x$ est la 1ère occurrence

```
def chercher(x, A):
    n = len(A)                  1 op    × 1 fois
    for i in range(n):          1 op
        if A[i] == x:           1 op
            return i            1 op
    return -1                   1 op
```

# Comptage des opérations si $A[k] = x$ est la 1ère occurrence

```
def chercher(x, A):
    n = len(A)                      1 op    × 1 fois
    for i in range(n):              1 op    × (k + 1) fois
        if A[i] == x:               1 op
            return i                1 op
    return -1                       1 op
```

# Comptage des opérations si $A[k] = x$ est la 1ère occurrence

```
def chercher(x, A):
    n = len(A)                      1 op    × 1 fois
    for i in range(n):              1 op    × (k + 1) fois
        if A[i] == x:               1 op    × (k + 1) fois
            return i                1 op
    return -1                       1 op
```

# Comptage des opérations si $A[k] = x$ est la 1ère occurrence

```
def chercher(x, A):
    n = len(A)                      1 op    × 1 fois
    for i in range(n):              1 op    × (k + 1) fois
        if A[i] == x:               1 op    × (k + 1) fois
            return i                1 op    × 1 fois
    return -1                       1 op
```

# Comptage des opérations si $A[k] = x$ est la 1ère occurrence

```
def chercher(x, A):
    n = len(A)                      1 op    × 1 fois
    for i in range(n):              1 op    × (k + 1) fois
        if A[i] == x:               1 op    × (k + 1) fois
            return i                1 op    × 1 fois
    return -1                       1 op    × 0 fois
```

# Comptage des opérations si $A[k] = x$ est la 1ère occurrence

```
def chercher(x, A):
    n = len(A)                      1 op    × 1 fois
    for i in range(n):              1 op    × (k + 1) fois
        if A[i] == x:               1 op    × (k + 1) fois
            return i                1 op    × 1 fois
    return -1                       1 op    × 0 fois
```

$$= 2k + 4$$

# Comptage des opérations si $x$ n'apparaît pas dans $A$

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

# Comptage des opérations si $x$ n'apparaît pas dans $A$

```
def chercher(x, A):
    n = len(A)                  1 op
    for i in range(n):          1 op
        if A[i] == x:           1 op
            return i            1 op
    return -1                   1 op
```

# Comptage des opérations si $x$ n'apparaît pas dans $A$

```
def chercher(x, A):
    n = len(A)                  1 op    × 1 fois
    for i in range(n):          1 op
        if A[i] == x:           1 op
            return i            1 op
    return -1                   1 op
```

# Comptage des opérations si $x$ n'apparaît pas dans $A$

```
def chercher(x, A):
    n = len(A)                  1 op    × 1 fois
    for i in range(n):          1 op
        if A[i] == x:           1 op    × n fois
            return i            1 op
    return -1                   1 op
```

# Comptage des opérations si $x$ n'apparaît pas dans $A$

```
def chercher(x, A):
    n = len(A)                  1 op    × 1 fois
    for i in range(n):          1 op
        if A[i] == x:           1 op    × n fois
            return i            1 op    × 0 fois
    return -1                   1 op
```

# Comptage des opérations si $x$ n'apparaît pas dans $A$

```
def chercher(x, A):
    n = len(A)                  1 op    × 1 fois
    for i in range(n):          1 op
        if A[i] == x:           1 op    × n fois
            return i            1 op    × 0 fois
    return -1                   1 op    × 1 fois
```

# Comptage des opérations si $x$ n'apparaît pas dans $A$

```
def chercher(x, A):
    n = len(A)                      1 op    × 1 fois
    for i in range(n):              1 op    × (n + 1) fois
        if A[i] == x:               1 op    × n fois
            return i                1 op    × 0 fois
    return -1                       1 op    × 1 fois
```

# Comptage des opérations si $x$ n'apparaît pas dans $A$

```
def chercher(x, A):
    n = len(A)                          1 op    × 1 fois
    for i in range(n):                  1 op    × (n + 1) fois
        if A[i] == x:                   1 op    × n fois
            return i                    1 op    × 0 fois
    return -1                           1 op    × 1 fois
```

$$= 2n + 3$$

# Efficacité

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

# Efficacité

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

- Si on a de la chance, on a $A[0] = x$ et on termine tout de suite en 4 opérations

# Efficacité

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

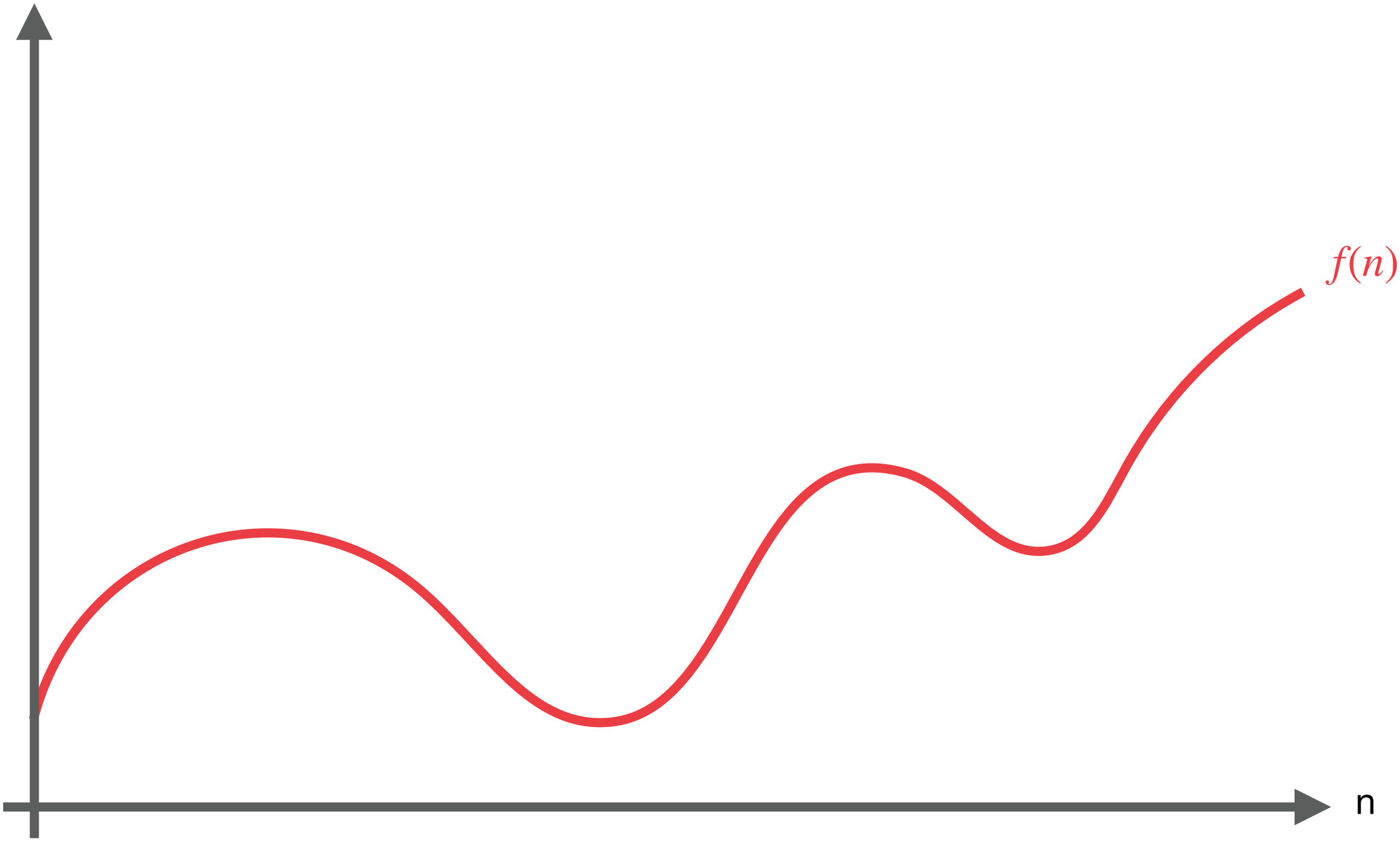
- Si on a de la chance, on a $A[0] = x$ et on termine tout de suite en 4 opérations
- Si $A[k] = x$ (1ère occurrence) on fait $2k + 4$ opérations

# Efficacité

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

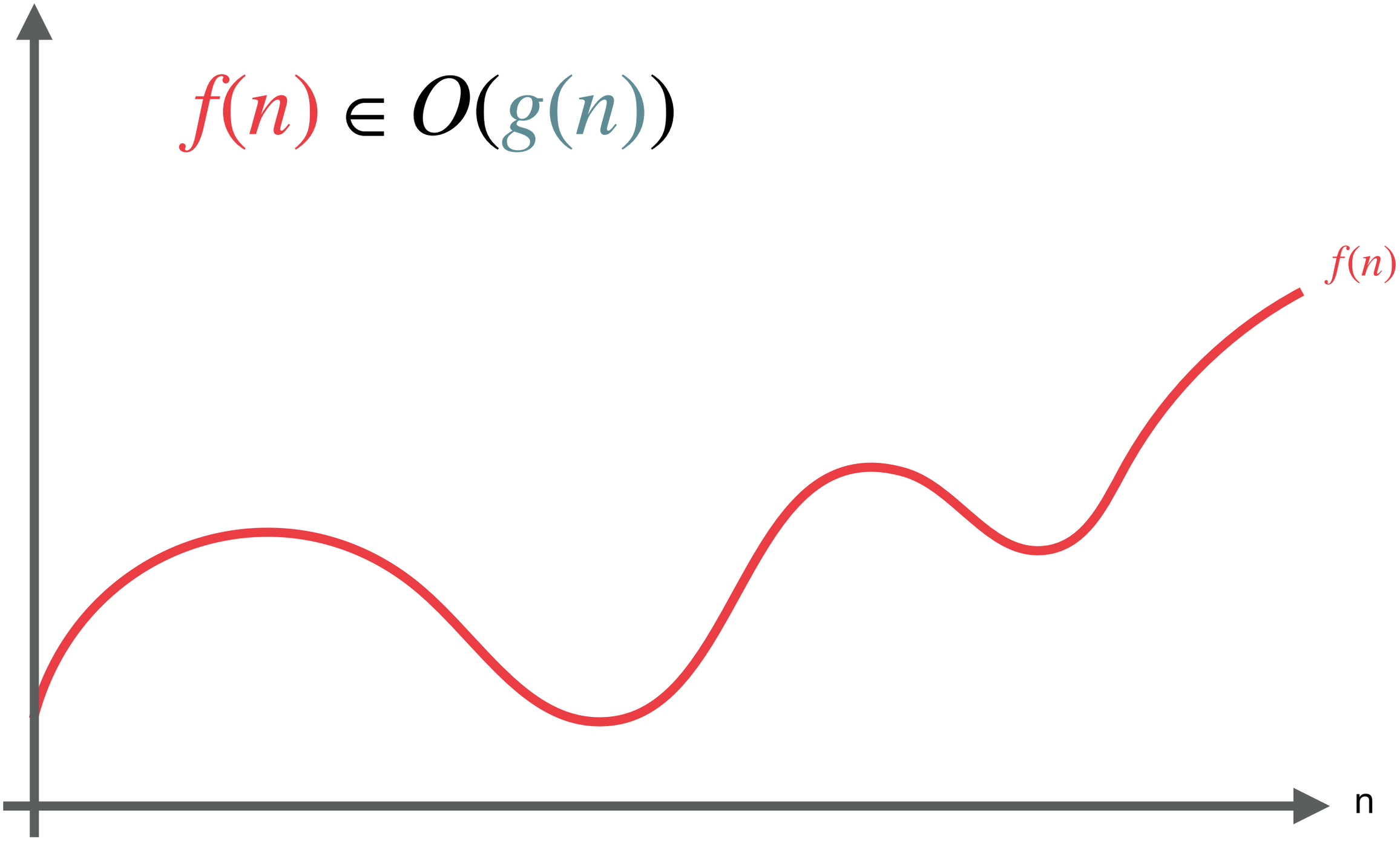
- Si on a de la chance, on a $A[0] = x$ et on termine tout de suite en 4 opérations
- Si $A[k] = x$ (1ère occurrence) on fait $2k + 4$ opérations
- Si $x$ n'est pas là on fait $2n + 3$ opérations

# Notation « grand O »



# Notation « grand O »

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

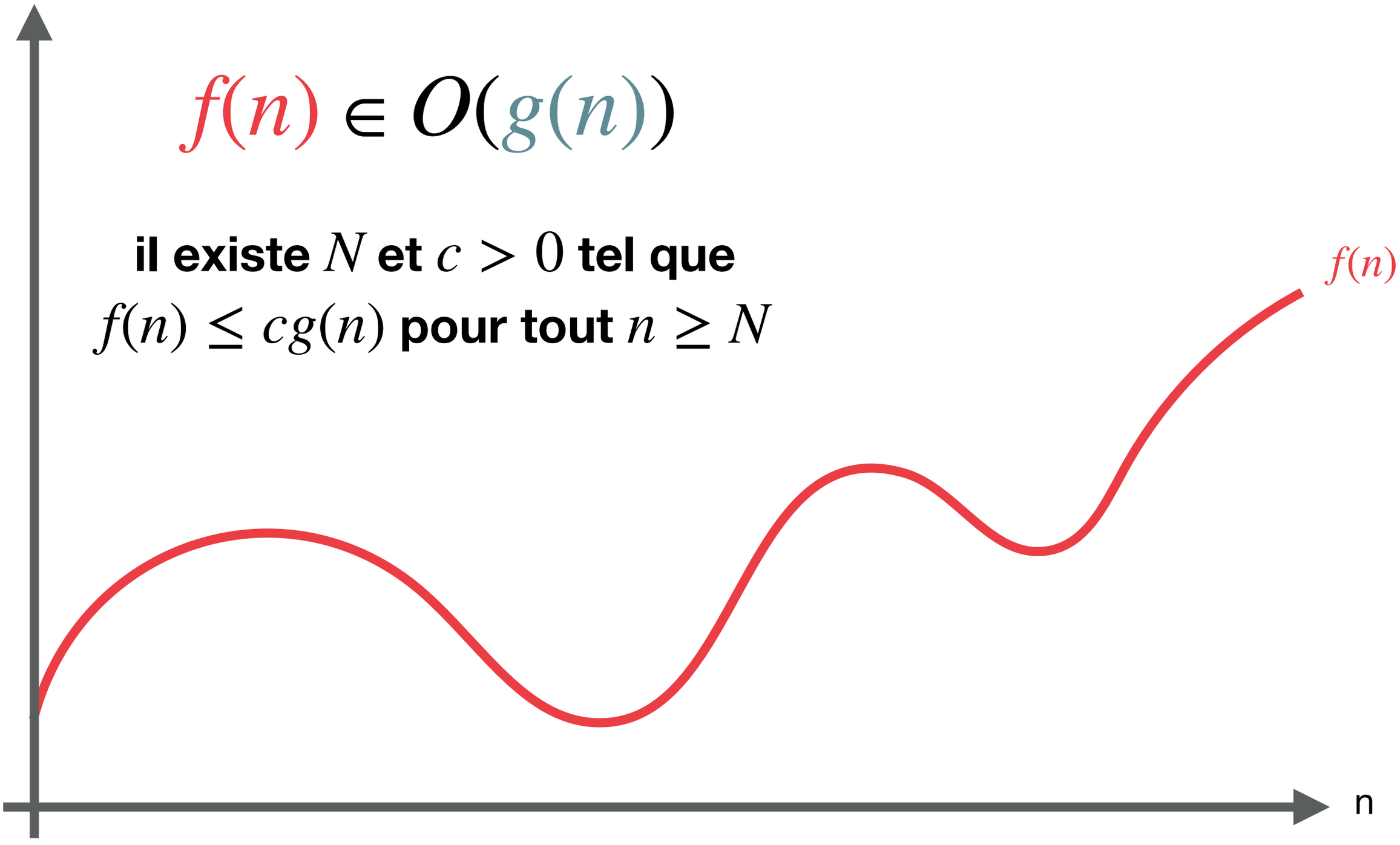


# Notation « grand O »

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

**il existe** $N$ **et** $c > 0$ **tel que**

$f(n) \leq cg(n)$ **pour tout** $n \geq N$

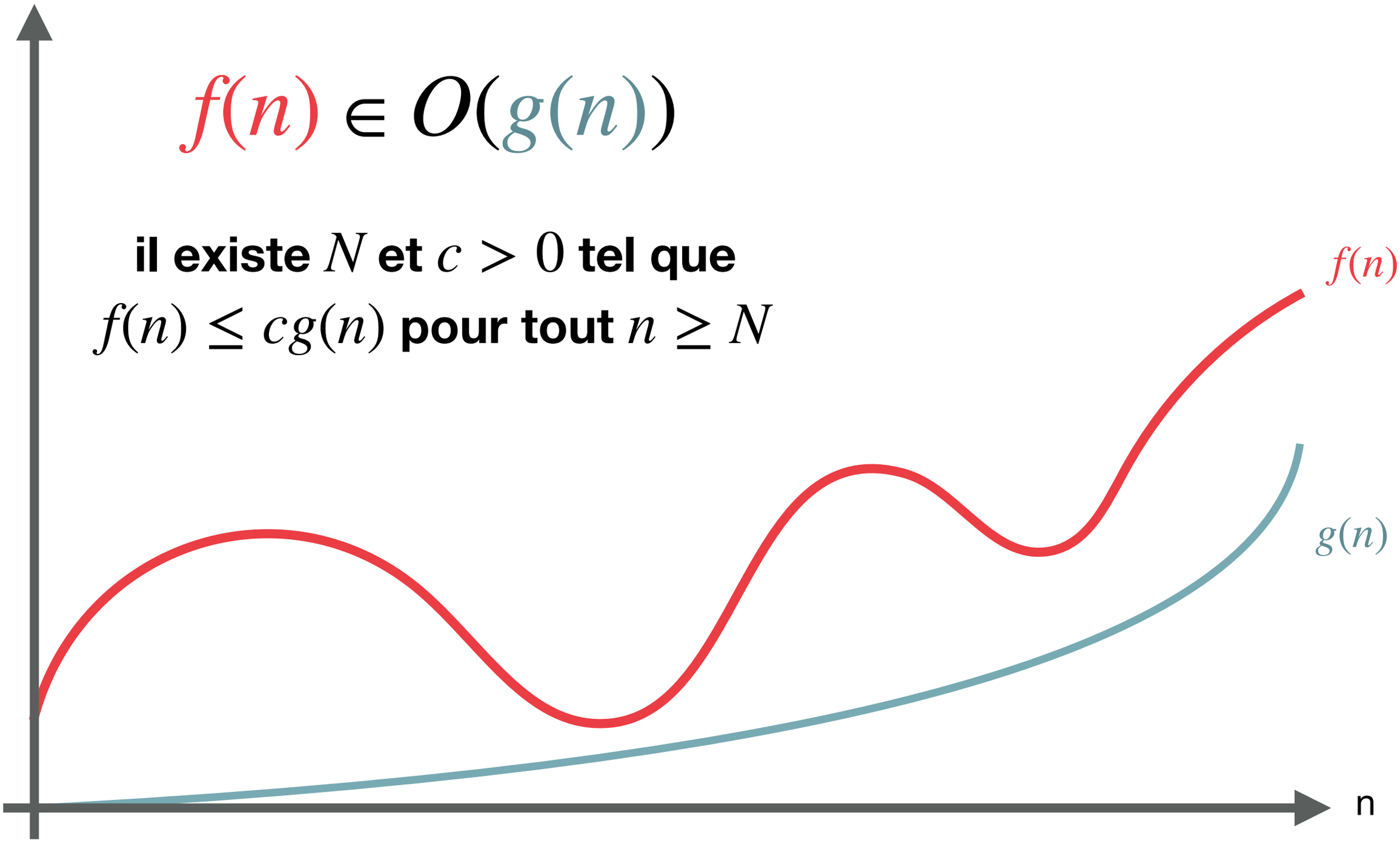


# Notation « grand O »

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

**il existe** $N$ **et** $c > 0$ **tel que**

$f(n) \leq cg(n)$ **pour tout** $n \geq N$

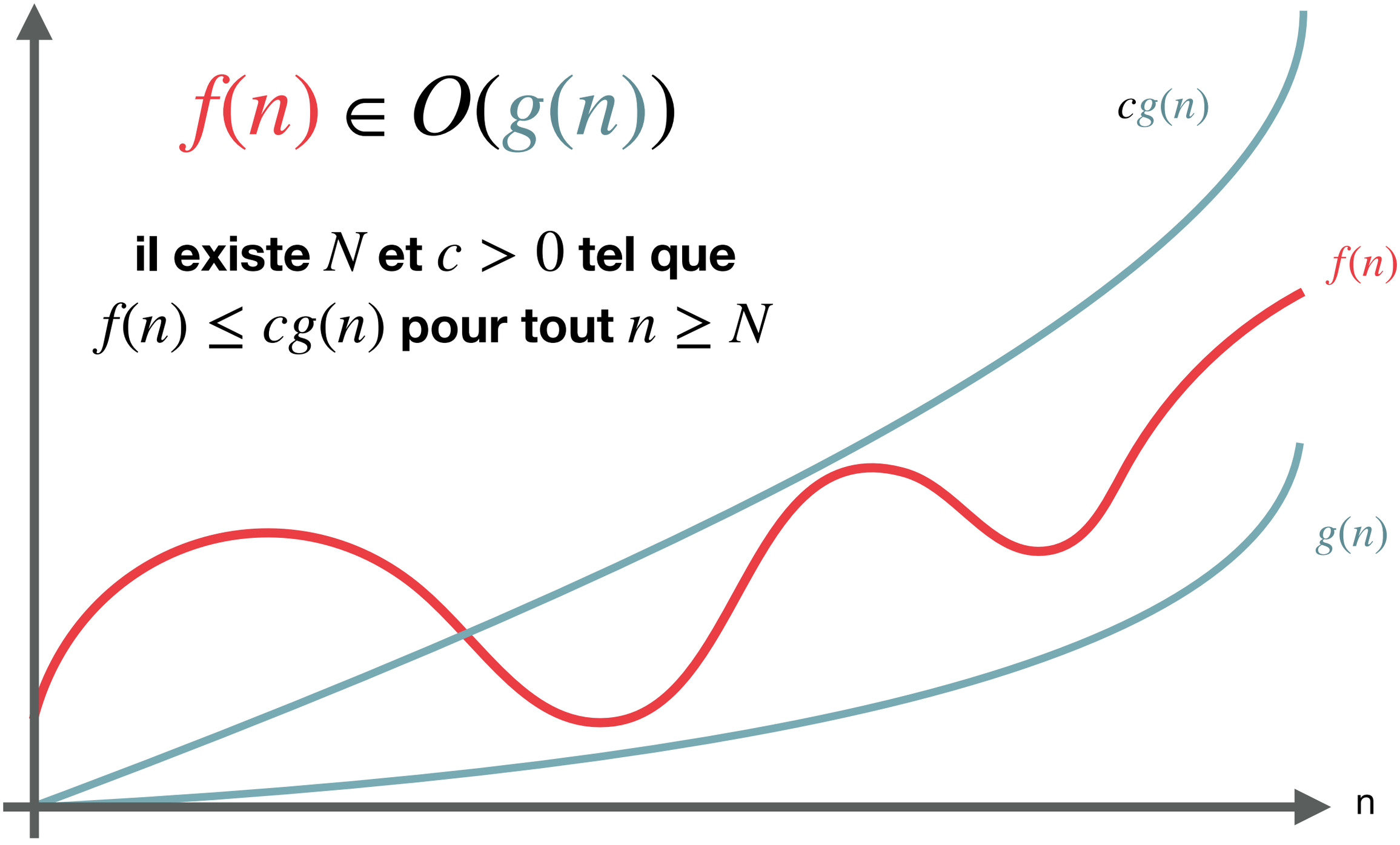


# Notation « grand O »

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

**il existe** $N$ **et** $c > 0$ **tel que**

$f(n) \leq cg(n)$ **pour tout** $n \geq N$



# Ordres de grandeur

Il existe $N$ et $c > 0$ tel que $f(n) \leq cg(n)$ pour tout $n \geq N$

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

# Efficacité

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

# Efficacité

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

- Le pire des cas: si $x$ n'est pas là on fait $2n + 3$ opérations

# Efficacité

```
def chercher(x, A):
    n = len(A)
    for i in range(n):
        if A[i] == x:
            return i
    return -1
```

- Le pire des cas: si $x$ n'est pas là on fait $2n + 3$ opérations

- En simplifiant avec la notation « grand O », ça donne $O(n)$

# Peut-on faire mieux que $O(n)$ pour la recherche dans un tableau ?

# Recherche dans un annuaire ou un dictionnaire ?

# Recherche dichotomique dans un tableau trié

```
def rechercher(x, A):
    n = len(A)
    i = 0
    j = n - 1
    while i <= j:
        m = (i + j) // 2
        if x == A[m]:
            return m
        elif x < A[m]:
            j = m - 1
        else:
            i = m + 1
    return -1
```

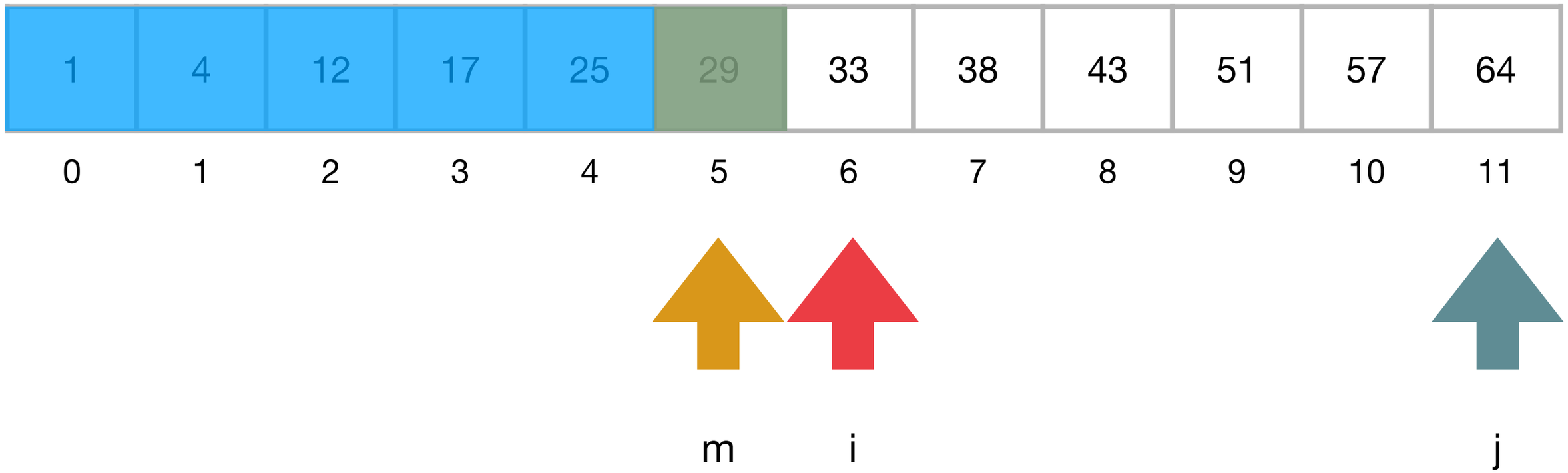
# Recherche dichotomique

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 |

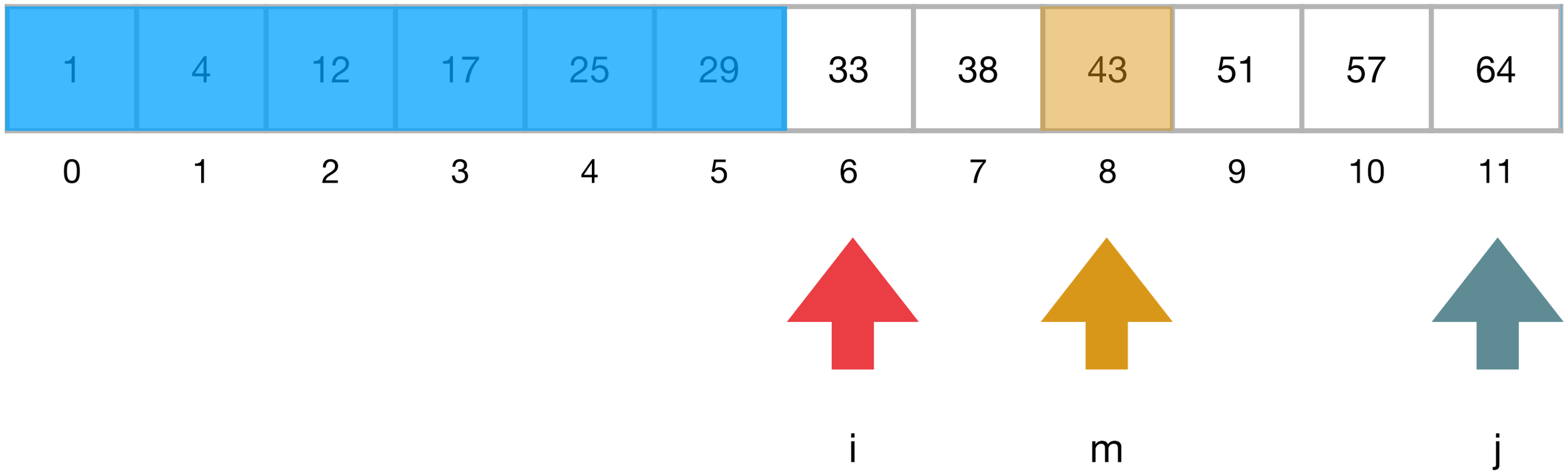
# Recherche dichotomique

Recherche de 33



# Recherche dichotomique

Recherche de 33



# Recherche dichotomique

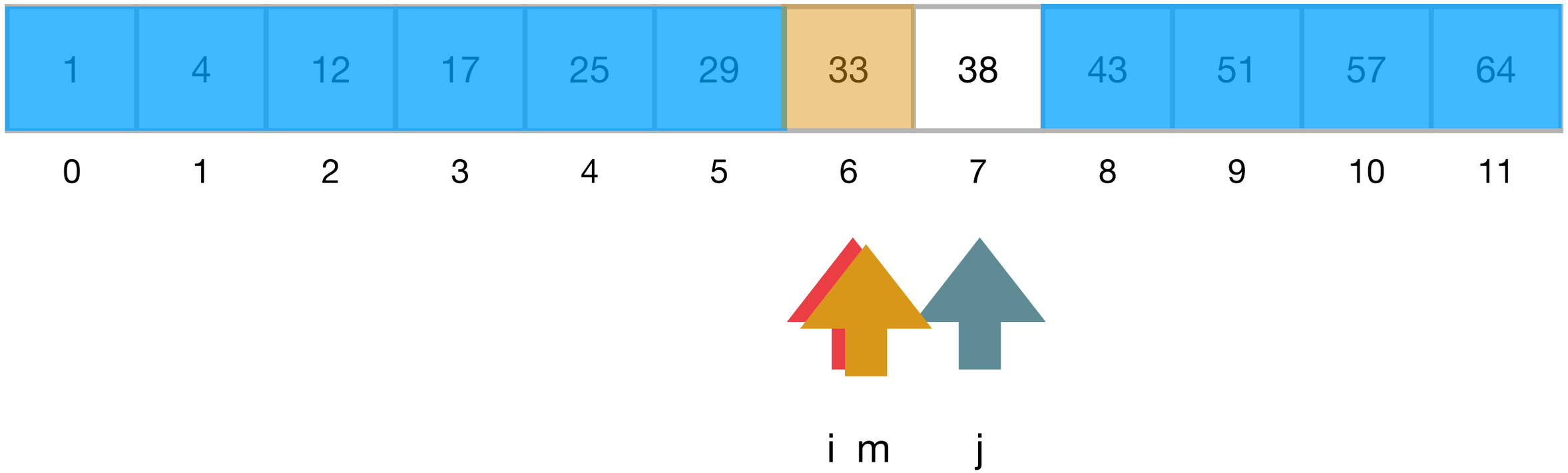
Recherche de 33

| 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 j m

# Recherche dichotomique

Recherche de 33



# Recherche dichotomique

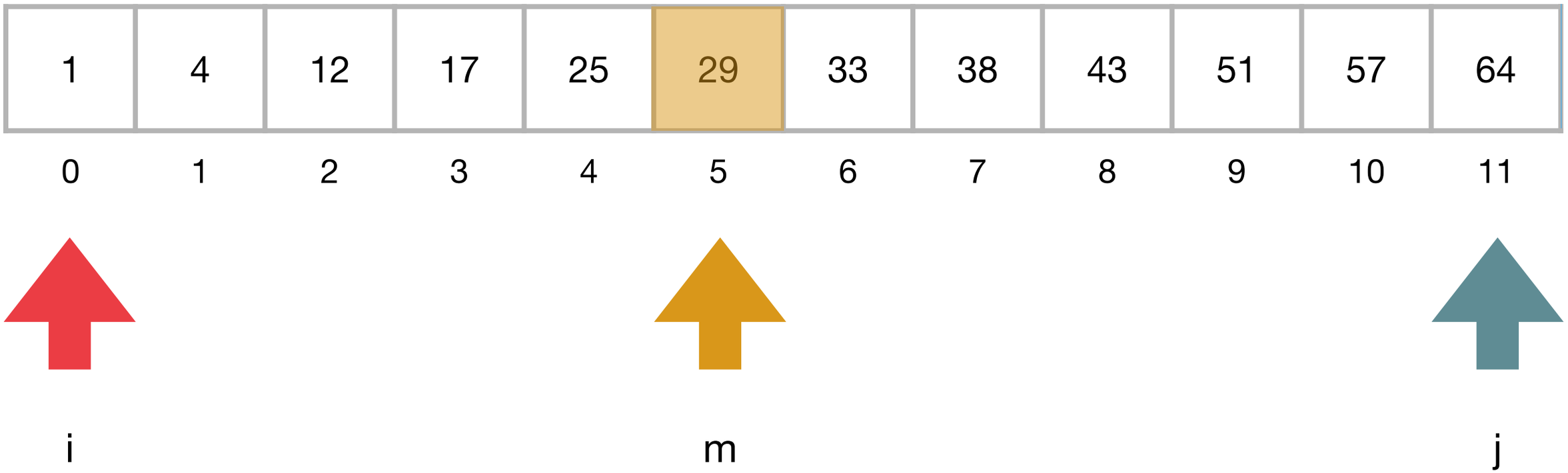
Recherche de 33

| 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

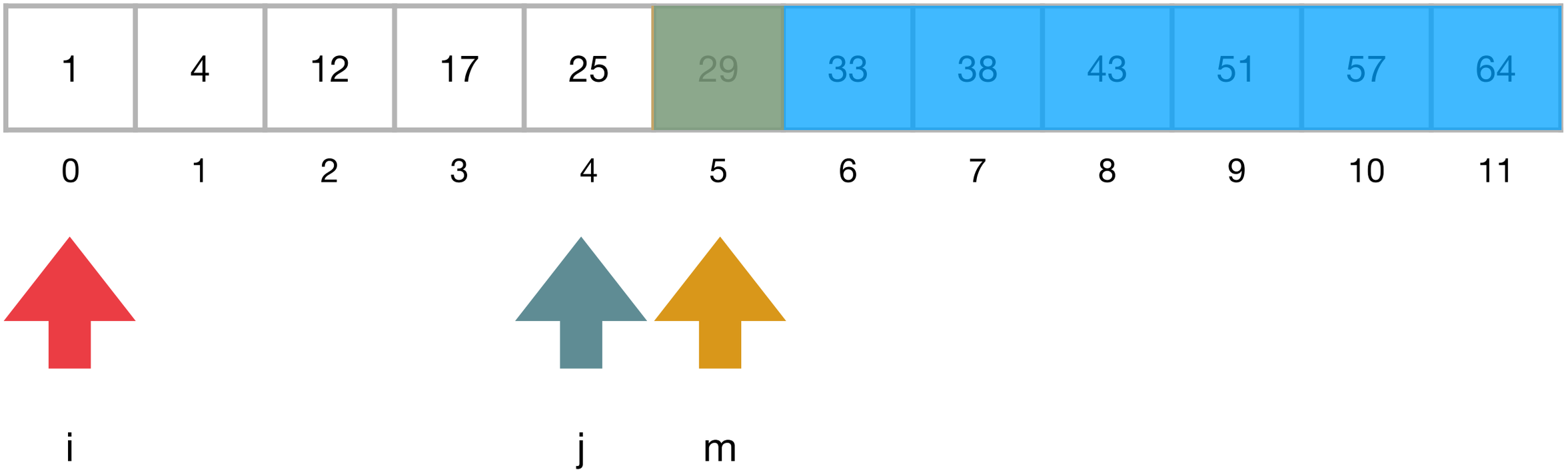
# Recherche dichotomique

Recherche de 16



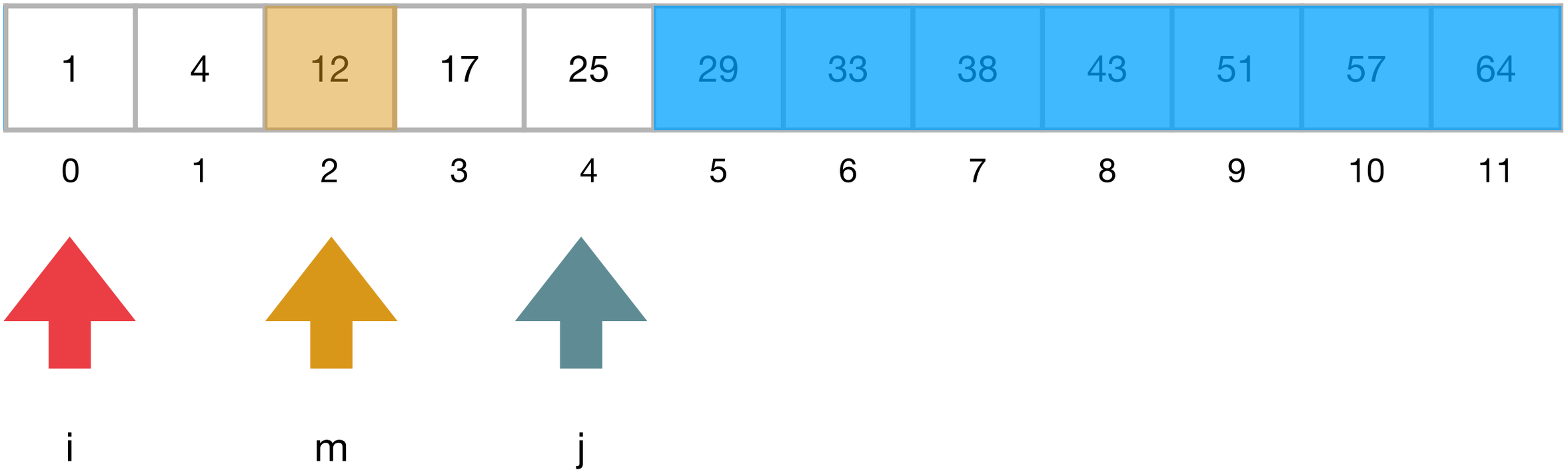
# Recherche dichotomique

Recherche de 16



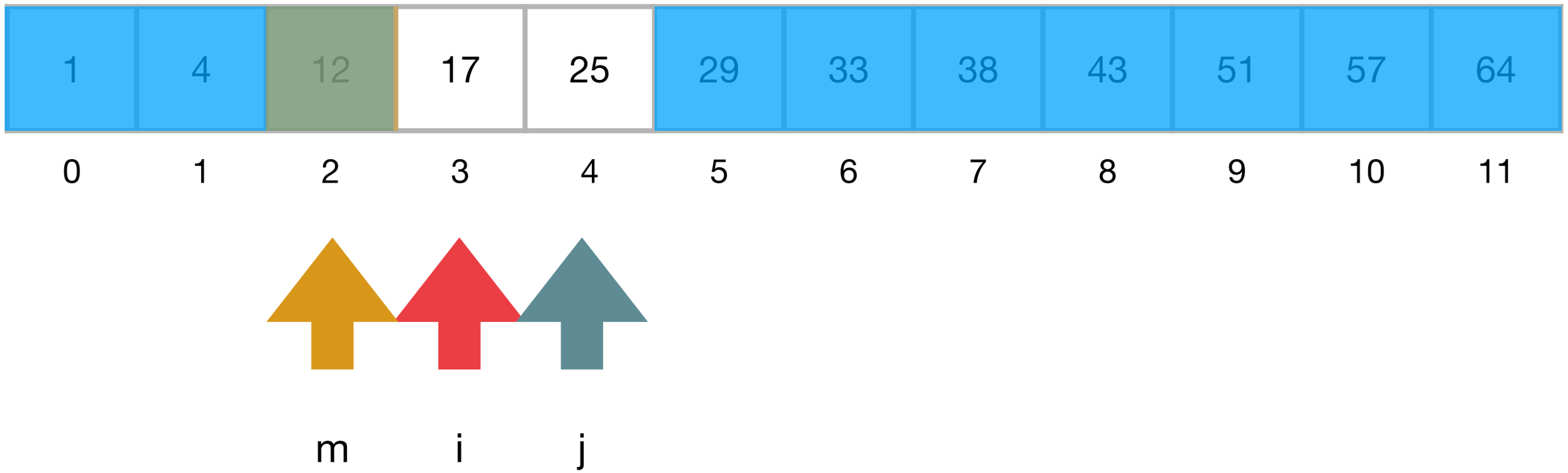
# Recherche dichotomique

Recherche de 16



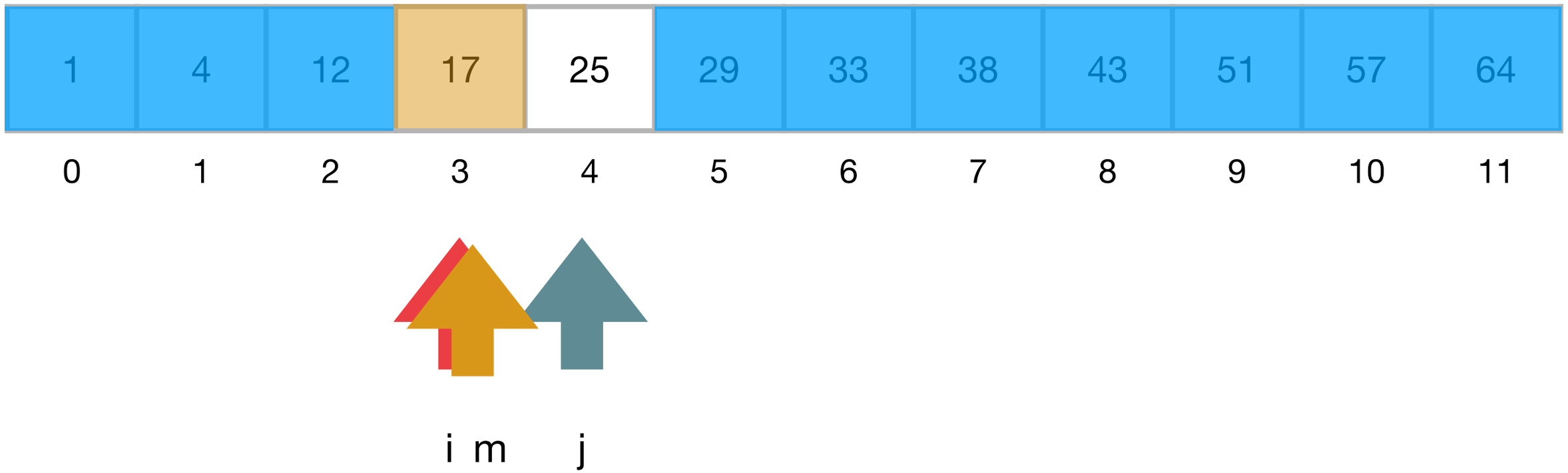
# Recherche dichotomique

Recherche de 16



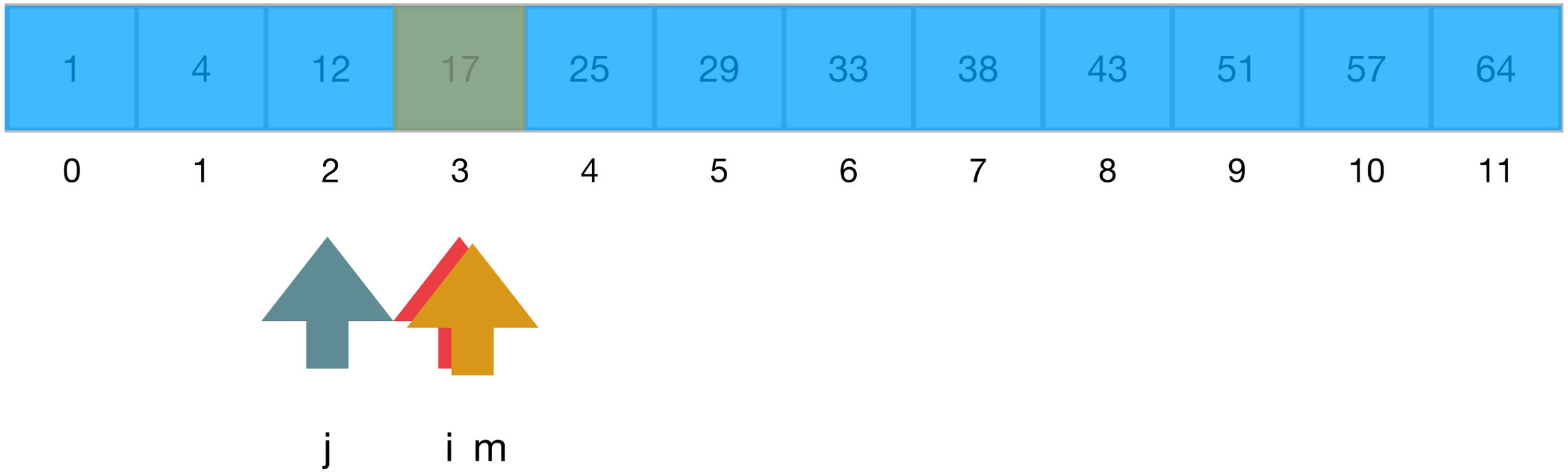
# Recherche dichotomique

Recherche de 16



# Recherche dichotomique

Recherche de 16



# 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 |

j i m

# Recherche dichotomique dans un tableau d'entiers trié

```
def rechercher(x, A):
    n = len(A)
    i = 0
    j = n - 1
    while i <= j:
        m = (i + j) // 2
        if x == A[m]:
            return m
        elif x < A[m]:
            j = m - 1
        else:
            i = m + 1
return -1
```

**Terminaison ?**

**Correction ?**

**Efficacité ?**

# Terminaison

```
def rechercher(x, A):
    n = len(A)
    i = 0
    j = n - 1
    while i <= j:
        m = (i + j) // 2
        if x == A[m]:
            return m
        elif x < A[m]:
            j = m - 1
        else:
            i = m + 1
    return -1
```

# Terminaison

```
def rechercher(x, A):
    n = len(A)
    i = 0
    j = n - 1
    while i <= j:
        m = (i + j) // 2
        if x == A[m]:
            return m
        elif x < A[m]:
            j = m - 1
        else:
            i = m + 1
    return -1
```

- On termine quand $i > j$, dans le pire des cas

# Terminaison

```
def rechercher(x, A):
    n = len(A)
    i = 0
    j = n - 1
    while i <= j:
        m = (i + j) // 2
        if x == A[m]:
            return m
        elif x < A[m]:
            j = m - 1
        else:
            i = m + 1
    return -1
```

- On termine quand $i > j$, dans le pire des cas
- À chaque itération, soit $i$ est incrémentée, soit $j$ est décrémentée strictement

# Terminaison

```
def rechercher(x, A):
    n = len(A)
    i = 0
    j = n - 1
    while i <= j:
        m = (i + j) // 2
        if x == A[m]:
            return m
        elif x < A[m]:
            j = m - 1
        else:
            i = m + 1
    return -1
```

- On termine quand $i > j$, dans le pire des cas
- À chaque itération, soit $i$ est incrémentée, soit $j$ est décrémentée strictement
- Soit on trouve $x$, et on s'arrête immédiatement, soit il n'est pas là, et donc tôt ou tard $i > j$

# Correction

```
def rechercher(x, A):
    n = len(A)
    i = 0
    j = n - 1
    while i <= j:
        m = (i + j) // 2
        if x == A[m]:
            return m
        elif x < A[m]:
            j = m - 1
        else:
            i = m + 1
    return -1
```

# Correction

```
def rechercher(x, A):
    n = len(A)
    i = 0
    j = n - 1
    while i <= j:
        m = (i + j) // 2
        if x == A[m]:
            return m
        elif x < A[m]:
            j = m - 1
        else:
            i = m + 1
    return -1
```

- Si $A$ est trié et que $x$ est dans $A$, alors il se trouve dans le sous-tableau $A[i, \ldots, j]$

# Correction

```
def rechercher(x, A):
    n = len(A)
    i = 0
    j = n - 1
while i <= j:
    m = (i + j) // 2
    if x == A[m]:
        return m
    elif x < A[m]:
        j = m - 1
    else:
        i = m + 1
return -1
```

- Si $A$ est trié et que $x$ est dans $A$, alors il se trouve dans le sous-tableau $A[i, \ldots, j]$
  - C'est vrai au début : $A[0, \ldots, n-1]$

# Correction

```
def rechercher(x, A):
    n = len(A)
    i = 0
    j = n - 1
while i <= j:
    m = (i + j) // 2
    if x == A[m]:
        return m
    elif x < A[m]:
        j = m - 1
    else:
        i = m + 1
return -1
```

- Si $A$ est trié et que $x$ est dans $A$, alors il se trouve dans le sous-tableau $A[i, \ldots, j]$

  - C'est vrai au début : $A[0, \ldots, n-1]$

- Ça reste vrai à chaque itération de la boucle, parce qu'on vérifie toujours si $x = A[m]$ ou $x < A[m]$ ou $x > A[m]$

# Correction

```
def rechercher(x, A):
    n = len(A)
    i = 0
    j = n - 1
    while i <= j:
        m = (i + j) // 2
        if x == A[m]:
            return m
        elif x < A[m]:
            j = m - 1
        else:
            i = m + 1
    return -1
```

- Si $A$ est trié et que $x$ est dans $A$, alors il se trouve dans le sous-tableau $A[i, \ldots, j]$
    - C'est vrai au début : $A[0, \ldots, n-1]$
    - Ça reste vrai à chaque itération de la boucle, parce qu'on vérifie toujours si $x = A[m]$ ou $x < A[m]$ ou $x > A[m]$
- Si on sort de la boucle avec $i > j$, alors $x$ n'est pas là

# Efficacité

```
def rechercher(x, A):
    n = len(A)
    i = 0
    j = n - 1
    while i <= j:
        m = (i + j) // 2
        if x == A[m]:
            return m
        elif x < A[m]:
            j = m - 1
        else:
            i = m + 1
    return -1
```

# Efficacité

```
def rechercher(x, A):
    n = len(A)
    i = 0
    j = n - 1
    while i <= j:
        m = (i + j) // 2
        if x == A[m]:
            return m
        elif x < A[m]:
            j = m - 1
        else:
            i = m + 1
    return -1
```

- Dans le pire des cas, $x$ n'est pas là

# Efficacité

```
def rechercher(x, A):
    n = len(A)
    i = 0
    j = n - 1
    while i <= j:
        m = (i + j) // 2
        if x == A[m]:
            return m
        elif x < A[m]:
            j = m - 1
        else:
            i = m + 1
    return -1
```

- 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

# Efficacité

```
def rechercher(x, A):
    n = len(A)
    i = 0
    j = n - 1
    while i <= j:
        m = (i + j) // 2
        if x == A[m]:
            return m
        elif x < A[m]:
            j = m - 1
        else:
            i = m + 1
    return -1
```

- 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

**Donc la recherche prend $O(n)$ pour un tableau quelconque, $O(\log_2 n)$ pour un tableau trié**