Science informatique

CM2

Antonio E. Porreca
aeporreca.org/scienceinfo

Algorithmes !

C’est quoi un algorithme ?

- La description non ambiguë d’une séquence finie d’instructions permettant de résoudre un problème

- Finitude = termine après un nombre fini d’étapes

- Non ambigu = précis (en termes d’opérations élémentaires)

- Entrées = données

- Sorties = résultat attendu

محمد بن موسی خوارزمی

Muḥammad ibn Mūsā al-Khwārizmī

Résoudre un problème

- On cherche un algorithme
- On le décrit précisément, de manière non ambigüe
- On prouve qu'il est correct
- On vérifie qu'il est efficace (idéalement, on choisit l'algorithme optimal)
- On le met en œuvre (pas dans cette UE)
- On le teste (pas dans cette UE)

Décrire des algorithmes

- En langage naturel (par exemple, en français avec un accent italien)

- En pseudocode (semi-formel)

- En langage de programmation (formel)

  - Par exemple, en Python dans l'UE Projet informatique

Recherche dans une séquence en langage naturel

- Pour chaque élément de la séquence à partir du premier :
    - Si cet élément est l'élément cherché, on a terminé
    - Sinon, on continue avec l'élément suivant
- S'il n'y a plus d'éléments et on n'a pas trouvé ce qu'on cherchait, alors il n'est pas là

Recherche dans une séquence en pseudocode

```
fonction chercher(élément, séquence)
    n ≔ longueur(séquence)
    i ≔ 0
    tant que i < n faire
        si séquence[i] = élément alors
            retourner i
        fin si
        i ≔ i + 1
    fin tant que
    retourner –1
fin fonction
```

Recherche dans une séquence en Python 🐍

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

Structures de contrôle

instruction$_1$
instruction$_2$
...
instruction$_n$

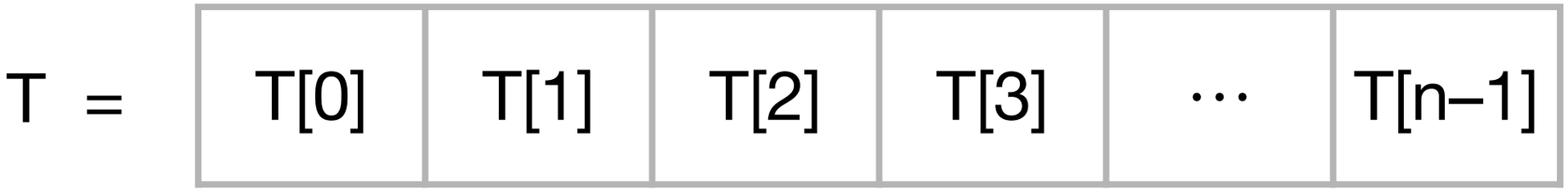
si *condition* **alors**
    instructions
sinon
    d'autres instructions
fin si

tant que *condition* **faire**
    instructions
fin tant que

Structures de données

- Variables entières
- Tableaux
- Matrices
- Arbres

Structures de données : les tableaux



Structures de données : les tableaux

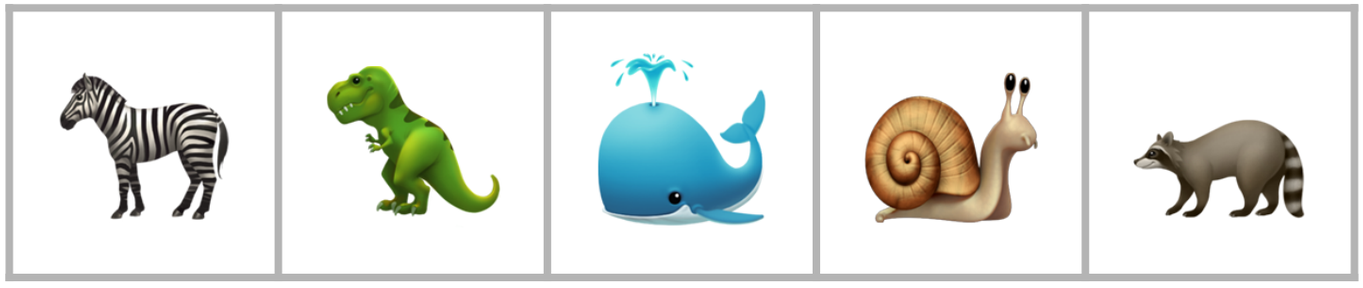
T =

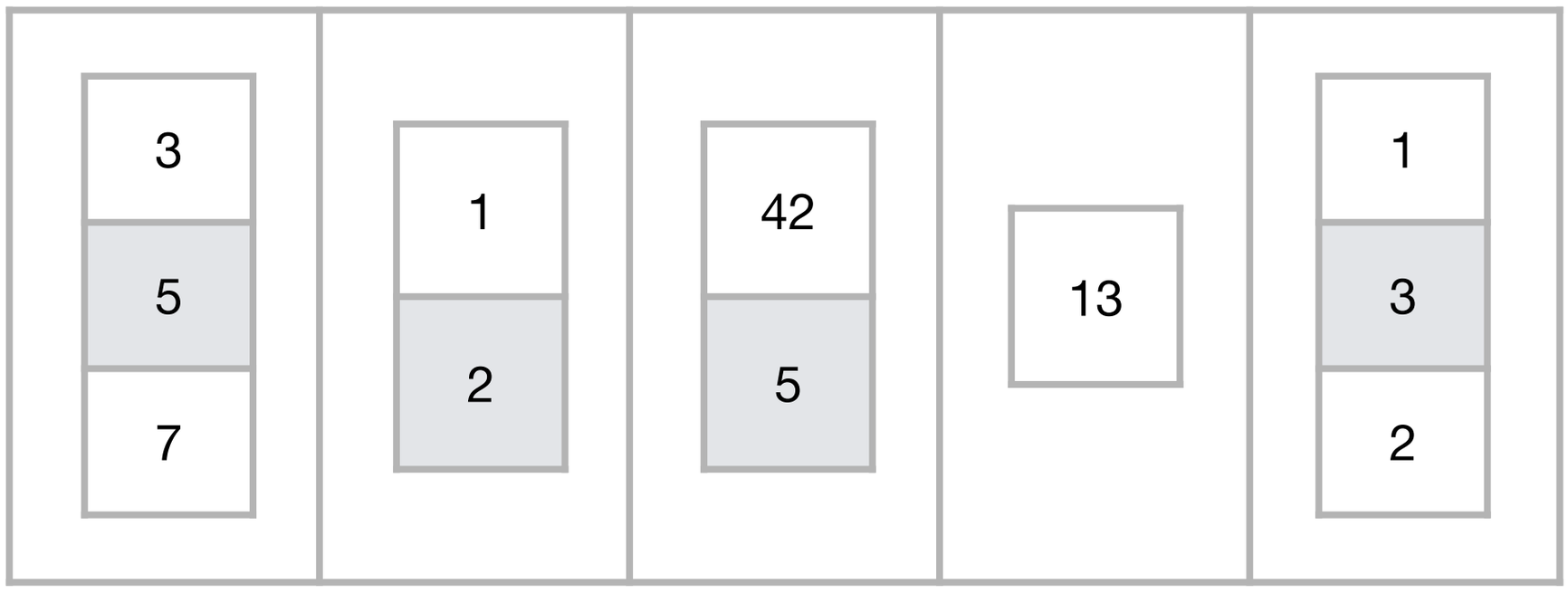
T[0]	T[1]	T[2]	T[3]	⋯	T[n–1]

3	4	2	3	1

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

Tableaux





Comment faire une recherche dans un dictionnaire ? 📖

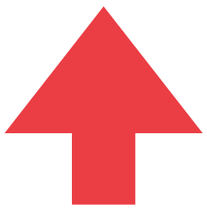
Recherche dans un tableau

```
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

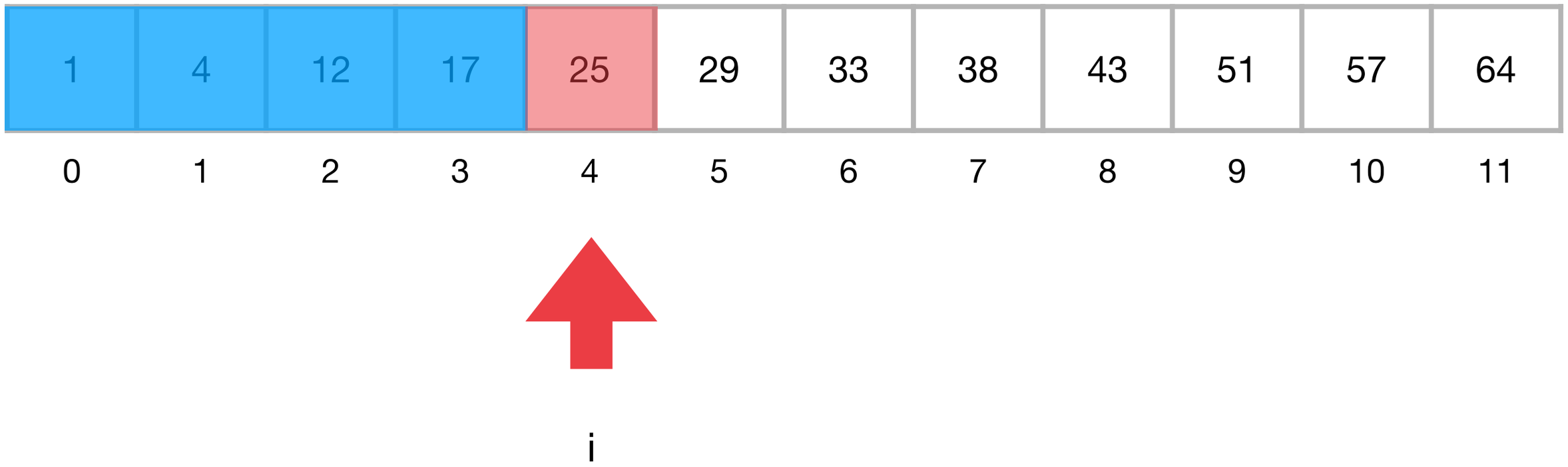
Recherche de 33

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

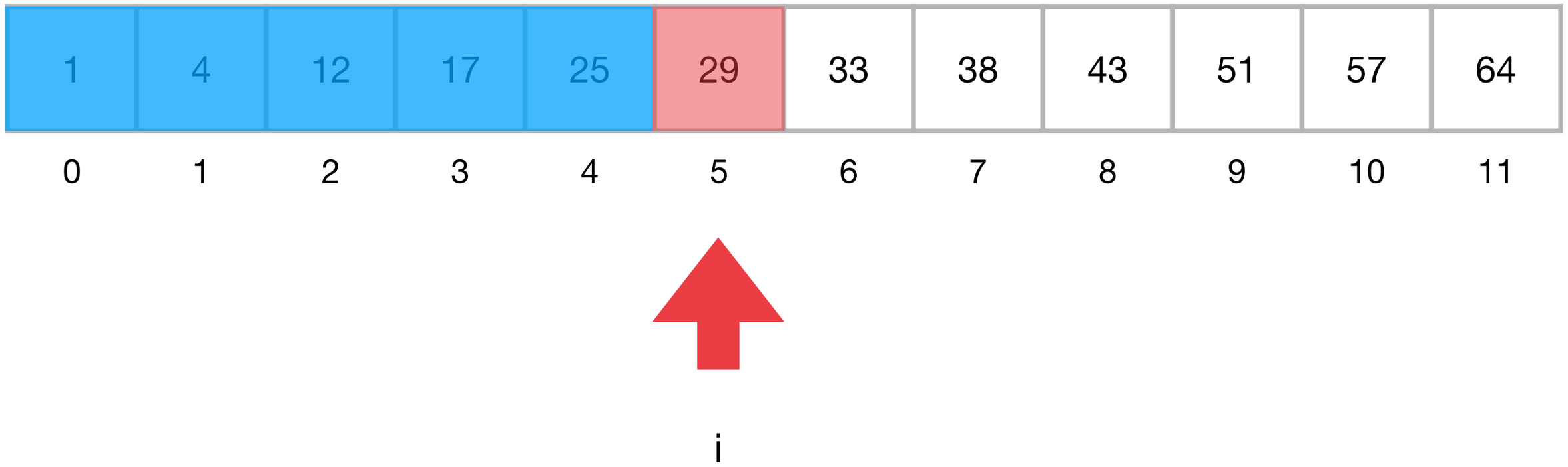
Recherche linéaire

Recherche de 33



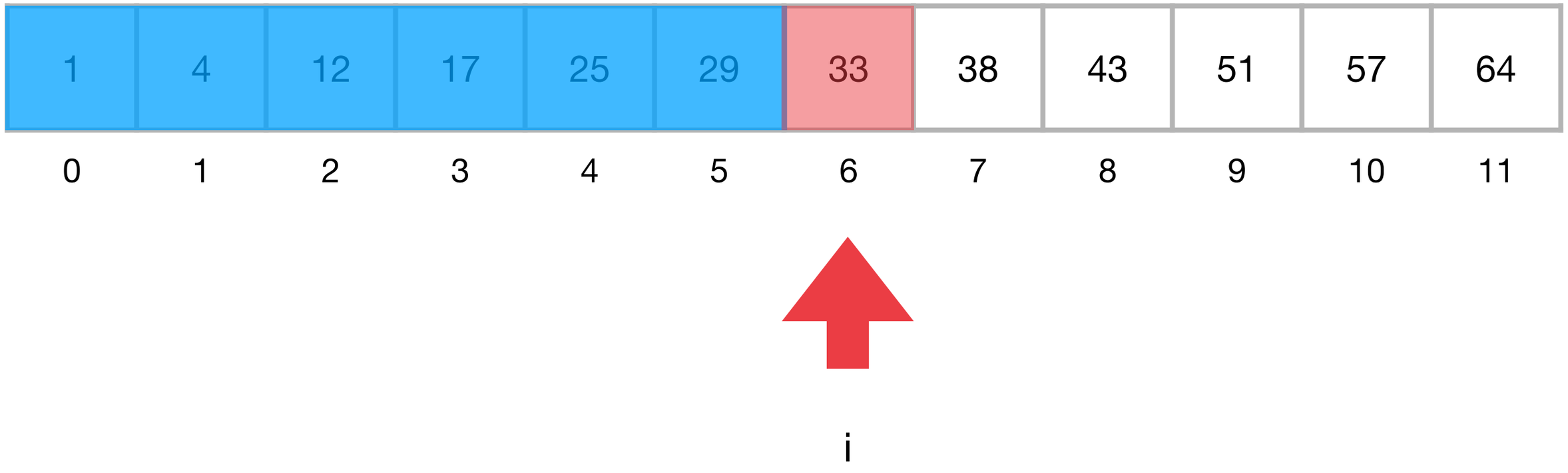
Recherche linéaire

Recherche de 33



Recherche linéaire

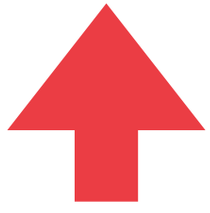
Recherche de 33



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

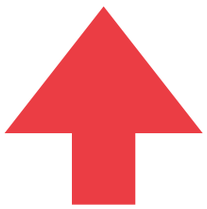


i

Recherche linéaire

Recherche de 3

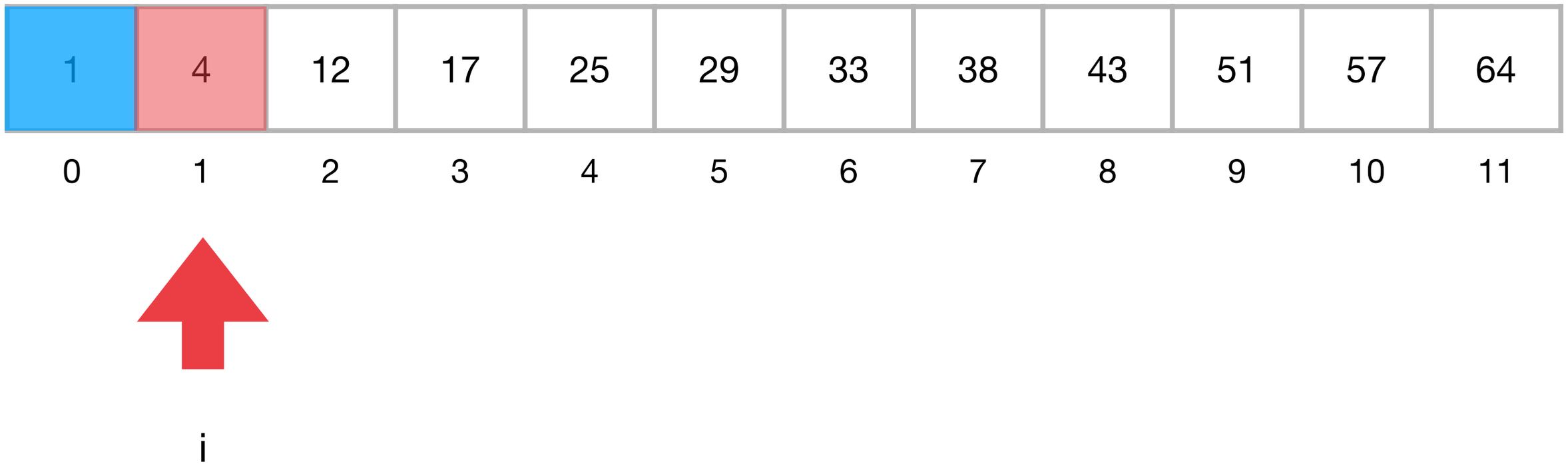
1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11



i

Recherche linéaire

Recherche de 3



Recherche linéaire

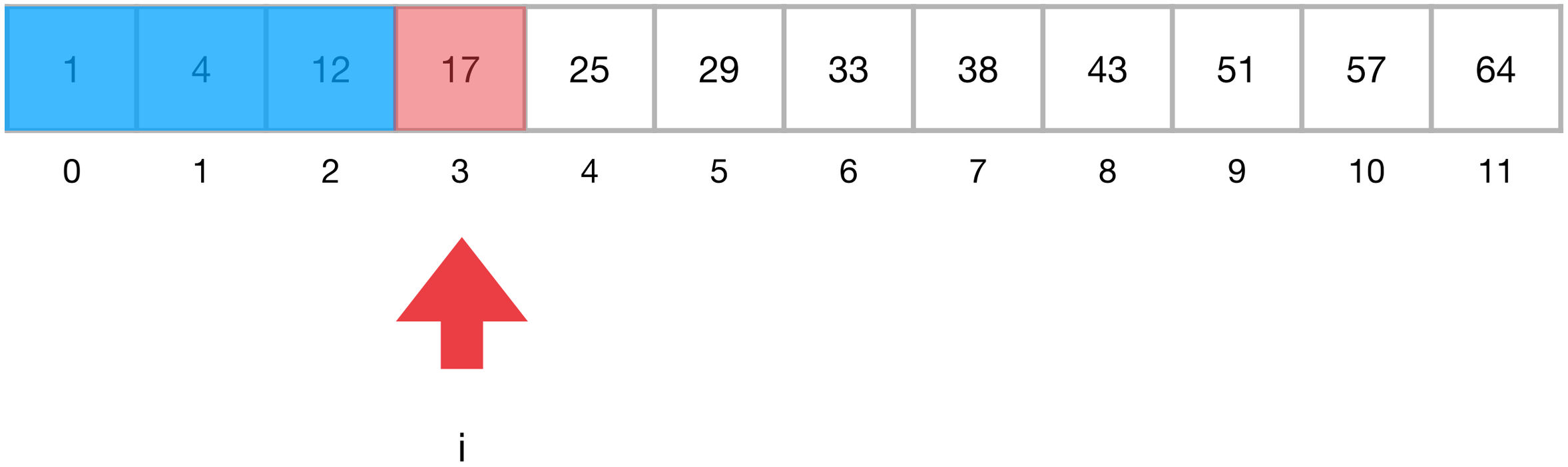
Recherche de 3

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

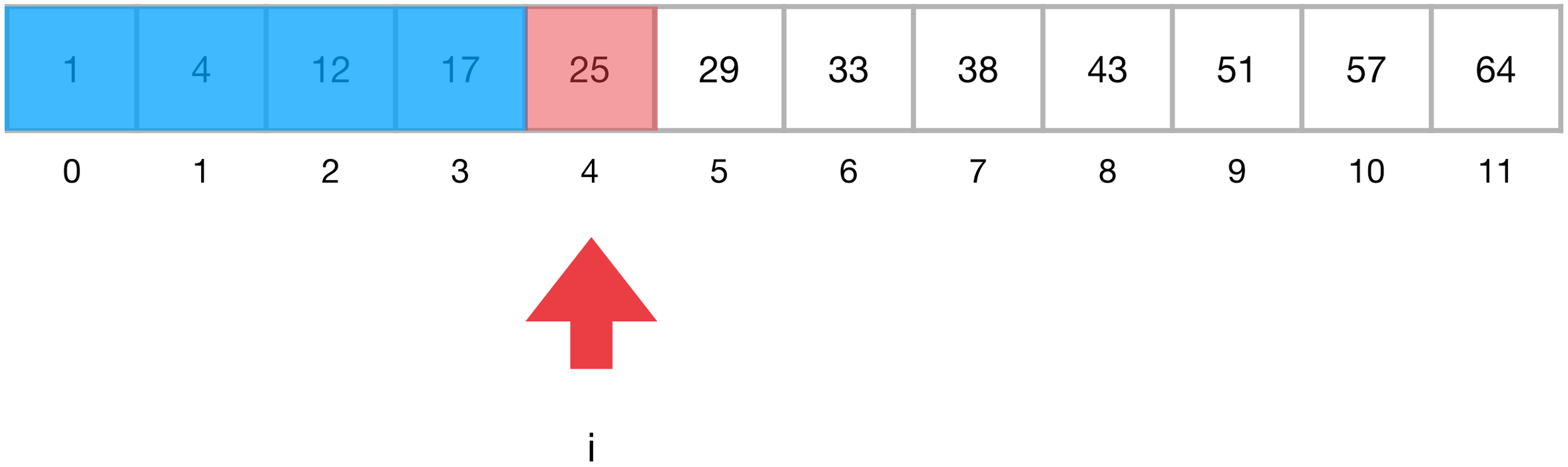
Recherche linéaire

Recherche de 3



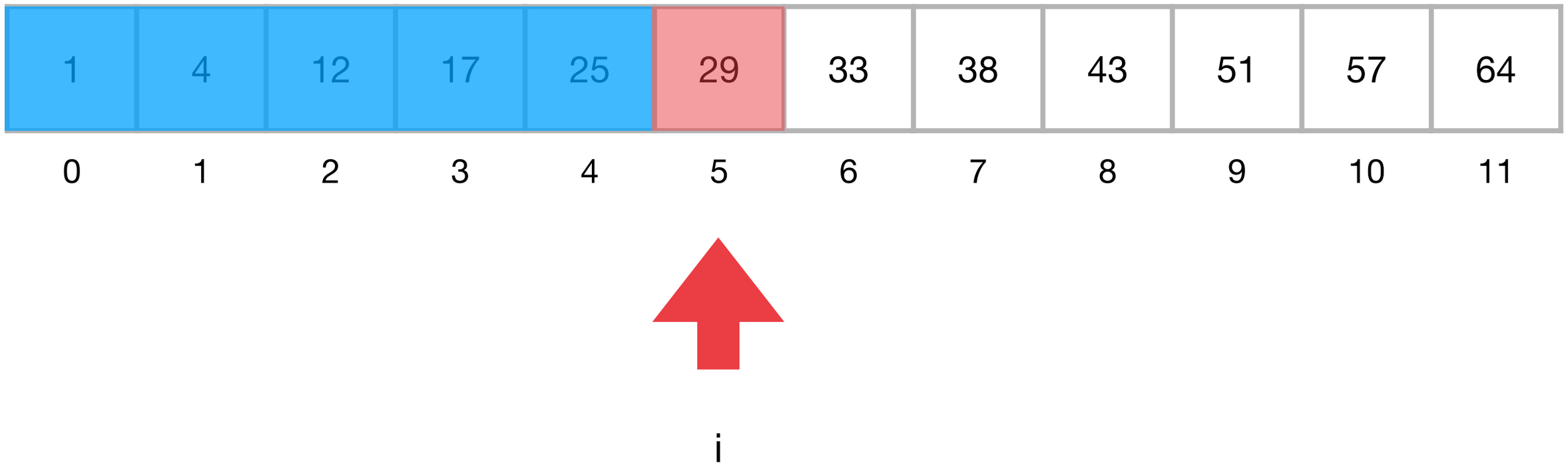
Recherche linéaire

Recherche de 3



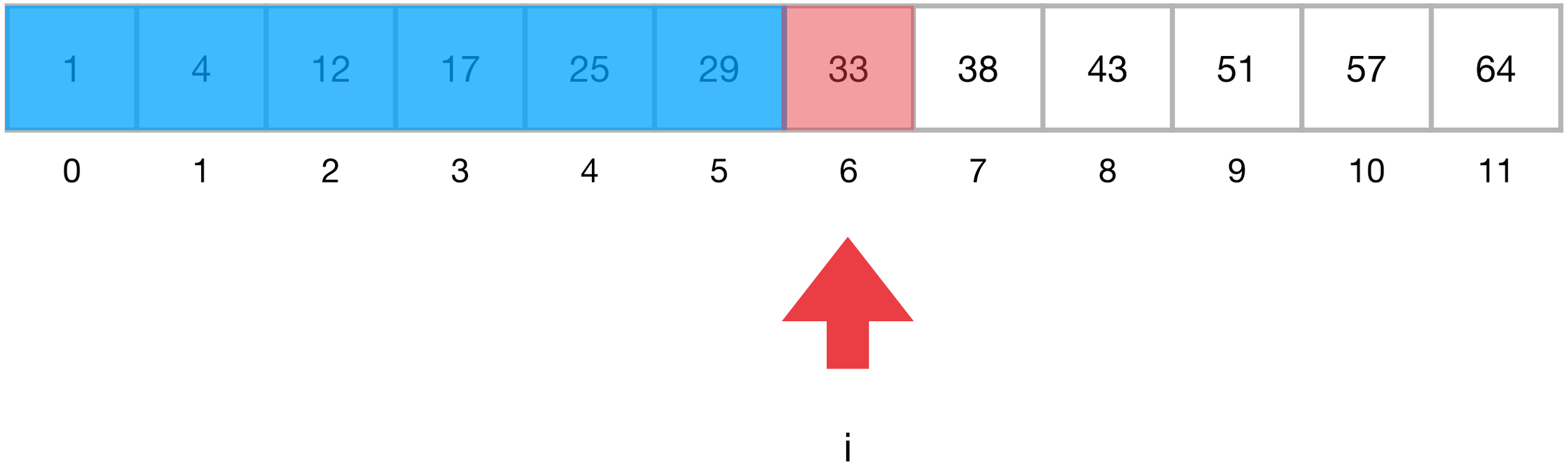
Recherche linéaire

Recherche de 3



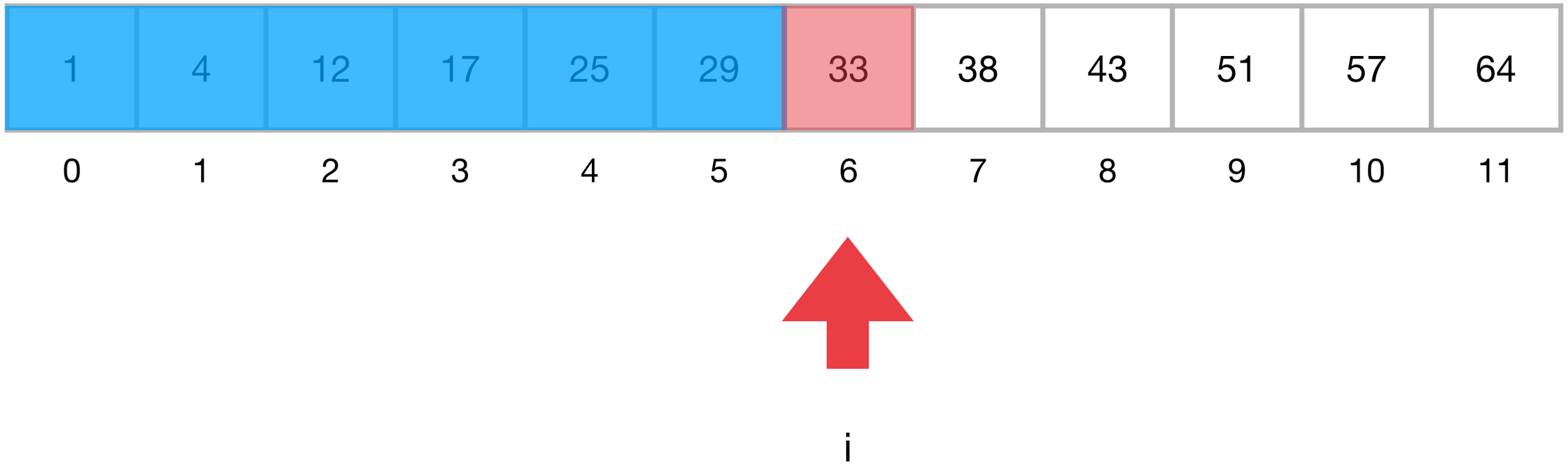
Recherche linéaire

Recherche de 3



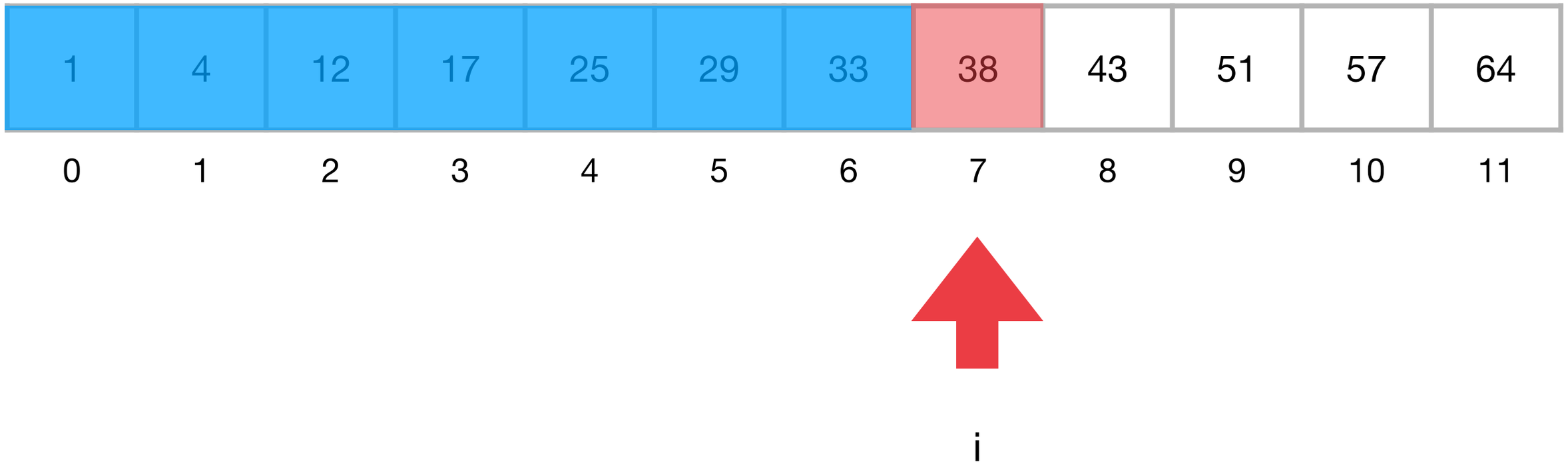
Recherche linéaire

Recherche de 3



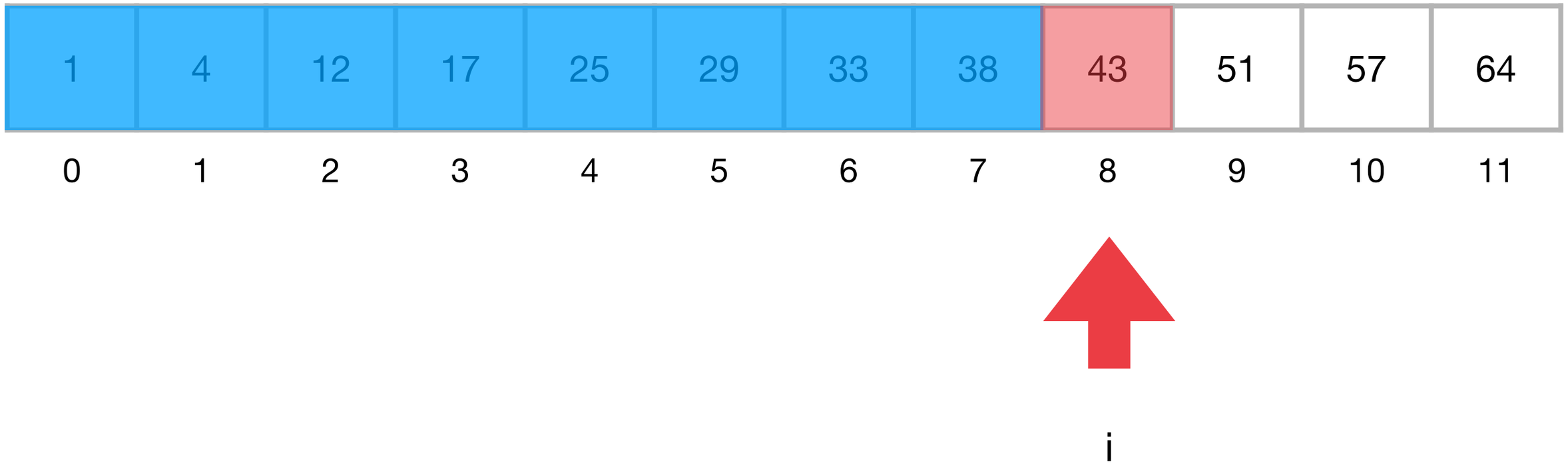
Recherche linéaire

Recherche de 3



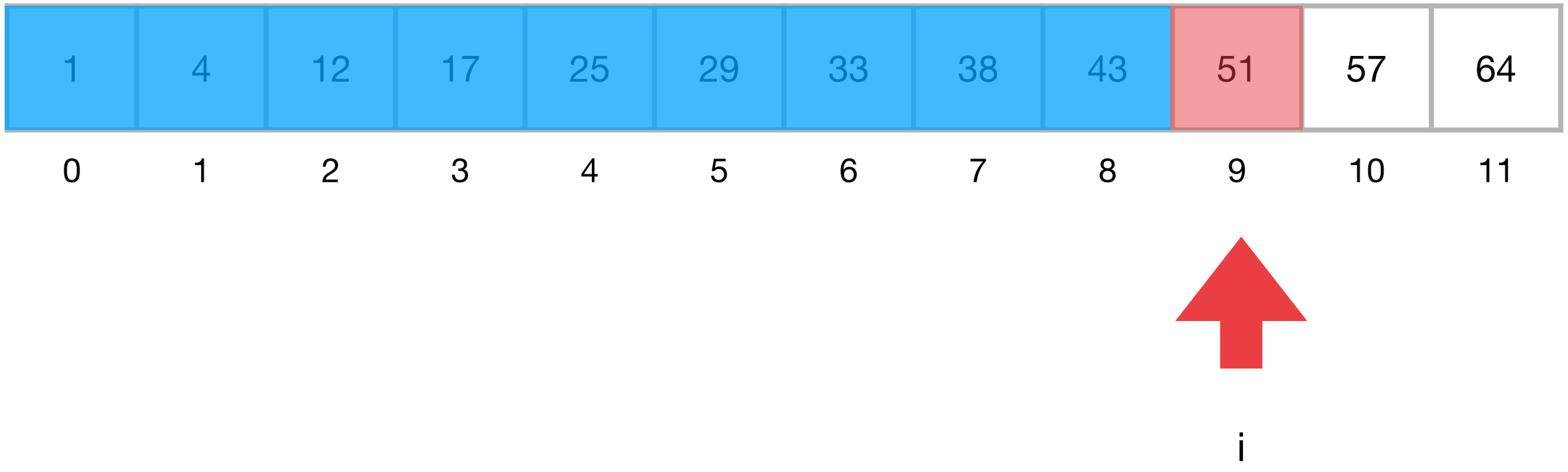
Recherche linéaire

Recherche de 3



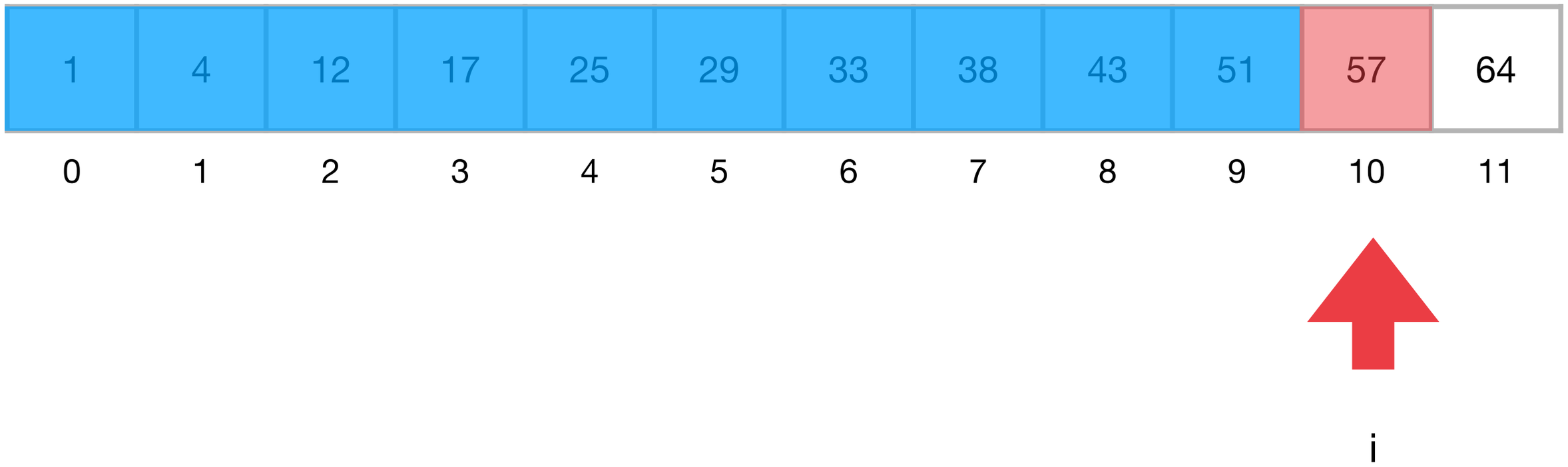
Recherche linéaire

Recherche de 3



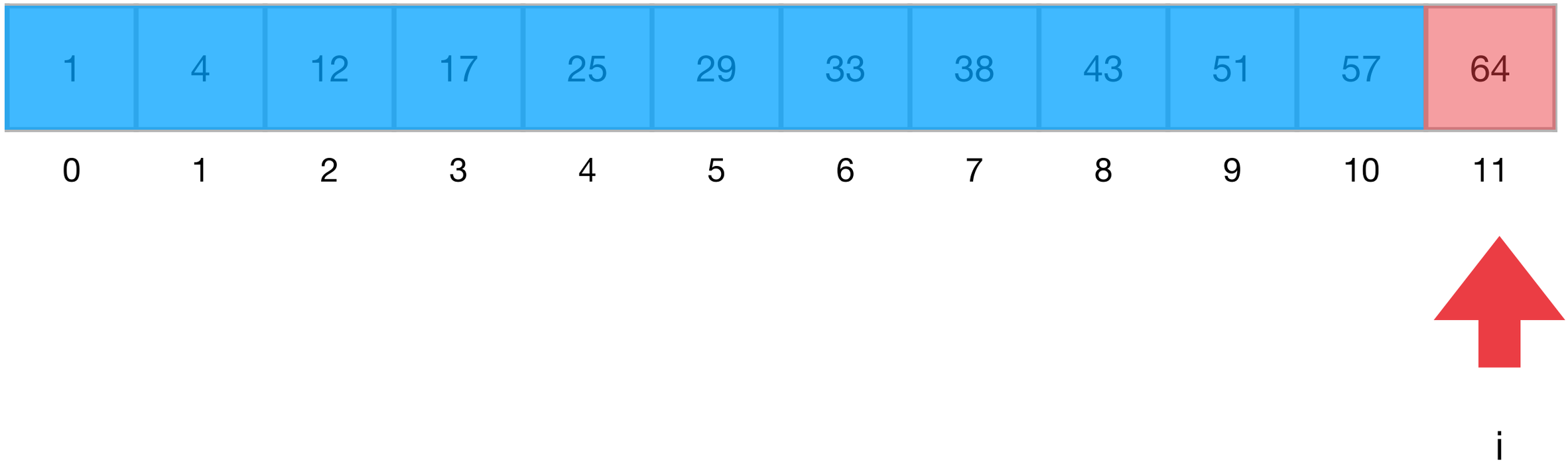
Recherche linéaire

Recherche de 3



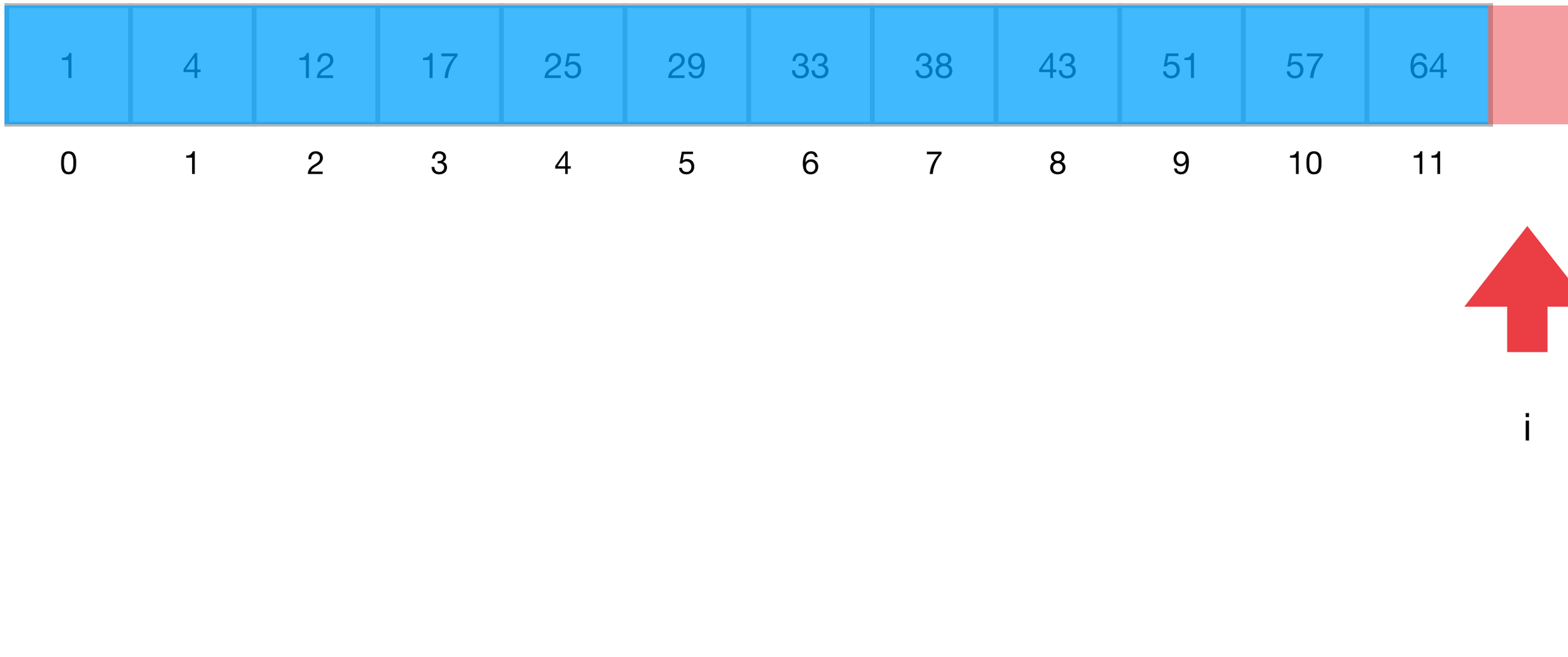
Recherche linéaire

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

```
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, …, 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)                          1
    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)                          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)	1
pour i ≔ 0 **à** n – 1 **faire**	1
si T[i] = x **alors**	1
👉 **retourner** i	1
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                      1
        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                      1
        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	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
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
```

si on a T[k] = x

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   }  k + 1 fois
            retourner i                                   1
        fin si
    fin pour
    retourner –1                                          1
fin fonction
```

si on a T[k] = x

Comptage des opérations

```
                                                          #opérations
fonction chercher(x, T)
    n ≔ longueur(T)                                       1
    pour i ≔ 0 à n – 1 faire                              1   } k + 1 fois
        si T[i] = x alors                                 1   }
            retourner i                                   1
        fin si
    fin pour
    retourner –1                                          1
fin fonction
```

si on a T[k] = x

2(k + 1) + 2 opérations

Comptage des opérations

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

si x n’est pas là

Comptage des opérations

#opérations

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

si x n’est pas là

2n + 3 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
```

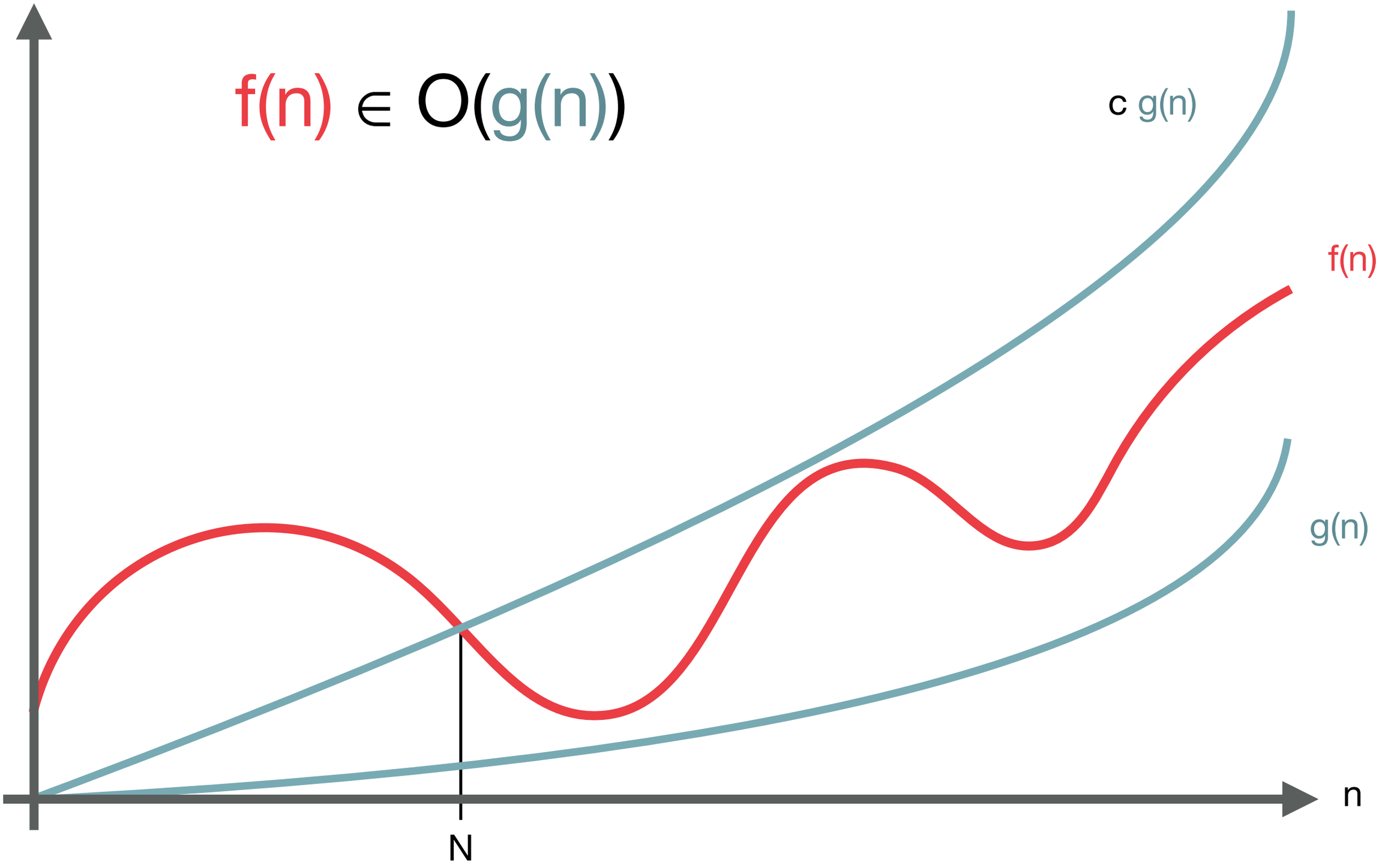
- Si on a de la chance, on a T[0] = x et on termine tout de suite en 4 opérations

- Si T[k] = x on fait 2(k + 1) + 2 = 2k + 4 opérations

- Si x n'est pas là on fait 2n + 3 opérations

Notation « grand O »

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

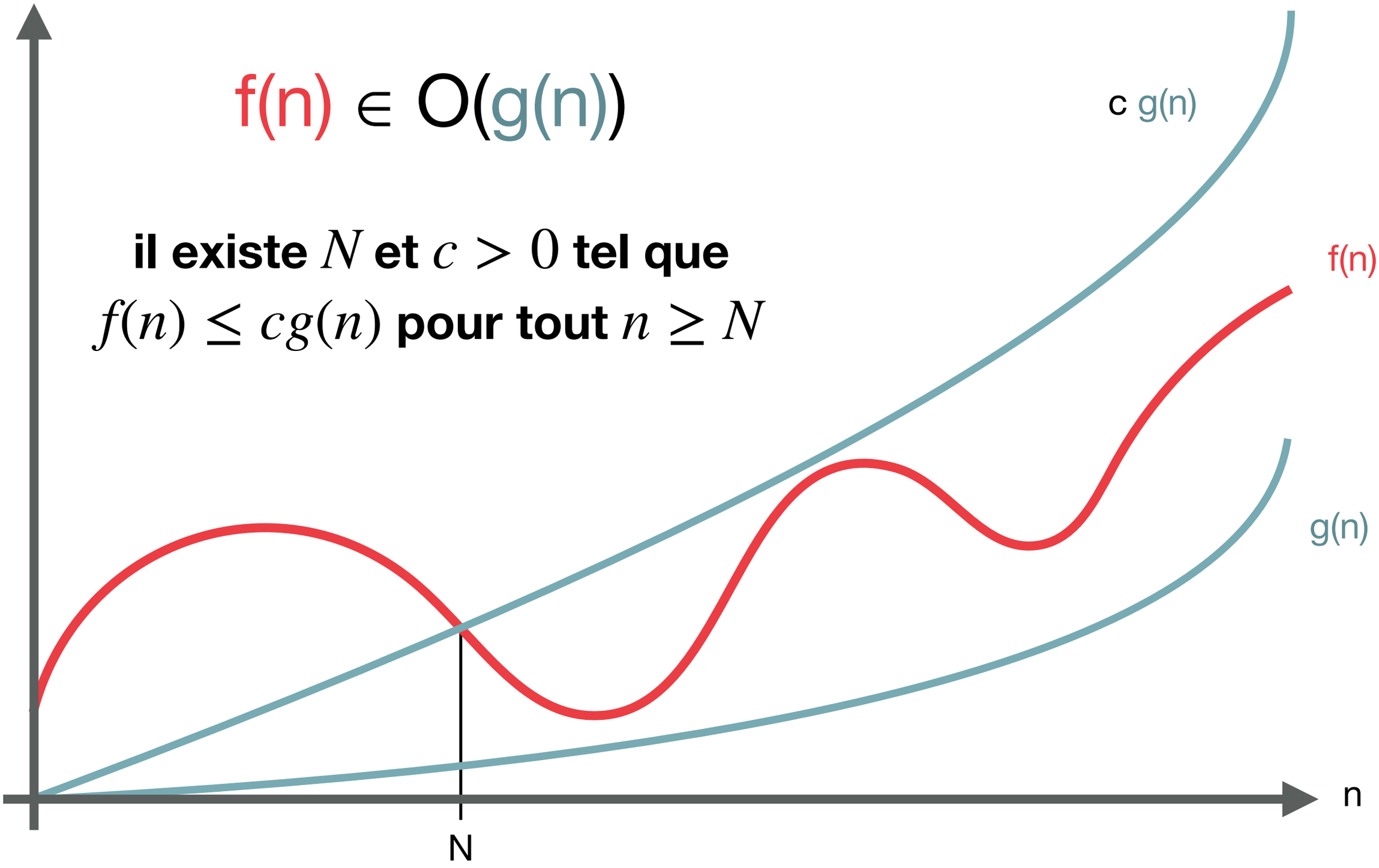


Notation « grand O »

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

il existe N et $c > 0$ tel que

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



Ordres de grandeur

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

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

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

Exercice 1 du TD1

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
```

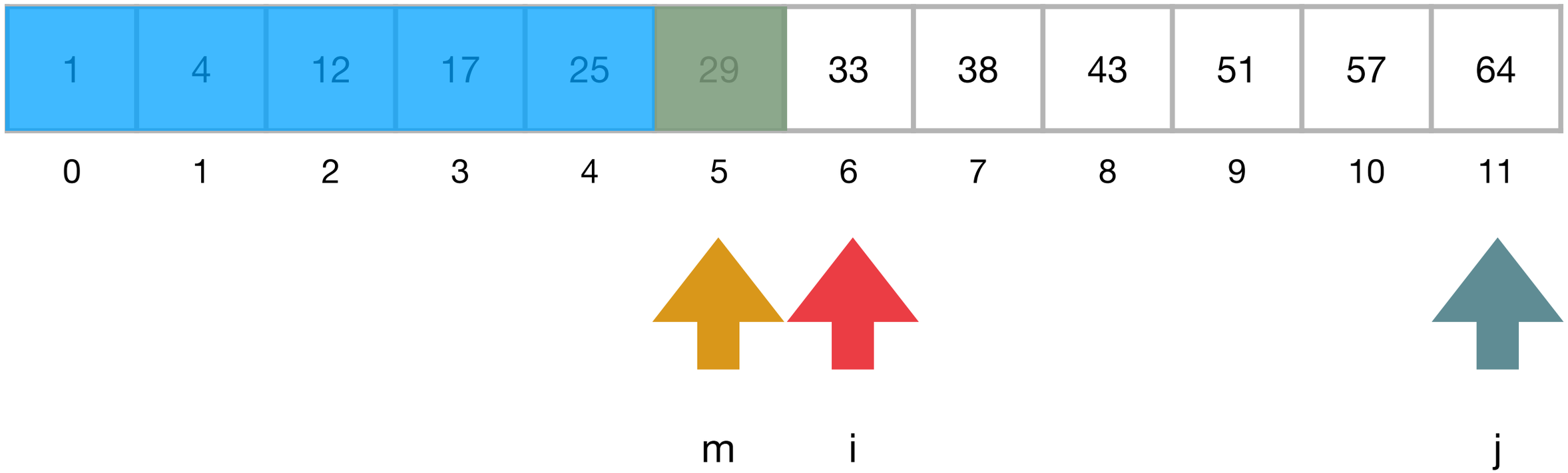
Recherche dichotomique

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

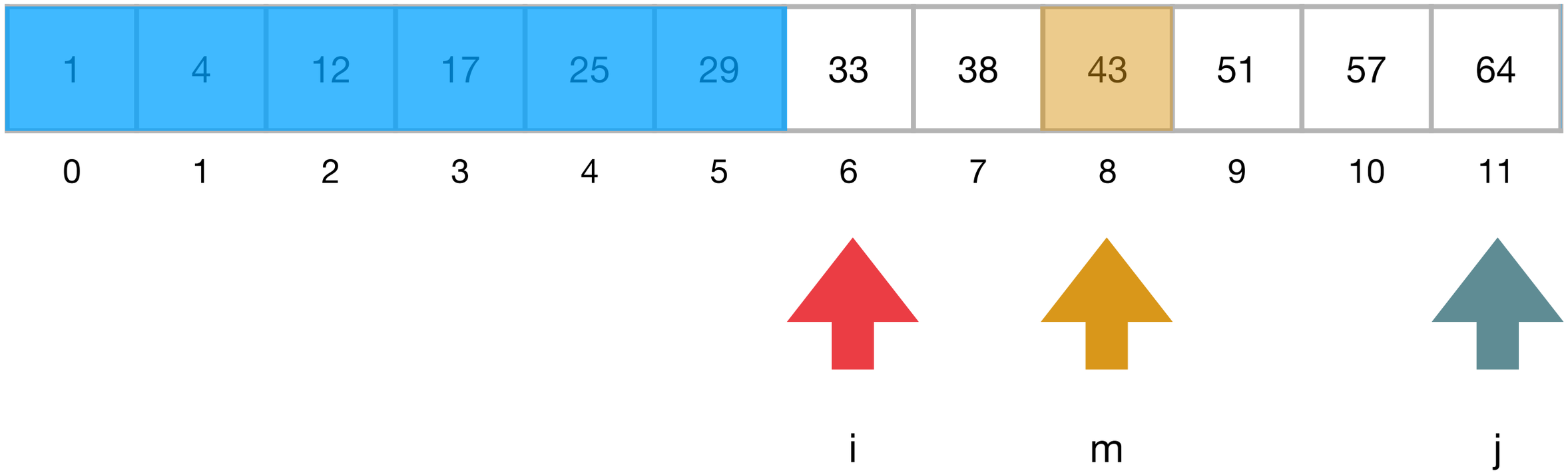
Recherche dichotomique

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

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

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

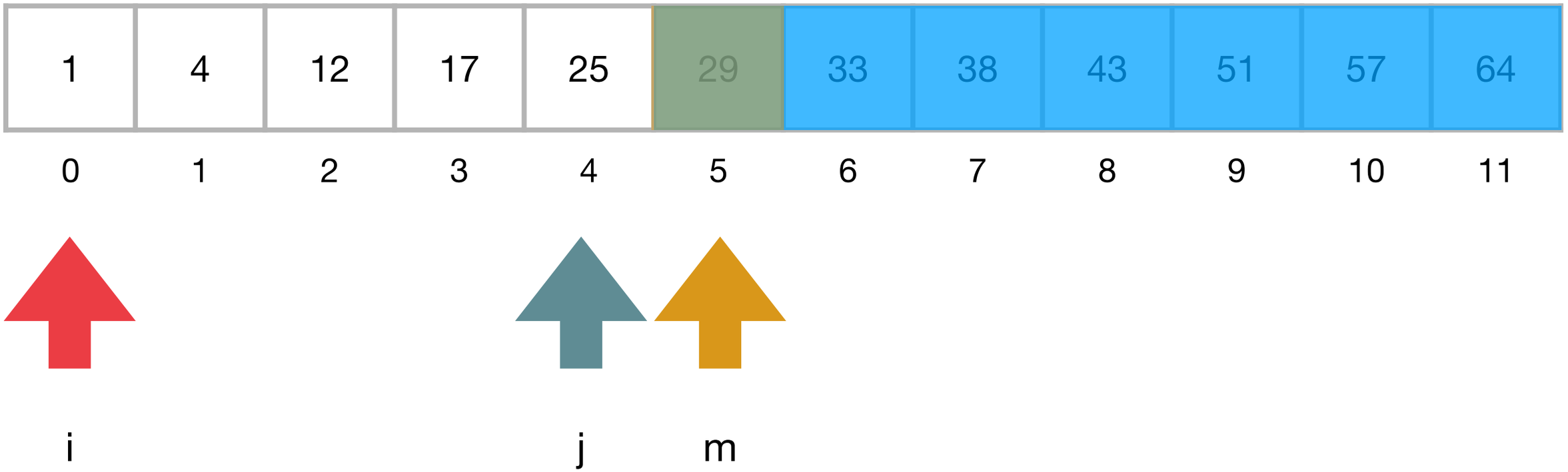
Recherche dichotomique

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

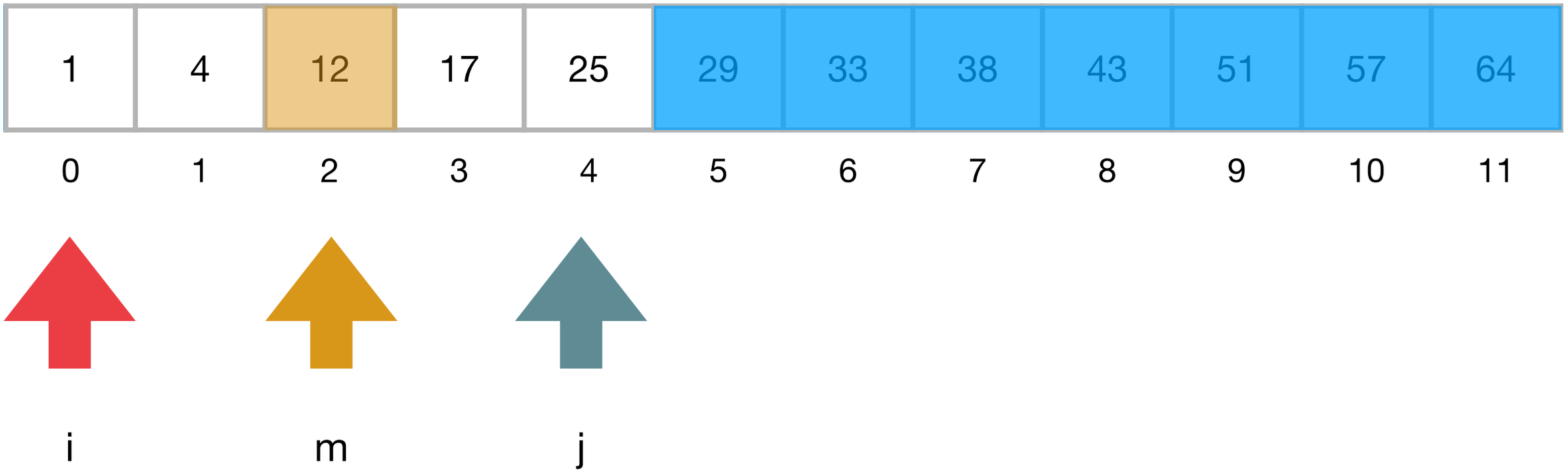
Recherche dichotomique

Recherche de 16



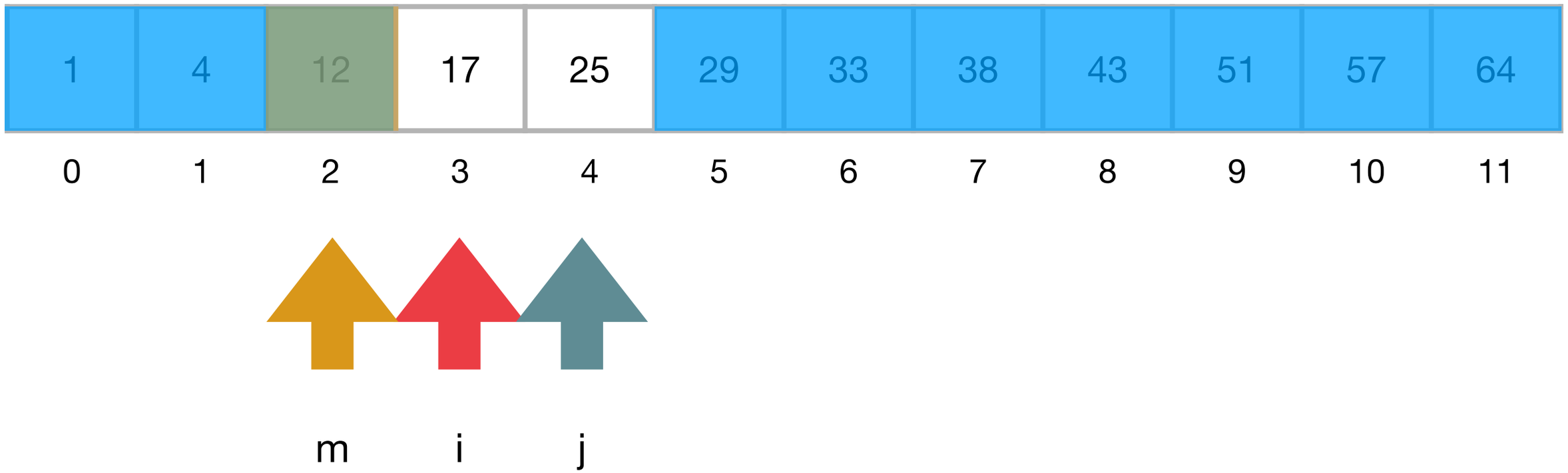
Recherche dichotomique

Recherche de 16



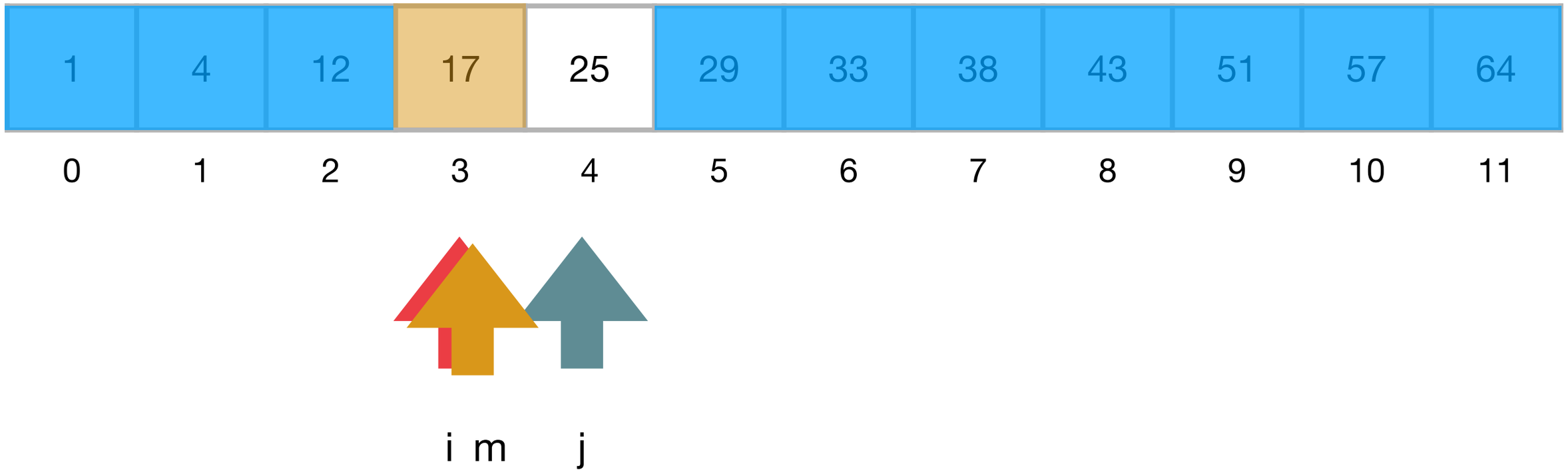
Recherche dichotomique

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

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

```
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, …, 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 > 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

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