Introduction à la science informatique

Semaine 4

Pour écrire des algorithmes,

il faut des structures de contrôle

et des structures de données

Pour écrire des algorithmes,

il faut des structures de contrôle

et des structures de données

Données

Pour écrire des algorithmes,

il faut des structures de contrôle

et des structures de données

Données

Données structurées

Pour écrire des algorithmes,

il faut des structures de contrôle

et des structures de données

Données

Données structurées

tableau :

D♦	7♦	D♠	7♠	J♠	7♥	R♣

Pour écrire des algorithmes,

il faut des structures de contrôle

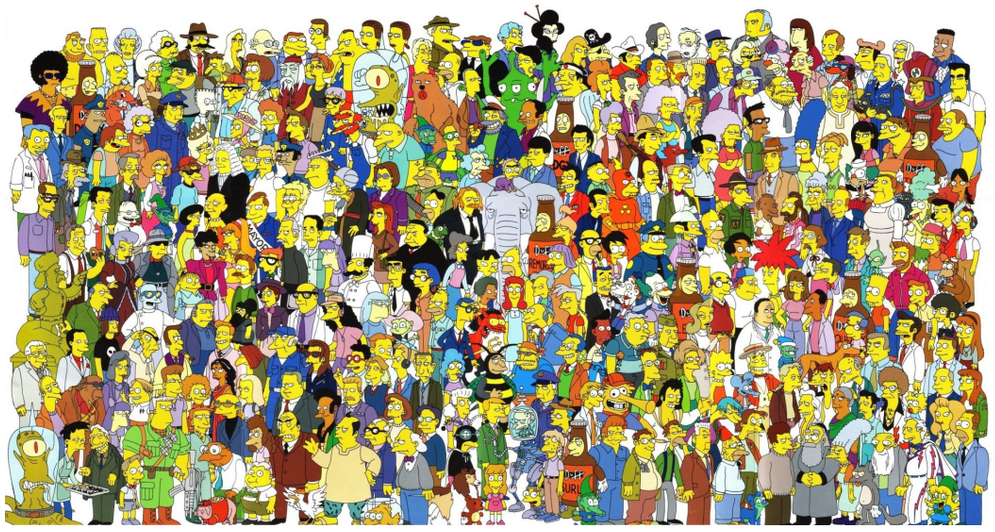
et des structures de données

Pour écrire des algorithmes,

il faut des structures de contrôle

et des structures de données

Données

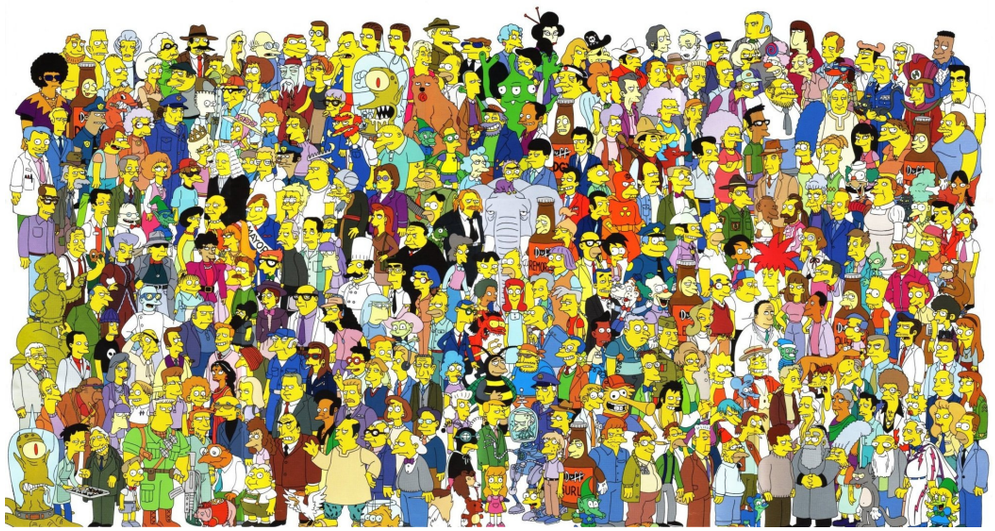


Pour écrire des algorithmes,

il faut des structures de contrôle

et des structures de données

Données



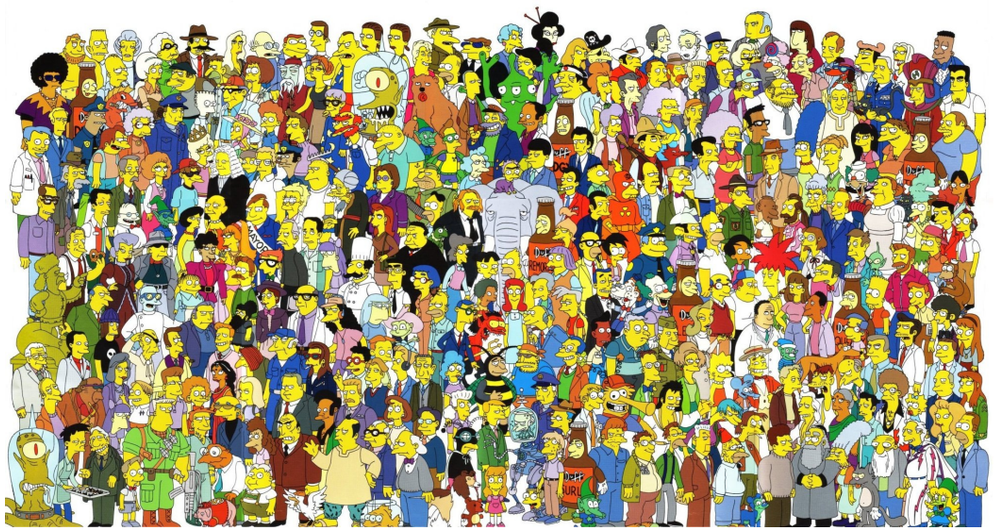
Données structurées

Pour écrire des algorithmes,

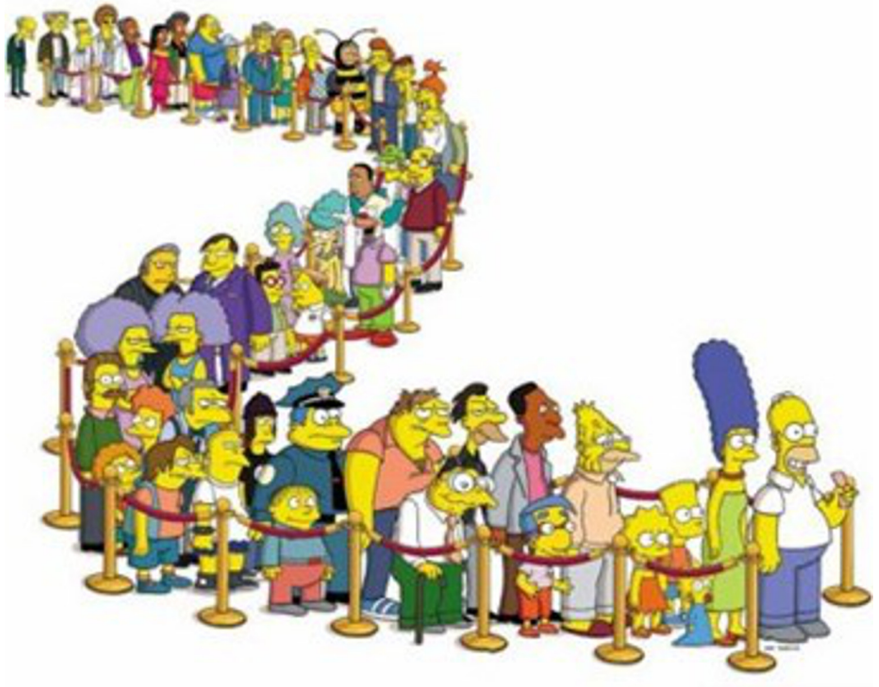
il faut des structures de contrôle

et des structures de données

Données



Données structurées



file (d’attente)

Pour écrire des algorithmes,

il faut des structures de contrôle

et des structures de données

Pour écrire des algorithmes,

il faut des structures de contrôle

et des structures de données

Données

Pour écrire des algorithmes,

il faut des structures de contrôle

et des structures de données

Données

Données structurées

Pour écrire des algorithmes,

il faut des structures de contrôle

et des structures de données

Données

Données structurées

pile (d’assiettes)

Pour écrire des algorithmes,

il faut des structures de contrôle

et des structures de données

Pour écrire des algorithmes,

il faut des structures de contrôle

et des structures de données

Données

Pour écrire des algorithmes,

il faut des structures de contrôle

et des structures de données

Données

Données structurées

Pour écrire des algorithmes,

il faut des structures de contrôle

et des structures de données

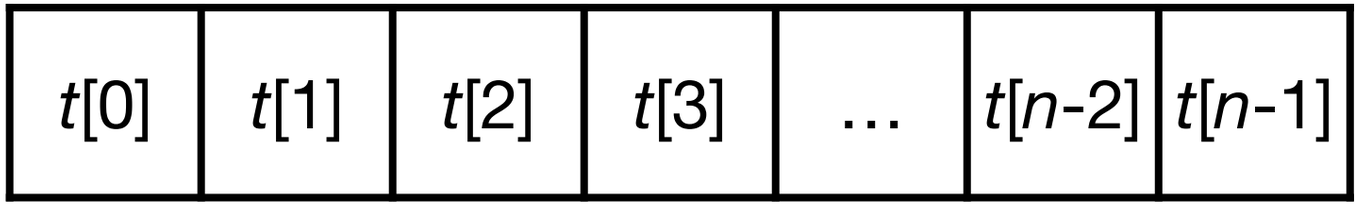
Données

Données structurées

matrice
tableau bidimensionnel

Tableaux

tableau *t* de **longueur** *n*



Tableaux

tableau *t* de **longueur** *n*

t[0]	*t*[1]	*t*[2]	*t*[3]	…	*t*[*n*-2]	*t*[*n*-1]

un tableau d’entiers de longueur 5

17	64	5	1	38

Tableaux

tableau *t* de **longueur** *n*

t[0]	*t*[1]	*t*[2]	*t*[3]	…	*t*[*n*-2]	*t*[*n*-1]

un tableau d’entiers de longueur 5

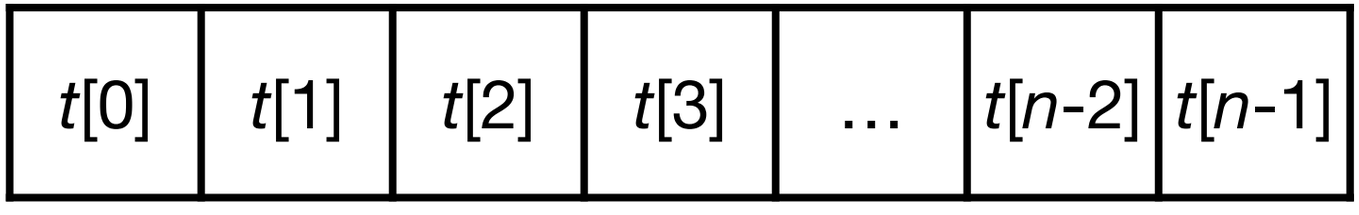
17	64	5	1	38

un tableau de booléens de longueur 4

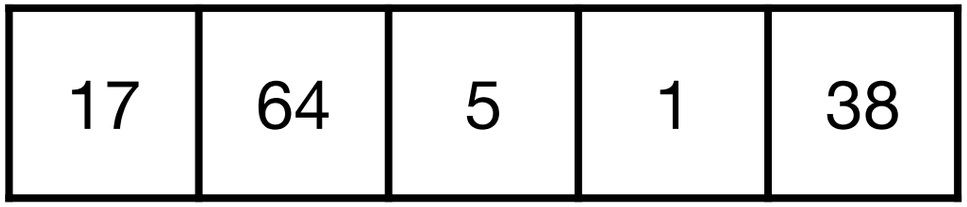
True	False	False	True

Tableaux

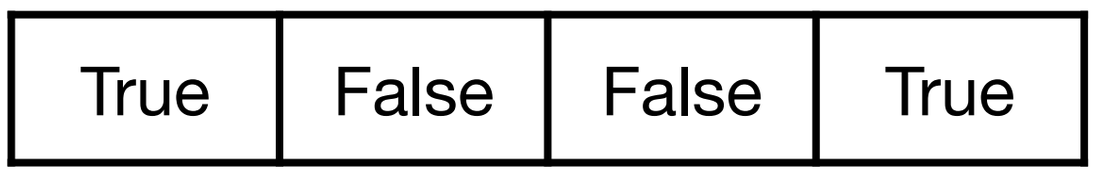
tableau *t* de **longueur** *n*



un tableau d'entiers de longueur 5



un tableau de booléens de longueur 4



Un tableau t de longueur n contenant des éléments de E est une application $t : \{0, \ldots, n-1\} \to E$.

Tableaux (listes) en Python

Tableaux (listes) en Python

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

Tableaux (listes) en Python

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

Tableaux (listes) en Python

```
>>> t = [5, 1, 4, 2, 3]
>>> t
[5, 1, 4, 2, 3]
>>> len(t)
5
```

Tableaux (listes) en Python

```
>>> t = [5, 1, 4, 2, 3]
>>> t
[5, 1, 4, 2, 3]
>>> len(t)
5
>>> t[0]
5
```

Tableaux (listes) en Python

```
>>> t = [5, 1, 4, 2, 3]
>>> t
[5, 1, 4, 2, 3]
>>> len(t)
5
>>> t[0]
5
>>> t[1] = 0
>>> t
```

Tableaux (listes) en Python

```
>>> t = [5, 1, 4, 2, 3]
>>> t
[5, 1, 4, 2, 3]
>>> len(t)
5
>>> t[0]
5
>>> t[1] = 0
>>> t
[5, 0, 4, 2, 3]
>>> t[5]
```

Tableaux (listes) en Python

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

Opération de base sur les tableaux : sommer les éléments

Données

Données structurées

tableau :

D♦	7♦	D♠	7♠	J♠	7♥	R♣

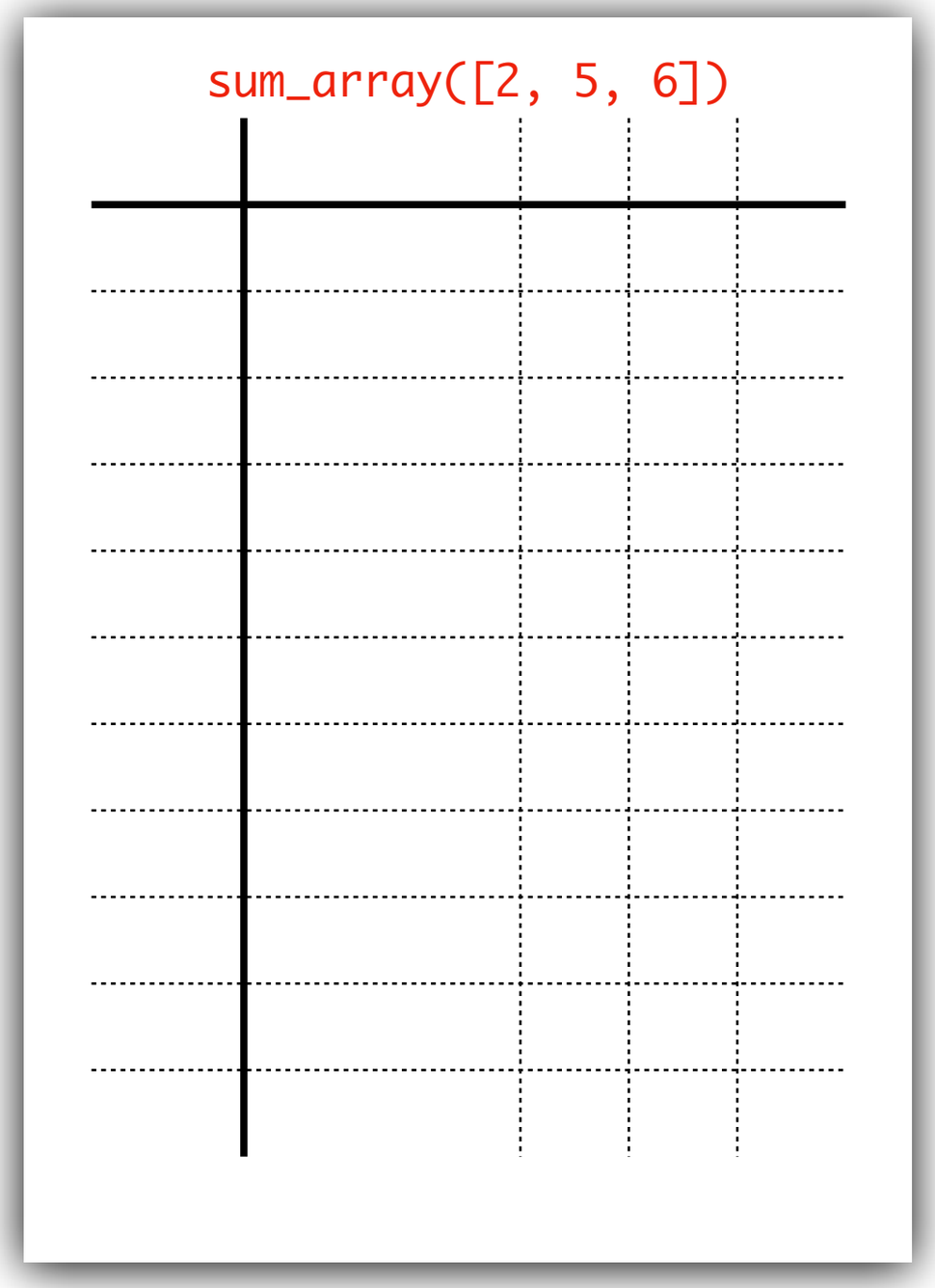
Combien de points contient ma main ?

Parcourir un tableau (pour sommer ses éléments)

```
1 def sum_array(t):
2     n = len(t)
3     s = 0
4     for i in range(n):
5         s = s + t[i]
6     return s
```

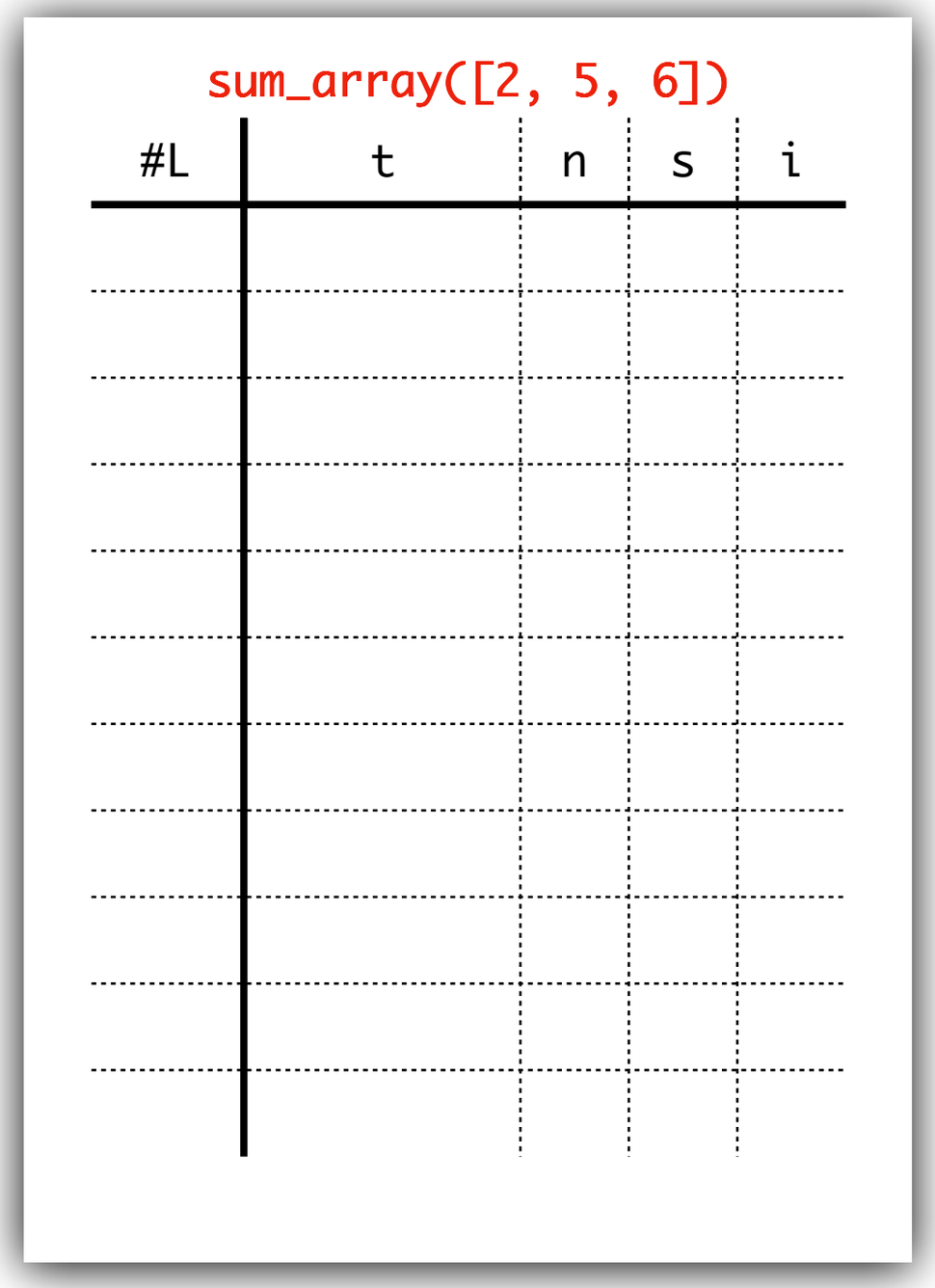
Parcourir un tableau (pour sommer ses éléments)

```
1  def sum_array(t):
2      n = len(t)
3      s = 0
4      for i in range(n):
5          s = s + t[i]
6      return s
```



Parcourir un tableau (pour sommer ses éléments)

```
1  def sum_array(t):
2      n = len(t)
3      s = 0
4      for i in range(n):
5          s = s + t[i]
6      return s
```



Parcourir un tableau (pour sommer ses éléments)

```
1  def sum_array(t):
2      n = len(t)
3      s = 0
4      for i in range(n):
5          s = s + t[i]
6      return s
```

sum_array([2, 5, 6])

#L	t	n	s	i
1	[2, 5, 6]			

Parcourir un tableau (pour sommer ses éléments)

```
1  def sum_array(t):
2      n = len(t)
3      s = 0
4      for i in range(n):
5          s = s + t[i]
6      return s
```

sum_array([2, 5, 6])

#L	t	n	s	i
1	[2, 5, 6]			
2		3		

Parcourir un tableau (pour sommer ses éléments)

```
1  def sum_array(t):
2      n = len(t)
3      s = 0
4      for i in range(n):
5          s = s + t[i]
6      return s
```

sum_array([2, 5, 6])

#L	t	n	s	i
1	[2, 5, 6]			
2		3		
3			0	

Parcourir un tableau (pour sommer ses éléments)

```
1  def sum_array(t):
2      n = len(t)
3      s = 0
4      for i in range(n):
5          s = s + t[i]
6      return s
```

sum_array([2, 5, 6])

#L	t	n	s	i
1	[2, 5, 6]			
2		3		
3			0	
4				0

Parcourir un tableau (pour sommer ses éléments)

```
1  def sum_array(t):
2      n = len(t)
3      s = 0
4      for i in range(n):
5          s = s + t[i]
6      return s
```

👉

sum_array([2, 5, 6])

#L	t	n	s	i
1	[2, 5, 6]			
2		3		
3			0	
4				0
5			2	

Parcourir un tableau (pour sommer ses éléments)

```
1  def sum_array(t):
2      n = len(t)
3      s = 0
4      for i in range(n):
5          s = s + t[i]
6      return s
```

sum_array([2, 5, 6])

#L	t	n	s	i
1	[2, 5, 6]			
2		3		
3			0	
4				0
5			2	
4				1

Parcourir un tableau (pour sommer ses éléments)

```
1  def sum_array(t):
2      n = len(t)
3      s = 0
4      for i in range(n):
5          s = s + t[i]
6      return s
```

👉

sum_array([2, 5, 6])

#L	t	n	s	i
1	[2, 5, 6]			
2		3		
3			0	
4				0
5			2	
4				1
5			7	

Parcourir un tableau (pour sommer ses éléments)

```
1  def sum_array(t):
2      n = len(t)
3      s = 0
4      for i in range(n):
5          s = s + t[i]
6      return s
```

sum_array([2, 5, 6])

#L	t	n	s	i
1	[2, 5, 6]			
2		3		
3			0	
4				0
5			2	
4				1
5			7	
4				2

Parcourir un tableau (pour sommer ses éléments)

```
1  def sum_array(t):
2      n = len(t)
3      s = 0
4      for i in range(n):
5          s = s + t[i]
6      return s
```

👉

sum_array([2, 5, 6])

#L	t	n	s	i
1	[2, 5, 6]			
2		3		
3			0	
4				0
5			2	
4				1
5			7	
4				2
5			13	

Parcourir un tableau (pour sommer ses éléments)

```
1  def sum_array(t):
2      n = len(t)
3      s = 0
4      for i in range(n):
5          s = s + t[i]
6      return s
```

sum_array([2, 5, 6])

#L	t	n	s	i
1	[2, 5, 6]			
2		3		
3			0	
4				0
5			2	
4				1
5			7	
4				2
5			13	
4				

Parcourir un tableau (pour sommer ses éléments)

```
1  def sum_array(t):
2      n = len(t)
3      s = 0
4      for i in range(n):
5          s = s + t[i]
👉 6    return s
```

sum_array([2, 5, 6])

#L	t	n	s	i
1	[2, 5, 6]			
2		3		
3			0	
4				0
5			2	
4				1
5			7	
4				2
5			13	
4				
6				

résultat : 13

Avec la boucle « while »

```
def sum_array(t):
    n = len(t)
    s = 0
    for i in range(n):
        s = s + t[i]
    return s
```

```
def sum_array(t):
    n = len(t)
    s = 0
    i = 0
    while i < n:
        s = s + t[i]
        i = i + 1
    return s
```

Exercice 1

Opération de base sur les tableaux : rechercher un élément

Données

Données structurées

tableau :

D♦	7♦	D♠	7♠	J♠	7♥	R♣

Ai-je le 7♥ dans ma main ?

Recherche séquentielle

Recherche séquentielle

```
def linear_search(t, x):
```

Recherche séquentielle

```
def linear_search(t, x):
    n = len(t)
```

Recherche séquentielle

```
def linear_search(t, x):
    n = len(t)
    for i in range(n):
```

Recherche séquentielle

```
def linear_search(t, x):
    n = len(t)
    for i in range(n):
        if t[i] == x:
```

Recherche séquentielle

```
def linear_search(t, x):
    n = len(t)
    for i in range(n):
        if t[i] == x:
            return i
```

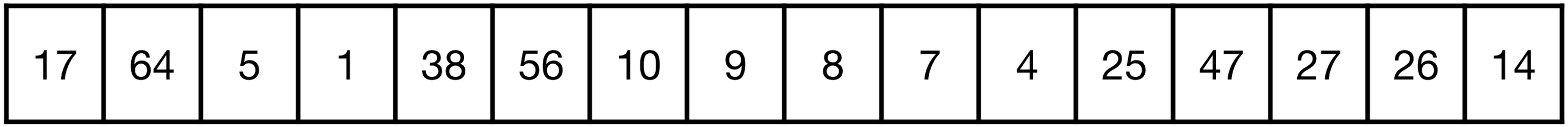
Recherche séquentielle

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

Recherche séquentielle

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

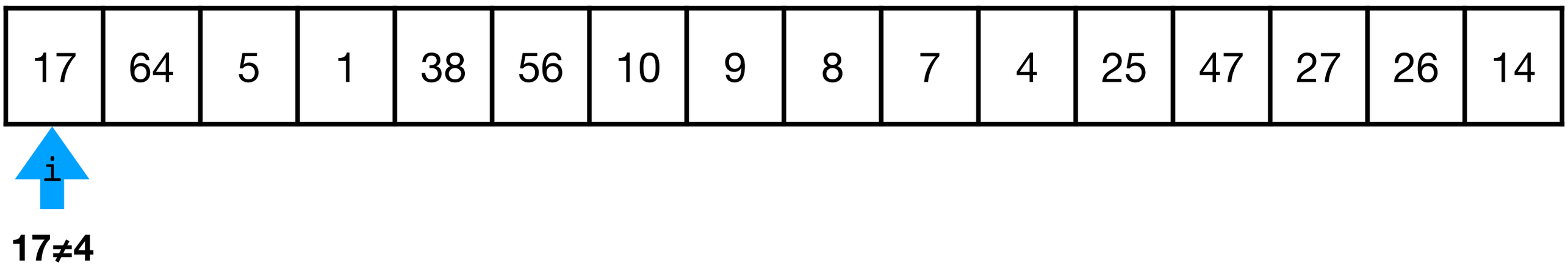
`linear_search(t, 4)` ?



Recherche séquentielle

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

linear_search(t, 4) ?

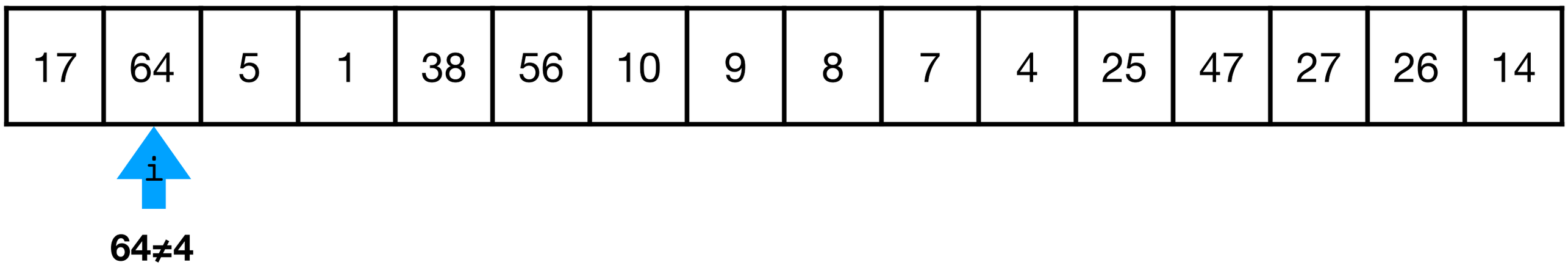


17≠4

Recherche séquentielle

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

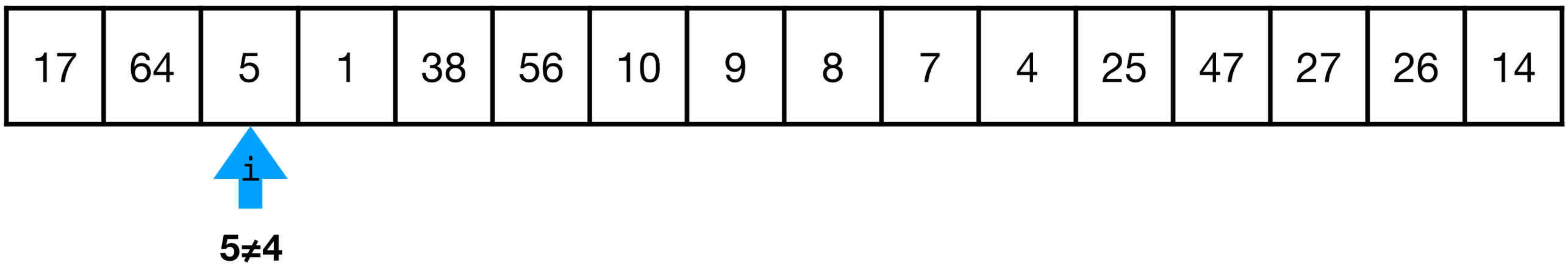
linear_search(t, 4) ?



Recherche séquentielle

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

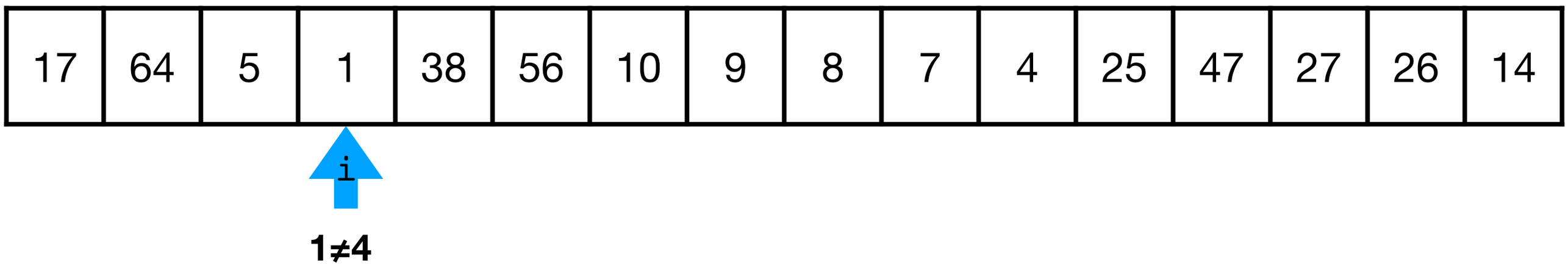
linear_search(t, 4) ?



Recherche séquentielle

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

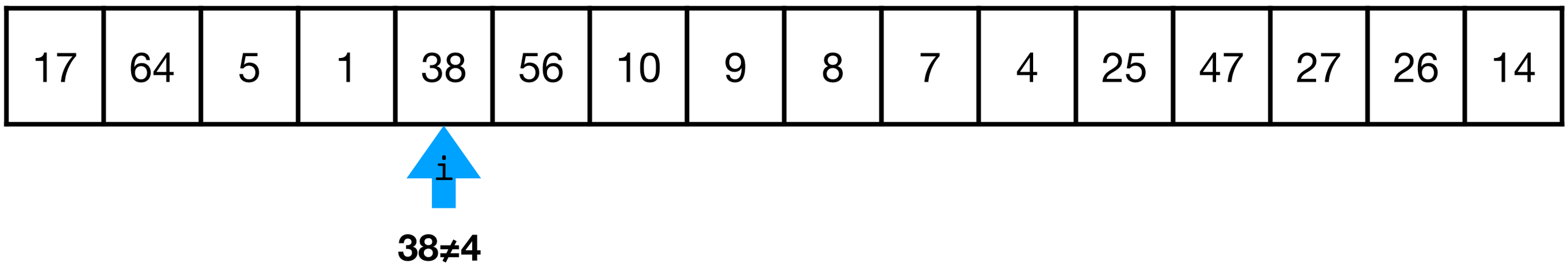
linear_search(t, 4) ?



Recherche séquentielle

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

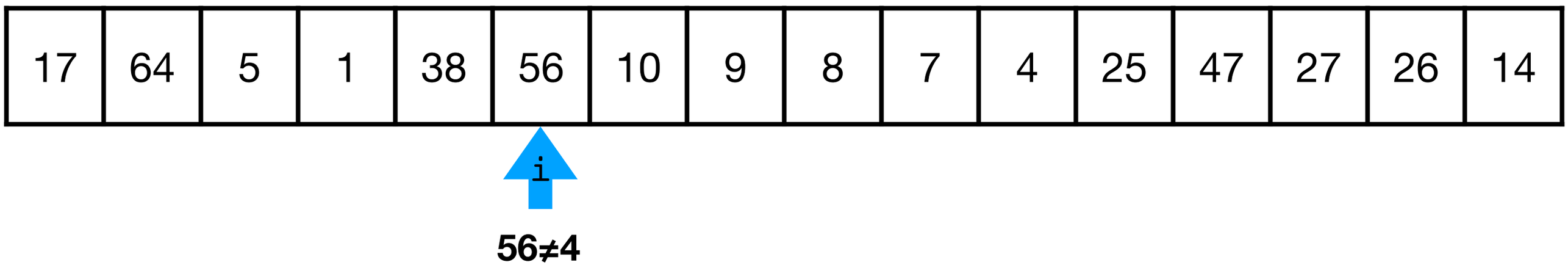
linear_search(t, 4) ?



Recherche séquentielle

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

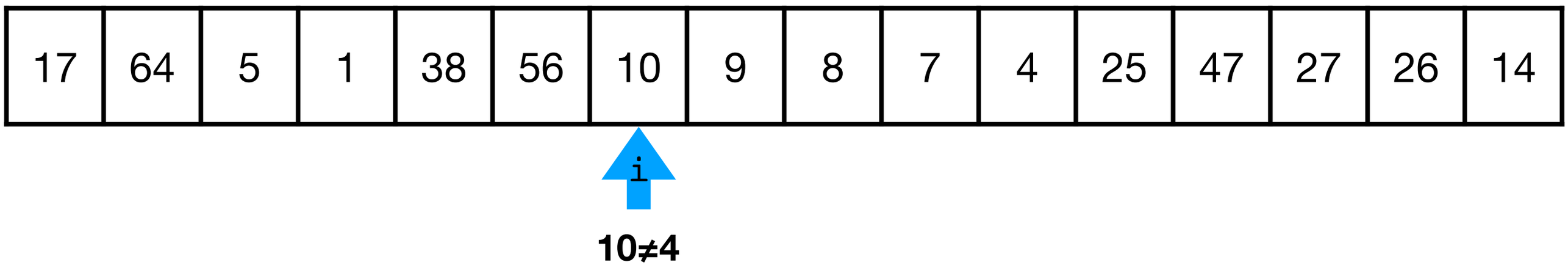
linear_search(t, 4) ?



Recherche séquentielle

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

linear_search(t, 4) ?

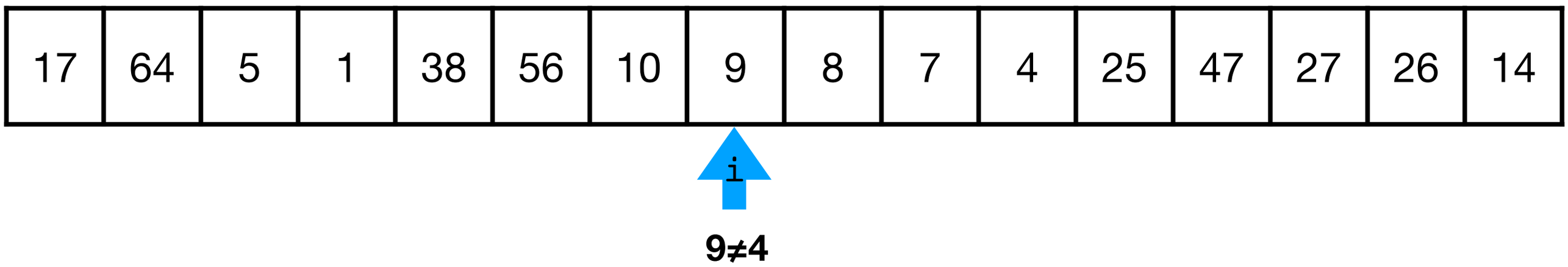


10≠4

Recherche séquentielle

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

linear_search(t, 4) ?

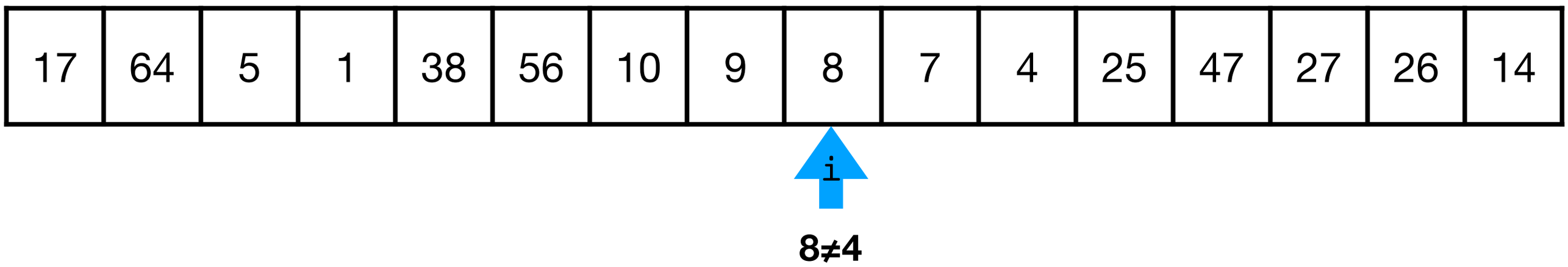


9≠4

Recherche séquentielle

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

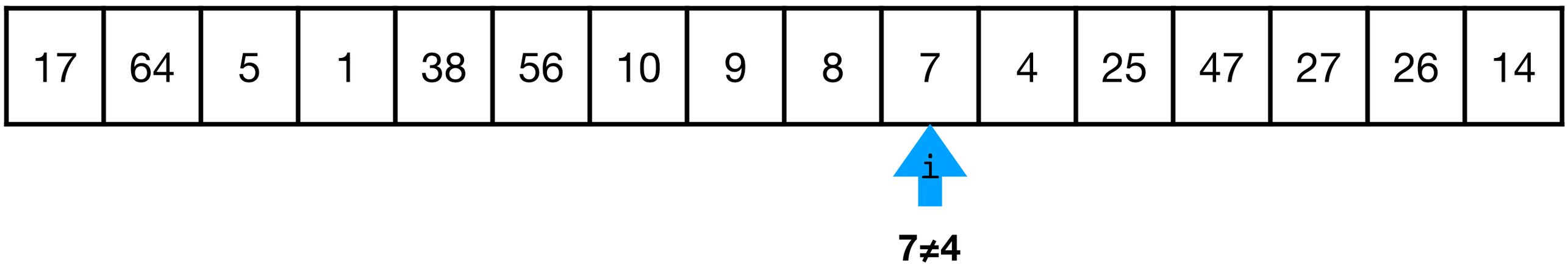
linear_search(t, 4) ?



Recherche séquentielle

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

linear_search(t, 4) ?

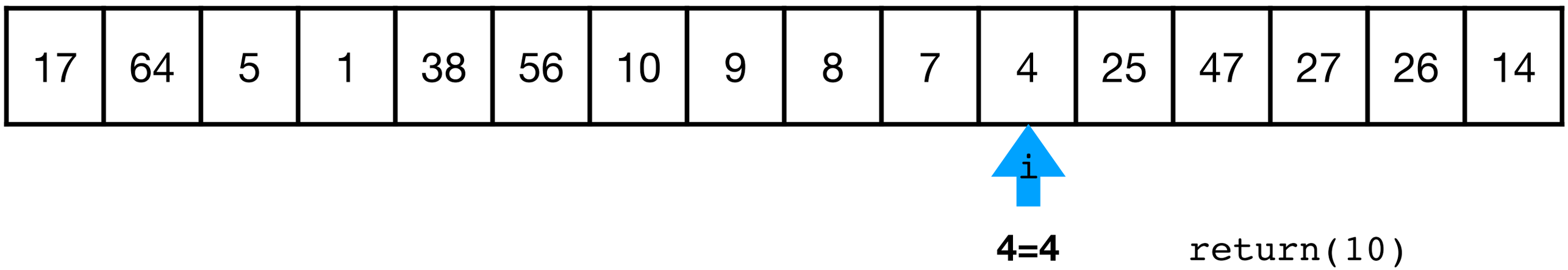


7≠4

Recherche séquentielle

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

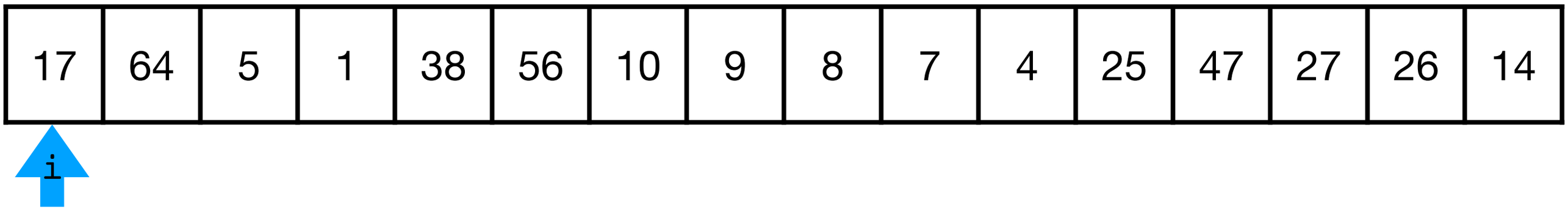
linear_search(t, 4) ?



Recherche séquentielle

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

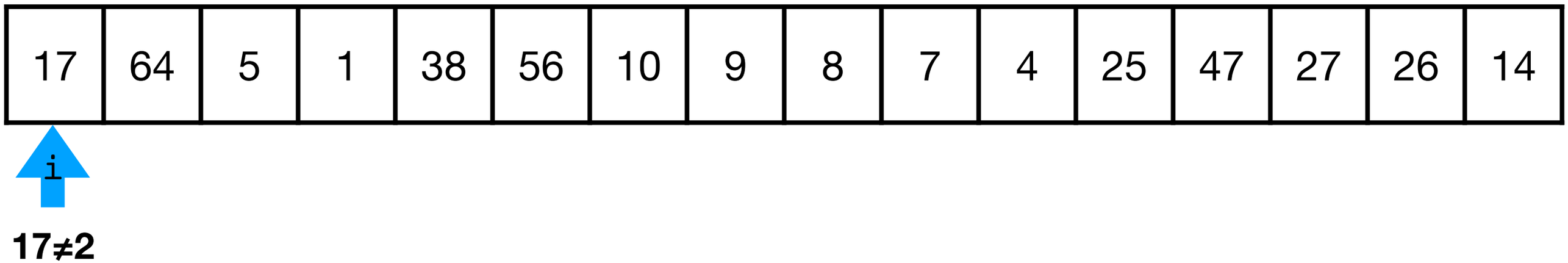
linear_search(t, 2) ?



Recherche séquentielle

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

linear_search(t, 2) ?

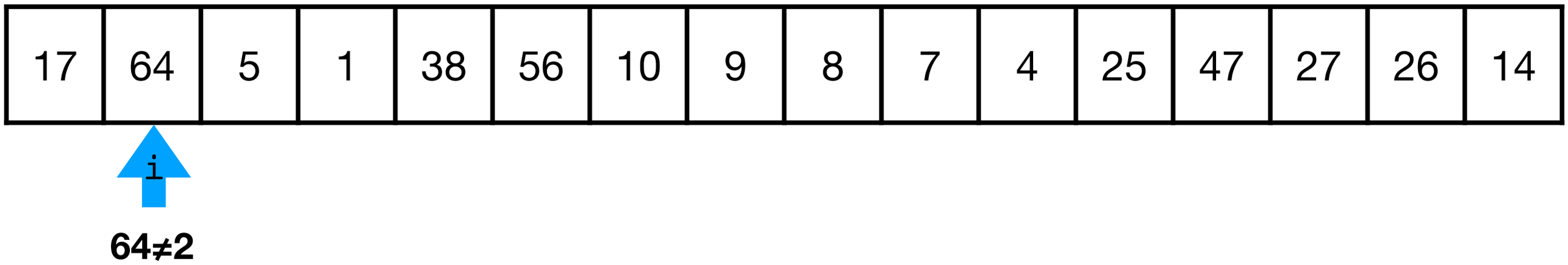


17≠2

Recherche séquentielle

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

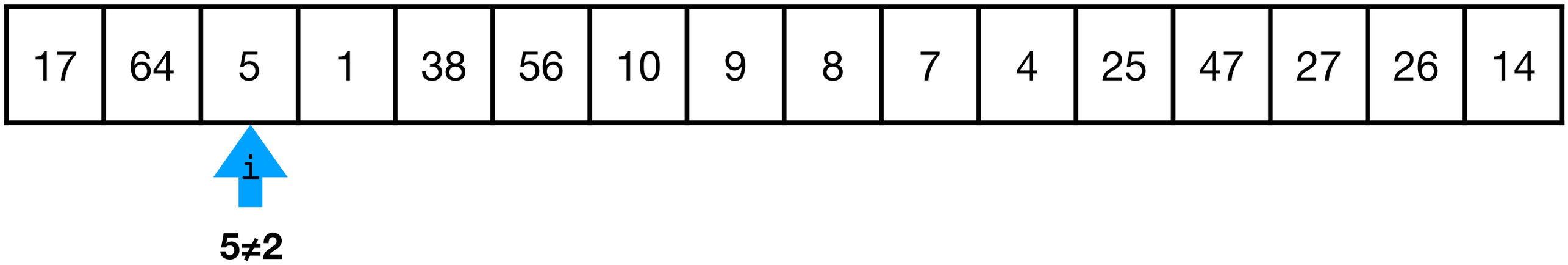
linear_search(t, 2) ?



Recherche séquentielle

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

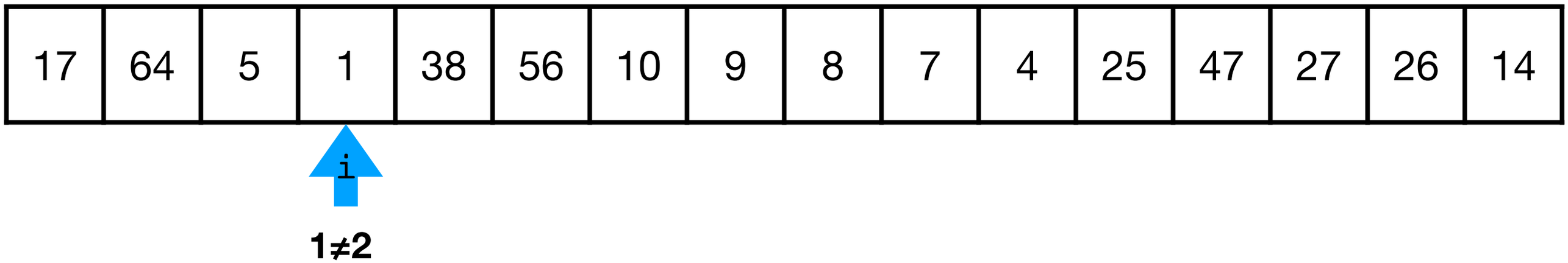
linear_search(t, 2) ?



Recherche séquentielle

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

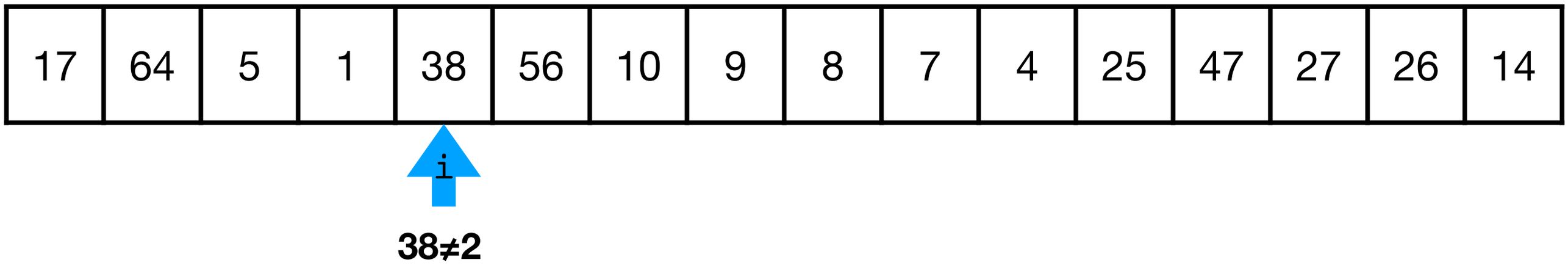
linear_search(t, 2) ?



Recherche séquentielle

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

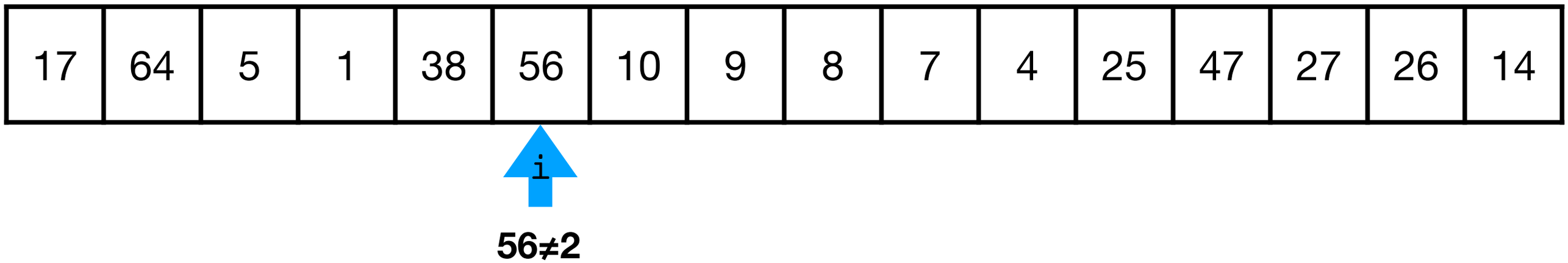
linear_search(t, 2) ?



Recherche séquentielle

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

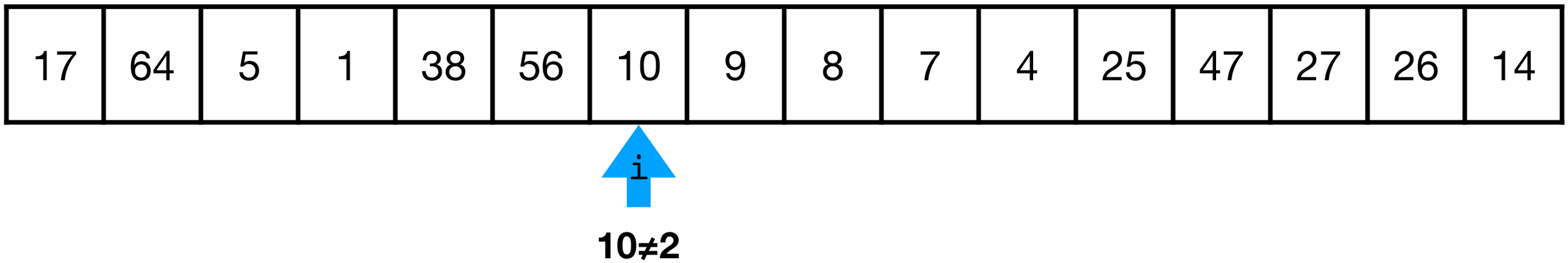
linear_search(t, 2) ?



Recherche séquentielle

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

`linear_search(t, 2)` ?

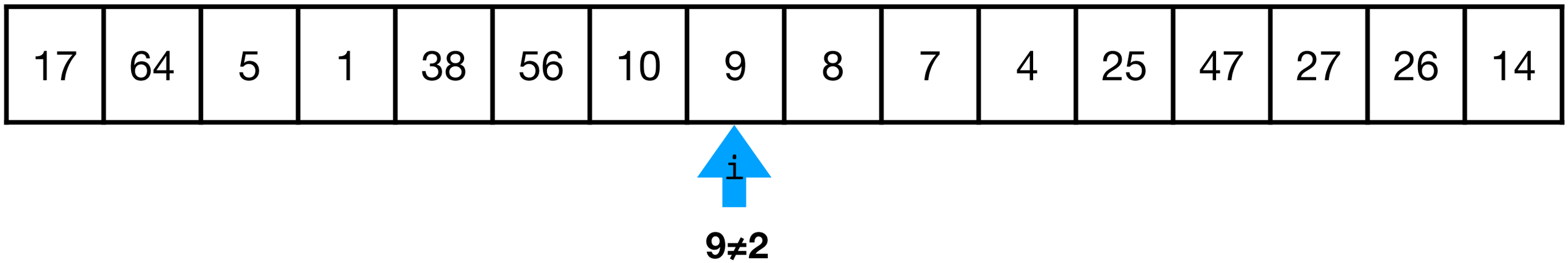


10≠2

Recherche séquentielle

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

linear_search(t, 2) ?

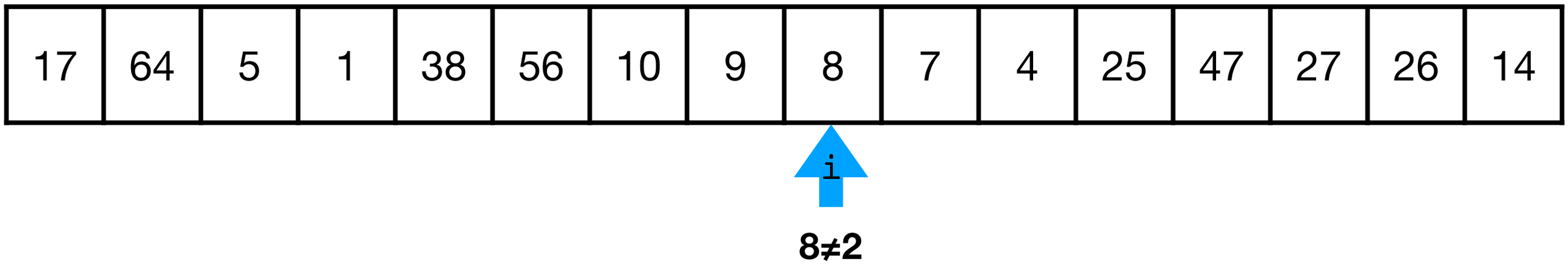


9≠2

Recherche séquentielle

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

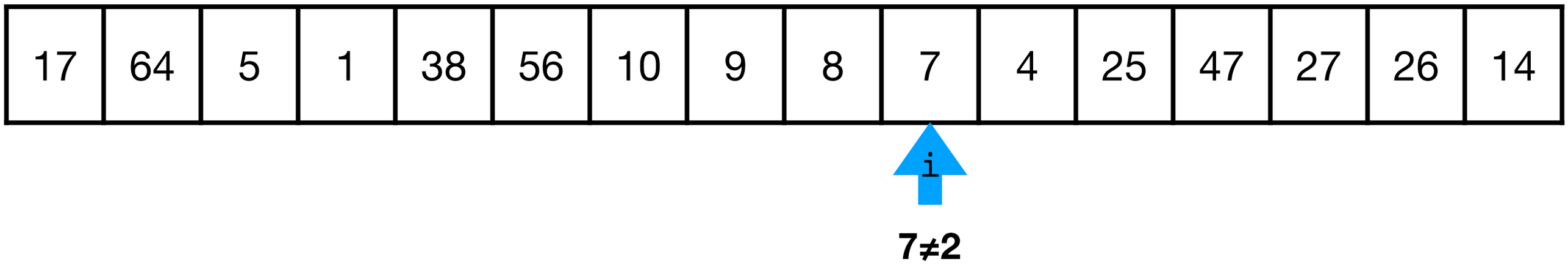
linear_search(t, 2) ?



Recherche séquentielle

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

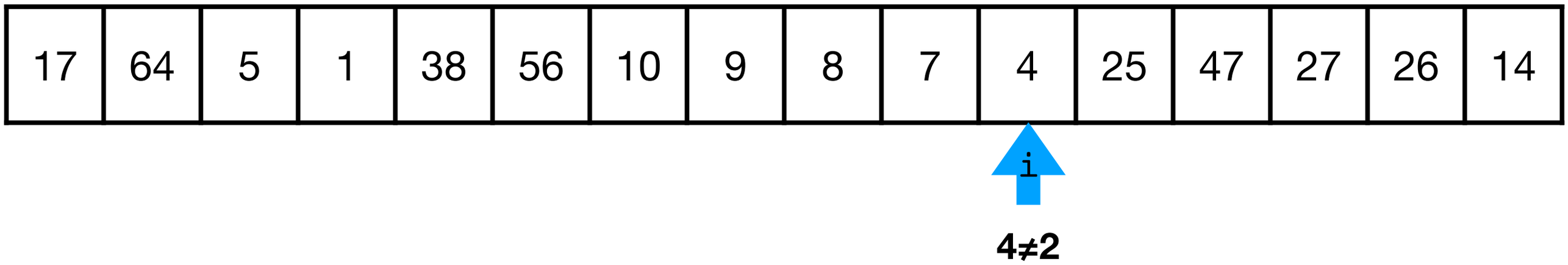
linear_search(t, 2) ?



Recherche séquentielle

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

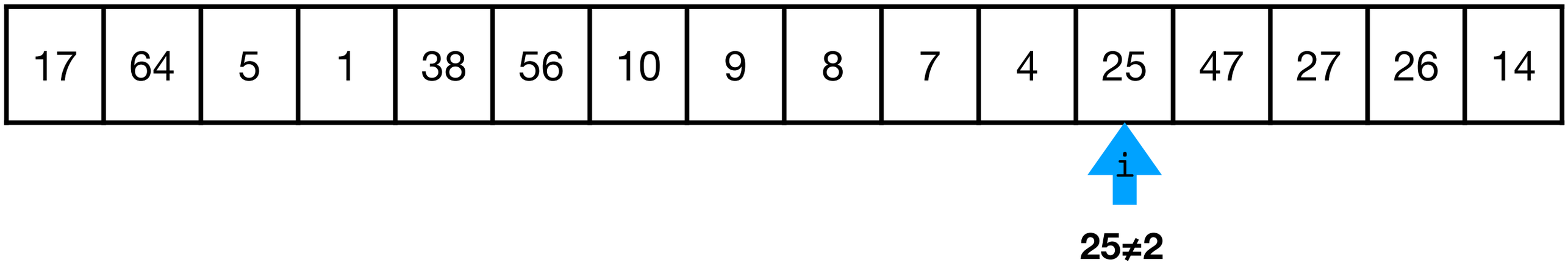
linear_search(t, 2) ?



Recherche séquentielle

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

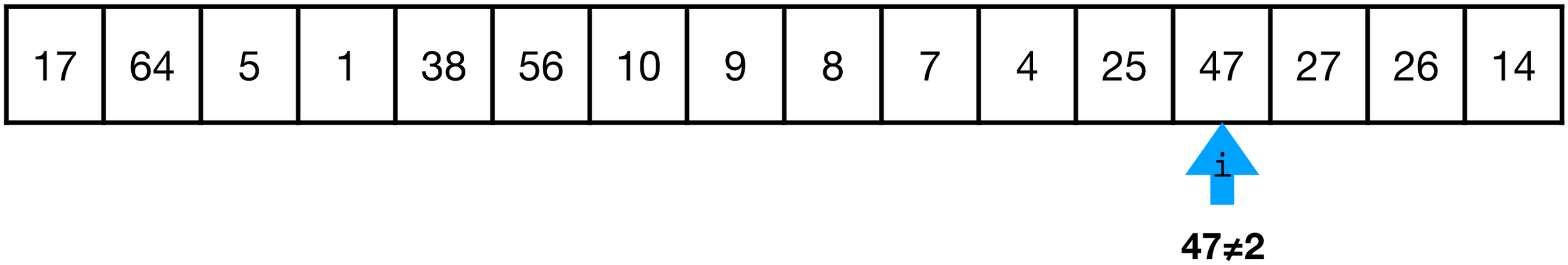
linear_search(t, 2) ?



Recherche séquentielle

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

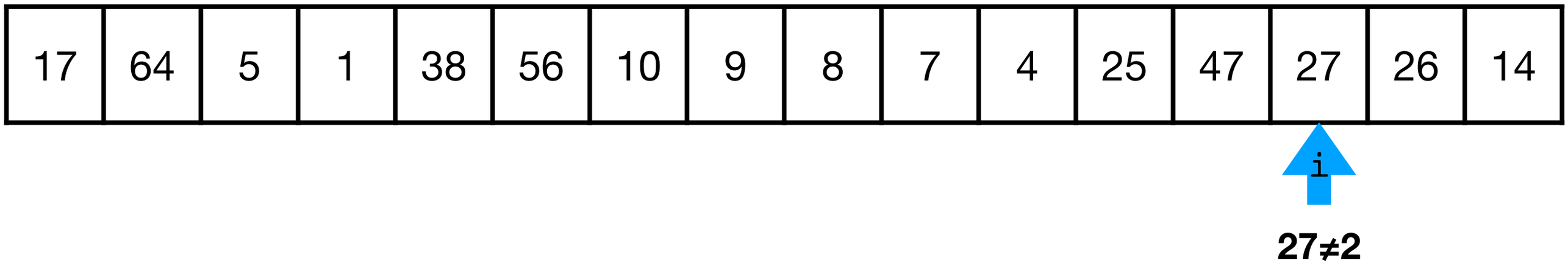
linear_search(t, 2) ?



Recherche séquentielle

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

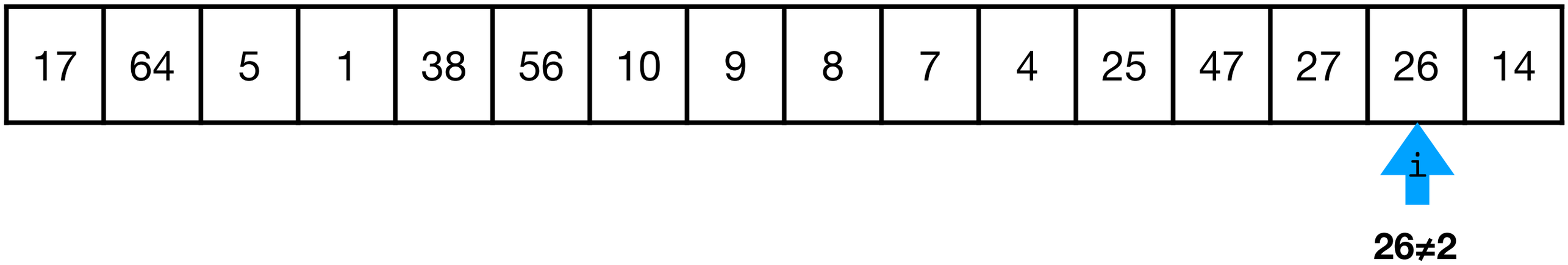
linear_search(t, 2) ?



Recherche séquentielle

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

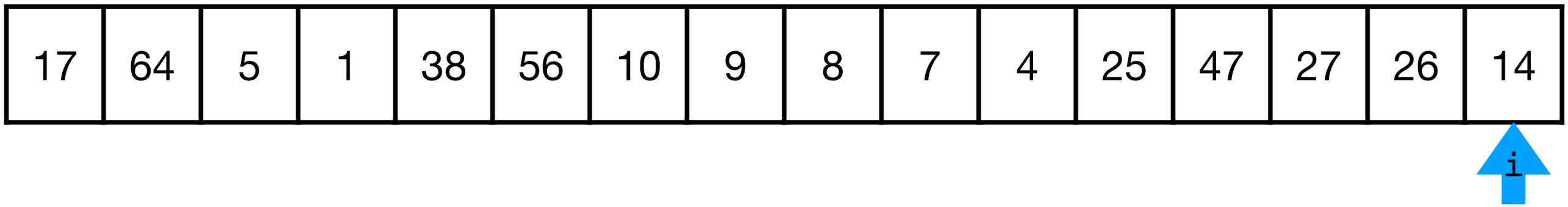
linear_search(t, 2) ?



Recherche séquentielle

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

linear_search(t, 2) ?



Recherche séquentielle

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

Terminaison ?

Efficacité ?

Terminaison

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

- Pas d'appel de fonction ou de boucle **while** donc la fonction termine toujours

Comptage des opérations

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$= 2k + 5$$

Comptage des opérations si x n'apparaît pas dans t

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

Comptage des opérations si x n'apparaît pas dans t

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

Comptage des opérations si x n'apparaît pas dans t

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

Comptage des opérations si x n'apparaît pas dans t

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

Comptage des opérations si x n'apparaît pas dans t

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

Comptage des opérations si x n'apparaît pas dans t

```
def linear_search(t, x):          1 op    × 1 fois
    n = len(t)                    1 op    × 1 fois
    for i in range(n):            1 op
        if t[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 t

```
def linear_search(t, x):          1 op    × 1 fois
    n = len(t)                    1 op    × 1 fois
    for i in range(n):            1 op
        if t[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 t

```
def linear_search(t, x):          1 op    × 1 fois
    n = len(t)                    1 op    × 1 fois
    for i in range(n):            1 op    × (n + 1) fois
        if t[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 t

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

$$= 2n + 4$$

Efficacité

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

Efficacité

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

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

Efficacité

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

- Si on a de la chance, on a $t[0] = x$ et on termine tout de suite en 5 opérations
- Si $t[k] = x$ (1ère occurrence) on fait $2k + 5$ opérations

Efficacité

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

- Si on a de la chance, on a $t[0] = x$ et on termine tout de suite en 5 opérations
- Si $t[k] = x$ (1ère occurrence) on fait $2k + 5$ opérations
- Si x n'est pas là on fait $2n + 4$ opérations

Efficacité

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

$2n + 4$ toujours supérieur ou égal à $2k + 5$ (car $k \leq n - 1$)

- Si on a de la chance, on a $t[0] = x$ et on termine tout de suite en 5 opérations
- Si $t[k] = x$ (1ère occurrence) on fait $2k + 5$ opérations
- Si x n'est pas là on fait $2n + 4$ opérations

Complexité

- Complexité : nombre d'opérations exécutées en fonction de la taille de l'entrée (longueur du tableau par exemple)

- Complexité dans le **meilleur des cas** : parmi toutes les entrées de taille *n*, quel est le plus **petit** nombre d'opérations qu'exécute l'algorithme ?

- Complexité dans le **pire des cas** : parmi toutes les entrées de taille *n*, quel est le plus **grand** nombre d'opérations qu'exécute l'algorithme ?

Complexité de la recherche séquentielle

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

Complexité de la recherche séquentielle

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

- Dans le meilleur des cas : 5 opérations (indépendant de la longueur du tableau)

Complexité de la recherche séquentielle

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

- Dans le meilleur des cas : 5 opérations (indépendant de la longueur du tableau)

- Dans le pire des cas : si le tableau est de longueur n, $2n + 4$ opérations

Exercice 2

Opération de base sur les tableaux : calculer des statistiques

Données

Données structurées

tableau :

D♦	7♦	D♠	7♠	J♠	7♥	R♣

Quel est le nombre moyen de points dans ma main ?

Exercice 3

Complexité des appels de fonction

```
def variance(t):
    m = mean(t)
    n = len(t)
    s = 0
    for i in range(n):
        s = s + (t[i] - m)**2
    return s / n
```

Complexité des appels de fonction

```
def variance(t):                          1 op
    m = mean(t)
    n = len(t)                            1 op
    s = 0                                 1 op
    for i in range(n):                    1 op
        s = s + (t[i] - m)**2             1 op
    return s / n                          1 op
```

Complexité des appels de fonction

```
def variance(t):                    1 op
    m = mean(t)                     1 + (2n + 5) op
    n = len(t)                      1 op
    s = 0                           1 op
    for i in range(n):              1 op
        s = s + (t[i] - m)**2       1 op
    return s / n                    1 op
```

Complexité des appels de fonction

```
def variance(t):                    1 op                    × 1 fois
    m = mean(t)                     1 + (2n + 5) op
    n = len(t)                      1 op
    s = 0                           1 op
    for i in range(n):              1 op
        s = s + (t[i] - m)**2       1 op
    return s / n                    1 op
```

Complexité des appels de fonction

```
def variance(t):                    1 op                    × 1 fois
    m = mean(t)                     1 + (2n + 5) op         × 1 fois
    n = len(t)                      1 op
    s = 0                           1 op
    for i in range(n):              1 op
        s = s + (t[i] - m)**2       1 op
    return s / n                    1 op
```

Complexité des appels de fonction

```
def variance(t):                    1 op                    × 1 fois
    m = mean(t)                     1 + (2n + 5) op         × 1 fois
    n = len(t)                      1 op                    × 1 fois
    s = 0                           1 op
    for i in range(n):              1 op
        s = s + (t[i] - m)**2       1 op
    return s / n                    1 op
```

Complexité des appels de fonction

```
def variance(t):                    1 op                    × 1 fois
    m = mean(t)                     1 + (2n + 5) op         × 1 fois
    n = len(t)                      1 op                    × 1 fois
    s = 0                           1 op                    × 1 fois
    for i in range(n):              1 op
        s = s + (t[i] - m)**2       1 op
    return s / n                    1 op
```

Complexité des appels de fonction

```
def variance(t):                    1 op                    × 1 fois
    m = mean(t)                     1 + (2n + 5) op         × 1 fois
    n = len(t)                      1 op                    × 1 fois
    s = 0                           1 op                    × 1 fois
    for i in range(n):              1 op                    × (n + 1) fois
        s = s + (t[i] - m)**2       1 op
    return s / n                    1 op
```

Complexité des appels de fonction

```
def variance(t):                    1 op                    × 1 fois
    m = mean(t)                     1 + (2n + 5) op         × 1 fois
    n = len(t)                      1 op                    × 1 fois
    s = 0                           1 op                    × 1 fois
    for i in range(n):              1 op                    × (n + 1) fois
        s = s + (t[i] - m)**2       1 op                    × n fois
    return s / n                    1 op
```

Complexité des appels de fonction

`def variance(t):`	1 op	$\times$ 1 fois
`    m = mean(t)`	$1 + (2n + 5)$ op	$\times$ 1 fois
`    n = len(t)`	1 op	$\times$ 1 fois
`    s = 0`	1 op	$\times$ 1 fois
`    for i in range(n):`	1 op	$\times (n + 1)$ fois
`        s = s + (t[i] - m)**2`	1 op	$\times n$ fois
`    return s / n`	1 op	$\times$ 1 fois

Complexité des appels de fonction

```
def variance(t):
    m = mean(t)
    n = len(t)
    s = 0
    for i in range(n):
        s = s + (t[i] - m)**2
    return s / n
```

1 op	× 1 fois
$1 + (2n + 5)$ op	× 1 fois
1 op	× 1 fois
1 op	× 1 fois
1 op	× $(n + 1)$ fois
1 op	× n fois
1 op	× 1 fois

$$= 4n + 11$$

Complexité des appels de fonction

```
def variance(t):                    1 op                    × 1 fois
    m = mean(t)                     1 + (2n + 5) op         × 1 fois
    n = len(t)                      1 op                    × 1 fois
    s = 0                           1 op                    × 1 fois
    for i in range(n):              1 op                    × (n + 1) fois
        s = s + (t[i] - m)**2       1 op                    × n fois
    return s / n                    1 op                    × 1 fois
```

Dans le pire et le meilleur des cas

$$= 4n + 11$$

Peut-on faire mieux que $2n + 4$ opérations dans le pire des cas pour la recherche dans un tableau ?

Peut-on faire mieux que $2n + 4$ opérations dans le pire des cas pour la recherche dans un tableau ?

… oui si le tableau est trié !

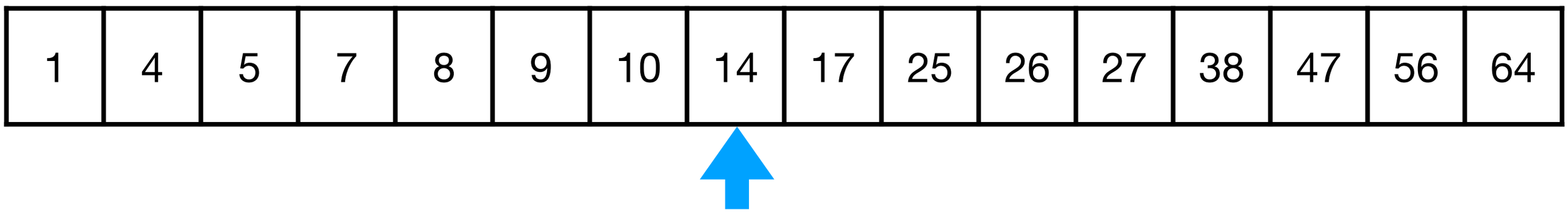
Recherche dichotomique dans un tableau trié

`binary_search(t, 26)` ?

1	4	5	7	8	9	10	14	17	25	26	27	38	47	56	64

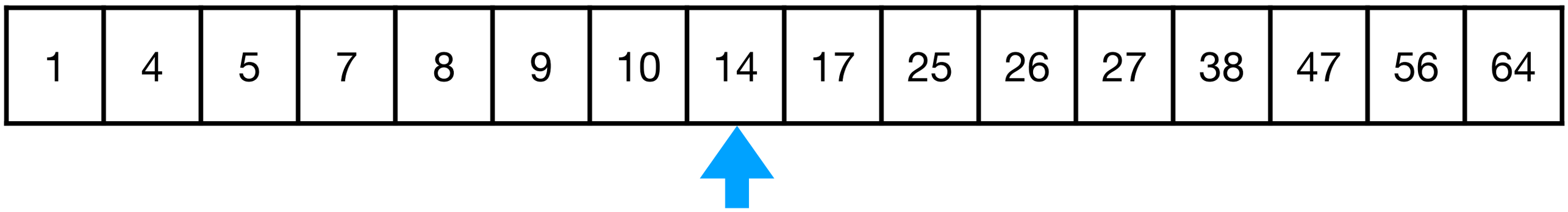
Recherche dichotomique dans un tableau trié

`binary_search(t, 26)` ?



Recherche dichotomique dans un tableau trié

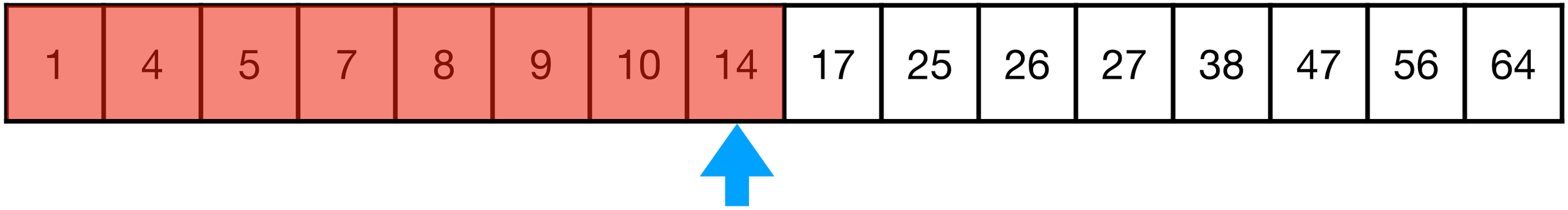
`binary_search(t, 26)` ?



14<26

Recherche dichotomique dans un tableau trié

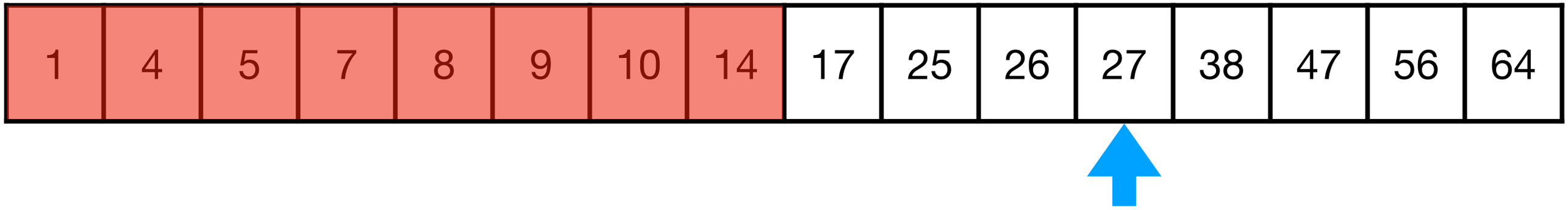
`binary_search(t, 26)` ?



14<26

Recherche dichotomique dans un tableau trié

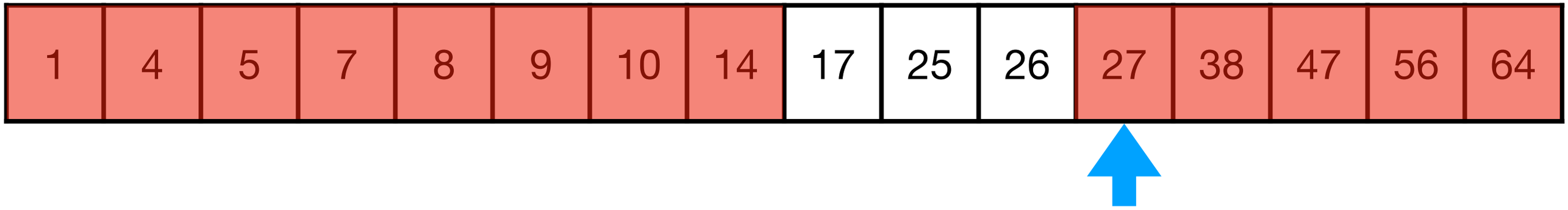
`binary_search(t, 26)` ?



27>26

Recherche dichotomique dans un tableau trié

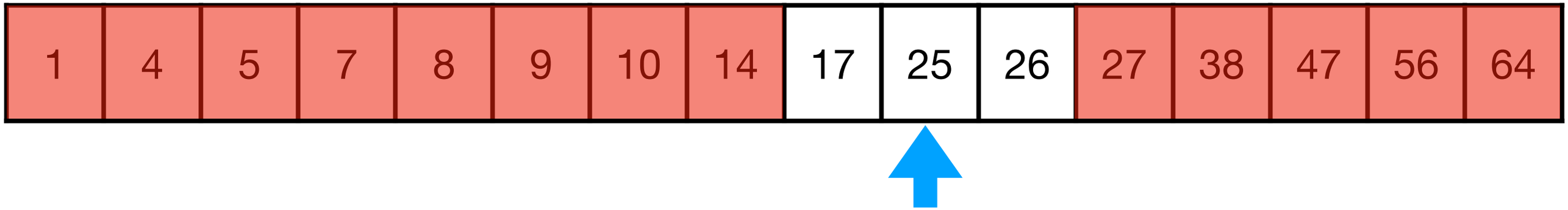
`binary_search(t, 26)` ?



27>26

Recherche dichotomique dans un tableau trié

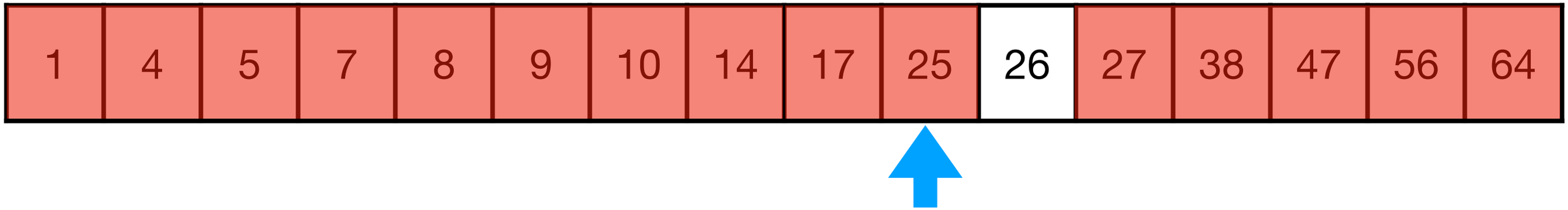
`binary_search(t, 26)` ?



25<26

Recherche dichotomique dans un tableau trié

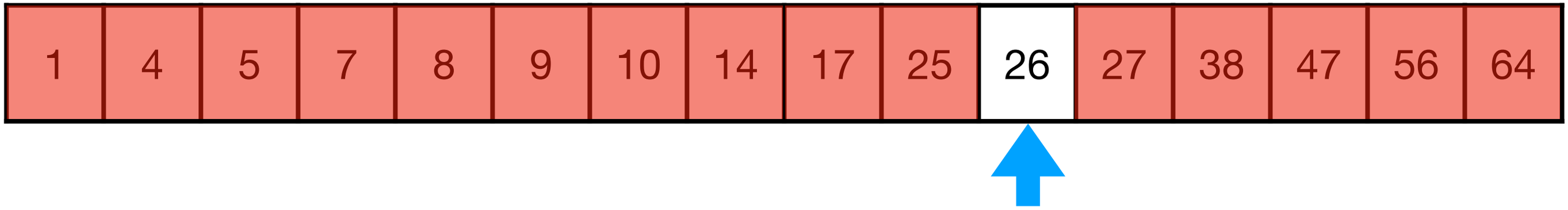
`binary_search(t, 26)` ?



25<26

Recherche dichotomique dans un tableau trié

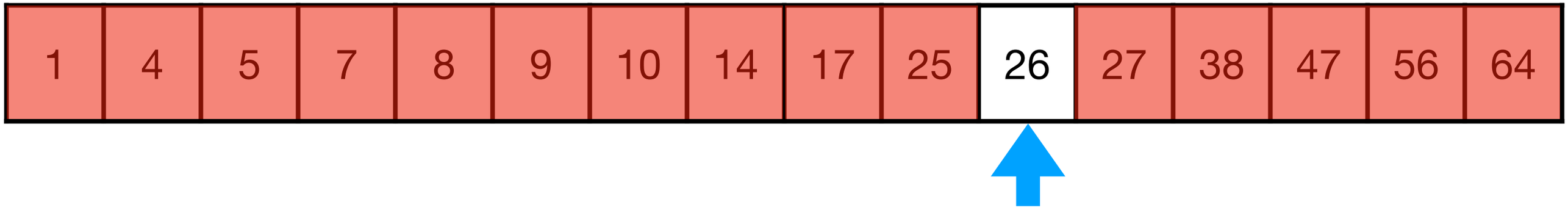
`binary_search(t, 26)` ?



`return(10)`

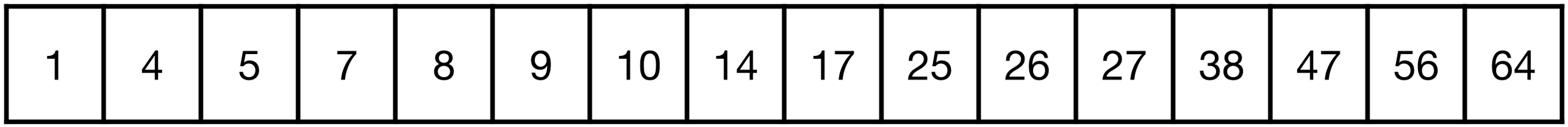
Recherche dichotomique dans un tableau trié

`binary_search(t, 26)` ?



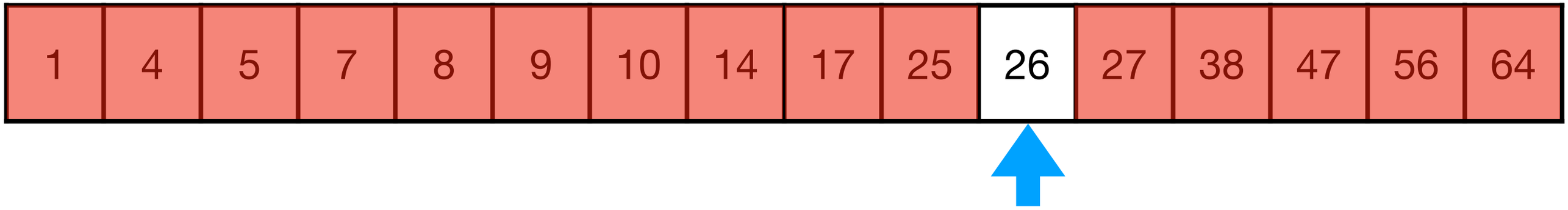
`return(10)`

`binary_search(t, 6)` ?



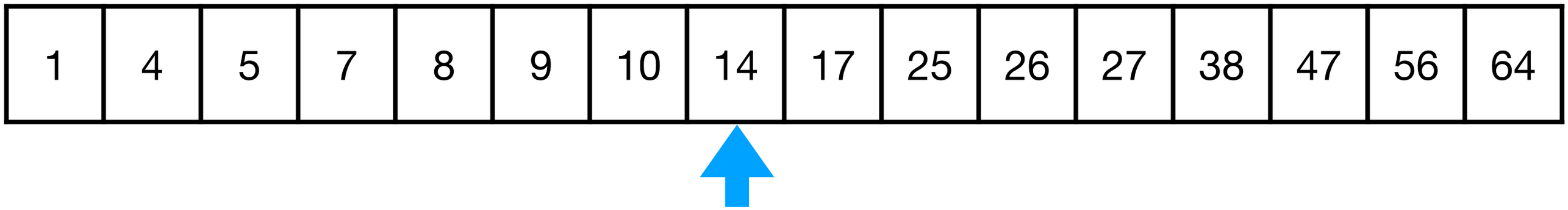
Recherche dichotomique dans un tableau trié

`binary_search(t, 26)` ?



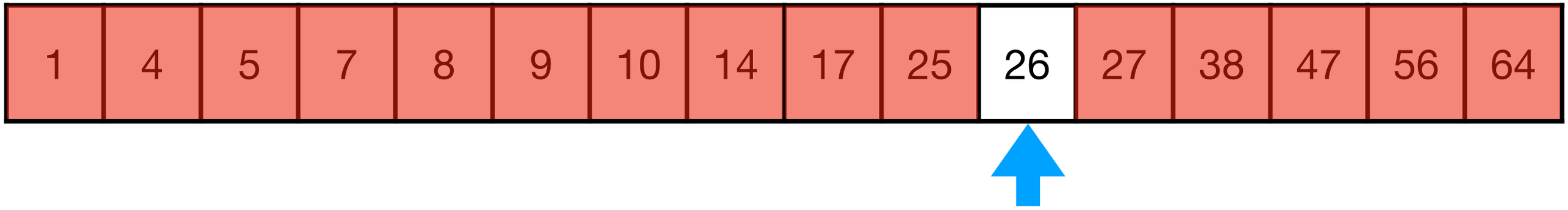
`return(10)`

`binary_search(t, 6)` ?



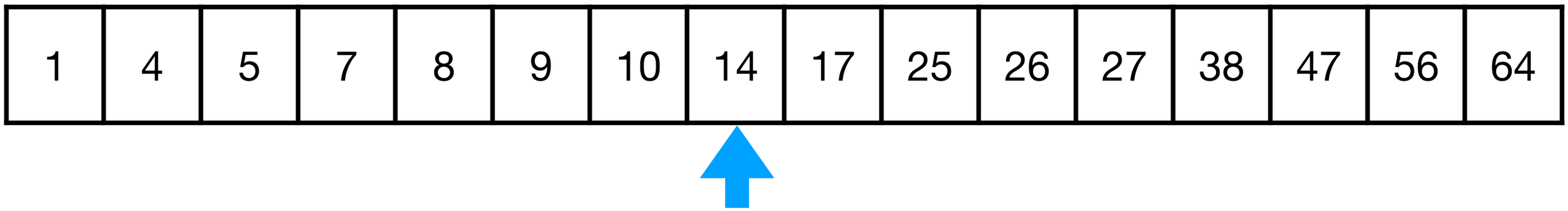
Recherche dichotomique dans un tableau trié

`binary_search(t, 26)` ?



`return(10)`

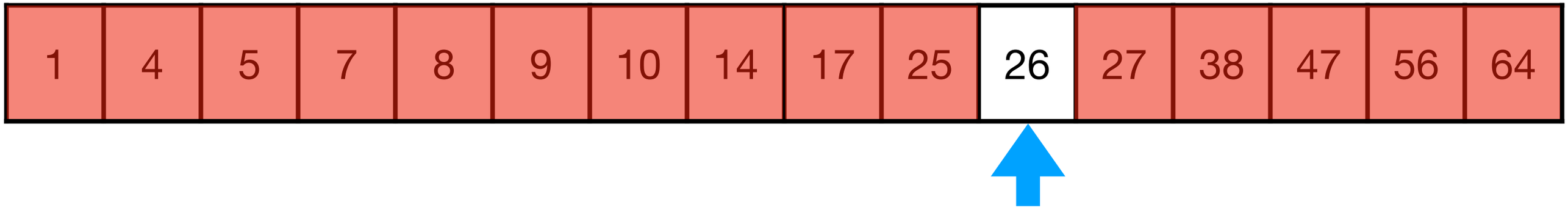
`binary_search(t, 6)` ?



14>6

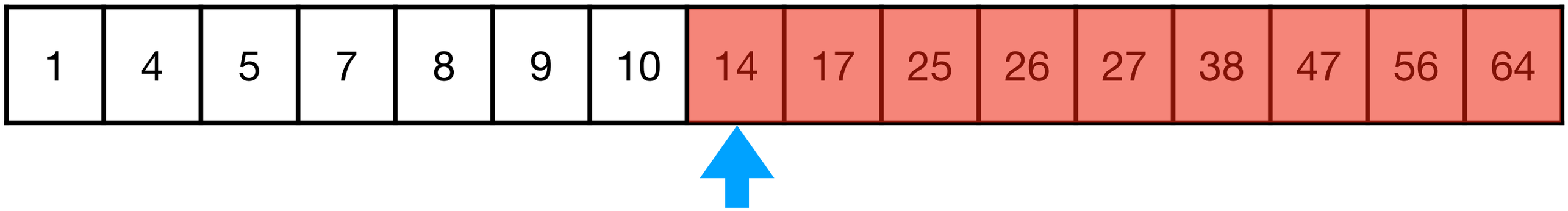
Recherche dichotomique dans un tableau trié

`binary_search(t, 26)` ?



`return(10)`

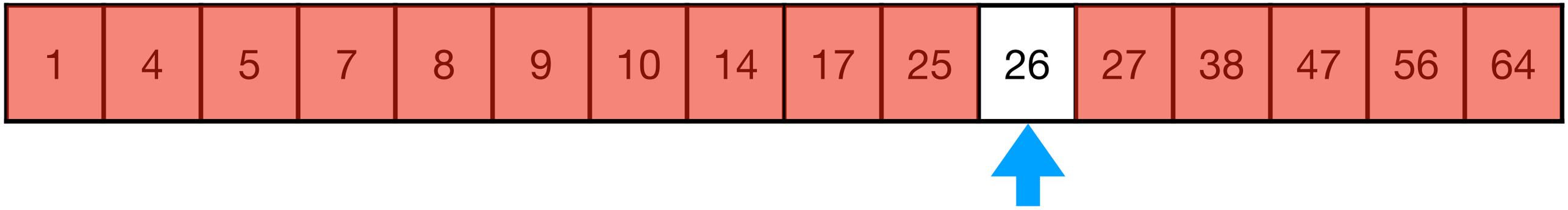
`binary_search(t, 6)` ?



14>6

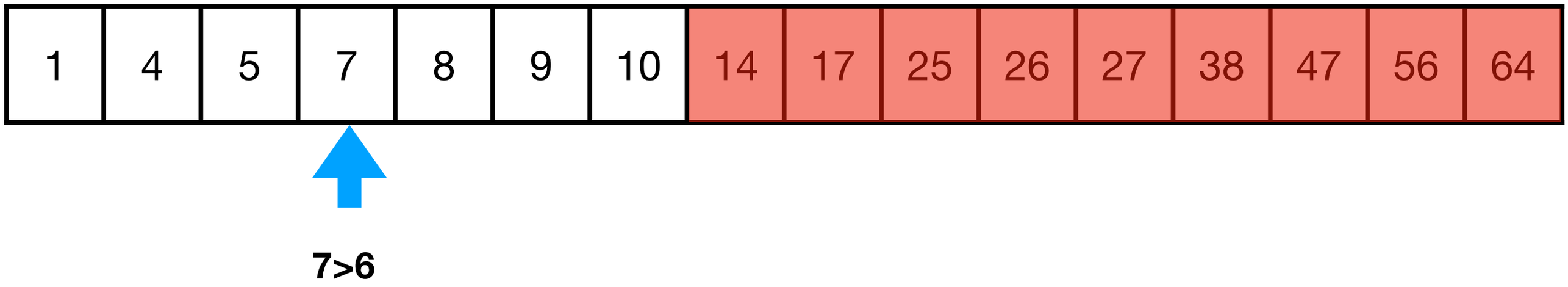
Recherche dichotomique dans un tableau trié

`binary_search(t, 26)` ?



`return(10)`

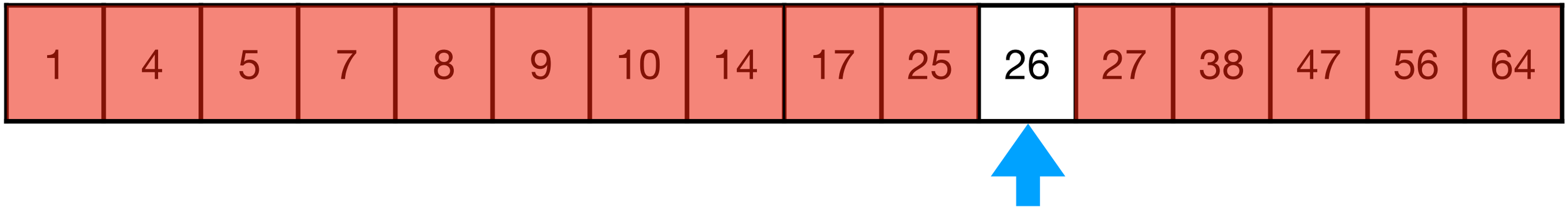
`binary_search(t, 6)` ?



7>6

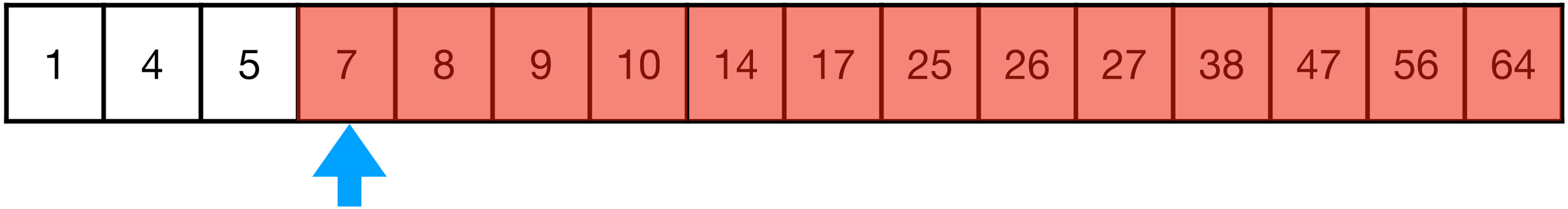
Recherche dichotomique dans un tableau trié

`binary_search(t, 26)` ?



`return(10)`

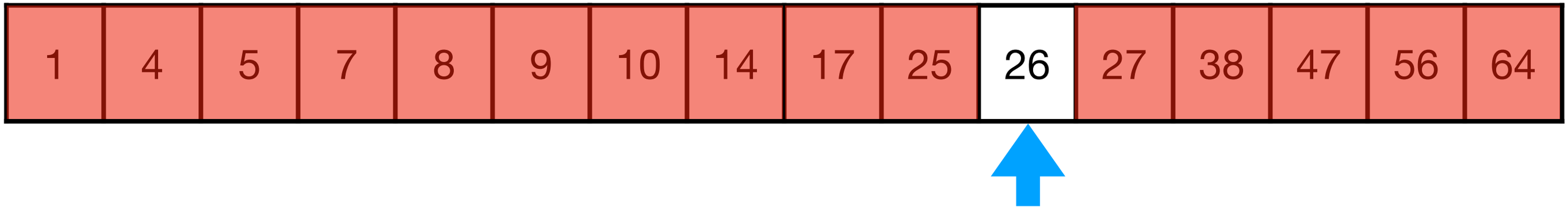
`binary_search(t, 6)` ?



7>6

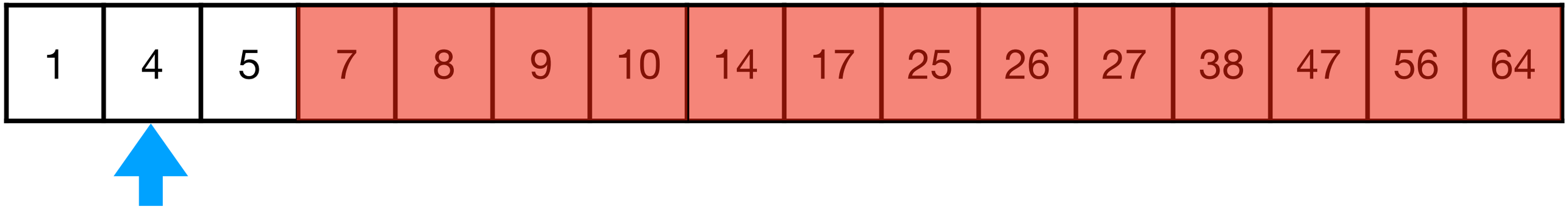
Recherche dichotomique dans un tableau trié

`binary_search(t, 26)` ?



`return(10)`

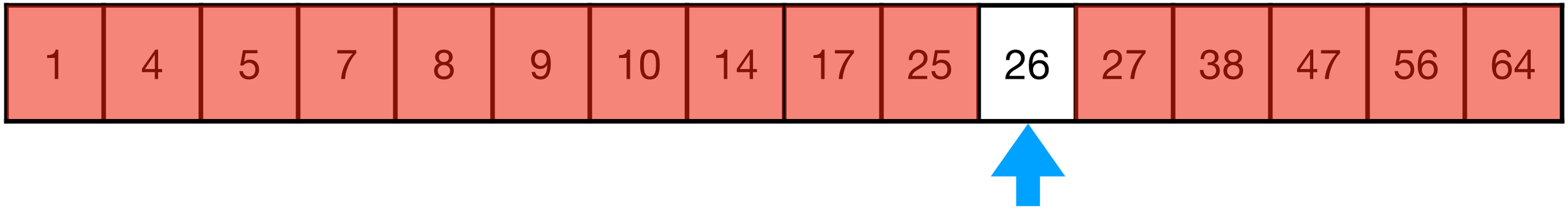
`binary_search(t, 6)` ?



4<6

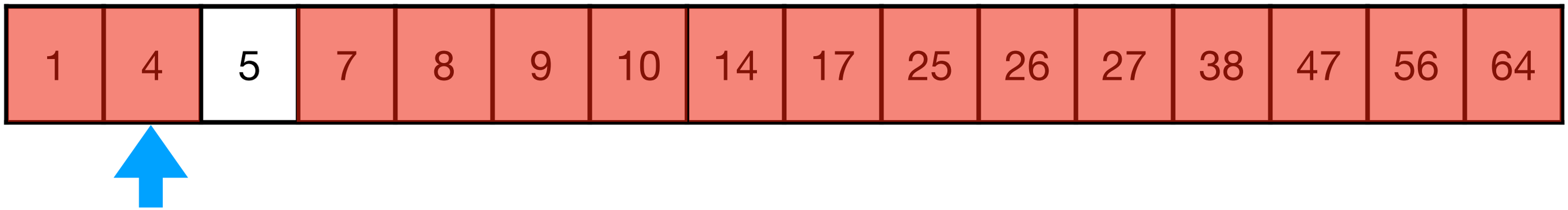
Recherche dichotomique dans un tableau trié

`binary_search(t, 26)` ?



`return(10)`

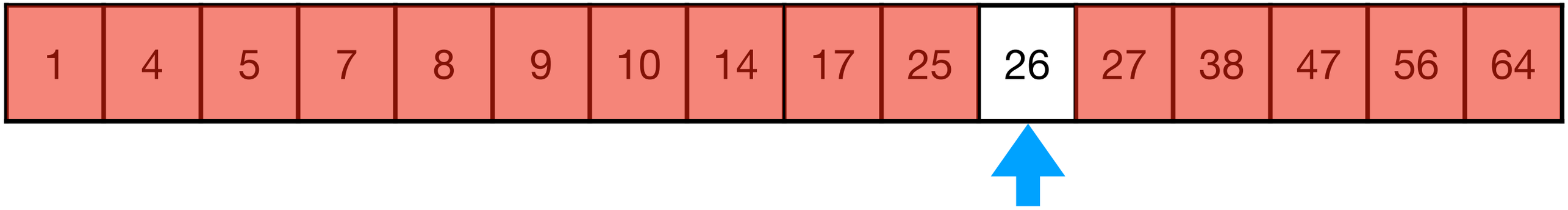
`binary_search(t, 6)` ?



4<6

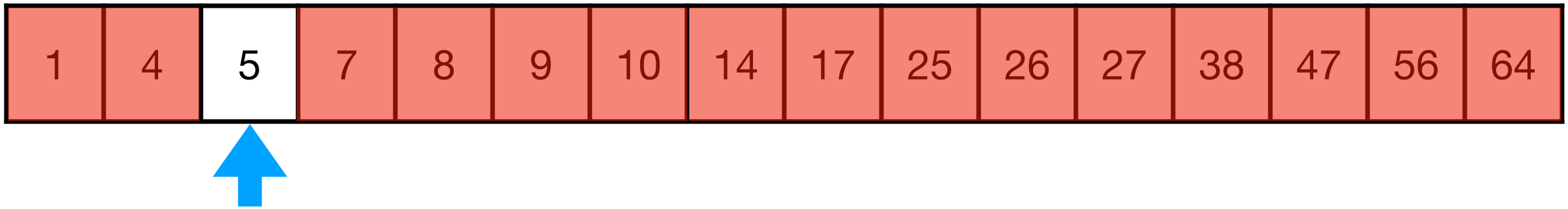
Recherche dichotomique dans un tableau trié

`binary_search(t, 26)` ?



`return(10)`

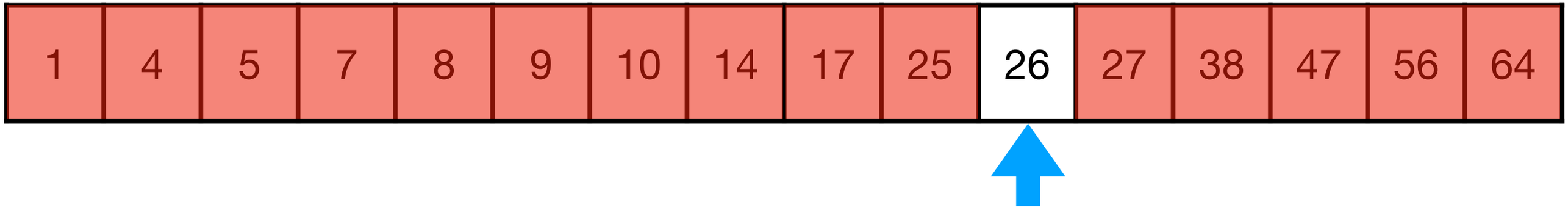
`binary_search(t, 6)` ?



5<6

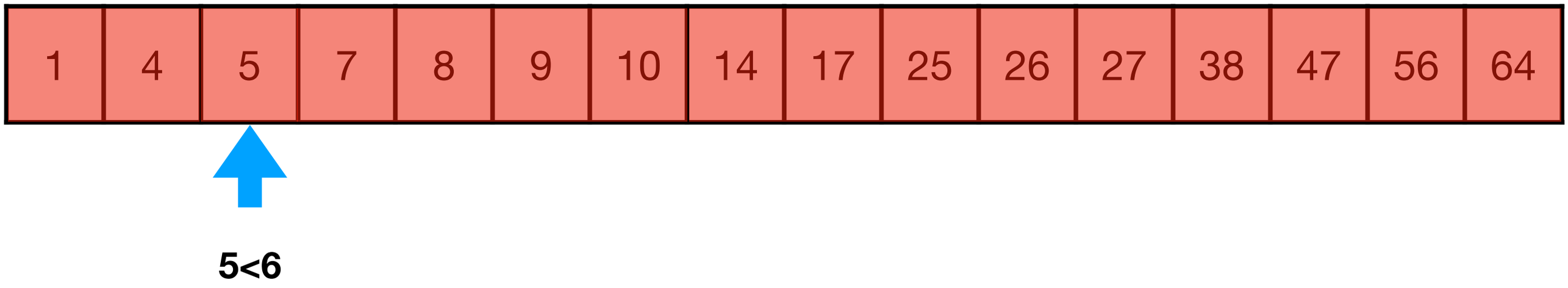
Recherche dichotomique dans un tableau trié

`binary_search(t, 26)` ?



`return(10)`

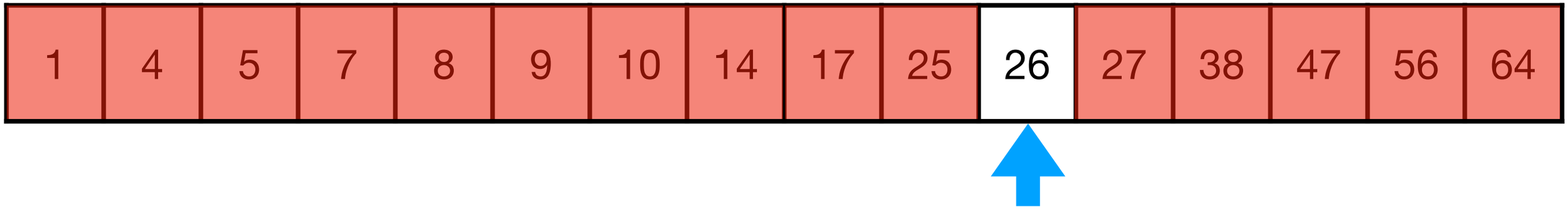
`binary_search(t, 6)` ?



5<6

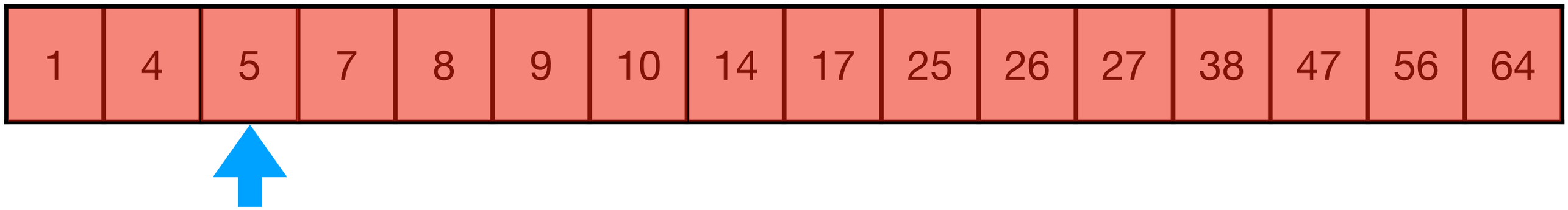
Recherche dichotomique dans un tableau trié

`binary_search(t, 26)` ?



`return(10)`

`binary_search(t, 6)` ?



5<6

`return(-1)`

Recherche dichotomique dans un tableau trié

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

Recherche dichotomique dans un tableau trié

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

Terminaison ?

Complexité ?

Terminaison

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

Terminaison

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

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

Terminaison

```
def binary_search(t, x):
    n = len(t)
    i = 0
    j = n - 1
    while i <= j:
        m = (i + j) // 2
        if x == t[m]:
            return m
        elif x < t[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 binary_search(t, x):
    n = len(t)
    i = 0
    j = n - 1
    while i <= j:
        m = (i + j) // 2
        if x == t[m]:
            return m
        elif x < t[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 $i > j$

Complexité

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

Complexité

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

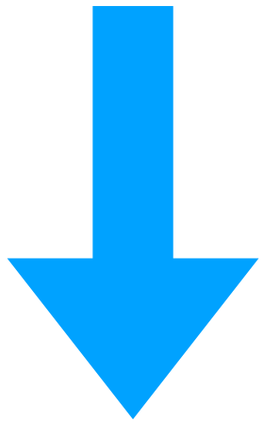
- Dans le meilleur des cas, on trouve x du premier coup : 8 opérations

Complexité

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

- Dans le meilleur des cas, on trouve x du premier coup : 8 opérations
- Dans le pire des cas, x n'est pas là. Comme on élimine à chaque itération la moitié du tableau jusqu'à ce qu'il devienne vide, on exécute la boucle $\lfloor \log_2 n \rfloor + 1$ fois au maximum : $6\lfloor \log_2 n \rfloor + 12$ opérations

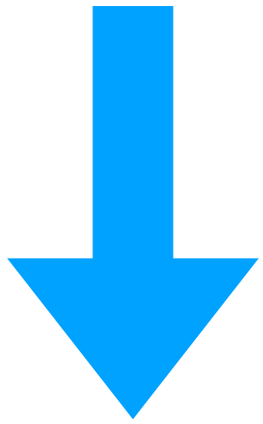
Recherche séquentielle dans un tableau quelconque



Recherche dichotomique dans un tableau trié

Recherche séquentielle dans un tableau quelconque

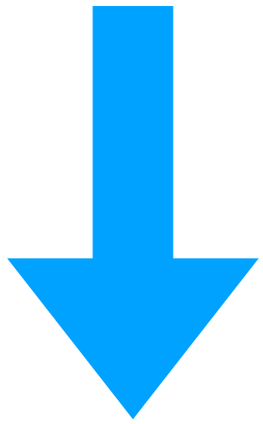
$2n + 4$ opérations dans le pire des cas



Recherche dichotomique dans un tableau trié

Recherche séquentielle dans un tableau quelconque

$2n + 4$ opérations dans le pire des cas

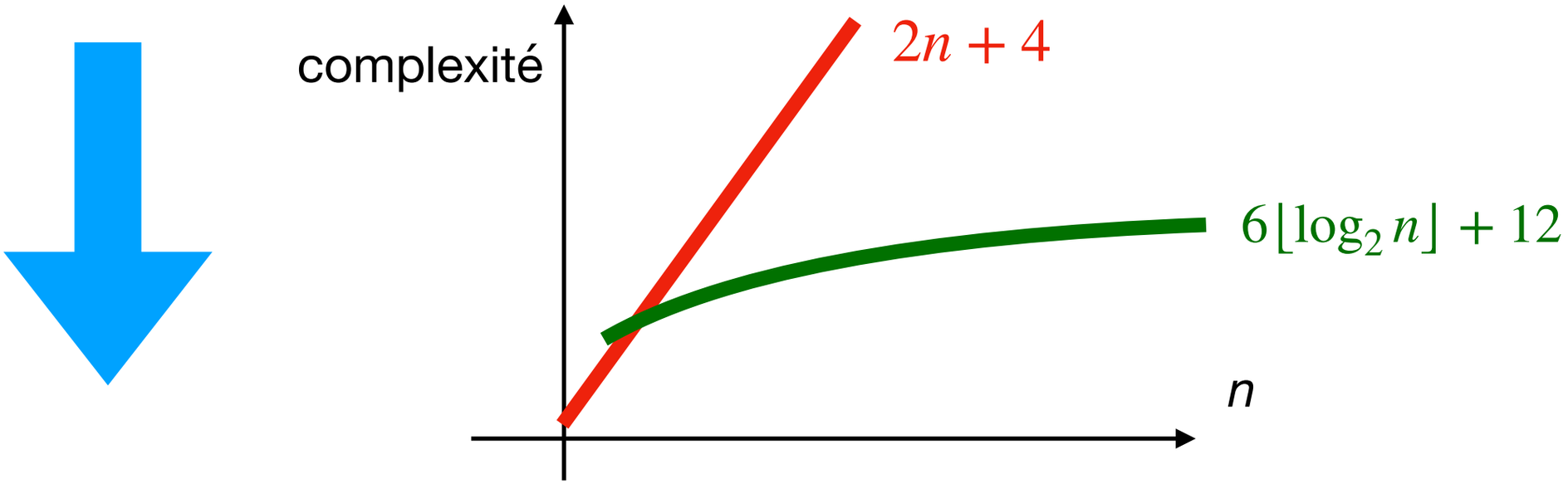


Recherche dichotomique dans un tableau trié

$6\lfloor \log_2 n \rfloor + 12$ opérations dans le pire des cas

Recherche séquentielle dans un tableau quelconque

$2n + 4$ opérations dans le pire des cas



Recherche dichotomique dans un tableau trié

$6\lfloor \log_2 n \rfloor + 12$ opérations dans le pire des cas

Exercice 4

Création et modification de tableaux

Création et modification de tableaux

```
>>> t = []                  # tableau vide
```

Création et modification de tableaux

```
>>> t = []                  # tableau vide
>>> t.append(0)             # ajout d'un élément à la fin
>>> t
```

Création et modification de tableaux

```
>>> t = []              # tableau vide
>>> t.append(0)         # ajout d'un élément à la fin
>>> t
[0]
>>> t.append(7)
```

Création et modification de tableaux

```
>>> t = []              # tableau vide
>>> t.append(0)         # ajout d'un élément à la fin
>>> t
[0]
>>> t.append(7)
>>> t
[0, 7]
```

Création et modification de tableaux

```
>>> t = []                  # tableau vide
>>> t.append(0)             # ajout d'un élément à la fin
>>> t
[0]
>>> t.append(7)
>>> t
[0, 7]
>>> t + [5, 6]              # concaténation de tableaux
[0, 7, 5, 6]
>>> u = [0 for i in range(4)]
>>> u
[0, 0, 0, 0]
>>> [2*i for i in range(5)]
[0, 2, 4, 6, 8]
```

Création et modification de tableaux

```
>>> t = []                  # tableau vide
>>> t.append(0)             # ajout d'un élément à la fin
>>> t
[0]
>>> t.append(7)
>>> t
[0, 7]
>>> t + [5, 6]              # concaténation de tableaux
[0, 7, 5, 6]
>>> u = [0 for i in range(4)]
>>> u
[0, 0, 0, 0]
>>> [2*i for i in range(5)]
[0, 2, 4, 6, 8]
```

raccourci pour :

```
u = []
for i in range(4):
    u.append(0)
```

Davantage d’options dans `range`

Davantage d'options dans `range`

```
range(5)                # génère les éléments 0, 1, 2, 3, 4
```

Davantage d'options dans `range`

```
range(5)              # génère les éléments 0, 1, 2, 3, 4
range(2, 5)           # génère les éléments 2, 3, 4
```

Davantage d'options dans `range`

```
range(5)              # génère les éléments 0, 1, 2, 3, 4
range(2, 5)           # génère les éléments 2, 3, 4
range(0, 5)           # génère les éléments 0, 1, 2, 3, 4
```

Davantage d'options dans `range`

```
range(5)                # génère les éléments 0, 1, 2, 3, 4
range(2, 5)             # génère les éléments 2, 3, 4
range(0, 5)             # génère les éléments 0, 1, 2, 3, 4
reversed(range(4)) # génère les éléments 3, 2, 1, 0
```

Davantage d'options dans `range`

```
range(5)                # génère les éléments 0, 1, 2, 3, 4
range(2, 5)             # génère les éléments 2, 3, 4
range(0, 5)             # génère les éléments 0, 1, 2, 3, 4
reversed(range(4)) # génère les éléments 3, 2, 1, 0
```

`range(a, b)` génère les éléments a, a+1, a+2, ..., b-1

Davantage d'options dans `range`

```
range(5)                # génère les éléments 0, 1, 2, 3, 4
range(2, 5)             # génère les éléments 2, 3, 4
range(0, 5)             # génère les éléments 0, 1, 2, 3, 4
reversed(range(4)) # génère les éléments 3, 2, 1, 0
```

`range(a, b)`	génère les éléments a, a+1, a+2, ..., b-1
`range(b)`	est un raccourci pour `range(0, b)`

Davantage d'options dans `range`

```
range(5)                # génère les éléments 0, 1, 2, 3, 4
range(2, 5)             # génère les éléments 2, 3, 4
range(0, 5)             # génère les éléments 0, 1, 2, 3, 4
reversed(range(4)) # génère les éléments 3, 2, 1, 0
```

`range(a, b)`	génère les éléments a, a+1, a+2, ..., b-1
`range(b)`	est un raccourci pour `range(0, b)`
`reversed(...)`	inverse l'ordre du `range`

Exercice 5