Recherche dichotomique dans un tableau trié

PCSI

Lycée Poincaré - NANCY

Position du problème

Position du problème

On cherche à savoir si un nombre est présent dans un tableau trié, et ce le plus rapidement possible.

Position du problème

On cherche à savoir si un nombre est présent dans un tableau trié, et ce le plus rapidement possible.
Dans le cas où il est présent on souhaite renvoyer vrai et/ou son indice, sinon faux.

Position du problème

On cherche à savoir si un nombre est présent dans un tableau trié, et ce le plus rapidement possible.
Dans le cas où il est présent on souhaite renvoyer vrai et/ou son indice, sinon faux.
On a déjà un algorithme qui fait ça pour un tableau trié ou non (et vous devez savoir l'écrire rapidement).

En parcourant tout le tableau

```
1 def recherche(t,x):
2     for elt in t:
3         if elt == x:
4             return True
5     return False
```

En parcourant tout le tableau

```
1 def recherche(t,x):
2     for elt in t:
3         if elt == x:
4             return True
5     return False
```

Problème, ce n'est pas très rapide et on peut faire bien mieux dans le cas où le tableau est déjà trié.

En parcourant tout le tableau

```
1 def recherche(t,x):
2     for elt in t:
3         if elt == x:
4             return True
5     return False
```

Problème, ce n'est pas très rapide et on peut faire bien mieux dans le cas où le tableau est déjà trié. Pensez à la manière dont vous chercher un mot dans le dictionnaire.

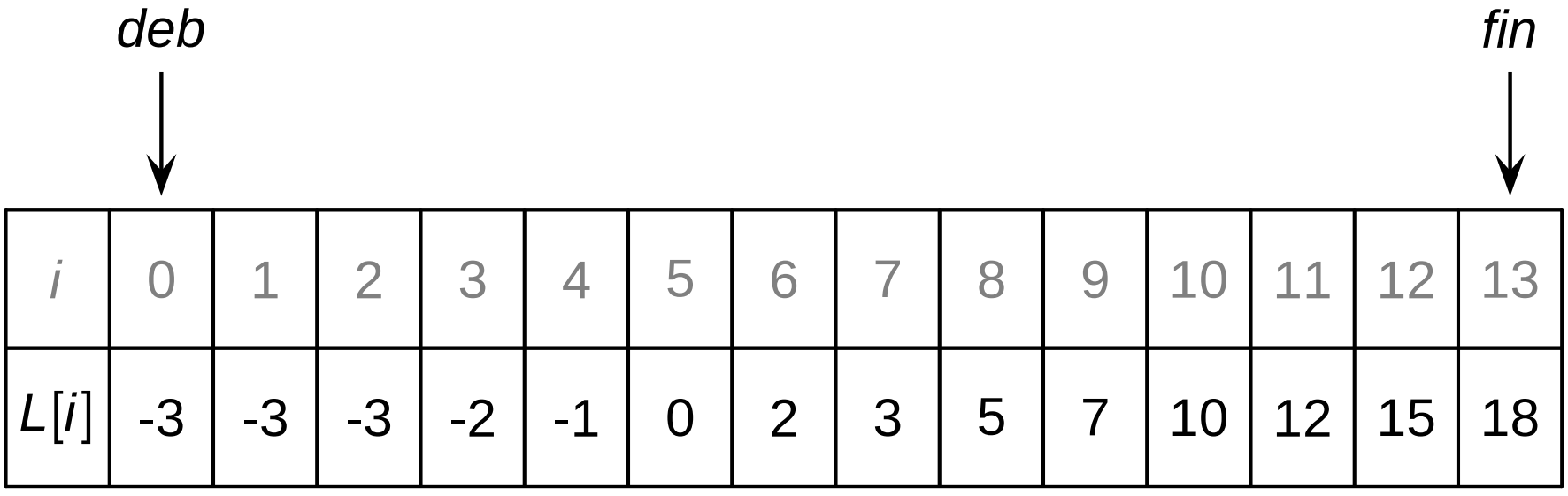
Principe de la méthode

Le principe est simple : on regarde au milieu et en comparant avec ce que l'on cherche, on continue de chercher est à droite ou à gauche.

Principe de la méthode : recherche de 5

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$L[i]$	-3	-3	-3	-2	-1	0	2	3	5	7	10	12	15	18

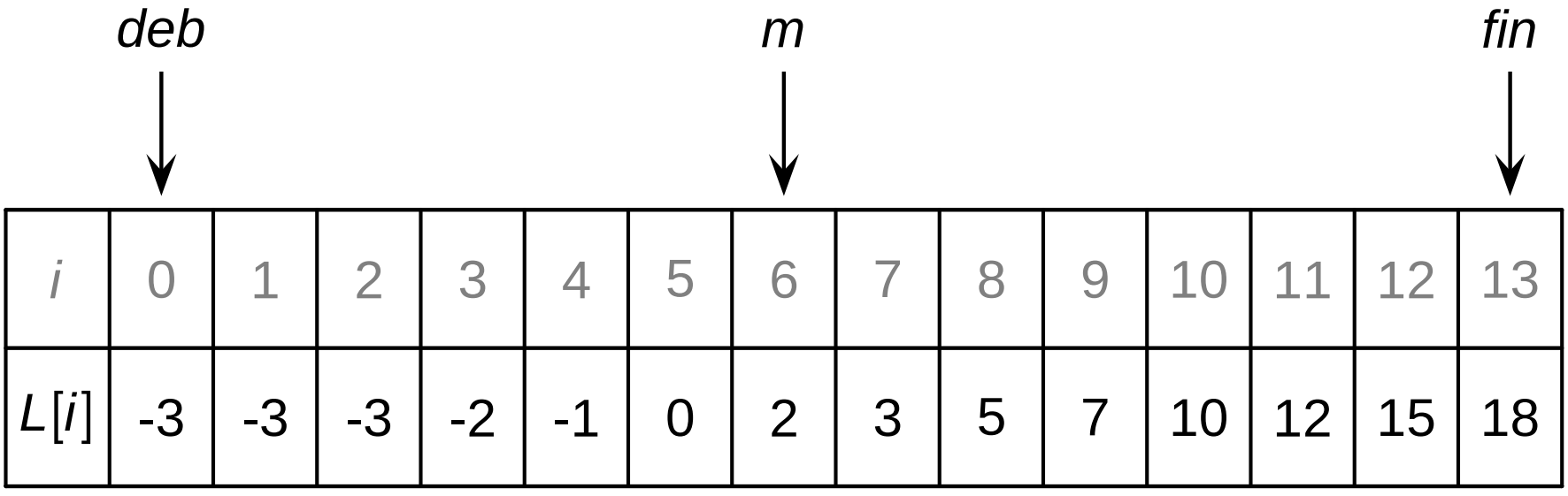
Principe de la méthode : recherche de 5



i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$L[i]$	-3	-3	-3	-2	-1	0	2	3	5	7	10	12	15	18

On va le chercher entre *deb* et *fin*

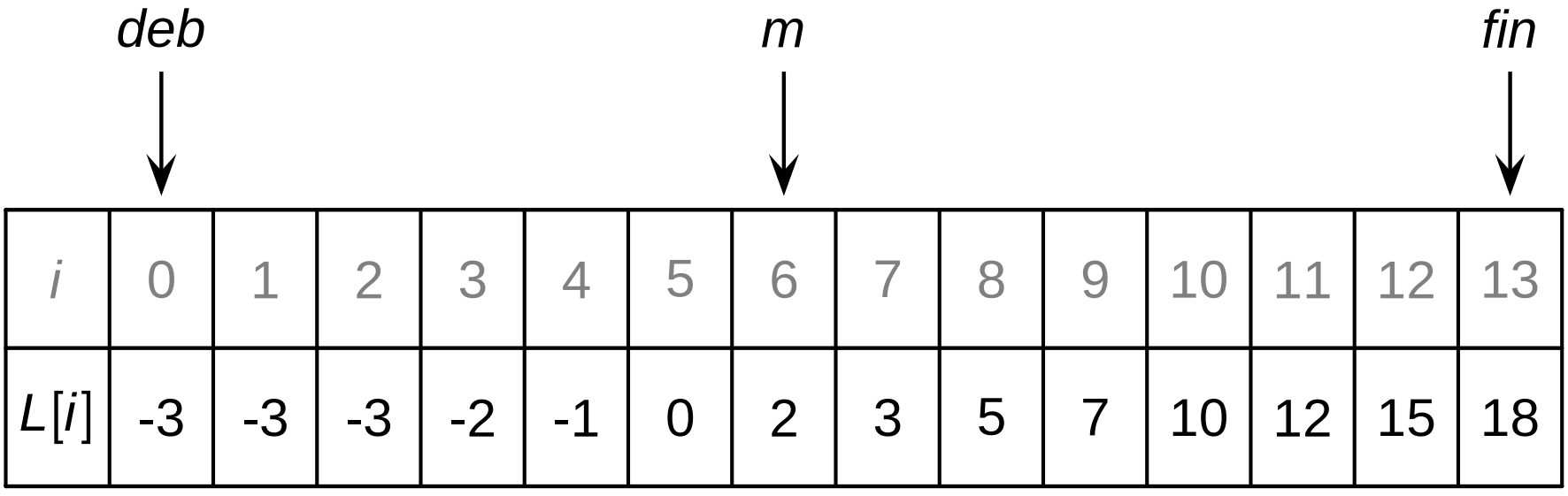
Principe de la méthode : recherche de 5



i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$L[i]$	-3	-3	-3	-2	-1	0	2	3	5	7	10	12	15	18

On regarde au « milieu » : m

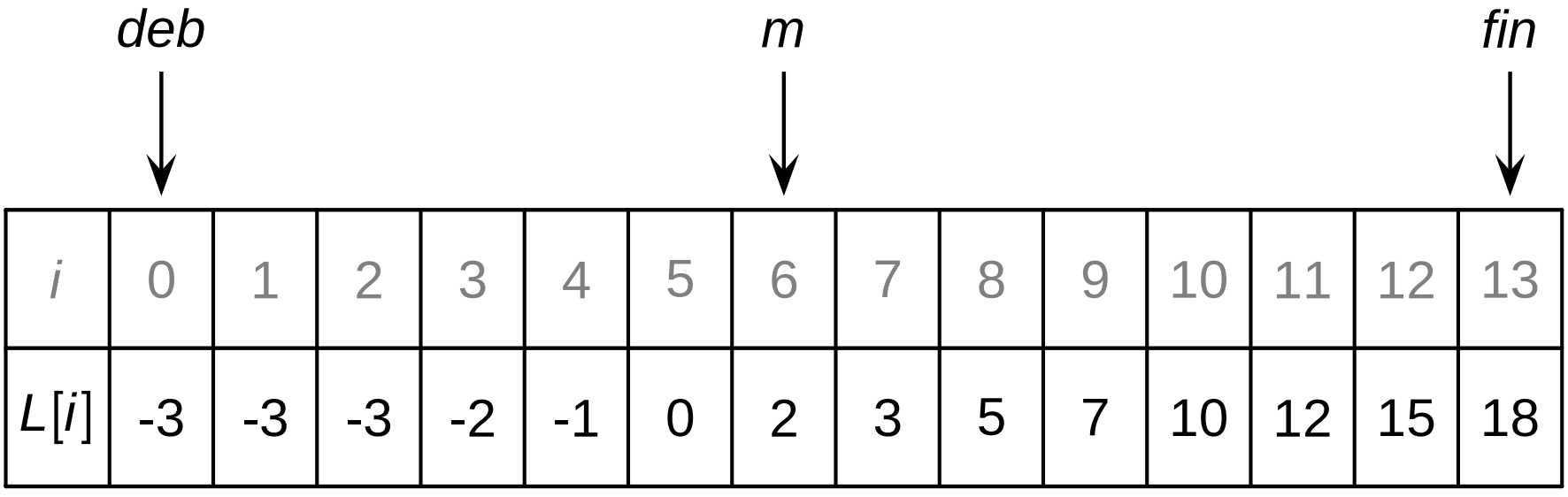
Principe de la méthode : recherche de 5



i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$L[i]$	-3	-3	-3	-2	-1	0	2	3	5	7	10	12	15	18

Attention : m doit être un entier $m = (a + b)//2$

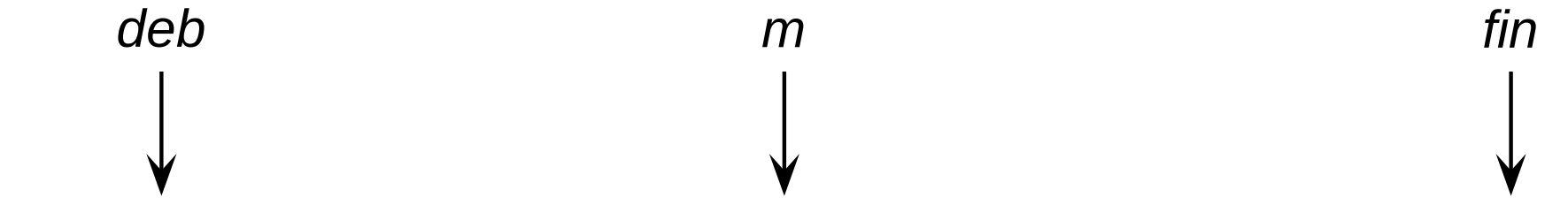
Principe de la méthode : recherche de 5



i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$L[i]$	-3	-3	-3	-2	-1	0	2	3	5	7	10	12	15	18

On compare $L[m]$ et x

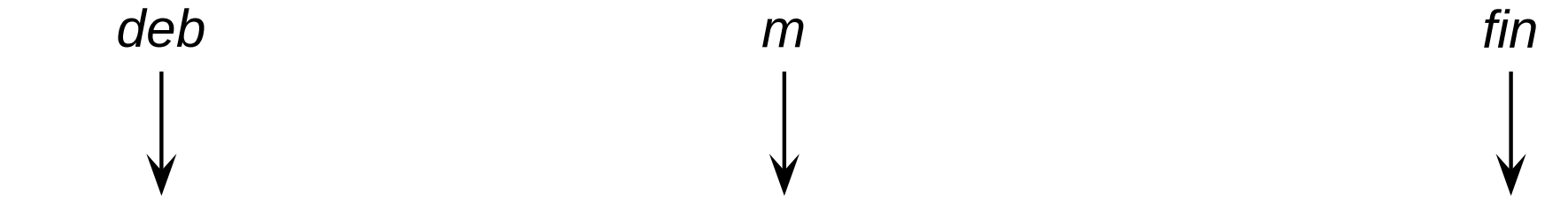
Principe de la méthode : recherche de 5



i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$L[i]$	-3	-3	-3	-2	-1	0	2	3	5	7	10	12	15	18

Ici $L[m] = 2 < x = 5$: il faut chercher à droite

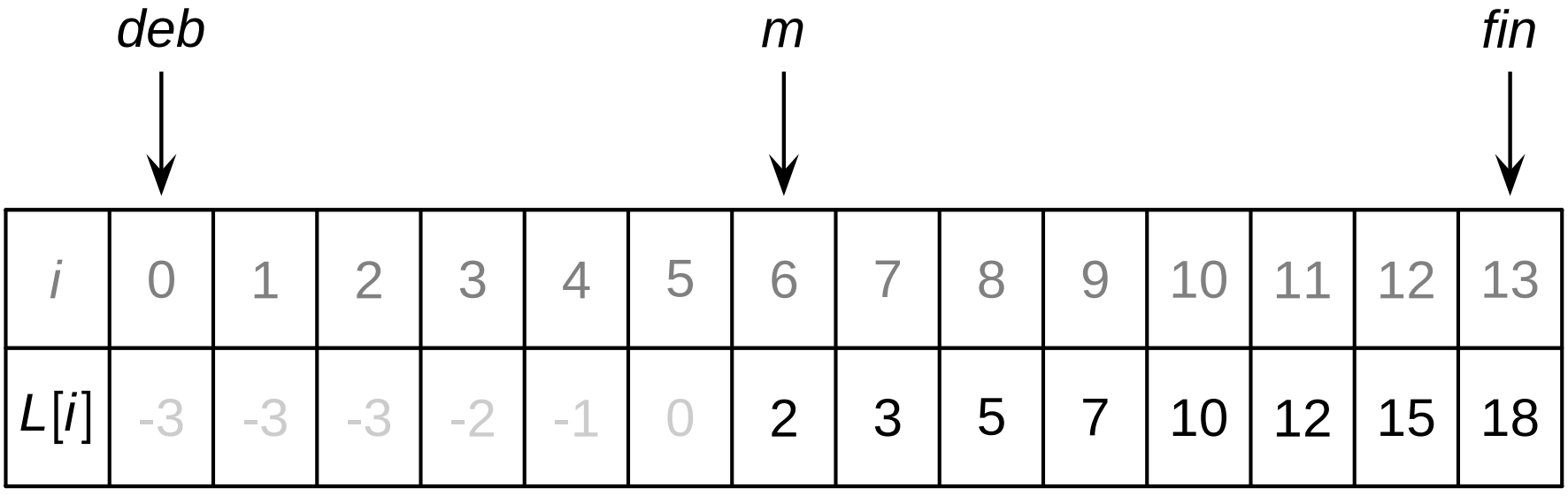
Principe de la méthode : recherche de 5



i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$L[i]$	-3	-3	-3	-2	-1	0	2	3	5	7	10	12	15	18

Ici $L[m] = 2 < x = 5$: il faut chercher à droite

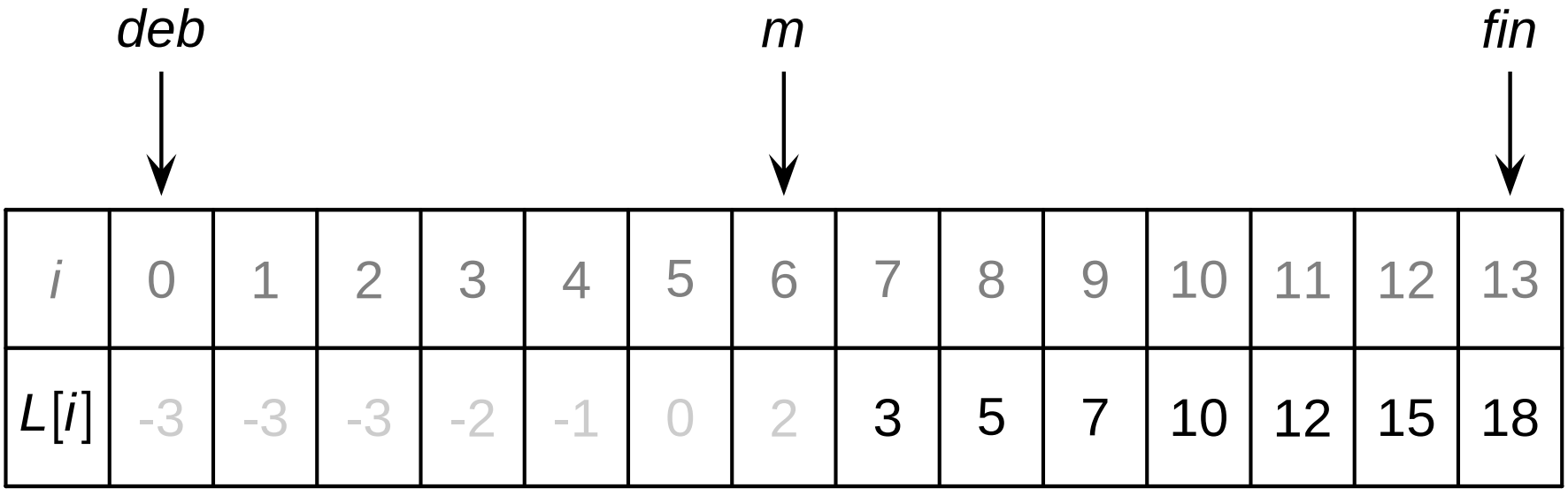
Principe de la méthode : recherche de 5



i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$L[i]$	-3	-3	-3	-2	-1	0	2	3	5	7	10	12	15	18

On peut même dire « strictement à droite » car $L[m] \neq x$

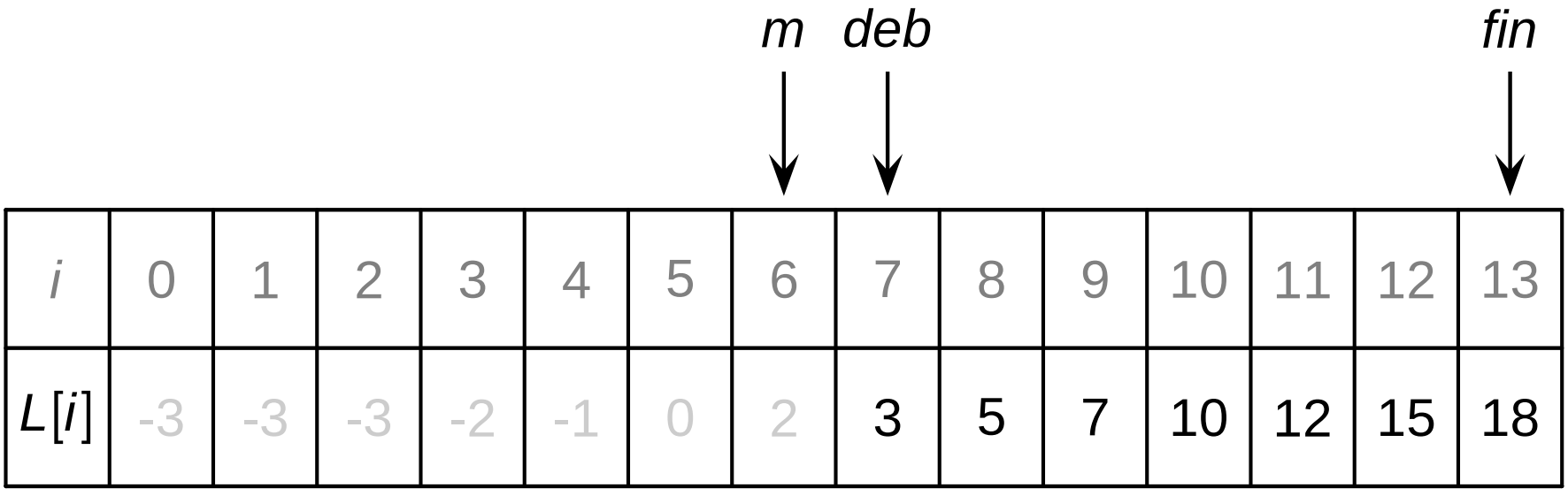
Principe de la méthode : recherche de 5



i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$L[i]$	-3	-3	-3	-2	-1	0	2	3	5	7	10	12	15	18

On recommence dans la partie de droite : $deb = m + 1$

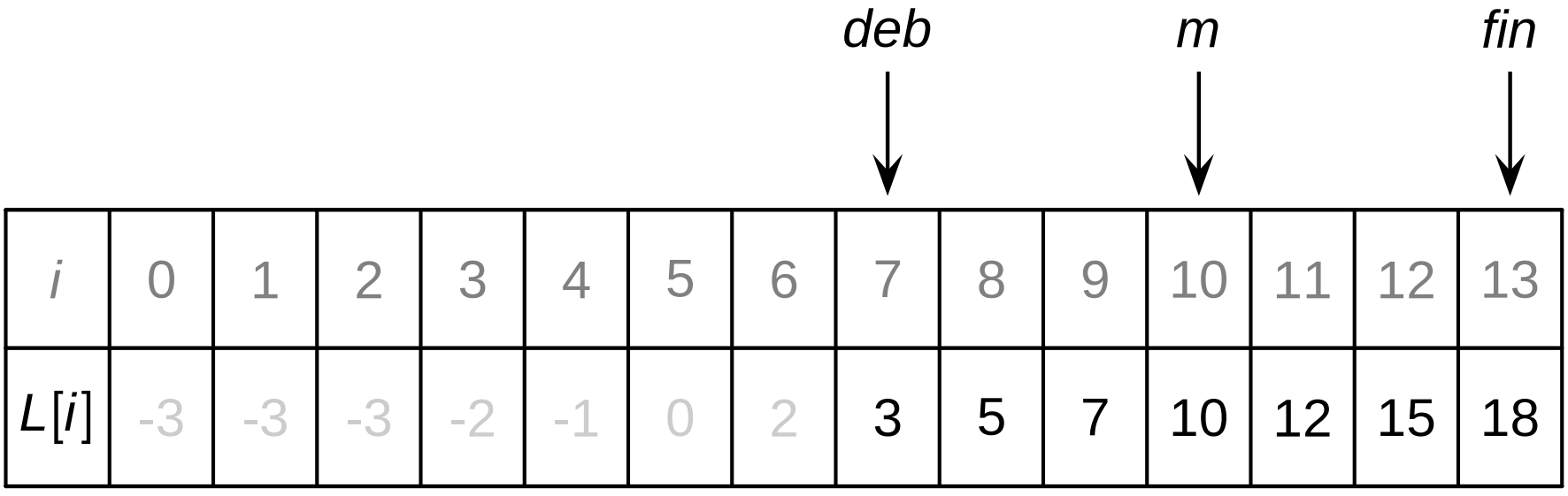
Principe de la méthode : recherche de 5



i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$L[i]$	-3	-3	-3	-2	-1	0	2	3	5	7	10	12	15	18

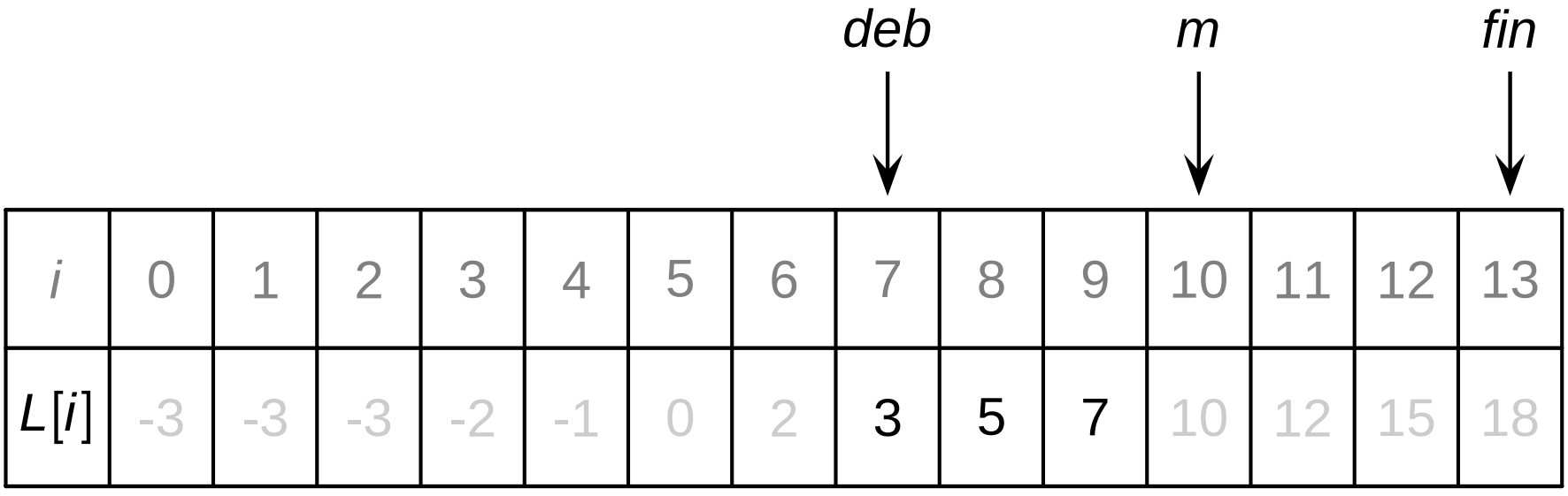
Ce +1 est très très important pour que le programme se termine.

Principe de la méthode : recherche de 5



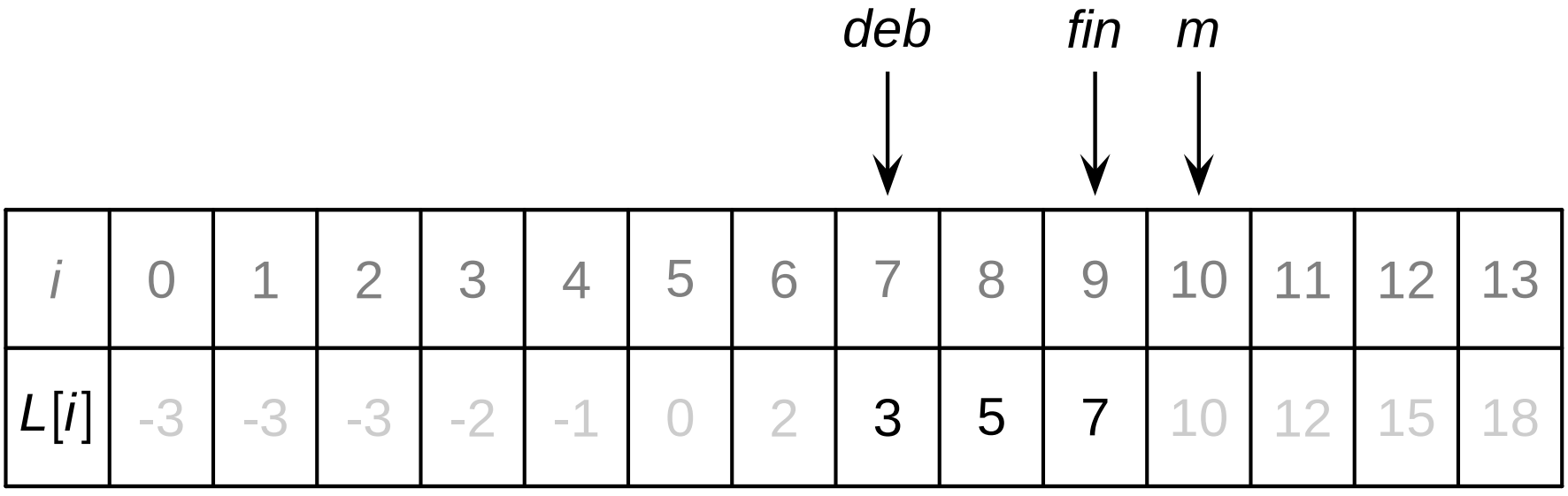
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$L[i]$	-3	-3	-3	-2	-1	0	2	3	5	7	10	12	15	18

Principe de la méthode : recherche de 5



i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$L[i]$	-3	-3	-3	-2	-1	0	2	3	5	7	10	12	15	18

Principe de la méthode : recherche de 5



i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$L[i]$	-3	-3	-3	-2	-1	0	2	3	5	7	10	12	15	18

Principe de la méthode : recherche de 5

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$L[i]$	-3	-3	-3	-2	-1	0	2	3	5	7	10	12	15	18

$L[m] = x$: on peut s'arrêter et renvoyer vrai.

On va illustrer le principe lorsque x n'est pas dans le tableau.

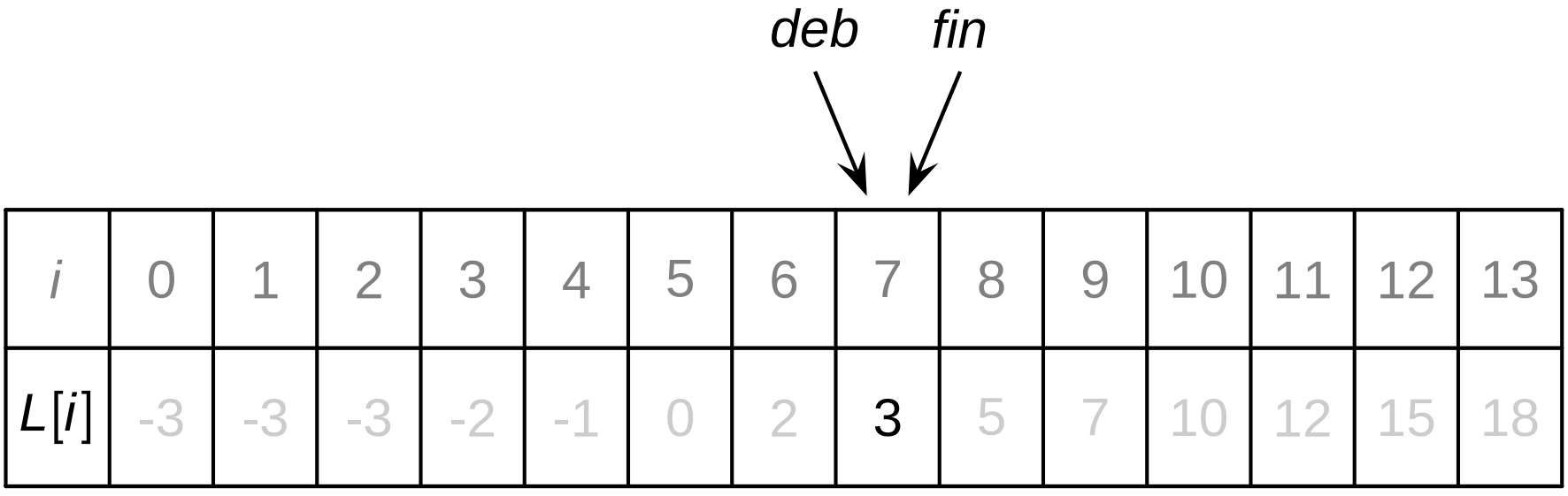
On va illustrer le principe lorsque x n'est pas dans le tableau.
On va prendre $x = 4$ les premières étapes sont les mêmes :

Principe de la méthode : recherche de 4

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$L[i]$	-3	-3	-3	-2	-1	0	2	3	5	7	10	12	15	18

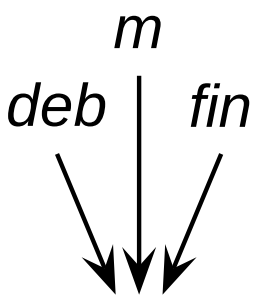
$L[m] > 4$: il faut chercher à gauche de m.

Principe de la méthode : recherche de 4



i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$L[i]$	-3	-3	-3	-2	-1	0	2	3	5	7	10	12	15	18

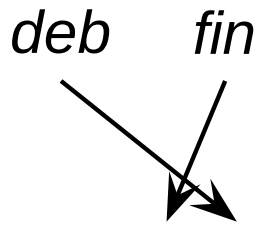
Principe de la méthode : recherche de 4



i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$L[i]$	-3	-3	-3	-2	-1	0	2	3	5	7	10	12	15	18

$L[m] = 3 < x = 4$: il faut chercher à droite de m : $deb = m + 1$.

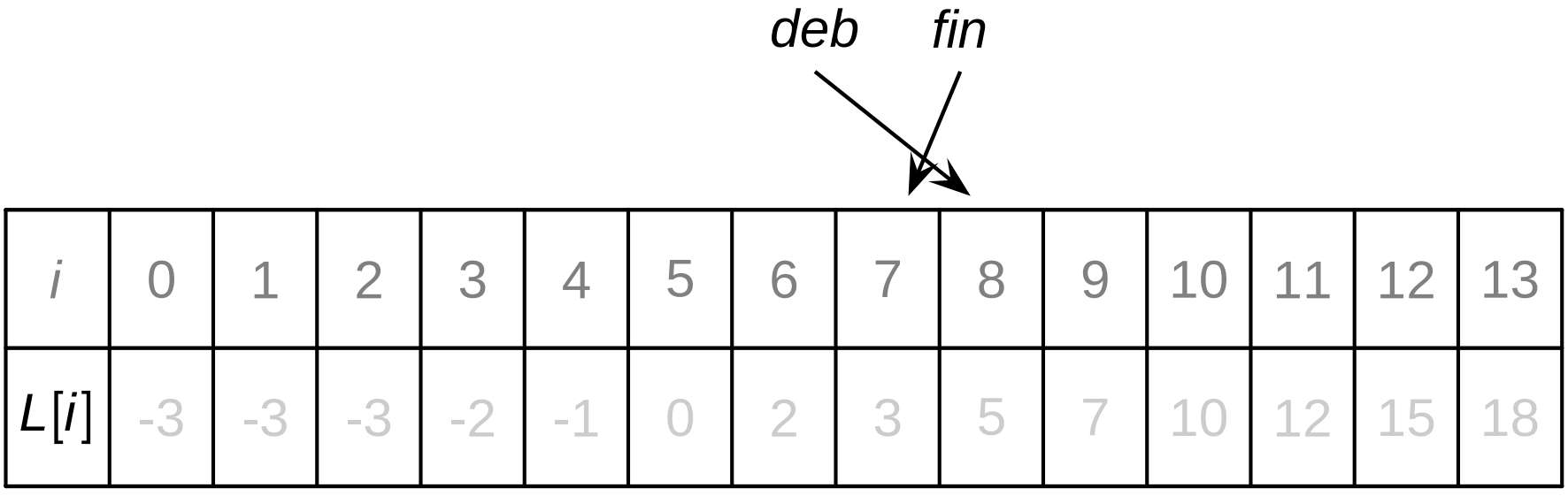
Principe de la méthode : recherche de 4



i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$L[i]$	-3	-3	-3	-2	-1	0	2	3	5	7	10	12	15	18

Cela n'a plus de sens de chercher x entre les indices *fin* et *deb* avec $deb \leqslant fin$ puisque dans notre cas $deb > fin$.

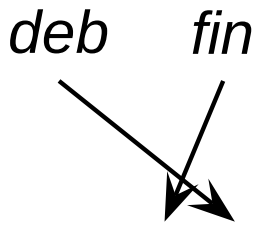
Principe de la méthode : recherche de 4



i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$L[i]$	-3	-3	-3	-2	-1	0	2	3	5	7	10	12	15	18

Voilà notre condition d'arrêt.

Principe de la méthode : recherche de 4



i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$L[i]$	-3	-3	-3	-2	-1	0	2	3	5	7	10	12	15	18

Voilà notre condition d'arrêt. On continue de chercher tant que $deb \leqslant fin$

Algorithme

Algorithme

Avant d’écrire un programme python,

Algorithme

Avant d'écrire un programme python, c'est une bonne idée d'écrire un algorithme avec des mots.

Algorithme

Avant d'écrire un programme python, c'est une bonne idée d'écrire un algorithme avec des mots.

...

Algorithme

Avant d'écrire un programme python, c'est une bonne idée d'écrire un algorithme avec des mots.
. . . À vous de jouer.

- entrée de l'algorithme : L, x ;

- entrée de l'algorithme : L, x ;
- on suppose que l'utilisateur est intelligent et que la liste est bien trié ;

- entrée de l'algorithme : L, x ;
- on suppose que l'utilisateur est intelligent et que la liste est bien trié ;
- tant que l'on a pas trouvé le nombre et que $deb \leqslant fin$:

- entrée de l'algorithme : L, x ;
- on suppose que l'utilisateur est intelligent et que la liste est bien trié ;
- tant que l'on a pas trouvé le nombre et que $deb \leqslant fin$:
  - m est le milieu de deb et fin, puis

- entrée de l'algorithme : L, x ;
- on suppose que l'utilisateur est intelligent et que la liste est bien trié ;
- tant que l'on a pas trouvé le nombre et que $deb \leqslant fin$:
  - m est le milieu de deb et fin, puis
    - si $L[m]$ est égal à x, alors renvoyer m (ou vrai) ;

- entrée de l'algorithme : L, x ;
- on suppose que l'utilisateur est intelligent et que la liste est bien trié ;
- tant que l'on a pas trouvé le nombre et que $deb \leqslant fin$:
  - m est le milieu de deb et fin, puis
    - si $L[m]$ est égal à x, alors renvoyer m (ou vrai) ;
    - sinon si $L[m] < x$ c'est que x est plus à droite : $deb = m + 1$;

- entrée de l'algorithme : L, x ;
- on suppose que l'utilisateur est intelligent et que la liste est bien trié ;
- tant que l'on a pas trouvé le nombre et que $deb \leqslant fin$:
  - m est le milieu de deb et fin, puis
    - si $L[m]$ est égal à x, alors renvoyer m (ou vrai) ;
    - sinon si $L[m] < x$ c'est que x est plus à droite : $deb = m + 1$;
    - sinon c'est que $L[m] > x$ et donc que x est plus à gauche : $fin = m - 1$;

- entrée de l'algorithme : L, x ;
- on suppose que l'utilisateur est intelligent et que la liste est bien trié ;
- tant que l'on a pas trouvé le nombre et que $deb \leqslant fin$:
  - m est le milieu de deb et fin, puis
    - si $L[m]$ est égal à x, alors renvoyer m (ou vrai) ;
    - sinon si $L[m] < x$ c'est que x est plus à droite : $deb = m + 1$;
    - sinon c'est que $L[m] > x$ et donc que x est plus à gauche : $fin = m - 1$;
- Je suis sorti de la boucle sans renvoyer m (ou vrai) : il n'est pas dedans, renvoyer None (ou faux).