Elements d'Algorithmique
CMTD5: Recherche dichotomique

Daniela Petrişan
Université de Paris, IRIF

Trouver une valeur dans un tableau

Entrée : un tableau `T` et un élément x

Sortie : i tel que `T[i]` = x ou `NonTrouvé`

```
1: fonction RECHERCHESÉQUENTIELLE (T, x)
2:     n ← longueur de T
3:     pour i ← 0 à n − 1 faire
4:         si T[i]=x alors
5:             retourne i
6:     retourne NonTrouvé
```

Trouver une valeur dans un tableau

Entrée : un tableau `T` et un élément x

Sortie : i tel que `T[i]` = x ou `NonTrouvé`

```
1: fonction RECHERCHESÉQUENTIELLE (T, x)
2:     n ← longueur de T
3:     pour i ← 0 à n − 1 faire
4:         si T[i]=x alors
5:             retourne i
6:     retourne NonTrouvé
```

Complexité dans le pire des cas : O(n)

Trouver une valeur dans un tableau

Entrée : un tableau `T` et un élément x

Sortie : i tel que `T[i]` = x ou `NonTrouvé`

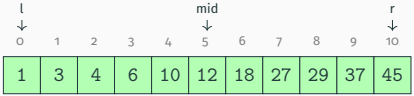
```
1: fonction RECHERCHESÉQUENTIELLE (T, x)
2:     n ← longueur de T
3:     pour i ← 0 à n − 1 faire
4:         si T[i]=x alors
5:             retourne i
6:     retourne NonTrouvé
```

Complexité dans le pire des cas : O(n)

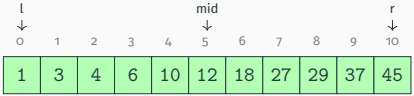
Nous pouvons faire mieux si le tableau est trié !

Recherche dichotomique: Principe

Exemple. Rechercher l'élément 27 dans le tableau suivant :



Exemple. Rechercher l'élément 27 dans le tableau suivant :

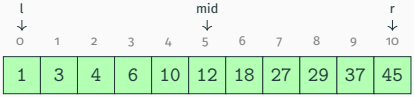


$\texttt{l} = 0, \texttt{r} = 10$

$\texttt{mid} = (\texttt{l} + \texttt{r})/2 = 5$

$27 > 12$

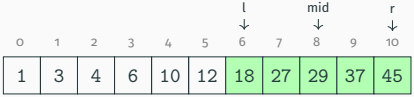
Exemple. Rechercher l'élément 27 dans le tableau suivant :



$l = 0, r = 10$

$mid = (l + r)/2 = 5$

$27 > 12$

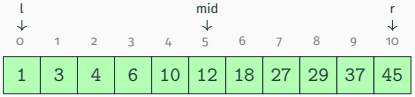


$l = 6, r = 10$

$mid = (l + r)/2 = 8$

$27 < 29$

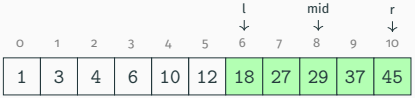
Exemple. Rechercher l'élément 27 dans le tableau suivant :



$l = 0, r = 10$

$mid = (l + r)/2 = 5$

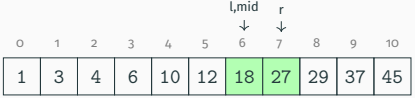
$27 > 12$



$l = 6, r = 10$

$mid = (l + r)/2 = 8$

$27 < 29$

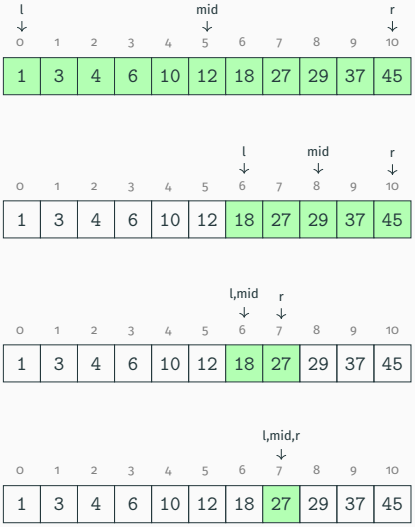


$l = 6, r = 7$

$mid = (l + r)/2 = 6$

$27 > 18$

Exemple. Rechercher l'élément 27 dans le tableau suivant :



$\texttt{l} = 0, \texttt{r} = 10$

$\texttt{mid} = (\texttt{l} + \texttt{r})/2 = 5$

$27 > 12$

$\texttt{l} = 6, \texttt{r} = 10$

$\texttt{mid} = (\texttt{l} + \texttt{r})/2 = 8$

$27 < 29$

$\texttt{l} = 6, \texttt{r} = 7$

$\texttt{mid} = (\texttt{l} + \texttt{r})/2 = 6$

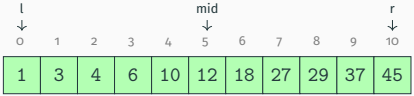
$27 > 18$

$\texttt{l} = 7, \texttt{r} = 7$

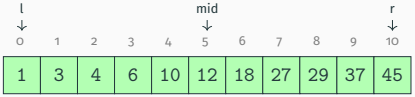
$\texttt{mid} = (\texttt{l} + \texttt{r})/2 = 7$

$27 = 27$, donc `renvoie 7`

Exemple. Rechercher l'élément 25 dans le tableau suivant :



Exemple. Rechercher l'élément 25 dans le tableau suivant :

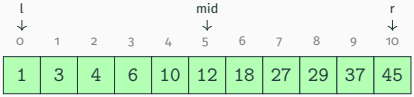


$\texttt{l} = 0, \texttt{r} = 10$

$\texttt{mid} = (\texttt{l} + \texttt{r})/2 = 5$

$25 > 12$

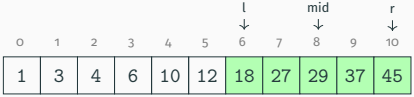
Exemple. Rechercher l'élément 25 dans le tableau suivant :



$l = 0, r = 10$

$mid = (l + r)/2 = 5$

$25 > 12$

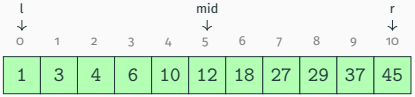


$l = 6, r = 10$

$mid = (l + r)/2 = 8$

$25 < 29$

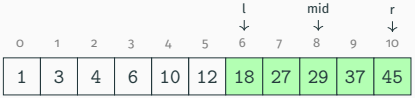
Exemple. Rechercher l'élément 25 dans le tableau suivant :



$l = 0, r = 10$

$mid = (l + r)/2 = 5$

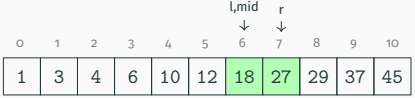
$25 > 12$



$l = 6, r = 10$

$mid = (l + r)/2 = 8$

$25 < 29$

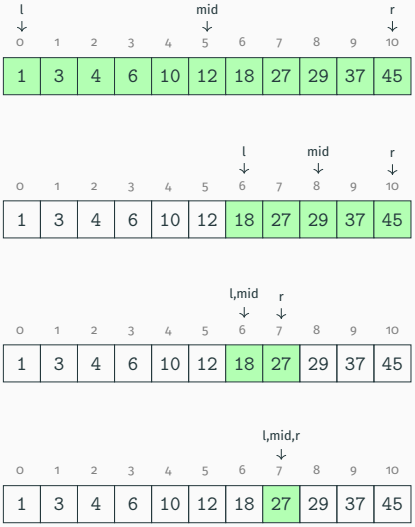


$l = 6, r = 7$

$mid = (l + r)/2 = 6$

$25 > 18$

Exemple. Rechercher l'élément 25 dans le tableau suivant :



$\texttt{l} = 0, \texttt{r} = 10$

$\texttt{mid} = (\texttt{l} + \texttt{r})/2 = 5$

$25 > 12$

$\texttt{l} = 6, \texttt{r} = 10$

$\texttt{mid} = (\texttt{l} + \texttt{r})/2 = 8$

$25 < 29$

$\texttt{l} = 6, \texttt{r} = 7$

$\texttt{mid} = (\texttt{l} + \texttt{r})/2 = 6$

$25 > 18$

$\texttt{l} = 7, \texttt{r} = 7$

$\texttt{mid} = (\texttt{l} + \texttt{r})/2 = 7$

$25 < 27$, donc $\texttt{r} := 6 < \texttt{l}$, renvoie NonTrouvé

Recherche dichotomique: Principe

- La recherche dichotomique est un algorithme permettant de trouver une valeur dans un tableau trié. Cet algorithme s'appuie sur l'approche «diviser pour régner».

Recherche dichotomique: Principe

- La recherche dichotomique est un algorithme permettant de trouver une valeur dans un tableau trié. Cet algorithme s'appuie sur l'approche «diviser pour régner».
- **Idée clé :** Si le tableau est trié, on peut comparer la valeur recherchée avec le milieu du tableau et on peut ignorer une moitié du tableau pour le reste de l'exécution.

Recherche dichotomique: Principe

- La recherche dichotomique est un algorithme permettant de trouver une valeur dans un tableau trié. Cet algorithme s'appuie sur l'approche «diviser pour régner».
- **Idée clé :** Si le tableau est trié, on peut comparer la valeur recherchée avec le milieu du tableau et on peut ignorer une moitié du tableau pour le reste de l'exécution.
- L'algorithme de recherche dichotomique répète cette procédure, en divisant à chaque étape par deux la taille du tableau restant.

Recherche dichotomique: Principe

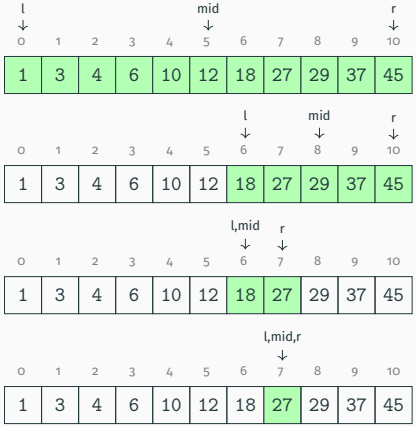
- La recherche dichotomique est un algorithme permettant de trouver une valeur dans un tableau trié. Cet algorithme s'appuie sur l'approche «diviser pour régner».
- **Idée clé :** Si le tableau est trié, on peut comparer la valeur recherchée avec le milieu du tableau et on peut ignorer une moitié du tableau pour le reste de l'exécution.
- L'algorithme de recherche dichotomique répète cette procédure, en divisant à chaque étape par deux la taille du tableau restant.
- Soit la valeur est trouvée $\Rightarrow$ retourner l'index;
  soit un tableau vide est obtenu $\Rightarrow$ retourner NonTrouvé.

Recherche dichotomique: Pseudo-code recursif

Entrée : un tableau trié `T` et un élément `x`
Sortie : `i` tel que `T[i]` = `x` ou `NonTrouvé`

1: **fonction** RECHDICHO(T, x)
2: $n \leftarrow$ longueur de T
3: **retourne** RECHDICHOREC($T, 0, n-1, x$)

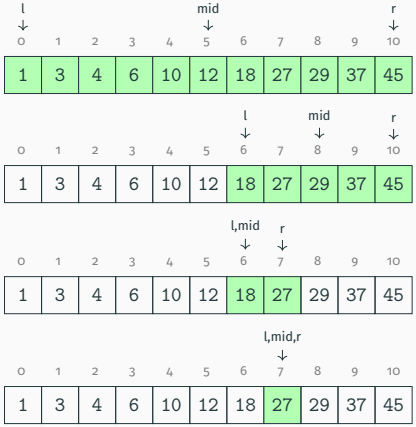
Exemple. Rechercher l'élément 27



Entrée : un tableau trié `T` et un élément `x`
Sortie : `i` tel que `T[i]` = `x` ou `NonTrouvé`

1: **fonction** RECHDICHO(`T`, `x`)
2: `n` ← longueur de `T`
3: **retourne** RECHDICHOREC(`T`, 0, `n` − 1, `x`)
4: **fonction** RECHDICHOREC(`T`, `l`, `r`, `x`)

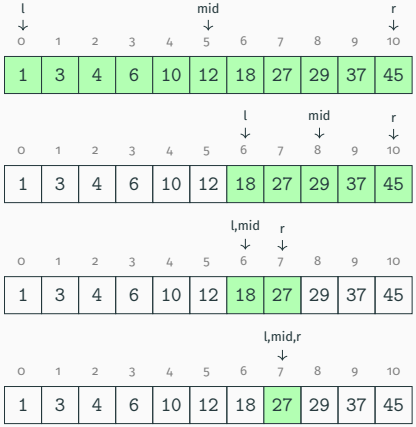
Exemple. Rechercher l'élément 27



Entrée : un tableau trié `T` et un élément `x`
Sortie : `i` tel que `T[i]` = `x` ou `NonTrouvé`

1: **fonction** RECHDICHO(`T`, `x`)
2: `n` ← longueur de `T`
3: **retourne** RECHDICHOREC(`T`, 0, `n` − 1, `x`)
4: **fonction** RECHDICHOREC(`T`, `l`, `r`, `x`)
5: **si** `r` < `l` **alors retourne** `NonTrouvé`

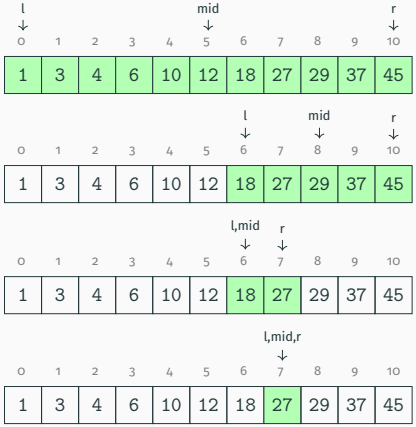
Exemple. Rechercher l'élément 27



Entrée : un tableau trié `T` et un élément `x`
Sortie : `i` tel que `T[i]` = `x` ou `NonTrouvé`

1: **fonction** RECHDICHO(`T`, `x`)
2: `n` ← longueur de `T`
3: **retourne** RECHDICHOREC(`T`, 0, `n` − 1, `x`)
4: **fonction** RECHDICHOREC(`T`, `l`, `r`, `x`)
5: **si** `r` < `l` **alors retourne** `NonTrouvé`
6: `mid` ← (`l` + `r`)/2

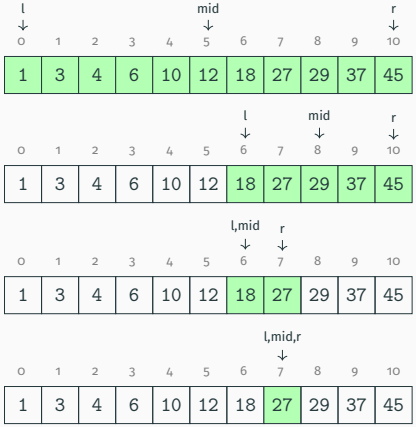
Exemple. Rechercher l'élément 27



Entrée : un tableau trié `T` et un élément `x`
Sortie : `i` tel que `T[i]` = `x` ou `NonTrouvé`

1: **fonction** RECHDICHO(T, x)
2: $n \leftarrow$ longueur de T
3: **retourne** RECHDICHOREC($T, 0, n-1, x$)
4: **fonction** RECHDICHOREC(T, l, r, x)
5: **si** $r < l$ **alors retourne** `NonTrouvé`
6: $mid \leftarrow (l + r)/2$
7: **si** `T[mid]=x` **alors**

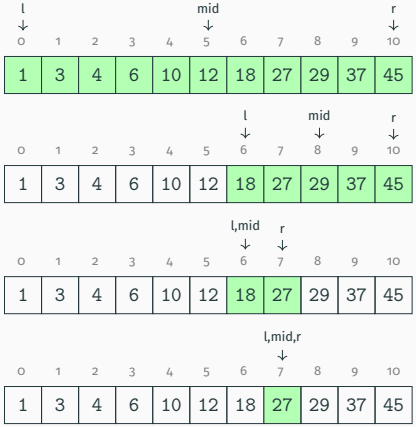
Exemple. Rechercher l'élément 27



Entrée : un tableau trié `T` et un élément `x`
Sortie : `i` tel que `T[i] = x` ou `NonTrouvé`

1: **fonction** RECHDICHO(T, x)
2: $n \leftarrow$ longueur de T
3: **retourne** RECHDICHOREC($T, 0, n-1, x$)
4: **fonction** RECHDICHOREC(T, l, r, x)
5: **si** $r < l$ **alors retourne** `NonTrouvé`
6: $mid \leftarrow (l + r)/2$
7: **si** `T[mid]=x` **alors retourne** `mid`

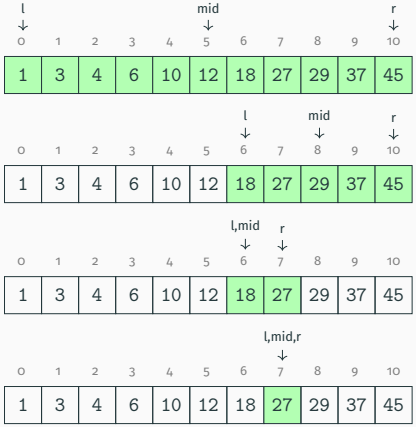
Exemple. Rechercher l'élément 27



Entrée : un tableau trié `T` et un élément `x`
Sortie : `i` tel que `T[i]` = `x` ou `NonTrouvé`

```
1:  fonction RechDicho(T, x)
2:      n ← longueur de T
3:      retourne RechDichoRec(T, 0, n − 1, x)
4:  fonction RechDichoRec(T, l, r, x)
5:      si r < l alors retourne NonTrouvé
6:      mid ← (l + r)/2
7:      si T[mid]=x alors retourne mid
8:      sinon
9:          si T[mid]<x alors
10:             retourne RechDichoRec(T, mid + 1, r, x)
```

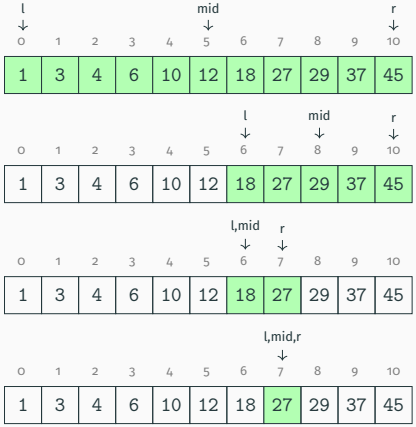
Exemple. Rechercher l'élément 27



Entrée : un tableau trié `T` et un élément `x`
Sortie : `i` tel que `T[i] = x` ou `NonTrouvé`

```
1: fonction RechDicho(T, x)
2:     n ← longueur de T
3:     retourne RechDichoRec(T, 0, n − 1, x)
4: fonction RechDichoRec(T, l, r, x)
5:     si r < l alors retourne NonTrouvé
6:     mid ← (l + r)/2
7:     si T[mid]=x alors retourne mid
8:     sinon
9:         si T[mid]<x alors
10:            retourne RechDichoRec(T, mid + 1, r, x)
11:        sinon
12:            retourne RechDichoRec(T, l, mid − 1, x)
```

Exemple. Rechercher l'élément 27

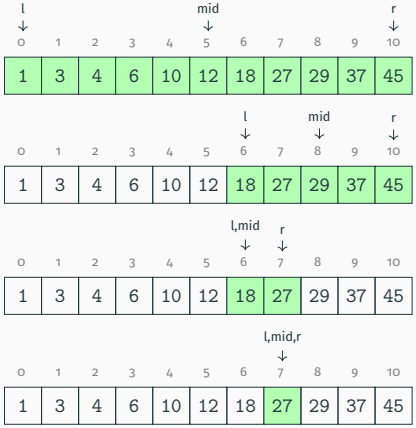


Recherche dichotomique: Pseudo-code itératif

Entrée : un tableau trié `T` et un élément `x`
Sortie : `i` tel que `T[i]` = `x` ou `NonTrouvé`

1: **fonction** RECHDICHOITÉRATIVE(`T`, `x`)
2: `n` ← longueur de `T`, `l` ← 0, `r` ← `n` − 1
3: **tant que** `l` ≤ `r` **faire**

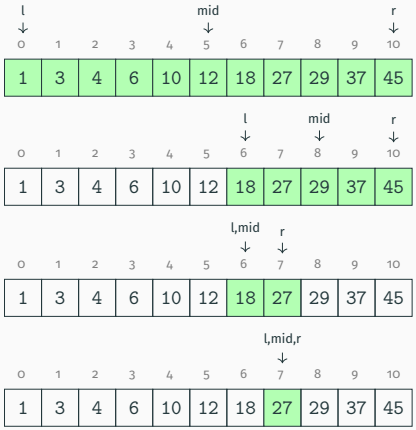
Exemple. Rechercher l'élément 27



Entrée : un tableau trié `T` et un élément `x`
Sortie : `i` tel que `T[i] = x` ou `NonTrouvé`

1: **fonction** RECHDICHOITÉRATIVE(`T`, `x`)
2: `n` ← longueur de `T`, `l` ← 0, `r` ← `n` − 1
3: **tant que** `l` ≤ `r` **faire**
4: `mid` ← (`l` + `r`)/2
5: **si** `T[mid]=x` **alors**
6: **retourne** `mid`

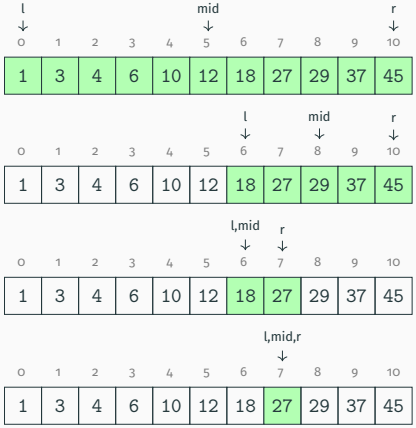
Exemple. Rechercher l'élément 27



Entrée : un tableau trié `T` et un élément `x`
Sortie : `i` tel que `T[i] = x` ou `NonTrouvé`

```
1: fonction RechDichoItérative(T, x)
2:     n ← longueur de T, l ← 0, r ← n − 1
3:     tant que l ≤ r faire
4:         mid← (l + r)/2
5:         si T[mid]=x alors
6:             retourne mid
7:         sinon
8:             si T[mid]<x alors
9:                 l ←
```

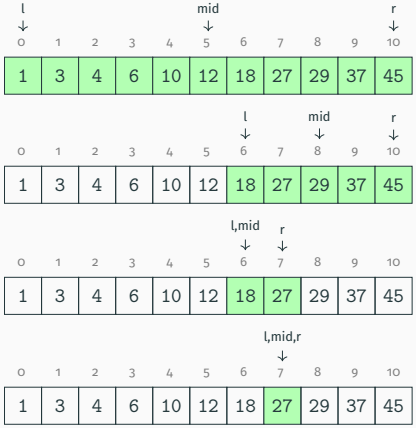
Exemple. Rechercher l'élément 27



Entrée : un tableau trié `T` et un élément `x`
Sortie : `i` tel que `T[i] = x` ou `NonTrouvé`

```
1: fonction RechDichoItérative(T, x)
2:     n ← longueur de T, l ← 0, r ← n − 1
3:     tant que l ≤ r faire
4:         mid← (l + r)/2
5:         si T[mid]=x alors
6:             retourne mid
7:         sinon
8:             si T[mid]<x alors
9:                 l ← mid + 1
```

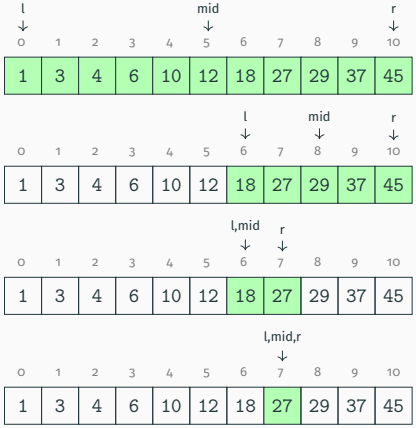
Exemple. Rechercher l'élément 27



Entrée : un tableau trié `T` et un élément `x`
Sortie : `i` tel que `T[i]` = `x` ou `NonTrouvé`

```
1:  fonction RechDichoItérative(T, x)
2:      n ← longueur de T, l ← 0, r ← n − 1
3:      tant que l ≤ r faire
4:          mid← (l + r)/2
5:          si T[mid]=x alors
6:              retourne mid
7:          sinon
8:              si T[mid]<x alors
9:                  l ← mid + 1
10:             sinon
11:                 r ←
```

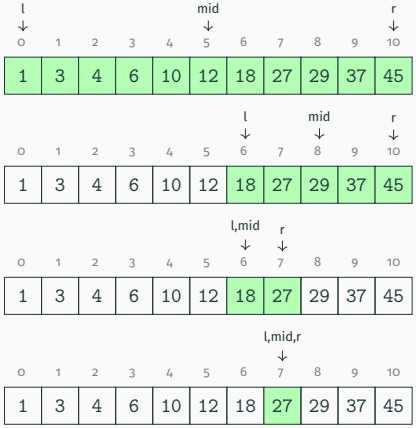
Exemple. Rechercher l'élément 27



Entrée : un tableau trié `T` et un élément `x`
Sortie : `i` tel que `T[i] = x` ou `NonTrouvé`

```
1:  fonction RechDichoItérative(T, x)
2:      n ← longueur de T, l ← 0, r ← n − 1
3:      tant que l ≤ r faire
4:          mid← (l + r)/2
5:          si T[mid]=x alors
6:              retourne mid
7:          sinon
8:              si T[mid]<x alors
9:                  l ← mid + 1
10:             sinon
11:                 r ← mid − 1
```

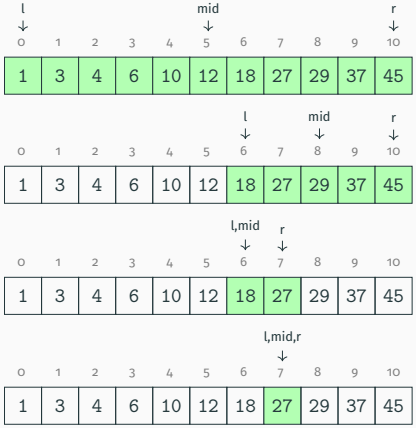
Exemple. Rechercher l'élément 27



Entrée : un tableau trié `T` et un élément `x`
Sortie : `i` tel que `T[i] = x` ou `NonTrouvé`

```
1:  fonction RechDichoItérative(T, x)
2:      n ← longueur de T, l ← 0, r ← n − 1
3:      tant que l ≤ r faire
4:          mid← (l + r)/2
5:          si T[mid]=x alors
6:              retourne mid
7:          sinon
8:              si T[mid]<x alors
9:                  l ← mid + 1
10:             sinon
11:                 r ← mid − 1
12:     retourne NonTrouvé
```

Exemple. Rechercher l'élément 27



Recherche dichotomique: Correction

RECHDICHOREC(T, l, r, x) retourne `i` tel que $l \leq i \leq r$ et `T[i] = x` ou `NonTrouvé` si `x` n'est pas dans le tableau dans l'intervalle `T[l,...,r]`.

RechDichoRec(`T`, `l`, `r`, `x`) retourne `i` tel que `l` $\leq$ `i` $\leq$ `r` et `T[i] = x` ou `NonTrouvé` si `x` n'est pas dans le tableau dans l'intervalle `T[l,...,r]`.

Preuve. Récurrence sur la taille de l'intervalle `r-l+1`.

RECHDICHOREC(T, l, r, x) retourne `i` tel que $l \leq i \leq r$ et `T[i] = x` ou `NonTrouvé` si `x` n'est pas dans le tableau dans l'intervalle `T[l,...,r]`.

Preuve. Récurrence sur la taille de l'intervalle `r-l+1`.

Cas de base: `r-l+1=0`. Dans ce cas `r<l`, c'est-à-dire la section du tableau est vide, donc `x` n'est pas là. L'algorithme retourne correctement `NonTrouvé` dans ce cas.

RECHDICHOREC(T, l, r, x) retourne i tel que $l \leq i \leq r$ et $T[i] = x$ ou NonTrouvé si x n'est pas dans le tableau dans l'intervalle $T[l, \ldots, r]$.

Preuve. Récurrence sur la taille de l'intervalle $r-l+1$.

Cas de base: $r-l+1=0$. Dans ce cas $r<l$, c'est-à-dire la section du tableau est vide, donc x n'est pas là. L'algorithme retourne correctement NonTrouvé dans ce cas.

Hérédité : Nous supposons que la propriété est vraie lorsque $r - l + 1 < n$, où $n \geq 1$, et nous la prouvons pour $r - l + 1 = n$.

RECHDICHOREC(T, l, r, x) retourne i tel que $l \leq i \leq r$ et $T[i] = x$ ou NonTrouvé si x n'est pas dans le tableau dans l'intervalle $T[l, \ldots, r]$.

Preuve. Récurrence sur la taille de l'intervalle $r-l+1$.

Cas de base: $r-l+1=0$. Dans ce cas $r<l$, c'est-à-dire la section du tableau est vide, donc x n'est pas là. L'algorithme retourne correctement NonTrouvé dans ce cas.

Hérédité : Nous supposons que la propriété est vraie lorsque $r - l + 1 < n$, où $n \geq 1$, et nous la prouvons pour $r - l + 1 = n$. Soit $mid = (l + r)/2$; il y a trois cas :

RechDichoRec(T, l, r, x) retourne i tel que l ≤ i ≤ r et T[i] = x ou NonTrouvé si x n'est pas dans le tableau dans l'intervalle T[l,...,r].

Preuve. Récurrence sur la taille de l'intervalle r-l+1.

Cas de base: r-l+1=0. Dans ce cas r<l, c'est-à-dire la section du tableau est vide, donc x n'est pas là. L'algorithme retourne correctement NonTrouvé dans ce cas.

Hérédité : Nous supposons que la propriété est vraie lorsque $r - l + 1 < n$, où $n \geq 1$, et nous la prouvons pour $r - l + 1 = n$. Soit $mid = (l + r)/2$; il y a trois cas :

- T[mid]=x. Dans ce cas RechDichoRec(T, l, r, x) retourne mid, ce qui est correct puisque x se trouve dans le tableau à la position mid.

RECHDICHOREC(T, l, r, x) retourne i tel que l ≤ i ≤ r et T[i] = x ou NonTrouvé si x n'est pas dans le tableau dans l'intervalle T[l, . . . , r].

Preuve. Récurrence sur la taille de l'intervalle r-l+1.

Cas de base: r-l+1=0. Dans ce cas r<l, c'est-à-dire la section du tableau est vide, donc x n'est pas là. L'algorithme retourne correctement NonTrouvé dans ce cas.

Hérédité : Nous supposons que la propriété est vraie lorsque r − l + 1 < n, où n ≥ 1, et nous la prouvons pour r − l + 1 = n. Soit mid = (l + r)/2; il y a trois cas :

- T[mid]=x. Dans ce cas RECHDICHOREC(T, l, r, x) retourne mid, ce qui est correct puisque x se trouve dans le tableau à la position mid.
- T[mid]<x. Dans ce cas l'algorithme renvoie RECHDICHOREC(T, mid + 1, r, x). Vu que le tableau *T* est trié, s'il existe l ≤ i ≤ r tel que T[i] = x, alors mid + 1 ≤ i ≤ r; on a aussi r − (mid + 1) + 1 < n. En utilisant l'hypothèse de récurrence, le résultat renvoyé par RECHDICHOREC(T, mid + 1, r, x) est correct.

RECHDICHOREC(T, l, r, x) retourne i tel que l ≤ i ≤ r et T[i] = x ou NonTrouvé si x n'est pas dans le tableau dans l'intervalle T[l, . . . , r].

Preuve. Récurrence sur la taille de l'intervalle r-l+1.

Cas de base: r-l+1=0. Dans ce cas r<l, c'est-à-dire la section du tableau est vide, donc x n'est pas là. L'algorithme retourne correctement NonTrouvé dans ce cas.

Hérédité : Nous supposons que la propriété est vraie lorsque r − l + 1 < n, où n ≥ 1, et nous la prouvons pour r − l + 1 = n. Soit mid = (l + r)/2; il y a trois cas :

- T[mid]=x. Dans ce cas RECHDICHOREC(T, l, r, x) retourne mid, ce qui est correct puisque x se trouve dans le tableau à la position mid.
- T[mid]<x. Dans ce cas l'algorithme renvoie RECHDICHOREC(T, mid + 1, r, x). Vu que le tableau *T* est trié, s'il existe l ≤ i ≤ r tel que T[i] = x, alors mid + 1 ≤ i ≤ r; on a aussi r − (mid + 1) + 1 < n. En utilisant l'hypothèse de récurrence, le résultat renvoyé par RECHDICHOREC(T, mid + 1, r, x) est correct.
- T[mid]>x. Similaire.

Recherche dichotomique: Terminaison et Complexité

→ L'algorithme s'exécute tant que $l \leq r$. Or, après chaque exécution de la boucle où l'algorithme n'est pas encore fini, soit $l \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor + 1$ (l augmente au moins d'un), soit $r \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor - 1$ (r diminue au moins d'un).

→ L'algorithme s'exécute tant que $l \leq r$. Or, après chaque exécution de la boucle où l'algorithme n'est pas encore fini, soit $l \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor + 1$ (l augmente au moins d'un), soit $r \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor - 1$ (r diminue au moins d'un).

→ L'algorithme s'exécute tant que $l \leq r$. Or, après chaque exécution de la boucle où l'algorithme n'est pas encore fini, soit $l \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor + 1$ (l augmente au moins d'un), soit $r \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor - 1$ (r diminue au moins d'un).

→ L'algorithme s'exécute tant que $l \leq r$. Or, après chaque exécution de la boucle où l'algorithme n'est pas encore fini, soit $l \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor + 1$ (l augmente au moins d'un), soit $r \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor - 1$ (r diminue au moins d'un).

→ Combien de fois la boucle s'exécute-t-elle au pire des cas (ex. quand $x \notin T[l \dots r]$) ?

→ L'algorithme s'exécute tant que $l \leq r$. Or, après chaque exécution de la boucle où l'algorithme n'est pas encore fini, soit $l \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor + 1$ (l augmente au moins d'un), soit $r \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor - 1$ (r diminue au moins d'un).

→ Combien de fois la boucle s'exécute-t-elle au pire des cas (ex. quand $x \notin T[l \dots r]$) ? Après chaque exécution (et si l'algorithme toujours n'a pas fini), la taille de l'intervalle `t = r-l+1` est réduite de moitié :

→ L'algorithme s'exécute tant que $l \leq r$. Or, après chaque exécution de la boucle où l'algorithme n'est pas encore fini, soit $l \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor + 1$ (l augmente au moins d'un), soit $r \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor - 1$ (r diminue au moins d'un).

→ Combien de fois la boucle s'exécute-t-elle au pire des cas (ex. quand $x \notin T[l \dots r]$) ?
Après chaque exécution (et si l'algorithme toujours n'a pas fini), la taille de l'intervalle `t = r-l+1` est réduite de moitié :
Après 1 exécution,

→ L'algorithme s'exécute tant que $l \leq r$. Or, après chaque exécution de la boucle où l'algorithme n'est pas encore fini, soit $l \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor + 1$ (l augmente au moins d'un), soit $r \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor - 1$ (r diminue au moins d'un).

→ Combien de fois la boucle s'exécute-t-elle au pire des cas (ex. quand $x \notin T[l \dots r]$) ?
Après chaque exécution (et si l'algorithme toujours n'a pas fini), la taille de l'intervalle `t = r-l+1` est réduite de moitié :
Après 1 exécution,

- Soit $r - l + 1 \leftarrow r - \left(\left\lfloor \frac{l+r}{2} \right\rfloor + 1 \right) + 1 \leq \left\lfloor \frac{r-l+1}{2} \right\rfloor = \left\lfloor \frac{t}{2} \right\rfloor$.

→ L'algorithme s'exécute tant que $l \leq r$. Or, après chaque exécution de la boucle où l'algorithme n'est pas encore fini, soit $l \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor + 1$ (l augmente au moins d'un), soit $r \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor - 1$ (r diminue au moins d'un).

→ Combien de fois la boucle s'exécute-t-elle au pire des cas (ex. quand $x \notin T[l \dots r]$) ?
Après chaque exécution (et si l'algorithme toujours n'a pas fini), la taille de l'intervalle `t = r-l+1` est réduite de moitié :
Après 1 exécution,

- Soit $r - l + 1 \leftarrow r - \left(\left\lfloor \frac{l+r}{2} \right\rfloor + 1 \right) + 1 \leq \left\lfloor \frac{r-l+1}{2} \right\rfloor = \left\lfloor \frac{t}{2} \right\rfloor$.
- Soit $r - l + 1 \leftarrow \left(\left\lfloor \frac{l+r}{2} \right\rfloor - 1 \right) - l + 1 \leq \left\lfloor \frac{r-l+1}{2} \right\rfloor = \left\lfloor \frac{t}{2} \right\rfloor$.

→ L'algorithme s'exécute tant que $l \leq r$. Or, après chaque exécution de la boucle où l'algorithme n'est pas encore fini, soit $l \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor + 1$ (l augmente au moins d'un), soit $r \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor - 1$ (r diminue au moins d'un).

→ Combien de fois la boucle s'exécute-t-elle au pire des cas (ex. quand $x \notin T[l \dots r]$) ?
Après chaque exécution (et si l'algorithme toujours n'a pas fini), la taille de l'intervalle `t = r-l+1` est réduite de moitié :
Après 1 exécution, $r - l + 1 \leq \lfloor t/2 \rfloor$.

→ L'algorithme s'exécute tant que $l \leq r$. Or, après chaque exécution de la boucle où l'algorithme n'est pas encore fini, soit $l \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor + 1$ (l augmente au moins d'un), soit $r \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor - 1$ (r diminue au moins d'un).

→ Combien de fois la boucle s'exécute-t-elle au pire des cas (ex. quand $x \notin T[l \dots r]$) ?
Après chaque exécution (et si l'algorithme toujours n'a pas fini), la taille de l'intervalle `t = r-l+1` est réduite de moitié :
Après 1 exécution, $r - l + 1 \leq \lfloor t/2 \rfloor$.
Après 2 exécutions, $r - l + 1 \leq \lfloor t/2^2 \rfloor$.

→ L'algorithme s'exécute tant que $l \leq r$. Or, après chaque exécution de la boucle où l'algorithme n'est pas encore fini, soit $l \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor + 1$ (l augmente au moins d'un), soit $r \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor - 1$ (r diminue au moins d'un).

→ Combien de fois la boucle s'exécute-t-elle au pire des cas (ex. quand $x \notin T[l \dots r]$) ?
Après chaque exécution (et si l'algorithme toujours n'a pas fini), la taille de l'intervalle `t = r-l+1` est réduite de moitié :
Après 1 exécution, $r - l + 1 \leq \lfloor t/2 \rfloor$.
Après 2 exécutions, $r - l + 1 \leq \lfloor t/2^2 \rfloor$.
Après 3 exécutions, $r - l + 1 \leq \lfloor t/2^3 \rfloor$.

→ L'algorithme s'exécute tant que $l \leq r$. Or, après chaque exécution de la boucle où l'algorithme n'est pas encore fini, soit $l \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor + 1$ (l augmente au moins d'un), soit $r \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor - 1$ (r diminue au moins d'un).

→ Combien de fois la boucle s'exécute-t-elle au pire des cas (ex. quand $x \notin T[l \dots r]$) ?
Après chaque exécution (et si l'algorithme toujours n'a pas fini), la taille de l'intervalle `t = r-l+1` est réduite de moitié :
Après 1 exécution, $r - l + 1 \leq \lfloor t/2 \rfloor$.
Après 2 exécutions, $r - l + 1 \leq \lfloor t/2^2 \rfloor$.
Après 3 exécutions, $r - l + 1 \leq \lfloor t/2^3 \rfloor$.
…Après i exécutions, $r - l + 1 \leq \lfloor t/2^i \rfloor$.

→ L'algorithme s'exécute tant que $l \leq r$. Or, après chaque exécution de la boucle où l'algorithme n'est pas encore fini, soit $l \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor + 1$ (l augmente au moins d'un), soit $r \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor - 1$ (r diminue au moins d'un).

→ Combien de fois la boucle s'exécute-t-elle au pire des cas (ex. quand $x \notin T[l \dots r]$) ?
Après chaque exécution (et si l'algorithme toujours n'a pas fini), la taille de l'intervalle `t = r-l+1` est réduite de moitié :
Après 1 exécution, $r - l + 1 \leq \lfloor t/2 \rfloor$.
Après 2 exécutions, $r - l + 1 \leq \lfloor t/2^2 \rfloor$.
Après 3 exécutions, $r - l + 1 \leq \lfloor t/2^3 \rfloor$.
…Après i exécutions, $r - l + 1 \leq \lfloor t/2^i \rfloor$.
Alors, pour que $r - l + 1 < 1$ il faut que $t/2^i < 1 \Rightarrow i > \log t$ exécutions.

→ L'algorithme s'exécute tant que $l \leq r$. Or, après chaque exécution de la boucle où l'algorithme n'est pas encore fini, soit $l \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor + 1$ (l augmente au moins d'un), soit $r \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor - 1$ (r diminue au moins d'un).

→ Combien de fois la boucle s'exécute-t-elle au pire des cas (ex. quand $x \notin T[l \dots r]$) ?
Après chaque exécution (et si l'algorithme toujours n'a pas fini), la taille de l'intervalle `t = r-l+1` est réduite de moitié :
Après 1 exécution, $r - l + 1 \leq \lfloor t/2 \rfloor$.
Après 2 exécutions, $r - l + 1 \leq \lfloor t/2^2 \rfloor$.
Après 3 exécutions, $r - l + 1 \leq \lfloor t/2^3 \rfloor$.
…Après i exécutions, $r - l + 1 \leq \lfloor t/2^i \rfloor$.
Alors, pour que $r - l + 1 < 1$ il faut que $t/2^i < 1 \Rightarrow i > \log t$ exécutions.
Après au plus $\log t + 1$ exécutions de la boucle, l'algorithme sera fini.

→ L'algorithme s'exécute tant que $l \leq r$. Or, après chaque exécution de la boucle où l'algorithme n'est pas encore fini, soit $l \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor + 1$ (l augmente au moins d'un), soit $r \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor - 1$ (r diminue au moins d'un).

→ Combien de fois la boucle s'exécute-t-elle au pire des cas (ex. quand $x \notin T[l \dots r]$) ?
Après chaque exécution (et si l'algorithme toujours n'a pas fini), la taille de l'intervalle `t = r-l+1` est réduite de moitié :
Après 1 exécution, $r - l + 1 \leq \lfloor t/2 \rfloor$.
Après 2 exécutions, $r - l + 1 \leq \lfloor t/2^2 \rfloor$.
Après 3 exécutions, $r - l + 1 \leq \lfloor t/2^3 \rfloor$.
…Après i exécutions, $r - l + 1 \leq \lfloor t/2^i \rfloor$.
Alors, pour que $r - l + 1 < 1$ il faut que $t/2^i < 1 \Rightarrow i > \log t$ exécutions.
Après au plus $\log t + 1$ exécutions de la boucle, l'algorithme sera fini. Au pire des cas, on a deux comparaisons dans la boucle $\Rightarrow 2(\log t + 1)$ comparaisons au total.

Definition

LOGARITHME $\log_2 x$: Combien de fois faut-il diviser x par 2 pour que cela donne 1 ?

Definition
LOGARITHME $\log_2 x$: Combien de fois faut-il diviser x par 2 pour que cela donne 1 ?

Exemple. $\log_2 64$

Definition
LOGARITHME $\log_2 x$: Combien de fois faut-il diviser x par 2 pour que cela donne 1 ?

Exemple. $\log_2 64$

$$\frac{64}{2} = 32 \qquad (1)$$

Definition
LOGARITHME $\log_2 x$: Combien de fois faut-il diviser x par 2 pour que cela donne 1 ?

Exemple. $\log_2 64$

$$\frac{64}{2} = 32 \qquad (1)$$

$$\frac{32}{2} = 16 \qquad (2)$$

Definition
LOGARITHME $\log_2 x$: Combien de fois faut-il diviser x par 2 pour que cela donne 1 ?

Exemple. $\log_2 64$

$$\frac{64}{2} = 32 \qquad (1)$$

$$\frac{32}{2} = 16 \qquad (2)$$

$$\frac{16}{2} = 8 \qquad (3)$$

Definition
LOGARITHME $\log_2 x$: Combien de fois faut-il diviser x par 2 pour que cela donne 1 ?

Exemple. $\log_2 64$

$$\frac{64}{2} = 32 \qquad (1)$$

$$\frac{32}{2} = 16 \qquad (2)$$

$$\frac{16}{2} = 8 \qquad (3)$$

$$\frac{8}{2} = 4 \qquad (4)$$

Definition
LOGARITHME $\log_2 x$: Combien de fois faut-il diviser x par 2 pour que cela donne 1 ?

Exemple. $\log_2 64$

$$\frac{64}{2} = 32 \qquad (1)$$

$$\frac{32}{2} = 16 \qquad (2)$$

$$\frac{16}{2} = 8 \qquad (3)$$

$$\frac{8}{2} = 4 \qquad (4)$$

$$\frac{4}{2} = 2 \qquad (5)$$

Definition
LOGARITHME $\log_2 x$: Combien de fois faut-il diviser x par 2 pour que cela donne 1 ?

Exemple. $\log_2 64$

$$\frac{64}{2} = 32 \qquad (1)$$

$$\frac{32}{2} = 16 \qquad (2)$$

$$\frac{16}{2} = 8 \qquad (3)$$

$$\frac{8}{2} = 4 \qquad (4)$$

$$\frac{4}{2} = 2 \qquad (5)$$

$$\frac{2}{2} = 1 \qquad (6)$$

Definition
LOGARITHME $\log_2 x$: Combien de fois faut-il diviser x par 2 pour que cela donne 1 ?

Exemple. $\log_2 64 = 6$

$$\frac{64}{2} = 32 \qquad (1)$$

$$\frac{32}{2} = 16 \qquad (2)$$

$$\frac{16}{2} = 8 \qquad (3)$$

$$\frac{8}{2} = 4 \qquad (4)$$

$$\frac{4}{2} = 2 \qquad (5)$$

$$\frac{2}{2} = 1 \qquad (6)$$

Logarithme de base 2.

$\log 1 =$	0	$\log 128 =$	7
$\log 2 =$	1	$\log 256 =$	8
$\log 4 =$	2	$\log 512 =$	9
$\log 8 =$	3	$\log 1024 =$	10
$\log 16 =$	4	$\log 10^6 \approx$	20
$\log 32 =$	5	$\log 10^9 \approx$	30
$\log 64 =$	6	$\log 10^{12} \approx$	40

Logarithme de base 2.

$\log 1 =$	0	$\log 128 =$	7
$\log 2 =$	1	$\log 256 =$	8
$\log 4 =$	2	$\log 512 =$	9
$\log 8 =$	3	$\log 1024 =$	10
$\log 16 =$	4	$\log 10^6 \approx$	20
$\log 32 =$	5	$\log 10^9 \approx$	30
$\log 64 =$	6	$\log 10^{12} \approx$	40

Le logarithme est une fonction qui croît très lentement !

Le logarithme est une fonction qui croît très lentement !

Le logarithme est une fonction qui croît très lentement !

Si on effectue ~ 10^9 operations per seconde :

n	$\log n$		n		$n \log n$		n^2	
10^3	10	instantané	10^3	instantané	10^4	instantané	10^6	instantané
10^4	13	— " —	10^4	— " —	$1{,}3 \cdot 10^5$	— " —	10^8	— " —
10^5	16,5	— " —	10^5	— " —	$1{,}6 \cdot 10^6$	— " —	10^{10}	10 sec
10^6	20	— " —	10^6	— " —	$2 \cdot 10^7$	— " —	10^{12}	15 min
10^7	23	— " —	10^7	— " —	$2{,}3 \cdot 10^8$	— " —	10^{14}	1 jour
10^8	26,5	— " —	10^8	— " —	$2{,}6 \cdot 10^9$	2,6 sec	10^{16}	4 mois
10^9	30	— " —	10^9	1 sec	$3 \cdot 10^{10}$	5 min	10^{18}	31 ans
10^{10}	33	— " —	10^{10}	10 sec	$3{,}3 \cdot 10^{11}$	1 h	10^{20}	...
10^{11}	36,5	— " —	10^{11}	1,5 min	$3{,}6 \cdot 10^{12}$	10 h	10^{22}	...
10^{12}	40	— " —	10^{12}	15 min	$4 \cdot 10^{13}$	4,5 jour	10^{24}	...