

Elements d'Algorithmique

CMTD4 : Fonctions Récursives

Mikaël Rabie

Université de Paris, IRIF



INSTITUT
DE RECHERCHE
EN INFORMATIQUE
FONDAMENTALE



Tri par Insertion - Correction de l'Algorithme

Tri par insertion - Principe

Le tri par insertion consiste à

- garder le début du tableau trié
- y insérer successivement, à leur place, les éléments non-triés.

10, **1**, 5, 19, 3, 3

1, 10, 5, 19, 3, 3

1, 10, **5**, 19, 3, 3

1, **5**, 10, 19, 3, 3

1, 5, 10, **19**, 3, 3

1, 5, 10, 19, **3**, 3

1, 5, 10, **3**, 19, 3

1, 5, **3**, 10, 19, 3

1, **3**, 5, 10, 19, 3

1, 3, 5, 10, 19, **3**

1, 3, 5, 10, **3**, 19

1, 3, 5, **3**, 10, 19

1, 3, **3**, 5, 10, 19

Tri par insertion - Pseudocode

Entrée : tableau T

```
1: fonction TRIPARTIEL( $T, i$ )  
2:    $j \leftarrow i$   
3:   tant que  $j > 0$  et  $T[j] < T[j - 1]$  faire  
4:     échanger  $T[j - 1]$  et  $T[j]$   
5:      $j \leftarrow j - 1$   
6: fonction TRIPARINSERTION( $T$ )  
7:    $n \leftarrow$  longueur de  $T$   
8:   pour  $i \leftarrow 1$  à  $n - 1$  faire  
9:     TRIPARTIEL( $T, i$ )
```

Tri par insertion - Pseudocode

Entrée : tableau T

```
1: fonction TRIPARTIEL( $T, i$ )  
2:    $j \leftarrow i$   
3:   tant que  $j > 0$  et  $T[j] < T[j - 1]$  faire  
4:     échanger  $T[j - 1]$  et  $T[j]$   
5:      $j \leftarrow j - 1$   
6: fonction TRIPARINSERTION( $T$ )  
7:    $n \leftarrow$  longueur de  $T$   
8:   pour  $i \leftarrow 1$  à  $n - 1$  faire  
9:     TRIPARTIEL( $T, i$ )
```

TriPartiel :

- Commence avec le nouvel élément en case i
- Insère le nouvel élément à la bonne place dans le tableau $T[0 \dots i]$

Tri par insertion - Pseudocode

Entrée : tableau T

```
1: fonction TRIPARTIEL( $T, i$ )
2:    $j \leftarrow i$ 
3:   tant que  $j > 0$  et  $T[j] < T[j - 1]$  faire
4:     échanger  $T[j - 1]$  et  $T[j]$ 
5:      $j \leftarrow j - 1$ 
6: fonction TRIPARINSERTION( $T$ )
7:    $n \leftarrow$  longueur de  $T$ 
8:   pour  $i \leftarrow 1$  à  $n - 1$  faire
9:     TRIPARTIEL( $T, i$ )
```

TriPartiel :

- Commence avec le nouvel élément en case i
- Insère le nouvel élément à la bonne place dans le tableau $T[0 \dots i]$

TriParInsertion :

- Ajoute les éléments un par un dans le tableau trié avec TriPartiel

Tri Partiel - Correction

Entrée : tableau T

```
1: fonction TRIPARTIEL( $T, i$ )  
2:    $j \leftarrow i$   
3:   tant que  $j > 0$  et  $T[j] < T[j - 1]$  faire  
4:     échanger  $T[j - 1]$  et  $T[j]$   
5:      $j \leftarrow j - 1$ 
```

Tri Partiel - Correction

Entrée : tableau T trié sur $T[0 \dots i - 1]$

1: **fonction** $\text{TRIPARTIEL}(T, i)$

2: $j \leftarrow i$

3: **tant que** $j > 0$ et $T[j] < T[j - 1]$ **faire**

4: échanger $T[j - 1]$ et $T[j]$

5: $j \leftarrow j - 1$

Tri Partiel - Correction

Entrée : tableau T trié sur $T[0 \dots i - 1]$

1: **fonction** `TRIPARTIEL`(T, i)

2: $j \leftarrow i$

3: **tant que** $j > 0$ et $T[j] < T[j - 1]$ **faire**

4: échanger $T[j - 1]$ et $T[j]$

5: $j \leftarrow j - 1$

But : Prouver qu'à la fin de `triPartiel`, le tableau $T[0 \dots i]$ est trié.

Invariant de Boucle

Tri Partiel - Correction

Entrée : tableau T trié sur $T[0 \dots i - 1]$

```
1: fonction TRIPARTIEL( $T, i$ )  
2:    $j \leftarrow i$   
3:   tant que  $j > 0$  et  $T[j] < T[j - 1]$  faire  
4:     échanger  $T[j - 1]$  et  $T[j]$   
5:      $j \leftarrow j - 1$ 
```

But : Prouver qu'à la fin de triPartiel, le tableau $T[0 \dots i]$ est trié.

Invariant de Boucle (preuve au tableau) :

Soit T_0 le tableau au départ. À chaque passage dans la boucle **tant que** :

- $T[j] = T_0[i]$
- Les valeurs dans $T_0[0 \dots i - 1]$ sont dans $T[0 \dots i]$, et leur ordre ne change pas
- $T[j]$ est plus petit que les valeurs dans $T[j + 1 \dots i - 1]$

Tri Partiel - Correction

But : Prouver qu'à la fin de `triPartiel`, le tableau $T[0 \dots i]$ est trié.

Invariant de Boucle (preuve au tableau) :

Soit T_0 le tableau au départ. À chaque passage dans la boucle **tant que** :

- $T[j] = T_0[i]$
- Les valeurs dans $T_0[0 \dots i - 1]$ sont dans $T[0 \dots i]$, et leur ordre ne change pas
- $T[j]$ est plus petit que les valeurs dans $T[j + 1 \dots i - 1]$

Conclusion :

- On a gardé les valeurs
- Les valeurs de $T_0[0 \dots i - 1]$ restent triées entre elles
- $T_0[i]$ est placé au bon endroit dans le tableau

Tri Partiel - Correction

But : Prouver qu'à la fin de `triPartiel`, le tableau $T[0 \dots i]$ est trié.

Invariant de Boucle (preuve au tableau) :

Soit T_0 le tableau au départ. À chaque passage dans la boucle **tant que** :

- $T[j] = T_0[i]$
- Les valeurs dans $T_0[0 \dots i - 1]$ sont dans $T[0 \dots i]$, et leur ordre ne change pas
- $T[j]$ est plus petit que les valeurs dans $T[j + 1 \dots i - 1]$

Conclusion :

- On a gardé les valeurs
- Les valeurs de $T_0[0 \dots i - 1]$ restent triées entre elles
- $T_0[i]$ est placé au bon endroit dans le tableau

$\Rightarrow T[0 \dots i]$ est trié

Tri par Insertion - Correction

Entrée : tableau T

```
1: fonction TRIPARINSERTION( $T$ )  
2:    $n \leftarrow$  longueur de  $T$   
3:   pour  $i \leftarrow 1$  à  $n - 1$  faire  
4:     TRIPARTIEL( $T, i$ )
```

TriPartiel : Si $T[0 \dots i - 1]$ est trié, alors $\text{triPartiel}(T, i)$ trie $T[0 \dots i]$.

Tri par Insertion - Correction

Entrée : tableau T

```
1: fonction TRIPARINSERTION( $T$ )  
2:    $n \leftarrow$  longueur de  $T$   
3:   pour  $i \leftarrow 1$  à  $n - 1$  faire  
4:     TRIPARTIEL( $T, i$ )
```

TriPartiel : Si $T[0 \dots i - 1]$ est trié, alors $\text{triPartiel}(T, i)$ trie $T[0 \dots i]$.

Par **récurrence**, après i appels de la fonction triPartiel , T est trié sur les cases $[0 \dots i]$.

- **Initialisation :** Avant le premier appel, la première case est bien triée.
- **Hérédité :** Si $T[0 \dots i]$ est trié, on sait que $\text{triPartiel}(T, i + 1)$ trie $T[0 \dots i + 1]$.

Tri par Insertion - Correction

Entrée : tableau T

```
1: fonction TRIPARINSERTION( $T$ )  
2:    $n \leftarrow$  longueur de  $T$   
3:   pour  $i \leftarrow 1$  à  $n - 1$  faire  
4:     TRIPARTIEL( $T, i$ )
```

TriPartiel : Si $T[0 \dots i - 1]$ est trié, alors $\text{triPartiel}(T, i)$ trie $T[0 \dots i]$.

Par **récurrence**, après i appels de la fonction triPartiel , T est trié sur les cases $[0 \dots i]$.

- **Initialisation** : Avant le premier appel, la première case est bien triée.
- **Hérédité** : Si $T[0 \dots i]$ est trié, on sait que $\text{triPartiel}(T, i + 1)$ trie $T[0 \dots i + 1]$.

Conclusion : triParInsertion trie les tableaux.

Fonctions Récursives

Répétition d'actions

Si on veut exécuter un certain de nombre de fois la même séquence d'actions :

- La boucle For
- La boucle Tant Que
- Les fonctions récursives

Factorielle

$$\text{factorielle}(n) = n! = n \times (n - 1) \times \dots \times 2 \times 1$$

Entrée : entier n

```
1: fonction FACTORIELLE( $n$ )  
2:    $f = 1$   
3:   pour  $i \leftarrow 2$  à  $n$  faire  
4:      $f = f \times i$   
5:   retourne  $f$ 
```

Factorielle

$$\text{factorielle}(n) = n! = n \times (n - 1) \times \dots \times 2 \times 1$$

Entrée : entier n

```
1: fonction FACTORIELLE( $n$ )  
2:    $f = 1$   
3:   pour  $i \leftarrow 2$  à  $n$  faire  
4:      $f = f \times i$   
5:   retourne  $f$ 
```

Entrée : entier n

```
1: fonction FACTREC( $n$ )  
2:   si  $n = 0$  alors  
3:     retourne 1  
4:   sinon  
5:     retourne  $n \times \text{FACTREC}(n - 1)$ 
```

$\log_2(n)$ = Plus grand entier k tel que $2^k \leq n$

Entrée : entier n

1: **fonction** LOGARITHME(n)

2: $l = 0$

3: **tant que** $n \geq 1$ **faire**

4: $n = n/2$

5: $l = l + 1$

6: **retourne** l

$\log_2(n)$ = Plus grand entier k tel que $2^k \leq n$

Entrée : entier n

```
1: fonction LOGARITHME( $n$ )  
2:    $l = 0$   
3:   tant que  $n \geq 1$  faire  
4:      $n = n/2$   
5:      $l = l + 1$   
6:   retourne  $l$ 
```

Entrée : entier n

```
1: fonction LOGREC( $n$ )  
2:   si  $n \leq 1$  alors  
3:     retourne 0  
4:   sinon  
5:     retourne  $1 + \text{LOGREC}(n/2)$ 
```

Pile d'exécution

Entrée : entier n

```
1: fonction FACTREC( $n$ )  
2:   si  $n = 0$  alors  
3:     retourne 1  
4:   sinon  
5:     retourne  $n \times \text{FACTREC}(n - 1)$ 
```

Pile d'exécution :

factRec(4) $f = 4 \times ?$

Pile d'exécution

Entrée : entier n

```
1: fonction FACTREC( $n$ )  
2:   si  $n = 0$  alors  
3:     retourne 1  
4:   sinon  
5:     retourne  $n \times \text{FACTREC}(n - 1)$ 
```

Pile d'exécution :

factRec(3)	
factRec(4)	$f = 4 \times \searrow$

Pile d'exécution

Entrée : entier n

```
1: fonction FACTREC( $n$ )  
2:   si  $n = 0$  alors  
3:     retourne 1  
4:   sinon  
5:     retourne  $n \times \text{FACTREC}(n - 1)$ 
```

Pile d'exécution :

factRec(2)	
factRec(3)	$f = 3 \times \searrow$
factRec(4)	$f = 4 \times \searrow$

Pile d'exécution

Entrée : entier n

```
1: fonction FACTREC( $n$ )  
2:   si  $n = 0$  alors  
3:     retourne 1  
4:   sinon  
5:     retourne  $n \times \text{FACTREC}(n - 1)$ 
```

Pile d'exécution :

factRec(1)	
factRec(2)	$f = 2 \times \searrow$
factRec(3)	$f = 3 \times \searrow$
factRec(4)	$f = 4 \times \searrow$

Pile d'exécution

Entrée : entier n

```
1: fonction FACTREC( $n$ )  
2:   si  $n = 0$  alors  
3:     retourne 1  
4:   sinon  
5:     retourne  $n \times \text{FACTREC}(n - 1)$ 
```

Pile d'exécution :

factRec(0)
factRec(1) $f = 1 \times \searrow$
factRec(2) $f = 2 \times \searrow$
factRec(3) $f = 3 \times \searrow$
factRec(4) $f = 4 \times \searrow$

Pile d'exécution

Entrée : entier n

```
1: fonction FACTREC( $n$ )  
2:   si  $n = 0$  alors  
3:     retourne 1  
4:   sinon  
5:     retourne  $n \times \text{FACTREC}(n - 1)$ 
```

Pile d'exécution :

factRec(0)	$f = 1$
factRec(1)	$f = 1 \times \searrow$
factRec(2)	$f = 2 \times \searrow$
factRec(3)	$f = 3 \times \searrow$
factRec(4)	$f = 4 \times \searrow$

Pile d'exécution

Entrée : entier n

```
1: fonction FACTREC( $n$ )  
2:   si  $n = 0$  alors  
3:     retourne 1  
4:   sinon  
5:     retourne  $n \times \text{FACTREC}(n - 1)$ 
```

Pile d'exécution :

factRec(1)	$f = 1 \times 1$
factRec(2)	$f = 2 \times \searrow$
factRec(3)	$f = 3 \times \searrow$
factRec(4)	$f = 4 \times \searrow$

Pile d'exécution

Entrée : entier n

```
1: fonction FACTREC( $n$ )  
2:   si  $n = 0$  alors  
3:     retourne 1  
4:   sinon  
5:     retourne  $n \times \text{FACTREC}(n - 1)$ 
```

Pile d'exécution :

factRec(2)	$f = 2 \times 1$
factRec(3)	$f = 3 \times \searrow$
factRec(4)	$f = 4 \times \searrow$

Pile d'exécution

Entrée : entier n

```
1: fonction FACTREC( $n$ )  
2:   si  $n = 0$  alors  
3:     retourne 1  
4:   sinon  
5:     retourne  $n \times \text{FACTREC}(n - 1)$ 
```

Pile d'exécution :

factRec(3)	$f = 3 \times 2$
factRec(4)	$f = 4 \times \searrow$

Pile d'exécution

Entrée : entier n

```
1: fonction FACTREC( $n$ )  
2:   si  $n = 0$  alors  
3:     retourne 1  
4:   sinon  
5:     retourne  $n \times \text{FACTREC}(n - 1)$ 
```

Pile d'exécution :

factRec(4) $f = 4 \times 6$

Fonction Récursive Terminale

Fonction récursive **terminale** : La dernière instruction est un appel récursif.

Entrée : entier n

1: **fonction** FACTREC(n)

2: **si** $n = 0$ **alors**

3: **retourne** 0

4: **sinon**

5: **retourne** $n \times \text{FACTREC}(n - 1)$

Entrée : entier n

1: **fonction** FACTINTER(n, f)

2: **si** $n = 0$ **alors**

3: **retourne** f

4: **sinon**

5: **retourne** FACTINTER($n - 1, n \times f$)

6: **fonction** FACTRECTER(n)

7: **retourne** FACTINTER($n, 1$)

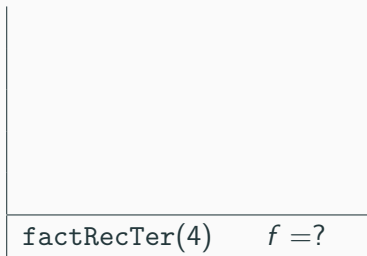
Fonction Récursive Terminale

Fonction récursive **terminale** : La dernière instruction est un appel récursif.

Entrée : entier n

```
1: fonction FACTINTER( $n, f$ )  
2:   si  $n = 0$  alors  
3:     retourne  $f$   
4:   sinon  
5:     retourne FACTINTER( $n - 1, n \times f$ )  
6: fonction FACTRECTER( $n$ )  
7:   retourne FACTINTER( $n, 1$ )
```

Pile d'exécution :



Fonction Récursive Terminale

Fonction récursive **terminale** : La dernière instruction est un appel récursif.

Entrée : entier n

```
1: fonction FACTINTER( $n, f$ )  
2:   si  $n = 0$  alors  
3:     retourne  $f$   
4:   sinon  
5:     retourne FACTINTER( $n - 1, n \times f$ )  
6: fonction FACTRECTER( $n$ )  
7:   retourne FACTINTER( $n, 1$ )
```

Pile d'exécution :

factInter(4, 1)	
factRecTer(4)	$f = \searrow$

Fonction Récursive Terminale

Fonction récursive **terminale** : La dernière instruction est un appel récursif.

Entrée : entier n

```
1: fonction FACTINTER( $n, f$ )
```

```
2:   si  $n = 0$  alors
```

```
3:     retourne f
```

4: sinon

```
5:   retourne FACTINTER( $n - 1, n \times f$ )
```

6: **fonction** FACTRECTER(n)

```

7:   retourne FACTINTER( $n, 1$ )

```

Pile d'exécution :

factInter(3, 4)	
factRecTer(4)	$f = \searrow$

Fonction Récursive Terminale

Fonction récursive **terminale** : La dernière instruction est un appel récursif.

Entrée : entier n

```
1: fonction FACTINTER( $n, f$ )  
2:   si  $n = 0$  alors  
3:     retourne  $f$   
4:   sinon  
5:     retourne FACTINTER( $n - 1, n \times f$ )  
6: fonction FACTRECTER( $n$ )  
7:   retourne FACTINTER( $n, 1$ )
```

Pile d'exécution :

factInter(2, 12)	
factRecTer(4)	$f = \searrow$

Fonction Récursive Terminale

Fonction récursive **terminale** : La dernière instruction est un appel récursif.

Entrée : entier n

```
1: fonction FACTINTER( $n, f$ )  
2:   si  $n = 0$  alors  
3:     retourne  $f$   
4:   sinon  
5:     retourne FACTINTER( $n - 1, n \times f$ )  
6: fonction FACTRECTER( $n$ )  
7:   retourne FACTINTER( $n, 1$ )
```

Pile d'exécution :

factInter(1, 24)	
factRecTer(4)	$f = \searrow$

Fonction Récursive Terminale

Fonction récursive **terminale** : La dernière instruction est un appel récursif.

Entrée : entier n

```
1: fonction FACTINTER( $n, f$ )  
2:   si  $n = 0$  alors  
3:     retourne  $f$   
4:   sinon  
5:     retourne FACTINTER( $n - 1, n \times f$ )  
6: fonction FACTRECTER( $n$ )  
7:   retourne FACTINTER( $n, 1$ )
```

Pile d'exécution :

factInter(0, 24)	
factRecTer(4)	$f = \searrow$

Fonction Récursive Terminale

Fonction récursive **terminale** : La dernière instruction est un appel récursif.

Entrée : entier n

```
1: fonction FACTINTER( $n, f$ )  
2:   si  $n = 0$  alors  
3:     retourne  $f$   
4:   sinon  
5:     retourne FACTINTER( $n - 1, n \times f$ )  
6: fonction FACTRECTER( $n$ )  
7:   retourne FACTINTER( $n, 1$ )
```

Pile d'exécution :

factRecTer(4) $f = 24$
