# Elements d'Algorithmique
# CMTD4 : Fonctions Récursives

Mikaël Rabie
Université de Paris, IRIF

INSTITUT
DE RECHERCHE
EN INFORMATIQUE
FONDAMENTALE

Université de Paris

# 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

**fonction** TRIPARTIEL(T, $i$)
    $j \leftarrow i$
    **tant que** $j > 0$ et $T[j] < T[j-1]$ **faire**
        échanger $T[j-1]$ et $T[j]$
        $j \leftarrow j - 1$
**fonction** TRIPARINSERTION(T)
    $n \leftarrow$ longueur de T
    **pour** $i \leftarrow 1$ à $n - 1$ **faire**
        TRIPARTIEL(T,$i$)

# Tri par insertion - Pseudocode

**Entrée :** tableau T

**fonction** TRIPARTIEL(T, $i$)
    $j \leftarrow i$
    **tant que** $j > 0$ et $\mathtt{T}[j] < \mathtt{T}[j-1]$ **faire**
        échanger $\mathtt{T}[j-1]$ et $\mathtt{T}[j]$
        $j \leftarrow j - 1$
**fonction** TRIPARINSERTION(T)
    $n \leftarrow$ longueur de T
    **pour** $i \leftarrow 1$ à $n - 1$ **faire**
        TRIPARTIEL(T,$i$)

**TriPartiel :**

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

# Tri par insertion - Pseudocode

**Entrée :** tableau T

```
fonction triPartiel(T, i)
    j ← i
    tant que j > 0 et T[j] < T[j − 1] faire
        échanger T[j − 1] et T[j]
        j ← j − 1
fonction triParInsertion(T)
    n ← longueur de T
    pour i ← 1 à n − 1 faire
        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 $\mathtt{T}$

**fonction** $\textsc{triPartiel}(\mathtt{T}, i)$
    $j \leftarrow i$
    **tant que** $j > 0$ et $\mathtt{T}[j] < \mathtt{T}[j-1]$ **faire**
        échanger $\mathtt{T}[j-1]$ et $\mathtt{T}[j]$
        $j \leftarrow j - 1$

# Tri Partiel - Correction

**Entrée :** tableau $\mathtt{T}$ trié sur $T[0 \ldots i-1]$

**fonction** $\textsc{triPartiel}(\mathtt{T}, i)$
    $j \leftarrow i$
    **tant que** $j > 0 \text{ et } \mathtt{T}[j] < \mathtt{T}[j-1]$ **faire**
        échanger $\mathtt{T}[j-1]$ et $\mathtt{T}[j]$
        $j \leftarrow j - 1$

## Tri Partiel - Correction

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

**fonction** $\textsc{triPartiel}(\texttt{T}, i)$
    $j \leftarrow i$
    **tant que** $j > 0$ et $\texttt{T}[j] < \texttt{T}[j-1]$ **faire**
        échanger $\texttt{T}[j-1]$ et $\texttt{T}[j]$
        $j \leftarrow j - 1$

**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 $T[0 \dots i-1]$

```
fonction triPartiel(T, i)
    j ← i
    tant que j > 0 et T[j] < T[j − 1] faire
        échanger T[j − 1] et T[j]
        j ← 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

$$\Rightarrow T[0 \dots i] \text{ est trié}$$

# Tri par Insertion - Correction

**Entrée :** tableau T

**fonction** triParInsertion(T)
  $n \leftarrow$ longueur de T
  **pour** $i \leftarrow 1$ à $n-1$ **faire**
    triPartiel(T,$i$)

**TriPartiel** : Si $T[0 \ldots i-1]$ est trié, alors `triPartiel`$(T, i)$ trie $T[0 \ldots i]$.

# Tri par Insertion - Correction

**Entrée :** tableau T

**fonction** TRIPARINSERTION(T)
  $n \leftarrow$ longueur de T
  **pour** $i \leftarrow 1$ à $n-1$ **faire**
    TRIPARTIEL(T,$i$)

**TriPartiel** : Si $T[0\ldots i-1]$ est trié, alors `triPartiel`$(T,i)$ trie $T[0\ldots i]$.

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

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

# Tri par Insertion - Correction

**Entrée :** tableau T

```
fonction triParInsertion(T)
    n ← longueur de T
    pour i ← 1 à n − 1 faire
        triPartiel(T,i)
```

**TriPartiel** : Si $T[0 \dots i-1]$ est trié, alors `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 `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

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

**Entrée :** entier $n$

**fonction** $\textsc{Factorielle}(n)$
    $f = 1$
    **pour** $i \leftarrow 2$ à $n$ **faire**
        $f = f \times i$
    **retourne** $f$

# Factorielle

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

**Entrée :** entier $n$

**fonction** FACTORIELLE($n$)
    $f = 1$
    **pour** $i \leftarrow 2$ à $n$ **faire**
        $f = f \times i$
    **retourne** $f$

**Entrée :** entier $n$

**fonction** FACTREC($n$)
    **si** $n = 0$ **alors**
        **retourne** $0$
    **sinon**
        **retourne** $n\times$ FACTREC($n-1$)

# Logarithme

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

**Entrée :** entier $n$

**fonction** LOGARITHME($n$)
    $l = 0$
    **tant que** $n \geq 1$ **faire**
        $n = n/2$
        $l = l + 1$
    **retourne** $l$

# Logarithme

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

**Entrée :** entier $n$

**fonction** $\textsc{Logarithme}(n)$<br>
    $l = 0$<br>
    **tant que** $n \geq 1$ **faire**<br>
        $n = n/2$<br>
        $l = l + 1$<br>
    **retourne** $l$

**Entrée :** entier $n$

**fonction** $\textsc{logRec}(n)$<br>
    **si** $n \leq 1$ **alors**<br>
        **retourne** $0$<br>
    **sinon**<br>
        **retourne** $1+ \textsc{logRec}(n/2)$

# Pile d'exécution

Pile d'éxécution :

**Entrée :** entier $n$

**fonction** $\textsc{factRec}(n)$
  **si** $n = 0$ **alors**
    **retourne** $0$
  **sinon**
    **retourne** $n\times$ $\textsc{factRec}(n-1)$

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

# Pile d'exécution

**Entrée :** entier $n$

```
fonction FACTREC(n)
    si n = 0 alors
        retourne 0
    sinon
        retourne n× FACTREC(n − 1)
```

Pile d'éxécution :

| |
|---|
| `factRec(3)` |
| `factRec(4)` $f = 4\times \searrow$ |

# Pile d'exécution

**Entrée :** entier $n$

**fonction** FACTREC($n$)
  **si** $n = 0$ **alors**
    **retourne** $0$
  **sinon**
    **retourne** $n \times$ FACTREC($n - 1$)

Pile d'éxécution :

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

# Pile d'exécution

**Entrée :** entier $n$

**fonction** $\textsc{factRec}(n)$
  **si** $n = 0$ **alors**
    **retourne** $0$
  **sinon**
    **retourne** $n \times \textsc{factRec}(n-1)$

Pile d'éxécution :

| | |
|---|---|
| `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 $n$

**fonction** $\textsc{factRec}(n)$
    **si** $n = 0$ **alors**
        **retourne** $0$
    **sinon**
        **retourne** $n\times$ $\textsc{factRec}(n-1)$

Pile d'éxécution :

| | |
|---|---|
| | |
| 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 $n$

**fonction** $\textsc{factRec}(n)$
  **si** $n = 0$ **alors**
    **retourne** $0$
  **sinon**
    **retourne** $n\times$ $\textsc{factRec}(n-1)$

Pile d'éxécution :

| | |
|---|---|
| | |
| `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 $n$

```
fonction FACTREC(n)
    si n = 0 alors
        retourne 0
    sinon
        retourne n× FACTREC(n − 1)
```

Pile d'éxécution :

| | |
|---|---|
| `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 $n$

```
fonction FACTREC(n)
    si n = 0 alors
        retourne 0
    sinon
        retourne n× FACTREC(n − 1)
```

Pile d'éxécution :

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

# Pile d'exécution

**Entrée :** entier $n$

**fonction** FACTREC($n$)
  **si** $n = 0$ **alors**
    **retourne** $0$
  **sinon**
    **retourne** $n \times$ FACTREC($n-1$)

Pile d'éxécution :

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

# Pile d'exécution

**Entrée :** entier $n$

**fonction** $\textsc{factRec}(n)$
  **si** $n = 0$ **alors**
    **retourne** $0$
  **sinon**
    **retourne** $n \times\ \textsc{factRec}(n-1)$

Pile d'éxécution :

| | |
|---|---|
| `factRec(4)` | $f = 4 \times 6$ |

# Fonction Récursive Terminale

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

**Entrée :** entier $n$

```
fonction FACTREC(n)
    si n = 0 alors
        retourne 0
    sinon
        retourne n× FACTREC(n − 1)
```

Pile d'éxécution :

**Entrée :** entier $n$

```
fonction FACTINTER(n, f)
    si n = 0 alors
        retourne f
    sinon
        retourne FACTINTER(n − 1, n × f)
fonction FACTRECTER(n)
    retourne FACTINTER(n, 1)
```

# Fonction Récursive Terminale

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

**Entrée :** entier $n$

```
fonction FACTINTER(n, f)
    si n = 0 alors
        retourne f
    sinon
        retourne FACTINTER(n − 1, n × f)
fonction FACTRECTER(n)
    retourne FACTINTER(n, 1)
```

Pile d'éxécution :

| | |
|---|---|
| `factRecTer`$(4)$ | $f = ?$ |

# Fonction Récursive Terminale

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

**Entrée :** entier $n$

**fonction** $\textsc{factInter}(n, f)$
  **si** $n = 0$ **alors**
    **retourne** f
  **sinon**
    **retourne** $\textsc{factInter}(n - 1, n \times f)$
**fonction** $\textsc{factRecTer}(n)$
  **retourne** $\textsc{factInter}(n, 1)$

Pile d'éxécution :

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

# Fonction Récursive Terminale

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

**Entrée :** entier $n$

**fonction** $\textsc{factInter}(n, f)$
    **si** $n = 0$ **alors**
        **retourne** f
    **sinon**
        **retourne** $\textsc{factInter}(n-1, n \times f)$
**fonction** $\textsc{factRecTer}(n)$
    **retourne** $\textsc{factInter}(n, 1)$

Pile d'éxécution :

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

# Fonction Récursive Terminale

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

**Entrée :** entier $n$

**fonction** $\textsc{factInter}(n, f)$
    **si** $n = 0$ **alors**
        **retourne** f
    **sinon**
        **retourne** $\textsc{factInter}(n-1, n \times f)$
**fonction** $\textsc{factRecTer}(n)$
    **retourne** $\textsc{factInter}(n, 1)$

Pile d'éxécution :

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

# Fonction Récursive Terminale

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

**Entrée :** entier $n$

**fonction** $\textsc{factInter}(n, f)$
    **si** $n = 0$ **alors**
        **retourne** f
    **sinon**
        **retourne** $\textsc{factInter}(n-1, n \times f)$
**fonction** $\textsc{factRecTer}(n)$
    **retourne** $\textsc{factInter}(n, 1)$

Pile d'éxécution :

| | |
|---|---|
| $\texttt{factInter}(1, 24)$ | |
| $\texttt{factRecTer}(4)$ | $f = \searrow$ |

# Fonction Récursive Terminale

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

**Entrée :** entier $n$

**fonction** FACTINTER($n, f$)
    **si** $n = 0$ **alors**
        **retourne** f
    **sinon**
        **retourne** FACTINTER($n-1, n \times f$)
**fonction** FACTRECTER($n$)
    **retourne** FACTINTER($n, 1$)

Pile d'éxécution :

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

# Fonction Récursive Terminale

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

**Entrée :** entier $n$

**fonction** $\textsc{factInter}(n, f)$
  **si** $n = 0$ **alors**
    **retourne** f
  **sinon**
    **retourne** $\textsc{factInter}(n-1, n \times f)$
**fonction** $\textsc{factRecTer}(n)$
  **retourne** $\textsc{factInter}(n, 1)$

Pile d'éxécution :

| | |
|---|---|
| `factRecTer`$(4)$ | $f = 24$ |