

Algorithmes de tri

Dr. Khadhraoui Mayara

2025/2026

PLAN

1. Introduction
2. Tri par sélection
3. Tri par insertion
4. Tri à bulle
5. Tri par fusion

- Le terme "trier" signifie «répartir des objets suivant certains critères».
- En algorithmique, le terme "tri" est souvent attaché au processus de classement d'une suite d'éléments dans un ordre donné.

Exemple:

- Trier N entier dans un ordres croissant.
- Trier N étudiants dans l'ordre décroissant de leurs moyennes (réel).
- Trier N noms dans l'ordre alphabétique croissant.
- Trier des fichiers selon leurs tailles.

- D'une façon générale, tout ensemble muni d'un ordre total peut fournir une suite d'éléments à trier.
- Deux catégories de tris :
 - **Les tris internes** : méthodes destinées à des masses de données limitées, stockées dans une structure de données se trouvant dans la mémoire centrale (Exemple : tableaux).
 - **Les tris externes** : méthodes destinées à de grandes masses de données, stockées dans des structures de données telle que les fichiers.

- Le tri est fondamental à beaucoup d'autres problèmes, et après le tri, beaucoup de problèmes deviennent faciles à résoudre. par exemple :
 - La recherche d'un élément.
 - Unicité d'éléments: après le tri tester les éléments adjacents.
 - Déterminer le plus petit et le plus grand élément.
 - Déterminer le kième plus grand élément.

Les algorithmes de tri se regroupent en plusieurs familles selon leur stratégie :

- Tris **simples** : faciles à comprendre et à implémenter, mais peu performants sur de grandes données.
 - Exemples : tri par sélection, tri par insertion, tri à bulles.
- Tris **efficaces** : plus complexes, mais beaucoup plus rapides sur de grands volumes.
 - Exemples : tri rapide (QuickSort), tri fusion (MergeSort), tri par tas (HeapSort).
- Tris **spécialisés** : exploitent la structure ou la nature des données au lieu de comparer les éléments un à un.
 - Exemples : tri par comptage (Counting Sort), tri radix (Radix Sort), tri par casier (Bucket Sort).

Principe:

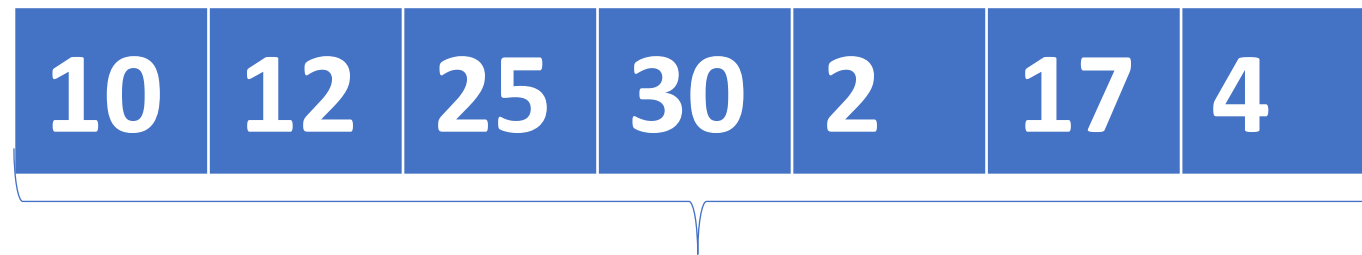
Répéter:

- Chercher le plus petit (plus grand) élément => **Sélection**.
- Mettre l'élément sélectionné au début de la partie non triée.

D'une autre manière

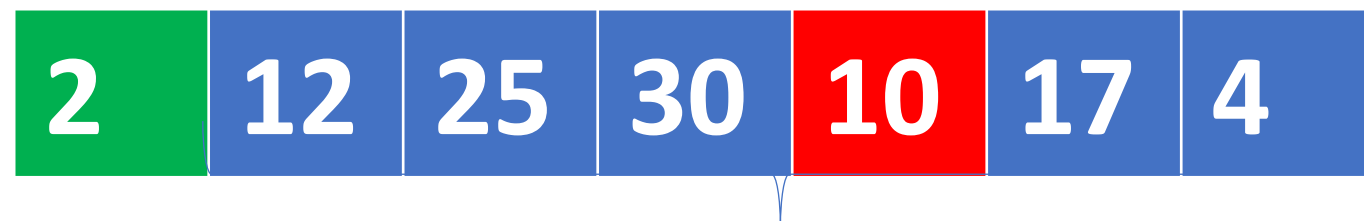
- Trouver le plus petit élément et le mettre au début du tableau
- Trouver le 2e plus petit élément et le mettre en seconde position
- Trouver le 3e plus petit élément et le mettre à la 3e place,
- ...

Exemple: T



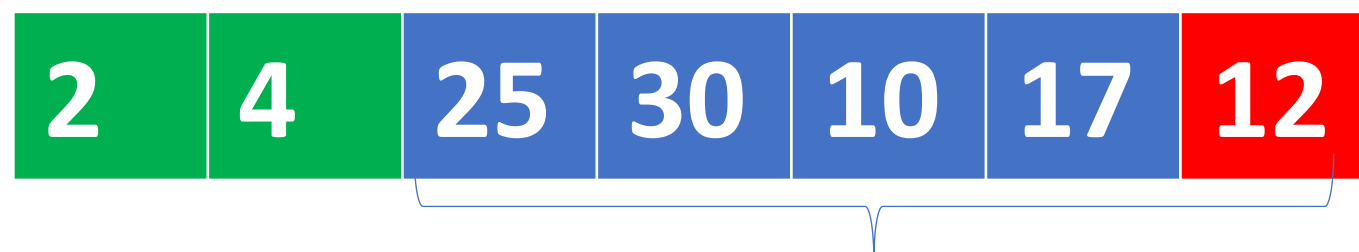
min = 2

Etape 1: Faire l'échange entre 2 (min) et 10



min = 4

Etape 2: Faire l'échange entre 4 (min) et 12



min = 10

Etape 3: Faire l'échange entre 10 (min) et 25



Etape 4: Faire l'échange entre 12 (min) et 30



Etape 5: Faire l'échange entre 17 (min) et 25



Etape 6: 25 (min) est à sa place



Pour $i \leftarrow 0$ à $n-2$ faire

$\text{min} \leftarrow i$

 Pour $j \leftarrow i+1$ à $n-1$ faire

 Si $\text{tab}[j] < \text{tab}[\text{min}]$ alors

/ recherche de l'indice du minimum */*

$\text{min} \leftarrow j$

 FinSi

FinPour

Si $\text{min} \neq i$ alors

échanger($\text{tab}[i]$, $\text{tab}[\text{min}]$)

/ échange des valeurs entre la case
 courante et le minimum */*

 FinSi

FinPour

```
int temp;           // variable temporaire
temp = tab[i];      // on garde la valeur de tab[i]
tab[i] = tab[min];  // on met la valeur de tab[min] à la place
tab[min] = temp;    // on met la valeur de tab[i] (ancienne) dans tab[min]
```

Complexité:

- Le tri par sélection fait toujours le même nombre de comparaisons, qu'importe si le tableau est déjà trié ou pas.
 - ♦ Principe de calcul
- Pour chaque position i , il cherche **le plus petit élément** dans la **partie non triée**.
- Pour n éléments :
 - 1er tour : $n-1$ comparaisons
 - 2e tour : $n-2$ comparaisons
 - etc.
- 👉 Total des comparaisons : $(n-1) + (n-2) + (n-3) + \dots + 1 = \frac{n(n-1)}{2}$
- ♦ Résultat : Complexité en temps : **$O(n^2)$** dans tous les cas (meilleur, moyen et pire).

Avantages :

- Simple à comprendre et à implémenter.
- Ne nécessite pas de mémoire supplémentaire (tri en place).

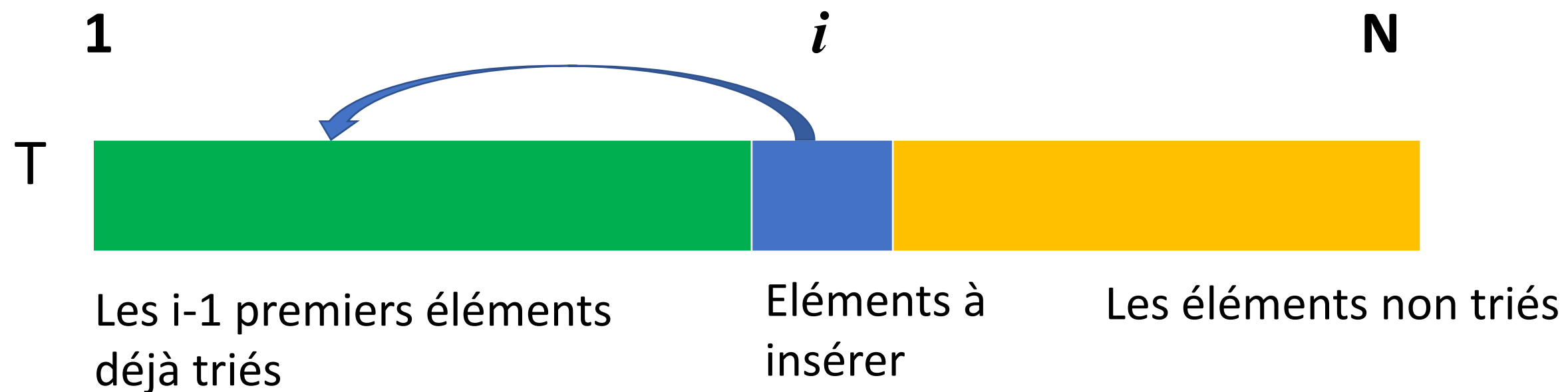
Inconvénients :

- Très lent pour les grandes listes (complexité ($O(n^2)$)).
- Inefficace sur les grands tableaux.

Principe:

Répéter:

- Insertion du prochain élément dans la partie qui est déjà triée précédemment.
- La partie de départ qui est triée est le premier élément.
- Il se pourrait qu'on a à déplacer plusieurs éléments pour l'insertion.



Principe:

(le tri du joueur de cartes !)

- Ordonner les deux premiers éléments
- Insérer le 3^e élément de manière à ce que les 3 premiers éléments soient triés
- Insérer le 4^e élément à “sa” place pour que. . .
- . . .
- Insérer le i -ième élément à sa place.

A la fin de la i -ème itération, les i premiers éléments de T sont triés.

Exemple:

T

10	12	5	30	2	17	4
----	----	---	----	---	----	---

Etape 1: insérer 12 à sa place

10	12	5	30	2	17	4
----	----	---	----	---	----	---

Etape 2: insérer 5 à sa place

5	10	12	30	2	17	4
---	----	----	----	---	----	---

Etape 3: insérer 30 à sa place



Etape 4: insérer 2 à sa place



Etape 5: insérer 17 à sa place



Etape 6: insérer 4 à sa place



```
Pour i ← 1 à n-1 faire  
  valeur ← tab[i]  
  j ← i - 1  
  Tant que j ≥ 0 et tab[j] > valeur faire  
    tab[j + 1] ← tab[j]  
    j ← j - 1  
  FinTantQue  
  tab[j + 1] ← valeur  
FinPour
```

Itération

Tableau initial : [5, 3, 4, 1]

- 1 i=1 → valeur=3 → on le place avant 5 → [3, 5, 4, 1]
- 2 i=2 → valeur=4 → on le place avant 5 → [3, 4, 5, 1]
- 3 i=3 → valeur=1 → on le place au début → [1, 3, 4, 5]

Complexité:

- Comme nous n'avons pas nécessairement à scanner toute la partie déjà triée, le pire cas, le meilleur cas et le cas moyen peuvent différer entre eux.
- Pire des cas: Chaque élément est inséré au début de la partie triée. Dans ce cas, tous les éléments de la partie triée doivent être déplacés à chaque itération.
- La complexité **du tri par insertion** dépend de l'ordre initial du tableau : dans le pire des cas (tableau inversé), elle est quadratique **$O(n^2)$** .

Avantages:

- Très efficace pour les petites listes ou les listes presque triées.
- Tri en place, donc peu de mémoire utilisée.

Inconvénients :

- Lent pour les grandes listes non triées ($O(n^2)$).
- Peu adapté aux grands ensembles de données.

Principe:

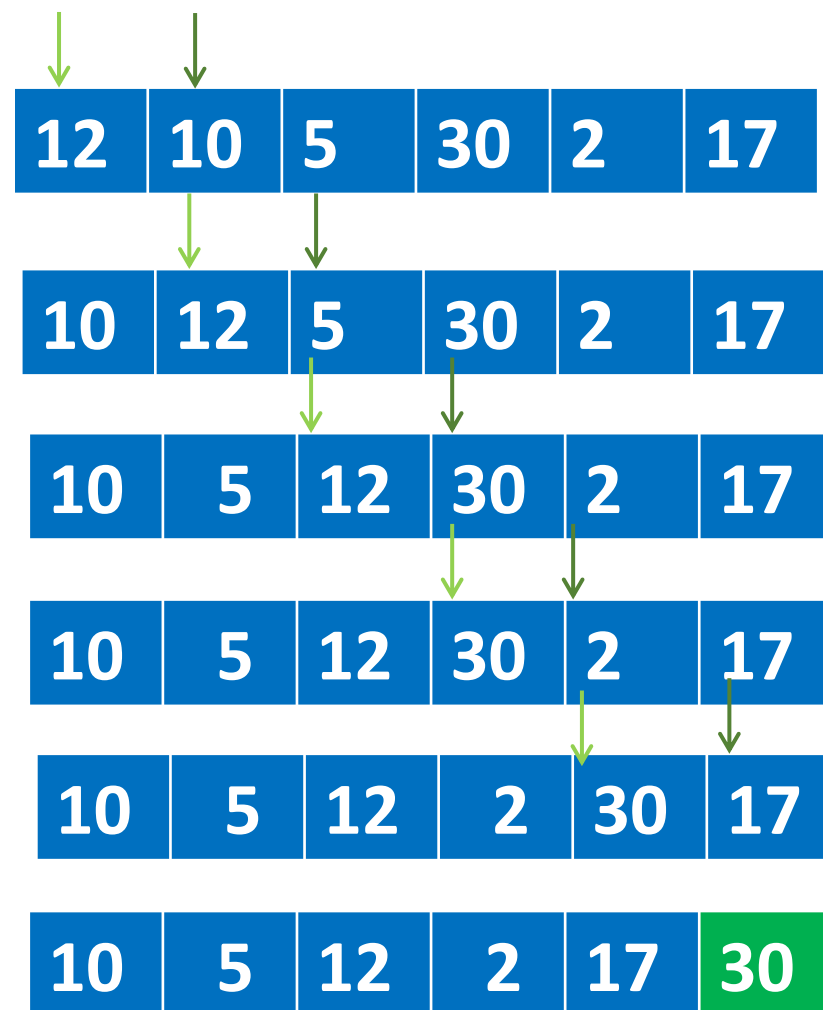
Répéter:

- Parcourir le tableau en comparant deux à deux les éléments successifs, permuter s'ils ne sont pas dans l'ordre.
 - Répéter tant que des permutations sont effectuées.
-
- Après le premier parcours, le plus grand élément étant à sa position définitive, il n'a plus à être traité.
 - Le reste du tableau est en revanche encore en désordre. Il faut donc le parcourir à nouveau, en s'arrêtant à l'avant-dernier élément.
 - Après ce deuxième parcours, les deux plus grands éléments sont à leur position définitive.
 - Il faut donc répéter les parcours du tableau, jusqu'à ce que les deux plus petits éléments soient placés à leur position définitive.

Exemple: T

12	10	5	30	2	17
----	----	---	----	---	----

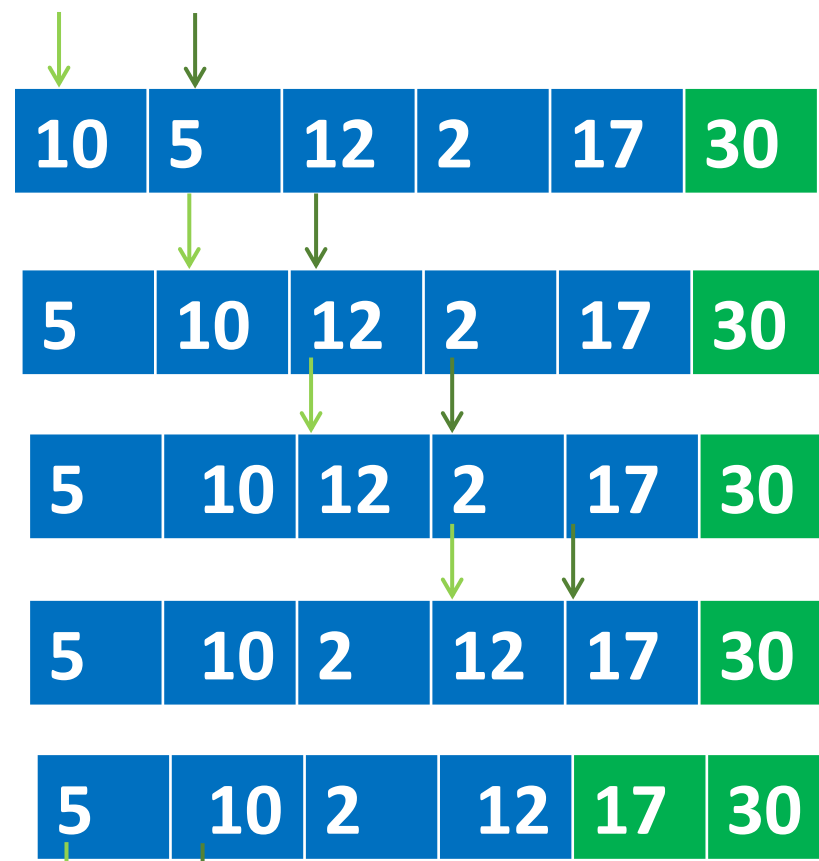
Itération 1:



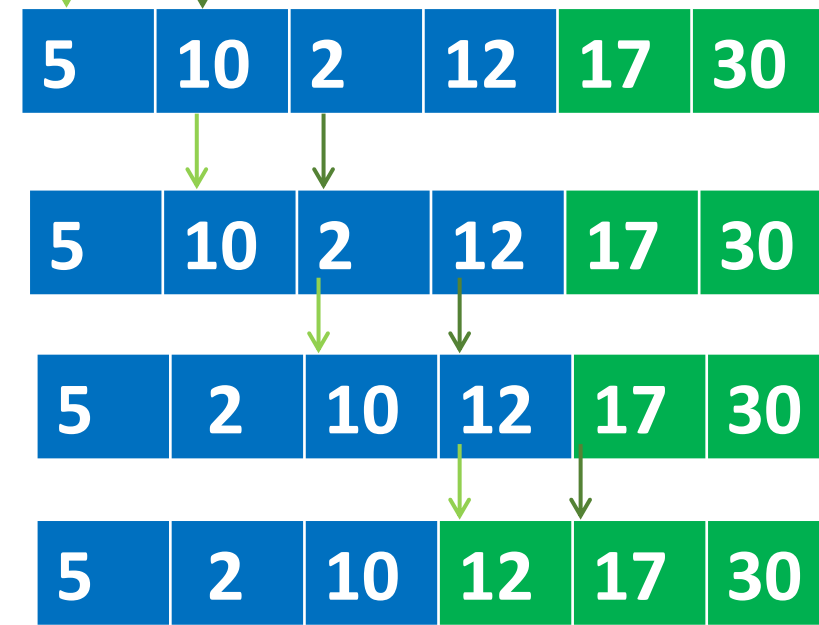
Exemple: T

12	10	5	30	2	17
----	----	---	----	---	----

Itération 2:



Itération 3:



Exemple: T

12	10	5	30	2	17
----	----	---	----	---	----

Itération 4:

5	2	10	12	17	30
2	5	10	12	17	30
2	5	10	12	17	30

Itération 5:

2	5	10	12	17	30
2	5	10	12	17	30

Aucun changement → arrêt de l'algorithme

```
void triBulle(int T[], int taille) {  
    for (int i = 0; i < taille - 1; i++)  
        for (int j = 0; j < taille - i - 1; j++)  
            if (T[j] > T[j + 1])  
                echanger(&T[j], &T[j + 1]);  
}  
  
void echanger(int *a, int *b) {  
    int tmp = *a;  
    *a = *b;  
    *b = tmp;  
}
```

i=0	T	12	10	20	25	0	35
		0	1	2	3	4	5
i=1	T	10	12	20	0	25	35
		0	1	2	3	4	5
i=2	T	10	12	0	20	25	35
		0	1	2	3	4	5
i=3	T	10	0	12	20	25	35
		0	1	2	3	4	5
i=4	T	0	10	12	20	25	35
		0	1	2	3	4	5

- Le tri à bulles est souvent enseigné en tant qu'exemple algorithmique, car son principe est simple.
- **Meilleur des cas:** le tableau est trié. Dans ce cas, nous n'avons à déplacer aucun élément.
- **Pire des cas:** le tableau est trié dans l'ordre inverse. Dans ce cas, la complexité est $O(n^2)$.

Avantages:

- Facile à comprendre et à implémenter.
- Peut s'arrêter plus tôt si le tableau devient trié avant la fin (version optimisée).
- **Inconvénients :**
- Très lent $O(n^2)$, même comparé à d'autres tris simples.
- Réalise beaucoup d'échanges inutiles.

Principe:

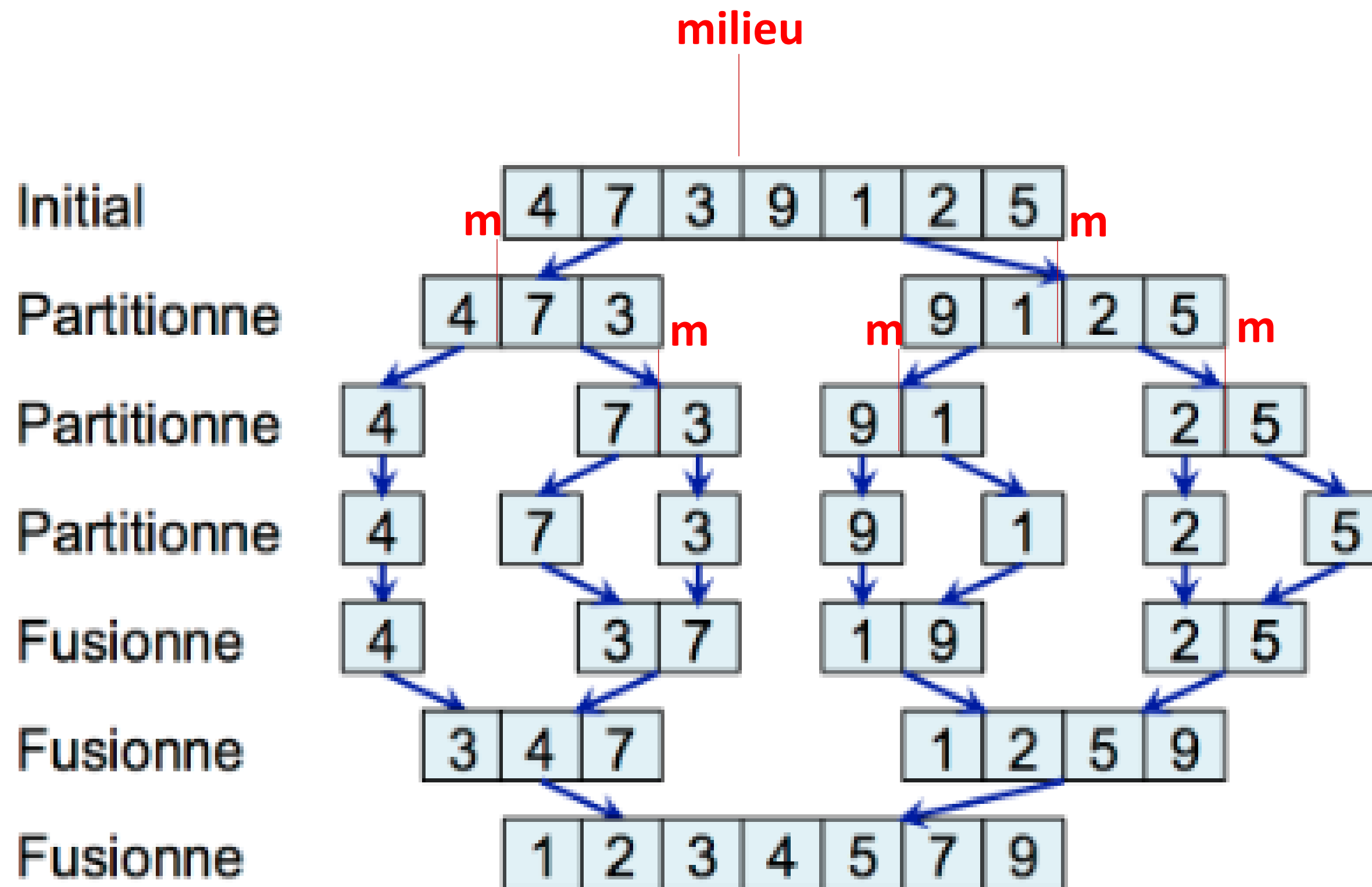
- Inventé en 1948 par Goldstine et Von Neumann.
- Applique le principe « **diviser pour régner** ».
- Basé sur l'idée que s'il y a deux suites d'éléments triés, il est très facile d'obtenir une troisième suite d'éléments triés, par « interclassement » (ou fusion) des deux précédentes suites. Le principe de ce tri consiste:
 - Diviser l'ensemble ou le tableau en deux sous-ensembles
 - Trier les deux sous ensembles.
 - Fusionner les deux sous ensembles

Principe:

Etant donné un tableau (ou une liste) de $T[1, \dots, n]$:

- Si $n = 1$, retourner le tableau T !
- Sinon :
 - Trier le sous-tableau $T[1 \dots n/2]$
 - Trier le sous-tableau $T[n/2 + 1 \dots n]$
 - Fusionner ces deux sous-tableaux. . .
- Il s'agit d'un algorithme “**diviser-pour-régner**”.

Principe:



Fonction **Tri_Fusion** (Var T : Tableau d'entiers, deb, fin: entier)

mil: entier

Début

Si deb < fin alors

mil $\leftarrow$ (deb + fin)/2

Tri_fusion (T,deb,mil);

Tri_fusion (T, mil+1, fin);

Fusion (T, deb, mil, fin);

Finsi

Fin

// Vérifie s'il y a plus d'un élément à trier

// Calcule le point milieu du tableau

// Trie récursivement la première moitié

// Trie récursivement la deuxième moitié

// Fusionne les deux moitiés triées

Procédure **Fusion** (var T : Tableau, deb, mil, fin : entier)

T1 : Tableau // Tableau temporaire pour la fusion

Début

i ← deb // i : début du premier sous-tableau

j ← mil + 1 // j : début du second sous-tableau

Pour k ← deb à fin faire // Étape 1 : Fusion des deux sous-tableaux dans T1 Si le 2e sous-tableau est épuisé

Si (j > fin) ou (i ≤ mil et T[i] ≤ T[j]) alors // ou si l'élément du 1er est plus petit

T1[k] ← T[i] // On prend l'élément du premier sous-tableau

i ← i + 1 // Avancer dans le 1er sous-tableau

Sinon

T1[k] ← T[j] // On prend l'élément du second sous-tableau

j ← j + 1 // Avancer dans le 2e sous-tableau

FinSi

FinPour

Copier(T, T1, deb, fin) // Copie les éléments de deb à fin

Fin

Procédure **Copier**(T, T1, deb, fin)

Début

Pour k allant de deb à fin faire

$T[k] \leftarrow T1[k]$

FinPour

Fin

Complexité:

- Le tri par fusion est un tri optimal sur les listes (tableaux), de complexité $O(n \log (n))$. Il s'agit de décomposer une liste en deux sous-listes chacune deux fois plus petites, de les trier séparément, puis de fusionner les résultats en une liste triée.
- Le nombre de comparaisons nécessaires est de l'ordre de **$O(n \log n)$** .

Avantages:

- Très rapide pour les grandes listes **$O(n \log n)$** .
- Stable (si deux éléments ont la même valeur, leur ordre relatif d'origine dans le tableau reste le même après le tri).

Inconvénients :

- Nécessite de la mémoire supplémentaire pour fusionner les sous-listes.
- Plus complexe à implémenter que les tris simples.