“ Algorithmiques avancée ”

2- Algorithmes de tri

Pr. Abdelhay HAQIQ (ahaqiq@esi.ac.ma)

Plan

- **Algorithmes de recherche**
- **Algorithmes de tri**
- **Comparaison des algorithmes de tri**

Plan

- **Algorithmes de recherche**
  - **Recherche séquentielle**
  - **Recherche dichotomique**
- Algorithmes de tri
- Comparaison des algorithmes de tri

Recherche séquentielle

- Le principe consiste à parcourir un tableau d'éléments dans l'ordre de ses indices jusqu'à ce qu'un élément recherché soit trouvé ou bien que la fin du tableau soit atteinte et l'élément recherché est alors inexistant.

- Exemple

5	10	4	2	7	15

  - Pour x =2, le programme affichera que ''2 existe''
  - Pour x =-1, le programme affichera que ''-1 n'existe pas''

Recherche séquentielle

```
int tri_sequentielle(int *tab, int n, int x)
{
    int i =0;
    bool trouve = false;

    while((i<n && !trouve))
    {
        if(tab[i] == x)
            trouve = true;
        i++;
    }
    return trouve;
}
```

Recherche dichotomique

- Un algorithme de recherche pour trouver la position d'un élément dans un **tableau trié**.
- Le principe consiste à comparer l'élément avec la valeur de la case au milieu du tableau ;
  - si la valeur est égale, la tâche est accomplie,
  - sinon on recommence dans la moitié du tableau pertinente.

Recherche dichotomique

- **Il y a 3 cas possibles:**
  - X < T[m], l'élément x s'il existe, il se trouve dans l'intervalle [premier... m-1]
  - x > T[m], l'élément de valeur x, s'il existe, se trouve dans l'intervalle [m+1... Dernier]
  - X = T[m], x est trouvé, la recherche est terminée
- La recherche dichotomique consiste à itérer ce processus jusqu'à ce que l'on trouve x ou que l'intervalle de recherche soit vide.

Recherche dichotomique

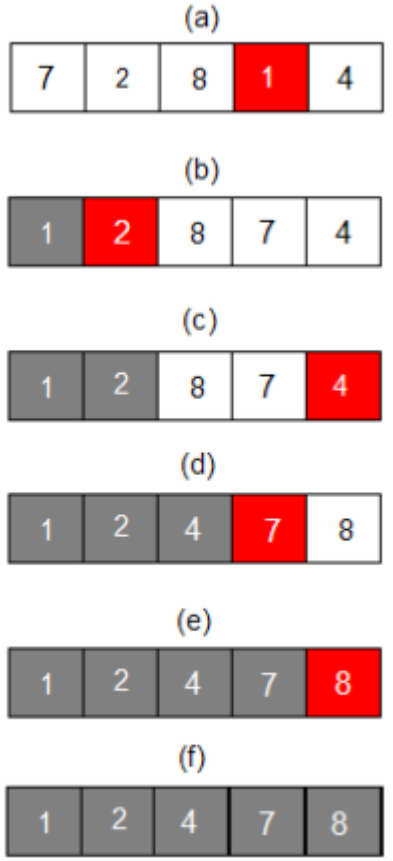
```c
void tri_dichotomique(int *tab, int n, int iRecherche)
{
    int iPremier = 0;
    int iDernier = n;
    int iTrouve = 0;
    int iMilieu;

    while ((iPremier <= iDernier) && (iTrouve == 0))
    {
        iMilieu = (iPremier + iDernier) / 2;
        if (tab[iMilieu] == iRecherche)
            iTrouve = 1;
        else
        {
            if (tab[iMilieu] > iRecherche)
                iDernier = iMilieu - 1;
            else
                iPremier = iMilieu + 1;
        }
    }
    if (!iTrouve)
       printf("La valeur %d n'appartient pas a la liste\n", iRecherche);
    else
       printf("La valeur %d appartient a la liste\n", iRecherche);
}
```

Plan

- Algorithmes de recherche
- **Algorithmes de tri**
  - **Tri par sélection**
  - **Tri à bulle**
  - **Tri par insertion**
  - **Tri rapide**
  - **Tri fusion**

Tri par sélection

- Le tri par sélection est un tri par comparaison.
- Le principe de tri de sélection est le suivant:
  - Rechercher le plus petit élément et l'échanger avec le premier élément t[1].
  - Rechercher le deuxième petit élément et l'échanger avec le deuxième élément t[2].
  - Faire la même chose avec le reste des éléments jusqu'à ce que le tableau soit trié.



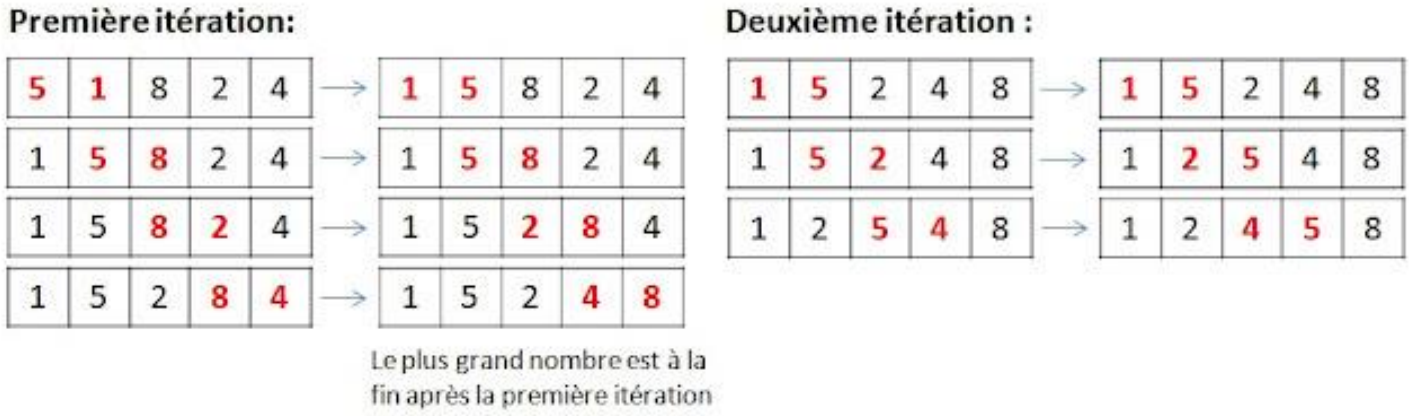
Tri par sélection

```
void tri_selection(int T[], int N)
{
    int i, j, tmp;
    for (i = 0; i < N - 1; i++)
        for (j = i + 1; j < N; j++)
            if (T[i] > T[j])
            {
                tmp = T[i];
                T[i] = T[j];
                T[j] = tmp;
            }
}
```

La complexité de l'algorithme est en $O(n^2)$

Tri à bulles

- Tri à bulles parcourt le tableau en comparant 2 cases successives, lorsqu'il trouve qu'elles ne sont pas dans l'ordre souhaité (croissant dans ce cas), il permute ces 2 cases.
- à la fin d'un parcours complet on aura le déplacement du maximum à la fin du tableau.
- En faisant cet opération N fois, le tableau sera donc trié .



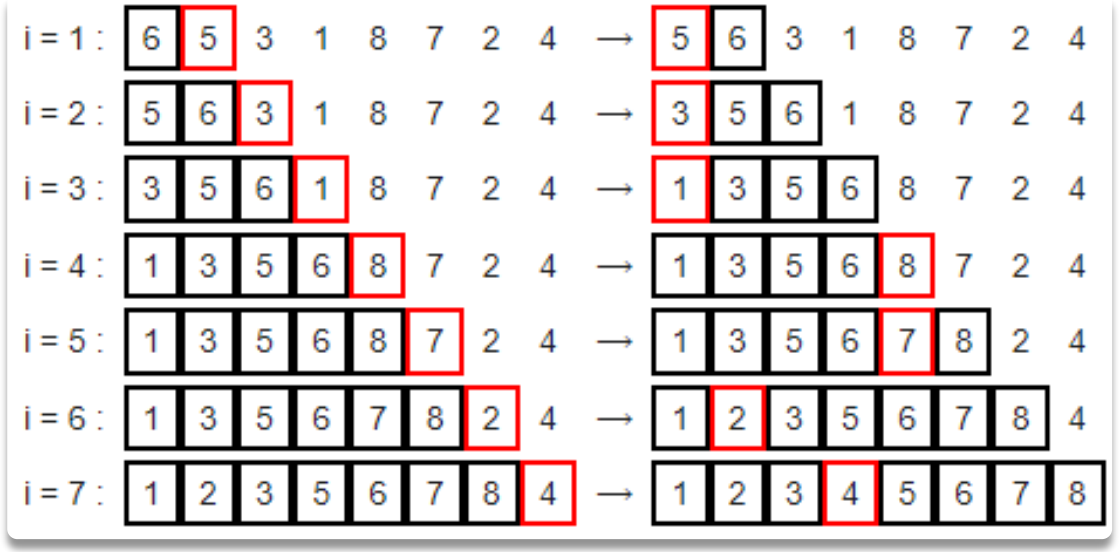
Tri à bulles

```
void tri_bulle(int T[], int N)
{
  int i, j, tmp;
  for (i = 0; i < N - 1; i++)
    for (j = 0; j < N - i - 1; j++)
    {
      if (tab[j] > tab[j + 1])
      {
        tmp = tab[j];
        tab[j] = tab[j + 1];
        tab[j + 1] = tmp;
      }
    }
}
```

La complexité de l'algorithme est en $O(n^2)$

Tri par insertion

- Le tri par insertion considère chaque élément du tableau et l'insère à la bonne place parmi les éléments déjà triés.
- Ainsi, au moment où on considère un élément, les éléments qui le précèdent sont déjà triés, tandis que les éléments qui le suivent ne sont pas encore triés.
- A chaque étape i :
  - le tableau T[0..(i-1)] est déjà trié
  - on insère T[i] à sa bonne place dans T[0..i]



Tri par insertion

```
void tri_insertion(int T[], int N)
{
    int i, j;
    int temp;

    for (i = 1; i < N; i++)
    {
        temp = T[i];
        j = i;
        while ((j > 0) && (temp < T[j - 1]))
        {
            T[j] = T[j - 1];
            j--;
        }
        T[j] = temp;
    }
}
```

La complexité de l'algorithme est en $O(n^2)$

Tri rapide

- L'algorithme consiste à placer un élément du tableau (appelé pivot) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui sont inférieurs au pivot soient à sa gauche et que tous ceux qui sont supérieurs au pivot soient à sa droite
  - Cette opération s'appelle le partitionnement.
- Pour chacun des sous-tableaux, on définit un nouveau pivot et on répète l'opération de partitionnement.
  - Ce processus est répété récursivement, jusqu'à ce que l'ensemble des éléments soit trié.

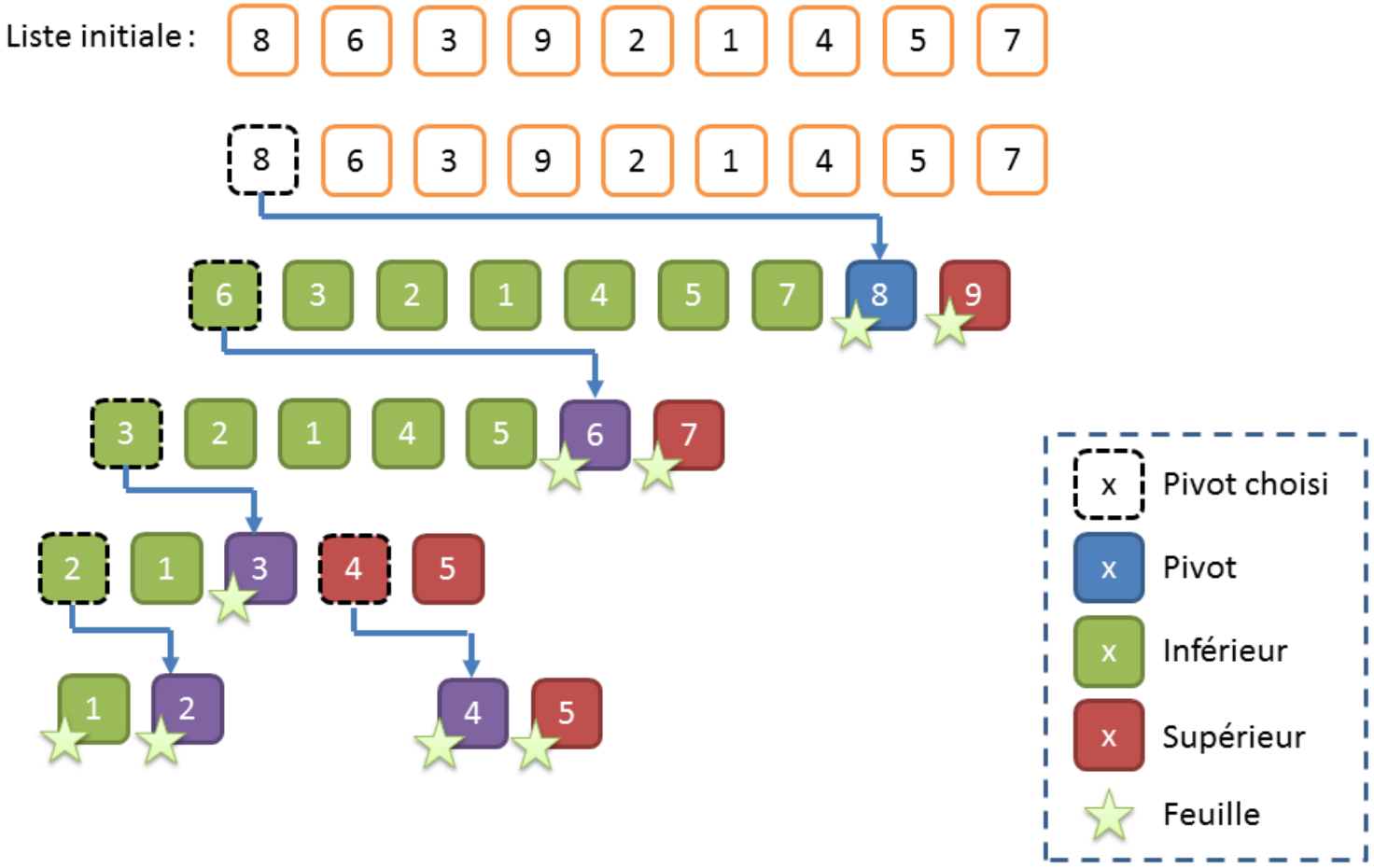
Tri rapide : Choix du Pivot

- Stratégies simples : premier élément ou dernier élément
- Stratégie efficace : pivot médian de trois éléments (le premier, le dernier et l'élément du milieu)
- Autre stratégie : élément aléatoire

Tri rapide

Tri rapide (Quick sort)

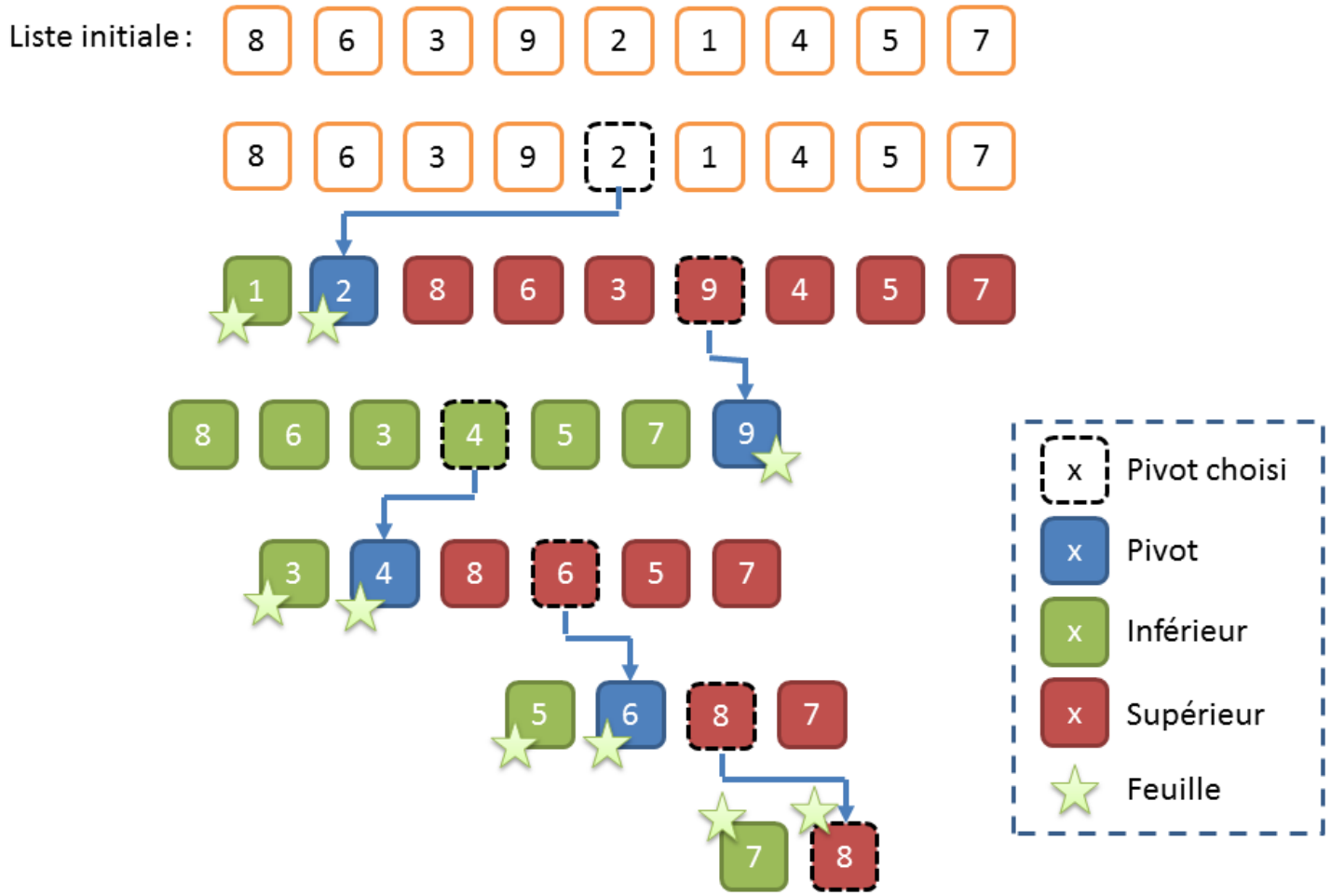
- pivot choisi en première position -



Tri rapide

Tri rapide (Quick sort)

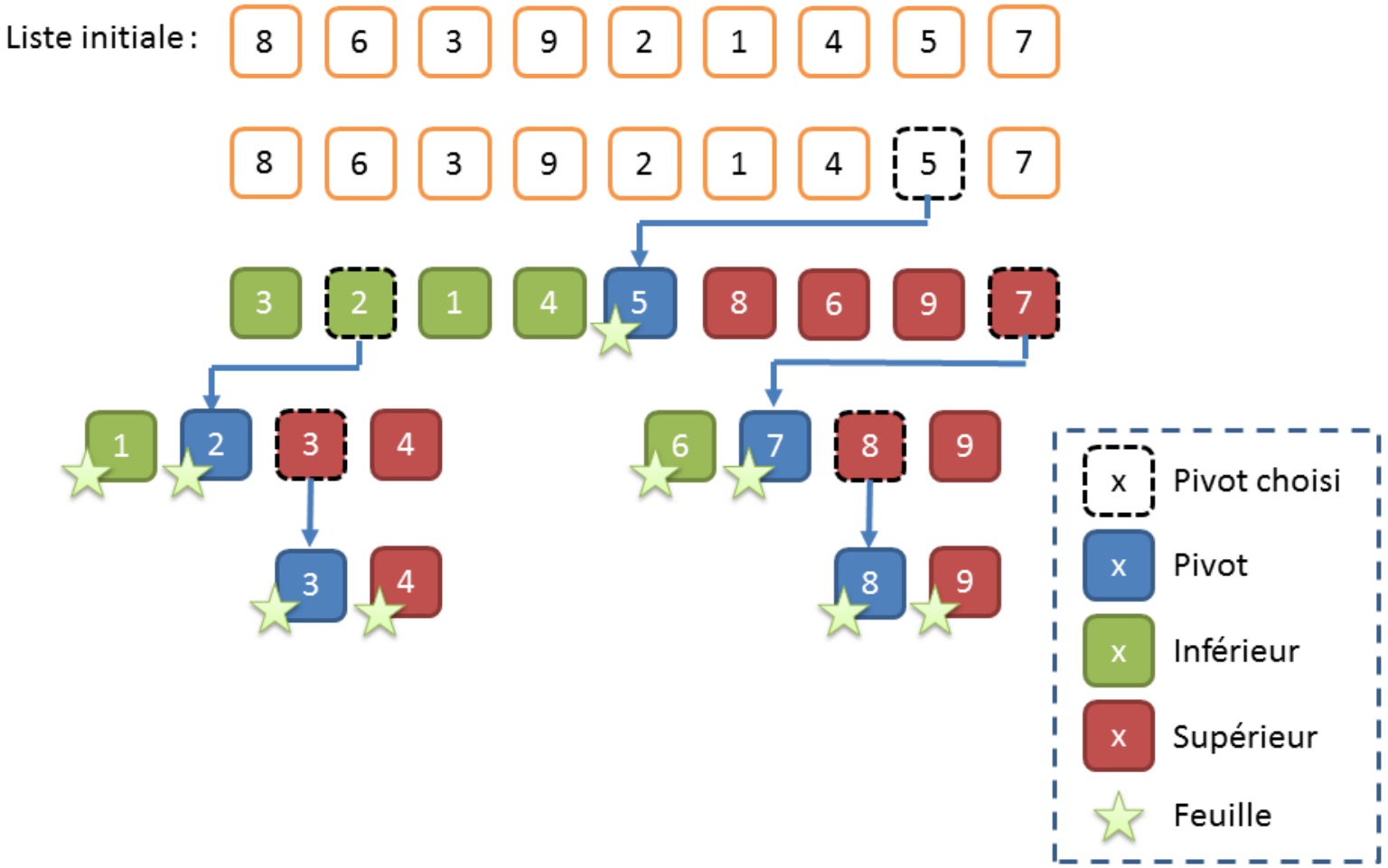
- pivot choisi au milieu -



Tri rapide

Tri rapide (Quick sort)

- pivot choisi aléatoirement et truqué -



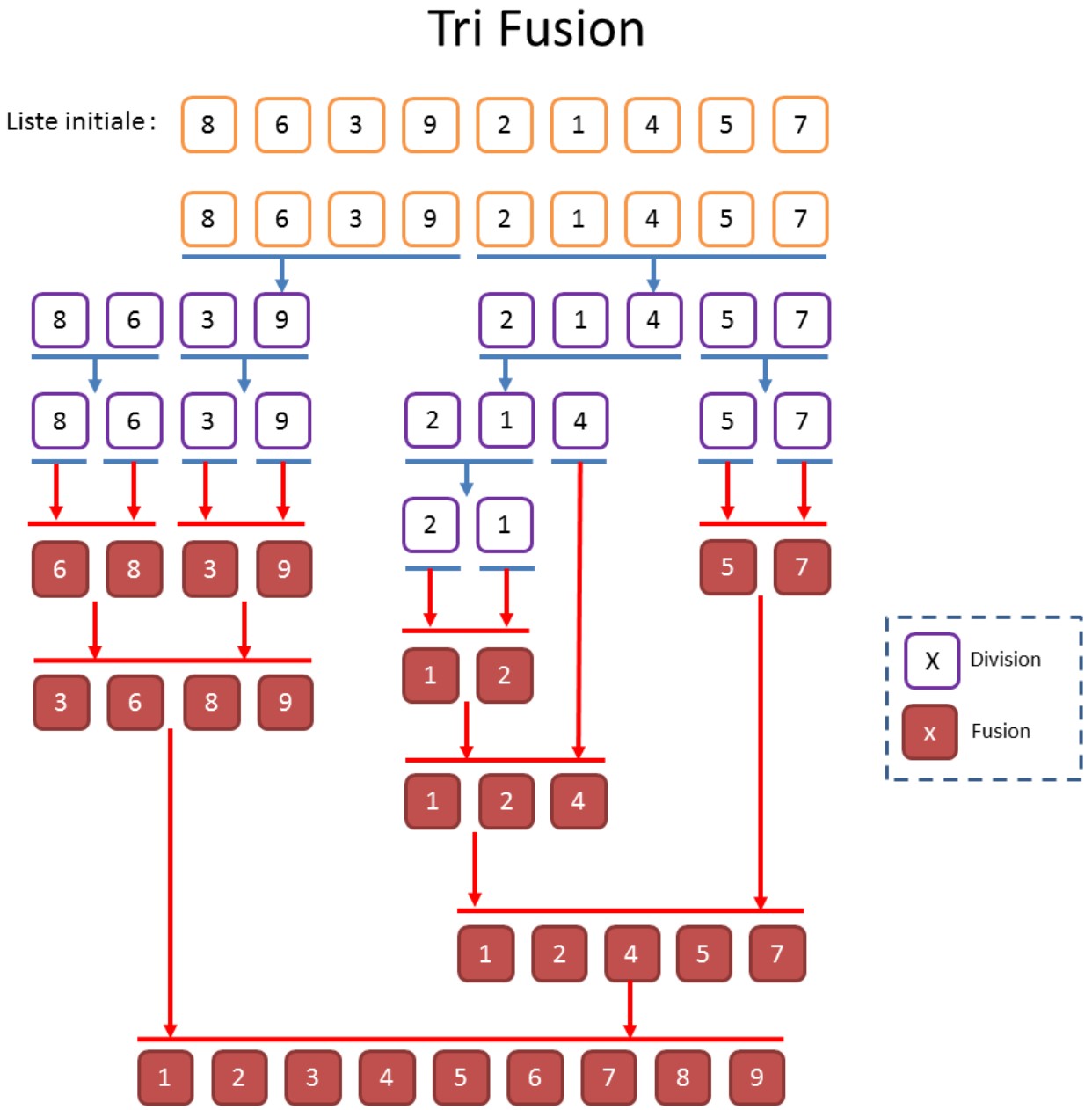
Tri rapide

```
void tri_rapide(int tab[], int first, int last) {
    int pivot, i, j;
    if(first < last) {
        pivot = first;
        i = first;
        j = last;
        while (i < j) {
            while(tab[i] <= tab[pivot] && i < last)
                i++;
            while(tab[j] > tab[pivot])
                j--;
            if(i < j) {
                permuter(&tab[i], &tab[j]);
            }
        }
        permuter(&tab[pivot], &tab[j]);
        tri_rapide(tab, first, j - 1);
        tri_rapide(tab, j + 1, last);
    }
}
```

Tri fusion

- **Tri fusion** ou **tri dichotomique** est basé sur la technique algorithmique diviser pour régner. L'opération principale de l'algorithme est la fusion, qui consiste à réunir deux listes triées en une seule
- Algorithme «diviser pour régner » :
  - Diviser : on divise les données en deux parties égales
  - Régner : on trie les deux parties en appliquant le tri fusion
  - Combiner : on fusionne les deux parties en un ensemble trié

Tri fusion



Tri fusion

```
// Tri par fusion
void triFusion(int tab[], int debut, int fin)
{
    if (debut < fin)
    {

        // Trouvez le point milieu pour diviser le
           tableau en deux moitiés
        int m = (debut + fin) / 2;

        triFusion(tab, debut, m);
        triFusion(tab, m + 1, fin);

        // Fusionnez les deux moitiés triées
        fusion(tab, debut, m, fin);
    }
}
```

```
void fusion(int tab[], int debut, int milieu, int fin)
    {
        int n1 = milieu - debut + 1;
        int n2 = fin - milieu;

        int G[n1], D[n2];

        for (int i = 0; i < n1; i++)
            G[i] = tab[debut + i];

        for (int j = 0; j < n2; j++)
            D[j] = tab[milieu + 1 + j];

        // maintient trois pointeurs, un pour chacun des deux tableaux et un pour
        // maintenir l'index actuel du tableau trié final
        int i, j, k;
        i = 0;
        j = 0;
        k = debut;

       .........
```

```
.........

while (i < n1 && j < n2)
    {
        if (G[i] <= D[j])
        {
            tab[k] = G[i];
            i++;
        }
        else
        {
            tab[k] = D[j];
            j++;
        }
        k++;
    }
```

```
.........

// Copiez tous les éléments restants du
   tableau non vide
while (i < n1)
{
    tab[k] = G[i];
    i++;
    k++;
}

while (j < n2)
{
    tab[k] = D[j];
    j++;
    k++;
}
}
```

Comparaison des algorithmes de tri

Algorithme de tri	Cas optimal	Cas moyen	Pire des cas
Tri par sélection	$O(n^2)$	$O(n^2)$	$O(n^2)$
Tri à bulle	$O(n)$	$O(n^2)$	$O(n^2)$
Tri par insertion	$O(n)$	$O(n^2)$	$O(n^2)$
Tri rapide	$O(n\log_2(n))$	$O(n\log_2(n))$	$O(n^2)$
Tri fusion	$O(n\log_2(n))$	$O(n\log_2(n))$	$O(n\log_2(n))$