# LES ALGORITHMES « DIVISER POUR MIEUX RÉGNER » ET LA RECURSIVITE

**1. Introduction : pourquoi « diviser pour mieux régner » ?**

Certains problèmes sont trop complexes pour être résolus directement. Une stratégie efficace consiste à **les découper en problèmes plus simples**, à résoudre chacun séparément, puis à **combiner les résultats**.

Cette stratégie s'appelle **« diviser pour mieux régner »**.

Elle est très utilisée en informatique et constitue une **application naturelle de la récursivité**.

**2. Principe général de la méthode**

Un algorithme de type « diviser pour régner » suit toujours les trois étapes suivantes :

1. **Diviser** le problème initial en plusieurs sous-problèmes plus petits.
2. **Résoudre** chaque sous-problème indépendamment.
3. **Combiner** les résultats partiels pour obtenir la solution finale.

Les sous-problèmes sont **du même type que le problème initial**, mais de taille plus petite.

**3. Lien avec la récursivité**

La méthode « diviser pour régner » est étroitement liée à la récursivité :

- résoudre un sous-problème revient souvent à **réappliquer le même algorithme** ;
- cela conduit naturellement à des **appels récursifs** ;
- lorsque plusieurs sous-problèmes sont générés, on parle de **récursivité multiple**.

👉 La récursivité permet donc d'implémenter simplement cette méthode.

**4. Exemple 1 : Recherche dichotomique (cas simple)**

**4.1 Idée générale (sans code)**

Pour chercher un élément dans un tableau trié :

- on compare l'élément recherché avec l'élément du milieu ;
- on élimine la moitié inutile du tableau ;
- on recommence sur la moitié restante.

On a bien :

- division du problème,
- résolution d'un sous-problème,
- mais **un seul sous-problème est traité**.

C'est un cas simple de « diviser pour régner ».

### 4.2 Version classique (itérative)

```
def recherche_dicho(t, x):
    gauche = 0
    droite = len(t) - 1
    while gauche <= droite:
        milieu = (gauche + droite) // 2
        if t[milieu] == x:
            return True
        elif t[milieu] < x:
            gauche = milieu + 1
        else:
            droite = milieu - 1
    return False
```

Cette version utilise une **boucle** pour répéter les divisions.

### 4.3 Version récursive

```
def recherche_dicho_rec(t, x, gauche, droite):
    if gauche > droite:
        return False
    milieu = (gauche + droite) // 2
    if t[milieu] == x:
        return True
    elif t[milieu] < x:
        return recherche_dicho_rec(t, x, milieu + 1, droite)
    else:
        return recherche_dicho_rec(t, x, gauche, milieu - 1)
```

Ici :

- chaque appel résout un problème plus petit,
- il n'y a **qu'un seul appel récursif** → récursivité simple.

En Python, il est possible de donner des **valeurs par défaut** aux paramètres. Cela permet d'appeler la fonction **plus simplement**, sans exposer les paramètres techniques aux élèves.

**Exemple avec la recherche dichotomique récursive**

```
def recherche_dicho_rec(t, x, gauche=0, droite=None):
    if droite is None:
        droite = len(t) - 1
    if gauche > droite:
        return False
    milieu = (gauche + droite) // 2
    if t[milieu] == x:
        return True
    elif t[milieu] < x:
        return recherche_dicho_rec(t, x, milieu + 1, droite)
    else:
        return recherche_dicho_rec(t, x, gauche, milieu - 1)
Appel simple :
recherche_dicho_rec([1, 3, 5, 7, 9], 5)
```

2️⃣ **Pourquoi ne pas mettre droite=len(t)-1 directement ?**

❌ Ceci est **interdit** :

def recherche_dicho_rec(t, x, gauche=0, droite=len(t)-1):

**Pourquoi ?**

- Les valeurs par défaut sont évaluées **au moment de la définition** de la fonction.
- Or t **n'existe pas encore** à ce moment-là.
- Cela provoque une erreur.

👉 La solution correcte est donc d'utiliser None comme valeur sentinelle.

## 5. Exemple 2 : Tri fusion (cas typique)

### 5.1 Principe du tri fusion

Le tri fusion est un algorithme emblématique du « diviser pour régner ».

Son fonctionnement est le suivant :

1. Diviser le tableau en deux sous-tableaux de taille à peu près égale.
2. Trier récursivement chaque sous-tableau.
3. Fusionner les deux tableaux triés en un seul tableau trié.

### 5.2 Observation essentielle

Trier une moitié du tableau, c'est **le même problème que trier le tableau entier**, mais en plus petit.

👉 C'est exactement ce qui justifie l'utilisation de la récursivité.

### 5.3 Schéma logique (sans code)

- Cas de base : un tableau de taille 0 ou 1 est déjà trié.
- Cas récursif :
    - trier la première moitié,
    - trier la seconde moitié,
    - fusionner les deux résultats.

Chaque appel récursif génère **deux appels récursifs** → récursivité multiple.

## 6. Récapitulatif : récursivité et « diviser pour régner »

| **Algorithme** | **Sous-problèmes** | **Type de récursivité** |
|---|---|---|
| Factorielle | 1 | Récursivité simple |
| Recherche dichotomique | 1 | Récursivité simple |
| Fibonacci | 2 | Récursivité multiple |
| Tri fusion | 2 | Récursivité multiple |

## 7. À retenir

Les algorithmes « diviser pour mieux régner » consistent à découper un problème en sous-problèmes du même type, à les résoudre récursivement, puis à combiner les résultats. Cette méthode est naturellement implémentée à l'aide de la récursivité, souvent multiple.

## 8. Conclusion pédagogique

- La méthode « diviser pour régner » donne du **sens** à la récursivité.
- La récursivité n'est pas une astuce de programmation, mais une **stratégie algorithmique**.
- Les versions itératives et récursives résolvent les mêmes problèmes, mais avec des approches différentes.