

Diviser pour régner

Fiche de synthèse

① LE PARADIGME

Définition 1 (Diviser pour régner) Un algorithme **diviser pour régner** procède en trois étapes :

1. **diviser** le problème en sous-problèmes plus petits de même nature ;
2. **régner** : résoudre chaque sous-problème récursivement, jusqu'à des cas assez simples pour être résolus directement ;
3. **combiner** les solutions des sous-problèmes en une solution du problème initial.

L'efficacité vient de la division : couper le problème en deux moitiés à chaque étape mène à $\log_2 n$ niveaux d'appels.

② RECHERCHE DICHOTOMIQUE

Dans un tableau **trié**, on compare l'élément cherché à celui du milieu : s'ils sont égaux, on l'a trouvé ; s'il est plus petit, on cherche dans la moitié gauche ; sinon, dans la moitié droite.

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

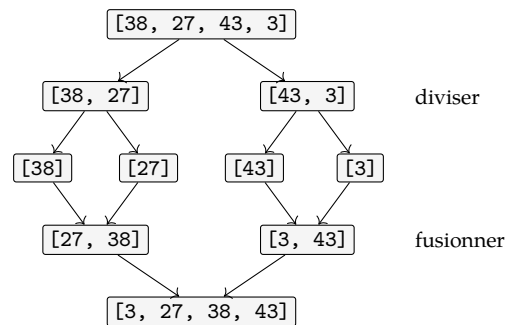
La fonction renvoie l'indice de x , ou -1 s'il est absent. La version récursive reçoit *gauche* et *droite* en paramètres et s'appelle sur la moitié retenue.

L'intervalle de recherche est divisé par deux à chaque étape : il y a au plus $\lfloor \log_2 n \rfloor + 1$ étapes, et la complexité est en $O(\log n)$. Pour un million d'éléments, 20 comparaisons suffisent, contre un million pour une recherche séquentielle.

③ TRI FUSION

1. Diviser : couper le tableau en deux moitiés.
2. Régner : trier chaque moitié par tri fusion.
3. Combiner : **fusionner** les deux moitiés triées.

Un tableau de zéro ou un élément est déjà trié : c'est le cas de base.



```
def fusion(gauche, droite):
    resultat = []
    i, j = 0, 0
    while i < len(gauche) and j < len(droite):
        if gauche[i] <= droite[j]:
            resultat.append(gauche[i])
            i += 1
        else:
            resultat.append(droite[j])
            j += 1
    resultat.extend(gauche[i:])
    resultat.extend(droite[j:])
    return resultat
```

```
def tri_fusion(tab):
    if len(tab) <= 1:
        return tab
    milieu = len(tab) // 2
    gauche = tri_fusion(tab[:milieu])
    droite = tri_fusion(tab[milieu:])
    return fusion(gauche, droite)
```

La fusion prend à chaque étape le plus petit des deux premiers éléments restants ; quand une liste est épuisée, **extend** ajoute à la fin du résultat les éléments restants de l'autre.

Il y a $\log_2 n$ niveaux de découpage, et la fusion parcourt les n éléments à chaque niveau : la complexité est en $O(n \log n)$ dans tous les cas, contre $O(n^2)$ pour les tris par sélection et par insertion. Le tri fusion n'est pas **en place** : il crée de nouvelles listes, ce qui demande de la mémoire supplémentaire.

④ EXPONENTIATION RAPIDE

$$a^n = \begin{cases} 1 & \text{si } n = 0, \\ (a^{n/2})^2 & \text{si } n \text{ est pair,} \\ a \times (a^{(n-1)/2})^2 & \text{si } n \text{ est impair.} \end{cases}$$

```
def puissance(a, n):
    if n == 0:
        return 1
```

```

demi = puissance(a, n // 2)
if n % 2 == 0:
    return demi * demi
else:
    return a * demi * demi

```

Pour n impair, $n // 2$ vaut $(n - 1)/2$. Le calcul demande $O(\log n)$ multiplications au lieu de $n - 1$.

	Diviser	Régner	Combiner
Dichotomie	couper l'intervalle en deux	chercher dans une moitié	renvoyer l'indice
Tri fusion	couper le tableau en deux	trier chaque moitié	fusionner
Puissance	diviser l'exposant par deux	calculer la puissance réduite	élever au carré, multiplier par a

📌 À retenir.

- Diviser pour régner : diviser, résoudre récursivement, combiner.
- La recherche dichotomique, dans un tableau trié, est en $O(\log n)$.
- Le tri fusion est en $O(n \log n)$ dans tous les cas, mais n'est pas en place.
- L'exponentiation rapide calcule a^n en $O(\log n)$ multiplications.