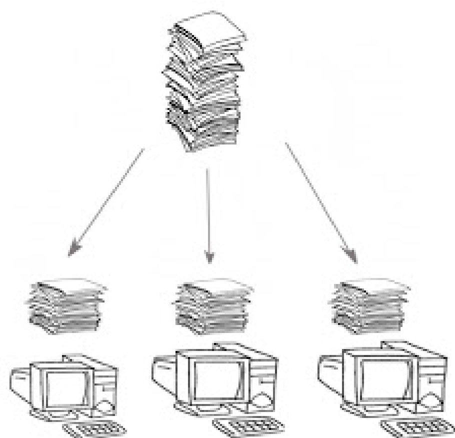
Chapitre 1

Diviser pour régner



1.1 Le principe « Diviser pour Régner »

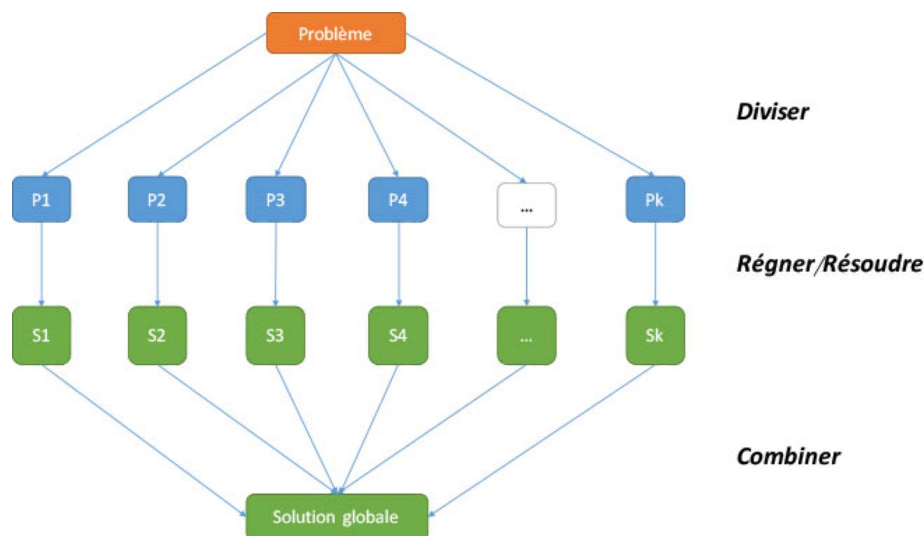
Le principe « Diviser pour Régner » est un principe consistant à **séparer un problème complexe en plusieurs sous-problèmes plus faciles** à résoudre. En algorithmique, cette méthode est souvent liée à la **récursivité**.

Pour résoudre un problème de taille N, un **algorithme récursif** fonctionne en général de la manière suivante :

- On construit à partir de notre entrée un problème de taille $N-1$
- On résoud le problème de taille $N-1$ (récursivité)
- On utilise cette solution pour résoudre le problème initial

La **méthode diviser pour régner** consiste, pour résoudre un problème de taille N, à :

- **Diviser** : on divise le problème d'origine en un certain nombre de sous-problèmes (généralement de taille $\frac{N}{2}$)
- **Régner** : on résout les sous-problèmes (plus faciles à résoudre que le problème d'origine)
- **Combiner** : les solutions des sous-problèmes sont combinées afin d'obtenir la solution du problème d'origine



On obtient en général **moins d'appels récursifs**. Dans certains cas la méthode diviser pour régner donne un algorithme de résolution plus rapide.

1.2 Exponentiation rapide

Problème. *Comment calculer x^n ?*

La méthode récursive classique se base sur les égalités suivantes (pour un nombre réel $x \neq 0$ et un entier naturel $n \geqslant 1$) :

$$\begin{cases} x^1 &= x \\ x^n &= x \times x^{n-1} \end{cases}$$

Question 01 Écrire une fonction récursive `puissance(x,n)` qui retourne le résultat de x^n.

Question 02 Combien faut-il d'opérations pour calculer 2^{10} ? 2^{37} ? 2^n ?

Question 03 Peut-on calculer 2^{2020} de cette manière ? Quel est le risque ?

Pour calculer plus efficacement x^n, on peut utiliser l'algorithme d'exponentiation rapide, basé sur les égalités suivantes :

$$\begin{cases} x^1 &= x \\ x^n &= \begin{cases} (x^2)^{\frac{n}{2}} & \text{si } n \text{ est pair} \\ x \times (x^2)^{\frac{n-1}{2}} & \text{si } n \text{ est impair} \end{cases} \end{cases}$$

C'est une méthode de type « diviser pour régner » :

- **Diviser** : on divise la puissance par 2
- **Régner** : on calcule récursivement la puissance de x^2 inférieure
- **Combiner** : on retourne le résultat, multiplié ou non par x selon la parité de n

Question 04 Implémenter cet algorithme dans une fonction `puissance_rapide(x,n)`.

Question 05 Combien faut-il d'opérations pour calculer 2^{10} ? 2^{37} ? 2^{2020} ? 2^n ?

Question 06 Si la pile d'éxécution est de taille 1000, quelle puissance de 2 peut-on calculer au maximum ?

1.3 Recherche du minimum d'une liste

Problème. *Comment déterminer le minimum d'une liste ?*

Question 07 Proposer une fonction `minimum(liste)` qui retourne le minimum de `liste`, de façon « naïve » (sans utiliser la fonction `min` !).

Question 08 Combien de comparaisons sont nécessaires pour déterminer le minimum d'une liste de cette façon ?

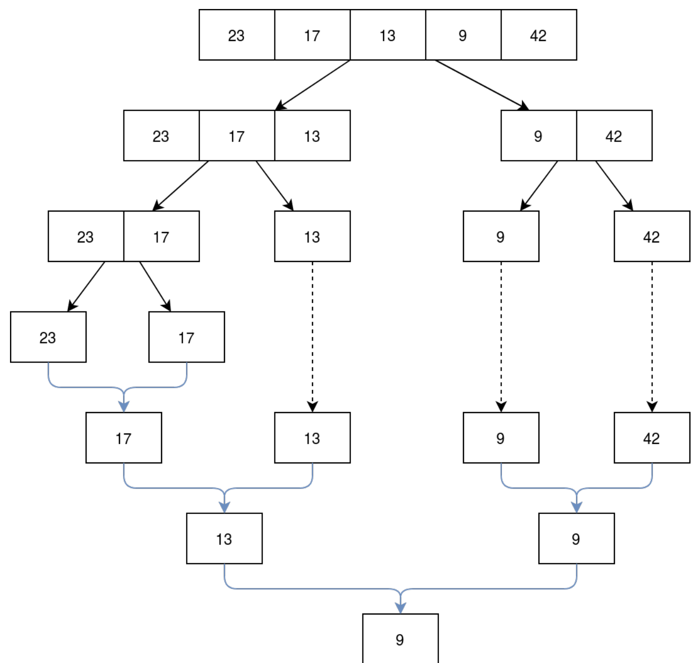
Une approche « diviser pour régner » est la suivante :

- **Diviser** : on divise la liste en deux sous-listes en la « coupant » en deux
- **Régner** : on calcule récursivement le minimum de chaque liste
- **Combiner** : on retourne le plus petit de ces deux minima

Algorithme

```
Fonction Minimum(liste, debut, fin):
Si debut == fin:
    Retourner liste[debut]
Sinon:
    milieu = (d+f) // 2
    x = Minimum(liste, debut, milieu)
    y = Minimum(liste, milieu + 1, fin)
    Si x < y:
        Retourner x
    Sinon:
        Retourner y
```

Voici un exemple avec la liste `[23,7,13,9,45]` :



Question 09 Déterminer le minimum de la liste `[23,12,4,56,35,57]` avec cette méthode.

Question 10 Avec cette méthode, combien de comparaisons sont nécessaires pour retourner le minimum d'une liste de longueur 10 ? 20 ? 100 ? n ?

Question 11 Implémenter cet algorithme en Python dans une fonction `minimum_dpr(liste, debut, fin)`.

☞ *On utilisera des valeurs par défaut pour les paramètres `debut` et `fin`*

Pour tester nos deux programmes et comparer leur rapidité, on peut générer par exemple 1000 listes de N nombres aléatoires. On compte le temps mis par chacune des fonctions pour déterminer le minimum des 1000 listes, et on en déduit le temps moyen de recherche d'un minimum pour une liste de N éléments en divisant ce temps par 1000.
En faisant cela pour chacune des deux fonctions précédentes, on peut alors estimer quelle fonction est la plus rapide.

Question 12 Écrire une fonction `test(N)` dont le rôle est de retourner le temps moyen de la détermination du minimum d'une liste pour chacune des fonctions précédentes.

Question 13 Compléter le tableau suivant grâce à la fonction `test` :

Temps moyens de recherche d'un minimum (en secondes)

Nombre d'éléments N de la liste	Minimum « naïf »	Minimum « diviser pour régner »
100		
1 000		
10 000		
20 000		

1.4 Tri fusion

Problème. *Comment trier une liste ?*

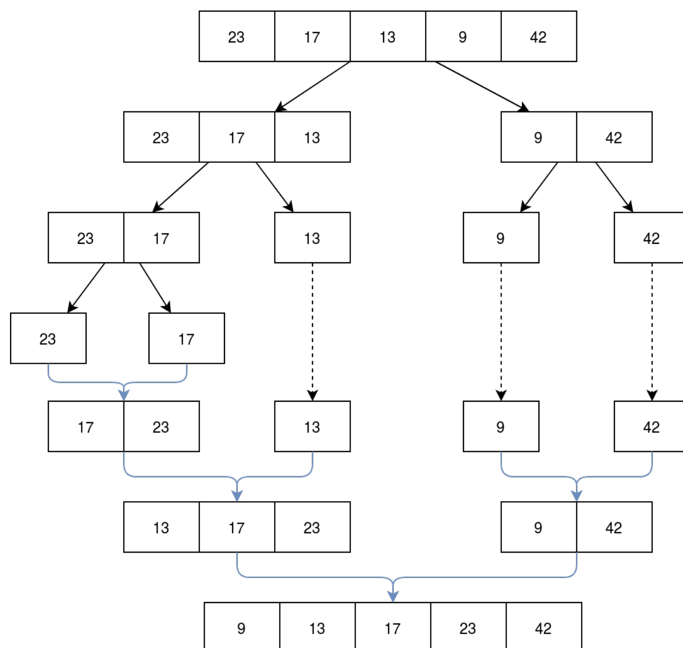
La solution à ce problème a déjà été rencontrée en classe de Première ! Nous avons même vu deux solutions :

- le tri par sélection
- le tri par insertion

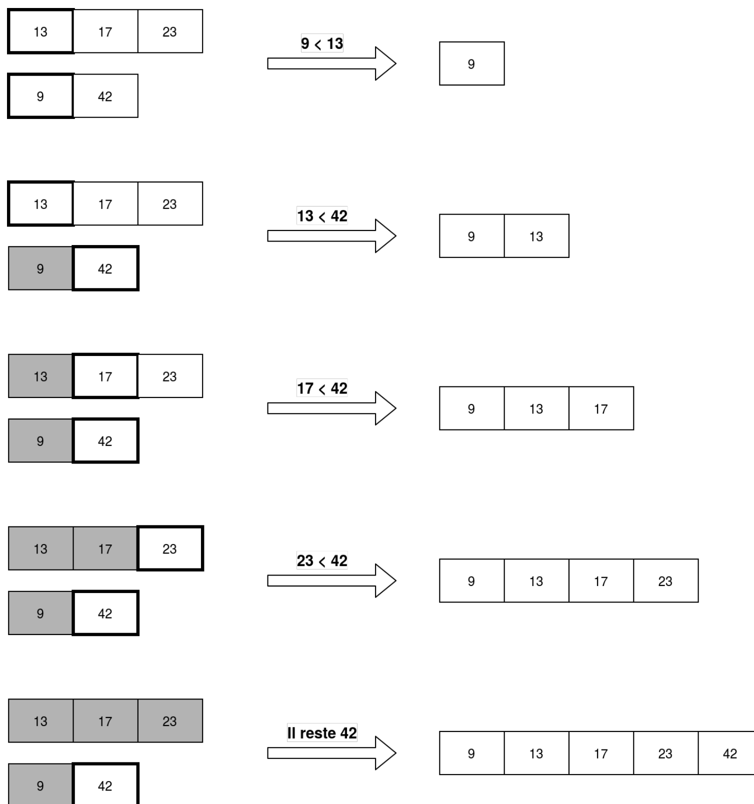
Il existe un très grand nombre de façon de trier une liste, mais c'est le **tri fusion** qui nous intéresse ici. Cette façon de trier n'est pas du tout naturelle, en revanche elle est bien plus rapide que les tris par sélection et insertion, notamment sur des listes de grande taille.

L'idée est identique à celle de la recherche d'un minimum, et fait intervenir le principe de « diviser pour régner » : on divise la liste en sous-listes, puis on fusionne les résultats de sorte à obtenir des listes triées de plus en plus grandes.

Voici un exemple avec la liste `[23,7,13,9,45]` :



L'idée qui permet d'avoir une fusion efficace (en bleu) repose sur le fait que les listes fusionnées sont **triées**. En effet, pour fusionner deux listes triées, il suffit de les parcourir de gauche à droite, les éléments les plus petits étant au début de la liste. Par exemple, la dernière fusion se décompose de la manière suivante :



Question 14 Trier la liste `[23,12,4,56,35,57]` avec cette méthode.

Question 15 Combien de découpages sont réalisés pour une liste de 10 éléments ? 100 éléments ? n éléments ?

Question 16 Si le tri par fusion d'une liste de n éléments nécessites d découpages et f fusions, combien de découpages et de fusions sont nécessaires pour une liste de longueur $2n$?

Question 17 Écrire une fonction `fusion(L1,L2)` qui fusionne deux listes triées `L1` et `L2`.

Question 18 Écrire une fonction `triFusion(liste)` qui retourne la liste `liste` triée avec la méthode de tri fusion.

Question 19 Comparer les méthodes de tri par sélection et par fusion en complétant le tableau suivant :

Temps moyens de tri d'une liste (en secondes)

Nombre d'éléments N de la liste	Tri par sélection	Tri par fusion
100		
1 000		
10 000		
20 000		

1.5 Exercices

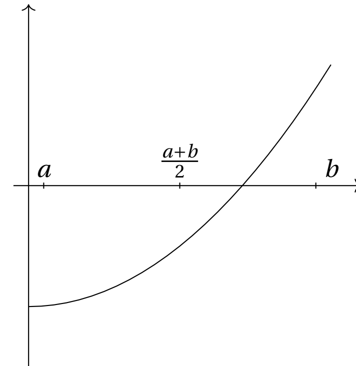
Exercice 01 ***Dichotomie***

On cherche à déterminer une solution d'une équation du type $f(x) = 0$, où f est une fonction continue sur un intervalle $[a; b]$. On suppose que $f(a) \times f(b) < 0$, de sorte que $f(a)$ et $f(b)$ sont de signes contraires.

Le principe est de rechercher cette racine **par dichotomie** :

- On calcule $m = \frac{a+b}{2}$
- Si $f(a) \times f(m) < 0$, alors f s'annule sur $]a; m[$
- Si $f(a) \times f(m) > 0$, alors f s'annule sur $]m; b[$

On répète ces calculs jusqu'à obtenir un encadrement aussi petit que l'on veut de la solution approchée.



1. Si $[a; b] = [0; 1]$, combien faut-il d'étapes pour obtenir une solution approchée à 10^{-5} près ?
2. Écrire une fonction `dichotomie(f,a,b)` qui retourne la valeur approchée de la solution de l'équation $f(x) = 0$ à 10^{-10} près sur l'intervalle $[a; b]$.