6.1 Introduction et premier exemple

6.1.1 Introduction

Le diviser pour régner est une méthode algorithmique basée sur le principe suivant :

On prend un problème (généralement complexe à résoudre), on divise ce problème en une multitude de petits problèmes, l'idée étant que les "petits problèmes" seront plus simples à résoudre que le problème original. Une fois les petits problèmes résolus, on recombine les "petits problèmes résolus" afin d'obtenir la solution du problème de départ.

Le paradigme "diviser pour régner" repose donc sur 3 étapes :

- DIVISER : le problème d'origine est divisé en un certain nombre de sous-problèmes
- RÉGNER : on résout les sous-problèmes (les sous-problèmes sont 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.

Les algorithmes basés sur le paradigme "diviser pour régner" sont souvent des algorithmes récursifs.

6.1.2 Un exemple pour comprendre

L'objectif est ici d'utiliser la méthode "diviser pour régner" afin de calculer le maximum d'un tableau de nombres.
En adoptant le paradigme « diviser pour régner », l'idée pour résoudre cette question est de calculer récursivement le maximum de la première moitié du tableau et celui de la seconde, puis de les comparer. Le plus grand des deux sera le maximum de tout le tableau. La condition d'arrêt à la récursivité sera l'obtention d'un tableau à un seul élément, son maximum étant bien sûr la valeur de cet élément.
Voici donc les trois étapes de la résolution de ce problème via cette méthode :

- Diviser le tableau en deux sous-tableaux en le « coupant » par la moitié.
- Calculer récursivement le maximum de chacun des sous-tableaux. Arrêter la récursion lorsque les tableaux n'ont plus qu'un seul élément.
- Retourner le plus grand des deux maximums précédents.

Exercice 6.1
Implémenter en python une telle fonction. Pour vous aider, voici le début du code, commenté :

```
def maxi_r(tableau, debut = 0,fin = -1):
    """
    debut et fin sont des indices
    debut est initialise a 0, fin a len(tableau) - 1
    """
```

```
    if fin == -1: # pour le premier appel
        fin = .....
    if debut == fin:
        return tableau[.....]
    milieu = (debut + fin) // 2
    return max(.................,.................)
```

6.1.3 Second exemple : le tri dichotomique

Rappelons que l'idée principale de la recherche dichotomique consiste à délimiter une portion du tableau tab dans laquelle la valeur v peut encore se trouver, avec deux indices g et d.

On compare la valeur au centre de cet intervalle avec la valeur v et, selon le cas, on signale qu'on a trouvé la valeur v ou on se déplace vers l moitié gauche ou la moitié droite.
☛ Il s'agit bien là d'une instance de l'idée « diviser pour régner », car on réduit le problème de la recherche dans l'intervalle [g ;d] à celui de la recherche dans un intervalle plus petit.
Voici donc les trois étapes de la résolution de ce problème via la méthode « diviser pour régner » :

- Diviser la liste en deux sous-listes de même taille (à un élément près) en la "coupant" par la moitié.
- Rechercher récursivement la présence de l'élément recherché dans la "bonne" des deux sous-listes après l'avoir comparé à l'élément situé au milieu de la liste.
- Pas de résultats à combiner puisque l'on ne « travaille » que sur l'une des deux sous-listes.

Exercice 6.2
Implémenter en python une telle fonction. Pour vous aider, voici le début du code, commenté :

```
def dichotomie_r(tableau,v,g = 0,d = -1):
  """
  g et d sont des indices
  g est initialise a 0, d a len(tableau) - 1
  la fonction renvoie l'indice de l'élement cherché si
  l'element est present
  Elle renvoie None sinon
  """
    if d == -1: # pour le premier appel
        d = .....
    if g > d :
        return None
      milieu = (g + d) // 2
      if tableau[milieu] < v:
        return ........
      elif tableau[milieu] > v:
        return .....
      else :
        return .......
```