

## Algorithmique

### TD n° 2 : Diviser pour régner

#### Exercice 1 : point fixe

On veut résoudre le problème suivant : l'entrée est un tableau  $t[1 \dots n]$  trié par ordre croissant  $\mathbf{t}$  de  $n$  entiers (relatifs) *tous distincts*, et la sortie est une valeur  $i$  telle que  $t[i] = i$  s'il en existe une, et le message "n'existe pas" sinon.

1. Répartissez vous en binômes de deux étudiants. Dans chaque binôme, un étudiant choisit un tableau  $t$  de taille  $n = 15$  qui contient des entiers tous distincts et avec au moins une entrée  $i$  telle que  $t[i] = i$ ; l'autre étudiant lui pose des questions du type "que vaut  $t[j]$  ?" pour diverses valeurs de  $j$ . Le jeu se termine lorsque le deuxième étudiant a trouvé  $i$  tel que  $t[i] = i$ . La valeur gagnée par le premier étudiant est le nombre de questions posées. Recommencer en inversant les rôles de  $A$  et de  $B$ .
2. Donner un algorithme *diviser pour régner* qui prend en entrée un tableau trié  $\mathbf{t}$  de 15 entiers (relatifs) *tous distincts*, et qui décide s'il existe un indice  $i$  tel que  $\mathbf{t}[i] = i$ . Quelle est sa complexité ?
3. Donner un algorithme *diviser pour régner* qui prend en entrée un tableau trié  $\mathbf{t}$  de  $n$  entiers (relatifs) *tous distincts*, et qui décide s'il existe un indice  $i$  tel que  $\mathbf{t}[i] = i$ .
4. Analyser sa complexité.

#### Exercice 2 : minimum et maximum

Soit  $\mathbf{t}$  un tableau de 5 éléments (comparables entre eux).

1. Montrer que le calcul du minimum de  $\mathbf{t}$  demande au moins 4 comparaisons.
  2. Proposer un algorithme optimal. Mêmes questions pour le maximum. Généraliser avec  $n$  au lieu de 5.
- On considère maintenant l'algorithme (écrit en Python) suivant, qui étant donné un tableau  $\mathbf{t}$  et deux indices  $g$  et  $d$  retourne un couple d'entier.

```
def minMax2(t,g,d):
    if g == d: return t[g],t[g]
    else:
        if t[g] < t[g+1]: return t[g],t[g+1]
        else: return t[g+1],t[g]

def minMax(t,g,d):
    if d <= g + 2 : return minMax2(t,g,d)
    milieu = (g+d)//2
    ming, maxg = minMax(t,g, milieu)
    mind, maxd = minMax(t,milieu, d)
    return min(ming, mind), max(maxg, maxd)
```

3. Montrer que l'algorithme calcule le minimum et le maximum d'un tableau  $\mathbf{t}$  si on l'appelle avec les paramètres 0 et  $\text{len}(\mathbf{t})$
4. Combien de comparaisons fait-il (pour un tableau de taille  $n = 2^k$ ) ? Est-ce contradictoire avec la question 2 ?
5. Proposer un algorithme non récursif aussi efficace.
6. Programmer l'algorithme récursif et l'algorithme non récursif. Comparer leur temps d'exécution sur des tableaux de grandes tailles.

**Rappel du cours** Le théorème suivant (*Master Theorem*) permet de résoudre les récurrences les plus fréquentes dans l'analyse des algorithmes de type *diviser pour régner*.

**Théorème.** Soient deux rationnels  $a \geq 1$  et  $b > 1$ . Soit  $f(n)$  une fonction positive. On considère la fonction  $t(n)$  définie par :

$$t(n) = \begin{cases} a \cdot t(\frac{n}{b}) + f(n) & \text{si } n > 1 \\ \Theta(1) & \text{si } n = 1 \end{cases}.$$

où  $\frac{n}{b}$  peut aussi désigner  $\lceil \frac{n}{b} \rceil$  ou  $\lfloor \frac{n}{b} \rfloor$ .

Alors on a :

1. Si  $f(n) = O(n^{\log_b(a)-\varepsilon})$  pour  $\varepsilon > 0$ , on a :  $t(n) = \Theta(n^{\log_b a})$ .
2. Si  $f(n) = \Theta(n^{\log_b a})$ , on a :  $t(n) = \Theta(n^{\log_b a} \cdot \log n)$ .
3. Si  $f(n) = \Omega(n^{\log_b(a)+\varepsilon})$  pour  $\varepsilon > 0$  et si il existe  $c < 1$  tel que  $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$  pour  $n$  assez grand, alors  $t(n) = \Theta(f(n))$ .

### Exercice 3 :

Donner les bornes asymptotiques obtenues grâce au Master Theorem pour les fonctions suivantes, sachant qu'elles sont constantes non nulles pour  $n \leq 2$  :

- $A(n) = A(\frac{9n}{10}) + n$
- $B(n) = 2 B(\frac{n}{4}) + \sqrt{n}$
- $C(n) = 7 C(\frac{n}{12}) + \log_2 n$
- $D(n) = 2 D(\frac{n}{2}) + \frac{n}{\log_2 n}$
- $E(n) = 5 E(\frac{n}{3}) + n^2$
- $F(n) = 4 F(\frac{n}{2}) + n^2 \log_2 n$

On travaille à concevoir un algorithme qui, pour une entrée de taille  $n$ , fait un appel récursif sur une partie des entrées, de taille  $n/2$ , puis fait un travail de complexité  $f(n)$ . Que doit valoir  $f(n)$  pour que la complexité soit sous-linéaire ? Et si on remplace  $n/2$  par  $n/10$  ? Et si on remplace  $n/2$  par  $9n/10$  ?

### Exercice 4 :

On veut fusionner  $k$  tableaux d'entiers triés. Chaque tableau à taille  $n$ . Un algorithme naïf est de fusionner le premier tableaux avec le deuxième, de fusionner le résultat avec le troisième, etc.

1. Analyser la complexité de cet algorithme (en terme de  $k$  et  $n$ ).
2. Donner un algorithme *diviser pour régner* et analyser sa complexité.