

## TD n°2

### Diviser pour Régner

#### Le théorème de résolution des récurrences

Le theoreme suivant permet de resoudre les recurrences les plus frequentes dans l'analyse des algorithmes de type diviser-pour-regner. Soit une fonction  $T$  qui represente la complexite d'un algorithme sur une donnee de taille  $n$  de la forme

$$T(n) = aT\left(\frac{n}{b}\right) + f(n):$$

Dans chacun des cas decrits ci-apres, la fonction  $T$  est de l'ordre de grandeur suivant :

**Cas “ $f$  petit”** : Si  $f(n) = 0$  ou, plus generalement,  $f(n) = O(n^{\log_b(a)-\epsilon})$  alors  $T(n) = O(n^{\log_b(a)})$

**Cas “ $f$  grand”** : Si  $f(n) = O(n^{\log_b(a)+\epsilon})$  et si  $af(\frac{n}{b}) \leq cf(n)$  avec  $c < 1$  alors  $T(n) = O(f(n))$

**Cas intermédiaire** : Si  $f(n) = O(n^{\log_b(a)})$  alors  $T(n) = O(n^{\log_b(a)} \log n)$

#### 1 Mise en bouche

**Exercice 1** En fonction de  $n$ , combien de lignes seront appees par l'appel de la fonction suivante ? On supposera que  $n$  est une puissance de deux. Ecrire et resoudre la recurrence.

```
void f(int n){
    if (n > 1){
        System.out.println("Encore une fois");
        f(n/2);
        f(n/2);
    }
}
```

**Exercice 2** Donner les bornes asymptotiques pour les fonctions suivantes sachant que pour  $n \leq 2$  elles sont constantes :

```
{ f(n) = f(9n-10) + n
{ g(n) = 2g(n-4) + √n
{ h(n) = h(√n) + 1
```

**Exercice 3** On cherche la somme d'un tableau  $B$  de  $n$  elements entiers.

Ecrire un algorithme de type diviser-pour-regner qui resout ce probleme. Analyser sa complexite.

#### 2 Fibonacci

**Exercice 4**

```
int fibo(int n) {
    if (n==0 || n==1) return n;
    else return fibo(n-1)+fibo(n-2);
}
```

Expliquer pourquoi l'algorithme naïf du calcul du  $n$ -ieme terme de la suite Fibonacci est inefficace et ne peut pas être vu comme un diviser pour regner.

**Exercice 5** Soit  $x$  et  $n$  deux entiers, donner un algorithme efficace permettant de calculer  $x^n$ . Et donner sa complexité.

**Exercice 6** A l'aide d'une matrice  $2 \times 2$  et de l'algorithme précédent, déduire un algorithme qui calcule le  $n$ -ième terme de la suite de Fibonacci en temps logarithmique.

### 3 Médiane et rang

**Exercice 7**  $m$  est la **médiane** d'un tableau  $T$  si  $T$  contient autant d'éléments supérieurs à  $m$  que d'inférieurs. Plus précisément,  $T$  doit pouvoir être partitionné en deux sous-tableaux  $T_1$  et  $T_2$ , tels que

- {  $T_1$  ne contient que des éléments  $\leq m$
- {  $T_2$  ne contient que des éléments  $\geq m$
- { si  $|T|$  est pair  $|T_1| = |T_2|$  sinon  $|T_1| = |T_2| + 1$
- { si plusieurs éléments vérifient les points ci-dessus,  $m$  est le plus petit d'entre eux

On suppose que l'on a une fonction  $\text{MÉDIANE}(T)$  qui renvoie l'élément médian du tableau  $T$  en temps  $O(n)$ , où  $n = |T|$ .

Montrer que l'on peut construire une fonction  $\text{RANG}(T; i)$ , qui renvoie le  $i^{\text{ème}}$  élément du tableau  $T$ , en temps  $O(n)$  seulement.

Penser à une méthode "diviser pour régner".

### 4 Pavage

La donnée du problème est une grille de taille  $2^k \times 2^k$  dont une case est *marquée*. On se propose de **paver** (recouvrir) la grille avec des tuiles de 3 cases en forme de L, de sorte que chaque case soit couverte par une et une seule tuile, sauf la case marquée au début.

Ce n'est pas *a priori* impossible car  $2^{2k} \equiv 1 \pmod{3}$  mais... est-ce possible ?

**Exercice 8** En coupant le carré en quatre carrés de taille  $2^{k-1} \times 2^{k-1}$  montrez comment on peut construire un pavage en se ramenant à quatre instances plus petites du problème.

**Exercice 9** Proposez une fonction récursive  $\text{PAVAGE}(k; X; Y; x; y)$  pour paver un carré, ou

- { Vous travaillez sur un carré de taille  $2^k \times 2^k$  dont le coin supérieur gauche est en  $(X; Y)$
- { Le point marqué est en  $(x; y)$  (avec comme precondition  $X \leq x \leq X + 2^k$  et  $Y \leq y \leq Y + 2^k$ )
- { Le carré de départ est de taille  $2^K \times 2^K$ , on appelle donc  $\text{PAVAGE}(K; 0; 0; x; y)$

Vous disposez d'une primitive  $\text{POSERTUILE}$  pour définir le pavage (vous êtes libres de définir sa syntaxe : elle pose une tuile aux coordonnées indiquées et avec l'orientation indiquée).

**Exercice 10** Quel est le temps d'exécution de votre algorithme sur une entrée de taille  $2^k \times 2^k$  ? Justifiez d'après le théorème de résolution de récurrences.