Université de Carthage
ENICarthage

Algorithmique Avancée & Complexité

TD5 – Analyse des Algorithmes Récursifs

Exercice 1

Déterminer de deux façons différentes les complexités respectives aux équations de récurrence suivantes :

a. $T(n) = 4T(n/2) + n$; $T(1) = 1$

1ère Méthode : application du théorème de résolution des récurrences.

On a :

$$\mathbf{a = 4 \;, b = 2 \Rightarrow \log_2 4 = 2 \quad et \quad f(n) = O(n)}$$

$$\mathbf{\Rightarrow f(n) = O(n^{(\log_2 4)-1}) \Rightarrow 1^{er}\ cas} \quad \boxed{\mathbf{T(n) = \theta(n^2)}}$$

2ème Méthode : Développement des récurrences $n= b^p= 2^p$

$$\begin{aligned}
\mathbf{T(n)} &\mathbf{= 4T(n/2) + n} \\
&\mathbf{= 4(4T(n/4) + n/2) + n} \\
&\mathbf{= 4(4(4T(n/8) + n/4) + n/2) + n} \\
&\vdots \\
&\mathbf{= 4(4(4 (\ldots\ldots(4T(n/2^p) + n/2^{p-1}) \ldots\ldots)+ n/4) + n/2) + n} \\
&\mathbf{= 4^p \underbrace{T(1)}_{1} + 4^{p-1} n/2^{p-1} + 4^{p-2} n/2^{p-2} + \ldots\ldots+ 4^1 n/2^1 + 4^0 n/2^0 \;/\; n = 2^p} \\
&\mathbf{= 4^p n/ 2^p + 4^{p-1} n/2^{p-1} + 4^{p-2} n/2^{p-2} + \ldots\ldots+ 4^1 n/2^1 + 4^0 n/2^0} \\
&\mathbf{= n * \textstyle\sum_{i=0}^{p} 2^i = n * (2^{p+1} -1) = n (2n-1) = 2n^2-n \Rightarrow} \quad \boxed{\mathbf{T(n) = O(n^2)}}
\end{aligned}$$

b. $T(n) = 2T(n/2) + n$; $T(1) = 1$

1ère Méthode : application du théorème de résolution des récurrences

$a = 2$, $b = 2$ $\Rightarrow$ $\log_2 2 = 1$ et $f(n) = O(n)$

$\Rightarrow$ $f(n) = O(n^{(\log_2 2)})$ $\Rightarrow$ 2ème cas $\boxed{T(n) = \theta(n\text{Log}(n))}$

2ème Méthode : Développement des récurrences

$$
\begin{aligned}
T(n) &= 2T(n/2) + n \\
&= 2(2T(n/4) + n/2) + n \\
&= 2(2(2T(n/8) + n/4) + n/2) + n \\
&\vdots \\
&= 2(2(2(\ldots\ldots(2T(n/2^p) + n/2^{p-1})\ldots\ldots) + n/4) + n/2) + n \\
&= 2^p \underbrace{T(1)} + 2^{p-1} n/2^{p-1} + 2^{p-2} n/2^{p-2} + \ldots\ldots + 2^1 n/2^1 + 2^0 n/2^0 \quad / \quad n = 2^p \\
&= \cancel{2^p} n/\cancel{2^p} + \cancel{2^{p-1}} n/\cancel{2^{p-1}} + \cancel{2^{p-2}} n/\cancel{2^{p-2}} + \ldots\ldots + \cancel{2^1} n/\cancel{2^1} + \cancel{2^0} n/\cancel{2^0} \\
&= (p+1)\, n \ / \ p = \text{Log}(n) \ \ (\text{ car } n = 2^p \Rightarrow \text{Log } n = p\text{Log}2 = p) \\
&= (\text{Log}(n) + 1)\, n = n\,\text{Log}(n) + n \Rightarrow \boxed{T(n) = O(n\text{Log}(n))}
\end{aligned}
$$

c. $T(n) = 3T(n/4) + n$; $T(1) = 1$

1ère Méthode : application du théorème de résolution des récurrences

$a = 3$, $b = 4$ $\Rightarrow$ $\log_4 3 < 1$ donc $f(n) = \Omega(n^{(\log_4 3)+\varepsilon})$ et $3 f(n/4) = 3(n/4)$

$3/4\ n \leq c \,.\, f(n)$ on peut choisir $c = 3/4$, $5/6$, tq $c < 1$

$\Rightarrow$ 3ème cas $\boxed{T(n) = \theta(f(n)) = \theta\ (n)}$

2ème Méthode : Développement des récurrences

$$
\begin{aligned}
T(n) &= 3T(n/4) + n \quad \underline{\text{on pose}}\ n = 4^p \\
&= 3(3T(n/4^2) + n/4) + n \\
&= 3(3(3T(n/4^3) + n/4^2) + n/4) + n \\
&\vdots \\
&= 3(3(3(\ldots\ldots(3T(n/4^p) + n/4^{p-1})\ldots\ldots) + n/4^2) + n/4) + n \\
&= 3^p \underbrace{T(1)}_{1} + 3^{p-1} n/4^{p-1} + 3^{p-2} n/4^{p-2} + \ldots\ldots + 3^1 n/4^1 + 3^0 n/4^0 \quad / \quad n = 4^p
\end{aligned}
$$

$= 3^p\ n/4^p + 3^{p-1}\ n/4^{p-1} + 3^{p-2}\ n/4^{p-2} + \ldots\ldots. + 3^1\ n/4^1 + 3^0\ n/4^0$

$$= n * \sum_{i=0}^{p} (3/4)^i = n\ \frac{(\frac{3}{4})^{p+1} - 1}{(\frac{3}{4}) - 1}$$

$= 3^p + 4 * (4^p - 3^p) = 3^{\log_4 n} + 4 * (n - 3^{\log_4 n})$

$= n^{\log_4 3} + 4\ n - 4\ n^{\log_4 3} = 4\ n - 3\ n^{\log_4 3}$

$\Rightarrow$ $\boxed{T(n) = O(n)}$

d. T(n) = 3T(n/2) + n² ; T(1) = 1

1ère Méthode : application du théorème de résolution des récurrences

$a = 3$, $b = 2$ $\Rightarrow$ $\log_2 3 = 1.58$ donc $f(n) = \Omega(n^{(\log_2 3)+\varepsilon})$ et $3 f(n/2) = 3(n/2)^2$

$3/4\ n^2 \leq c \cdot n^2$ on peut choisir $c = 3/5$, tq $c < 1$

$\Rightarrow$ 3ème cas $\boxed{T(n) = \theta(f(n)) = \theta(n^2)}$

Exercice 2

1. Quelle est la complexité de l'algorithme de recherche séquentielle d'un élément dans un tableau à n éléments ? justifier.

```
Fonction   Recherche_Elt ( T : Tab , x, n : entier)  Booléen

Var  i : entier

Début
        i ←— 1

Tantque  i ≤ n Faire
        Si T[i] = x  alors
                Retourner ( Vrai )
        Fsi
        i  ←—  i+1
finTque

        Retourner ( Vrai )
Fin
```

$\Rightarrow$ $T(n) = 2 * n = O(n)$

2. La recherche dichotomique d'un élément x dans un tableau trié T (à n éléments) se présente sous la fonction récursive suivante :

```
Fonction Dichotomie(T :Tab, x; d; f : entier) : booléen
var Milieu : entier
début
        si (d>f)alors Dichotomie ←Faux
        sinon
                si (x = T[Milieu]) alors Dichotomie ← Vrai
                sinon
                   si (x < T[milieu]) alors Dichotomie ←Dichotomie(T; x; d; milieu)
                   sinon Dichotomie ← Dichotomie(T; x; milieu; f)
                   fsi
                fsi
        fsi
Fin
```

a. Sachant que la complexité de l'algorithme de recherche dichotomique dans un tableau à n éléments est T(n), montrer que $T(n) = T(n/2) + O(1)$.

On a : $T(n) = T(n/2) + 3 = T(n/2) + O(1)$

b. Déduire une estimation asymptotique de T(n) en appliquant le théorème de résolution des récurrences.

$a = 1$, $b = 2 \Rightarrow \log_2 1 = 0$ **et** $f(n) = O(1)$

$\Rightarrow f(n) = O(n^{(\log_2 1)}) \Rightarrow 2^{ème}$ **cas** $\boxed{T(n) = \theta(n^0 Log_2(n)) = \theta(Log_2(n))}$

c. Calculer cette même estimation d'une autre manière.

$T(n) = T(n/2) + O(1) = T(n/2) + C = (T(n/4) + C) + C = ((T(n/8) + C) + C) + C$
$= T(1) + C + \ldots\ldots + C + C = 0 + C * p$ **<u>or</u>** $p = \log_2 n$

Donc $T(n) = C * Log_2 n \Rightarrow T(n) = O(Log_2\, n)$

<u>Exercice 3</u>

Concevoir un algorithme récursif m-aire (m>= 2) calculant le maximum d'une liste de taille n. Déterminer sa complexité. Comparer avec l'algorithme itératif classique. Conclure.

<u>Fonction</u> Max_Tab (T :Tab, d, f : entier) : entier

<u>Début</u>

<u>Si</u> (d= f) <u>alors</u> Retourner (T[d])

<u>Sinon</u>

$M_1 \longleftarrow$ Max_Tab (T, d, d + n/m – 1)
$M_2 \longleftarrow$ Max_Tab (T, d + n/m, d +2 n/m – 1)

$\vdots$

$M_m \longleftarrow$ Max_Tab (T, f- n/m + 1, f)

Retourner (Max (M_1, M_2, ……, M_m))

<u>Fsi</u>

<u>Fin</u>

$$\Rightarrow \quad T(n) = \begin{cases} 0 & \textbf{\underline{si}}\ n = 1 \\ m * T(n/m) + (m-1) & \textbf{\underline{si}}\ n > 1 \end{cases}$$

Pour l'algorithme itératif classique on a T(n) = n - 1 = O (n)

$$\begin{aligned} T(n) &= m * T(n/m) + (m-1) \quad / \quad n = m^p \text{ et } T(1) = 0 \\ &= m * (m * T(n/m^2) + (m-1)) + (m-1) \\ &= \ldots\ldots. \\ &= m^p * T(1) + m^{p-1} (m-1) + \ldots\ldots\ldots. + m^1 (m-1) + m^0 (m-1) = \\ &\quad (m-1) \sum_{i=0}^{p-1} m^i = (m \cancel{-} 1) \frac{m^p - 1}{\cancel{m - 1}} \\ &= m^p - 1 = O(n) \end{aligned}$$

$\Rightarrow$ les deux algorithmes sont de même complexité