TD 2 : D&C, THEOREME MASTER ET COMPLEXITE DES ALGORITHMES RECURSIFS

Exercice 1 –

[O.Bodini, 2014] Dans le cadre d'utiliser le théorème master pour donner des ordres de grandeur asymptotique pour des fonctions définies par récurrence.

1. En utilisant le théorème master, donner un ordre de grandeur asymptotique pour T(n) :

a. $T(1) = 1$, $T(n) = 2T(n/2) + n^2$;
b. $T(1) = 0$, $T(n) = 2T(n/2) + n$;
c. $T(1) = 1$, $T(n) = 2T(n/2) + \sqrt{n}$;
d. $T(1) = 0$, $T(n) = T(n/2) + \Theta(1)$.

2. Essayer de donner des exemples d'algorithmes dont le coût est calculé par l'une de ces récurrences

Exercice 2 –

1. Un algorithme s'exécute dans le meilleur des cas en réalisant $cm(n) = 2n + 1$ opérations et dans le pire des cas en $cp(n) = 2^n + 4n-2$ opérations pour une donnée de taille n. Indiquez et prouvez son comportement asymptotique.
2. Le nombre d'appels récursifs d'un algorithme récursif est donné par la formule :
$$c(n) = \begin{cases} 1 & \text{si } n = 0 \text{ ou } 1 \\ 16c\left(\frac{n}{4}\right) + n^3 & \text{sinon} \end{cases}$$
   - Indiquez et prouvez son comportement asymptotique.
3. Étant donné que : $T(n) = 2T([n/4]) + 3\log n$
   - Trouvez le comportement asymptotique de T(n). Justifiez votre réponse.

Exercice 3 –

[O.Bodini, 2014] Soit un algorithme dont la complexité T(n) est donnée par la relation de récurrence : $T(1) = 1$,
$$T(n) = 3T([n/2]) + n^2$$

1. Calculer T(n) en résolvant la récurrence.
2. Déterminer T(n) à l'aide du théorème maitre.

$$\text{On a, } \quad a^{\log_x b} = b^{\log_x a} \quad \text{et} \sum_{i=0}^{k} x^i = \frac{x^{k+1}-1}{x-1}$$

Exercice 4 –

[H.Bherer, 2006] Vous décidez d'utiliser la technique diviser pour régner pour résoudre un certain type de problème.

Pour $n \geq 4$, vous savez que vous pouvez obtenir la solution à un exemplaire de taille n en résolvant a sous-exemplaires de taille $[n/4]$. Le temps requis pour la décomposition de l'exemplaire original en a sous-exemplaires est dans $O(n^2/\log n)$ et le temps requis pour la recombinaison des sous-solutions est dans $O(n^2)$. Supposez pour simplifier que n est une puissance de 4.

1. Donnez l'équation de récurrence asymptotique exprimant le temps d'exécution $t(n)$ de cet algorithme en fonction de la taille n de l'exemplaire.
2. Donnez l'ordre exact de $t(n)$, sous la forme la plus simple possible, (i) lorsque $a = 8$; (ii) lorsque $a = 16$; (iii) lorsque $a = 32$.
3. Les réponses obtenues en 2) s'appliquent-elles toujours si n n'est pas une puissance de 4? Justifiez brièvement votre réponse.

Exercice 5 –

[E. Hebrard, 2018] Calculer la complexité dans le pire des cas et dans le meilleur des cas de

```
Algorithme: rechercheDicho(T)
Données: un tableau trié T de taille n et un entier x;
Résultat: le premier indice i tel que T[i]=x, ou bien -1 si x∉ T
m,i,j: entier;
début
    i←1;  j←n;
    tant que i<j faire
      m ← (i+j)/2;
      si x=T[m] alors   retourner m;
      si x<T[m] alors   j← m;  sinon    i← m+1;  retourner -1;
fin.
```

- Préciser à quoi correspond chaque cas ?

Exercice 6 –

1. [S.Émilie] Un algorithme A est concu en utilisant l'approche « Diviser Pour Régner ». Afin de résoudre un certain type de problème de taille n dans un temps TA (n), il décompose ce dernier en 3 sous-exemplaires de taille $[\frac{n}{4}]$. Les étapes de décomposition en sous-exemplaires, de combinaison des solutions intermédiaires et de gestion des appels récursifs demandent un temps $f(n) = n \log n$. Un deuxième algorithme B résout le meme type de problème dans un temps $TB(n) \in \theta(\frac{1}{3} n \log n2 + \sqrt{n})$.

A et B sont-ils équivalents? Justifiez.

2. Montrez que le temps d'exécution TC (n) de l'algorithme C suivant est dans $\Omega(2^{\sqrt{n}})$ (on suppose ici que le temps de calcul de f_{2n} est dans $\theta(1)$).

Remarque : montrez par induction que pour tout $n \geq 0$ nous avons $f_{2n} \geq 2^n - 1$ et utilisez ce résultat pour votre analyse.

```
AlgoC(n)
REP = 0
  for i = 0 to f 2n
     REP++
Return REP
```

Bon courage