TD 02 – Divide and Conquer

Exercice 1. *Suivez le maître*

Appliquer le *Master Theorem* sur les cas suivants :

1. $T(n) = 9T(n/3) + n$;
2. $T(n) = T(2n/3) + 1$;
3. $T(n) = 3T(n/4) + n \log n$;
4. $T(n) = 2T(n/2) + n \log n$;
5. $T(n) = 2T(n/2) + n^3$;
6. $T(n) = T(9n/10) + n$;
7. $T(n) = 7T(n/3) + n^2$;
8. $T(n) = T(n-1) + n$;
9. $T(n) = T(\sqrt{n}) + 1$.

Exercice 2. *Meetic*

Un site internet cherche à regrouper ses membres en fonction des goûts musicaux de chacun. Pour cela, chaque membre doit classer par ordre de préférence une liste d'artistes [1]. On dit que deux membres, Arthur et Béatrice, ont des goûts musicaux proches lorsque qu'il y a peu d'*inversions* dans leurs classements : une inversion est une paire d'artiste $\{L, M\}$ telle qu'Arthur préfère L à M et Béatrice préfère M à L. On cherche donc à compter le nombre d'inversion dans les classements d'Arthur et Béatrice.

1. Compter le nombre d'inversion les classements suivants :

   Arthur : Britney Spears, Lady Gaga, Michael Jackson, Madonna, Céline Dion ;

   Béatrice : Lady Gaga, Madonna, Britney Spears, Michael Jackson, Céline Dion.

2. Proposer un algorithme naïf qui résout le problème. Quelle est sa complexité ?

On cherche maintenant à améliorer l'algorithme précédent en utilisant le paradigme Diviser-Pour-Régner. Pour cela, on coupe le classement de chaque membre en deux sous-classements de même taille, celui des artistes préférés (classement supérieur) et celui des autres artistes (classement inférieur). On compte alors les inversions (L, M) qui peuvent être de deux types : soit L et M apparaissent dans le même sous-classement de Béatrice, soit L et M apparaissent dans deux différents sous-classements de Béatrice (inversions mixtes).

3. On suppose que les deux sous-classements de Béatrice sont triés en fonction du classement d'Arthur [2]. Montrer qu'on peut alors compter les inversions mixtes en temps linéaire.
4. Donner un algorithme de type Diviser-Pour-Régner qui fonctionne en temps $\mathcal{O}(n \log n)$.

Exercice 3. *Plus grand et deuxième plus grand de n entiers*

On s'intéresse dans cet exercice à la **complexité dans le pire des cas et en nombre de comparaisons** des algorithmes.

1. Pour rechercher le plus grand et deuxième plus grand élément de n entiers, donner un algorithme naïf et sa complexité.
2. Pour améliorer les performances, on se propose d'envisager la solution consistant à calculer le maximum suivant le principe d'un *tournoi* (tournoi de tennis par exemple). Plaçons-nous d'abord dans le cas où il y a $n = 2^k$ nombres qui s'affrontent dans le tournoi. Comment retrouve-t-on, une fois le tournoi terminé, le deuxième plus grand ? Quelle est la complexité de l'algorithme ? Dans le cas général, comment adapter la méthode pour traiter n quelconque ?
3. Montrons l'optimalité de cet algorithme en fournissant une borne inférieure sur le nombre de comparaisons à effectuer. Nous utiliserons la méthode des *arbres de décision*.
   - **(a)** Montrer que tout arbre de décision qui calcule le maximum de N entiers a au moins 2^{N-1} feuilles.
   - **(b)** Montrer que tout arbre binaire de hauteur h et avec f feuilles vérifie $2^h \geq f$.

1. Le classement est un ordre total.
2. Si L et M apparaissent dans le même sous-classement de Béatrice, alors ils apparaissent dans le même ordre que dans le classement d'Arthur.

(c) Soit A un arbre de décision résolvant le problème du plus grand et deuxième plus grand de n entiers, minorer son nombre de feuilles. En déduire une borne inférieure sur le nombre de comparaisons à effectuer.

Exercice 4. *Multiplication de deux entiers*

Soient deux entiers x et y codés sur n bits (on supposera que n est une puissance de 2). On souhaite multiplier ces deux entiers entre eux, en travaillant au niveau des bits. La multiplication de deux entiers de n bits se fait de manière triviale en $\mathcal{O}(n^2)$, la multiplication d'un entier par une puissance de 2, et l'addition de 2 entiers se font en temps linéaire $\mathcal{O}(n)$.

1. La méthode diviser pour régner n'est pas toujours meilleure qu'un algorithme naïf. Pour illustrer cela, donnez un algorithme diviser pour régner ayant une complexité en $\mathcal{O}(n^2)$ pour multiplier deux entiers x et y de n bits chacun.
2. Proposez un algorithme diviser pour régner ayant une complexité inférieure à $\mathcal{O}(n^2)$. Donnez la formule de récurrence pour votre algorithme et sa complexité finale (en $\mathcal{O}$). On supposera pour simplifier que l'addition/multiplication de deux nombres de taille n est un nombre de taille n.