TD 2 – Où est Charlie ?

Exercice 1. *Tableaux*

Dans cet exercice, on s'intéresse à deux problèmes de sélections d'éléments dans un tableau.

1. Étant donné un tableau trié d'entiers deux à deux distincts $A[1,\dots,n]$, on veut décider s'il existe un entier i tel que $A[i] = i$. Donner un algorithme en temps $\mathcal{O}(\log n)$ pour ce problème.
2. Soit $A[1,\dots,n]$ un tableau d'entiers triés dont on ne connaît pas la longueur (n est inconnu). On peut demander la valeur de $A[i]$ pour n'importe quelle valeur entière de i : si $i \leq n$, on reçoit la valeur de $A[i]$, sinon on reçoit ∞. Donner un algorithme en temps $\mathcal{O}(\log n)$ qui prend en entrée un entier x et trouve l'indice du tableau où se trouve x (s'il en existe un).

Exercice 2. *Les deux plus grands*

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 $2^h \geq f$.
   - **(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 3. *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).

1. Le classement est un ordre total.

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 $\mathcal{O}(n \log n)$.

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.