TD d'algorithmique numéro 1
Natacha Portier et Daniel Hirschkoff
semaine du 27 septembre au 1er octobre 1999

On s'intéresse à des ensembles d'entiers tous distincts rangés dans un tableau : $L[1]...L[n]$. Les algorithmes considérés dans ce TD effectuent des **affectations** et leur seul critère de décision (ou de bifurcation) est la **comparaison** de deux éléments ($=$ et $<$). En aucun cas ils ne peuvent effectuer des opérations arithmétiques, comme l'addition et la multiplication.

Question 1.1 ⋆ maximum de n entiers ⋆

Écrire un algorithme (naïf!) qui calcule le maximum de n entiers. Quelle en est la complexité (en nombre de comparaisons effectuées, en nombre d'affectations effectuées, dans le pire des cas, le meilleur, en moyenne) ?

Indications pour le calcul des affectations en moyenne : Soit $P_{n,k}$ le nombre de permutations σ de $\{1, ..., n\}$ telles que sur la donnée $L[1] = \sigma(1)...L[n] = \sigma(n)$ l'algorithme effectue k affectations. Donner une relation de récurrence pour $P_{n,k}$. Soit $G_n(z) = \sum P_{n,k} z^k$. Montrer que $G_n(z) = z(z+1)...(z+n-1)$. En déduire le résultat.

Question 1.2

L'algorithme ci-dessus est-il optimal pour la complexité en comparaisons dans le pire des cas ?

A partir de maintenant, on ne s'intéresse plus qu'à **la complexité dans le pire des cas et en nombre de comparaisons** des algorithmes considérés.

Question 2.1 ⋆ plus petit et plus grand ⋆

On s'intéresse maintenant au calcul (simultané) du maximum et du minimum de n entiers. Donner l'algorithme naïf et sa complexité.

Question 2.2

L'idée pour améliorer l'algorithme est de regrouper les éléments à comparer *par paires*, de manière à diminuer le nombre de comparaisons à effectuer. Décrire l'algorithme, en évaluer la complexité.

Question 2.3

Montrons l'optimalité de cet algorithme. Soit A un algorithme qui trouve le maximum et le minimum. Pour une donnée fixée, au cours du déroulement

de l'algorithme, on appelle novice (N) un élément qui n'a jamais subit de comparaison, gagnant (G) un élément qui a été comparé au moins une fois et a toujours été supérieur aux éléments auquels il a été comparé, perdant (P) un élément qui a été comparé au moins une fois et a toujours été inférieur aux éléments auquels il a été comparé, et moyens (M) les autres. Le nombre de ces éléments est représenté par un quadruplet d'entiers (i, j, k, l) et on a bien sur $i + j + k + l = n$. Donner la valeur de ce quadruplet au début et à la fin de l'algorithme, et en déduire un nombre minimal de tests à effectuer. On choisira bien sur la donnée de manière à maximiser ce nombre. Conclure.

Question 3.1 ⋆ deuxième plus grand ⋆

Recherche du deuxième plus grand : quelle est la complexité de la méthode naïve consistant à appliquer deux fois la méthode vue à la question 1 ?

Question 3.2

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 tout d'abord dans le cas où il y a $n = 2^k$ nombres à trier. 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 ?

Question 3.3

Montrer l'optimalité de cet algorithme en minorant le nombre de feuilles d'un arbre de décision.