SANDWICH AU JAMBON

Le problème dit du *sandwich au jambon* ou bien encore appelé théorème de *Stone-Tukey* s'énonce de la manière suivante : un ensemble de n points en dimension d peut toujours être séparé en deux parties de cardinal au plus [n/2] par un hyperplan de dimension d-1 (certains points peuvent être dans l'hyperplan), où [n/2] désigne la partie entière de n/2. *De manière concrète, un ensemble de points dans l'espace peut être séparé en deux parties quasi-égales par un plan. De même un ensemble de points dans le plan peut être séparé en deux par une droite et même en 4 à l'aide de deux droites. Ce sujet porte sur la résolution algorithmique de ce problème et de problèmes connexes selon différentes méthodes.*

Dans tout le problème, les tableaux sont indicés à partir de 0. L'accès à la i-ème case d'un tableau tab est noté tab[i]. Quel que soit le langage utilisé, on suppose que les tableaux peuvent être passés comme arguments des fonctions. En outre, il existe une primitive allouer(m; c) pour créer un tableau de taille m dont chaque case contient c à l'origine, ainsi qu'une primitive taille(t) qui renvoie la taille d'un tableau t. Enfin, on disposera d'une fonction floor(x) qui renvoie la partie entière [x] pour tout réel $x \geq 0$.

La complexité, ou le temps d'exécution, d'un programme P (fonction ou procédure) est le nombre d'opérations élémentaires (addition, soustraction, multiplication, division, affectation, etc...) nécessaires à l'exécution de P. Lorsque cette complexité dépend d'un paramètre n, on dira que P a une complexité en O(f(n)), s'il existe $K > 0$ tel que la complexité de P est au plus $K\,f(n)$, pour tout n. Lorsqu'il est demandé de garantir une certaine complexité, le candidat devra justifier cette dernière si elle ne se déduit pas directement de la lecture du code.

1 Grand, Petit et Médian

Dans cette partie, nous supposerons donné un tableau tab contenant n nombres réels. Les indices du tableau vont de 0 à n-1.

Nous utiliserons le tableau de taille 11 suivant pour nos exemples :

3	2	5	8	1	34	21	6	9	14	8

Q1. Ecrire une fonction **calculeIndiceMaximum(tab)** qui renvoie l'indice d'une case où se trouve le plus grand réel de tab. Sur le tableau précédent, la fonction renverra 5 car la case 5 contient la valeur 34.

Q2. Ecrire une fonction **nombrePlusPetit(tab,val)** qui renvoie le nombre d'éléments dans le tableau tab dont la valeur est plus petite ou égale à val. Sur le tableau exemple, pour une valeur de val égale à 5, la fonction devra renvoyer la valeur 4 car seuls les nombres 3, 2, 5, 1 sont inférieurs ou égaux à 5.

Nous allons maintenant calculer un médian d'un tableau. Rappelons qu'une valeur médiane m d'un ensemble E de nombres est un élément de E tel que les deux ensembles $E_{<m}$ (les nombres de E strictement plus petits que m) et $E_{>m}$ (les nombres de E strictement plus grands que m) vérifient $|E_{<m}| = |n/2|$ et $|E_{>m}| = |n/2|$. Notez que le problème du médian est une reformulation de problème dit du *sandwich au jambon* pris en dimension 1. Une méthode naïve consiste donc à parcourir les éléments de l'ensemble et à calculer pour chacun d'eux les valeurs de $|E_{<m}|$ et $|E_{>m}|$ mais il est possible de faire mieux comme nous allons le voir dans la partie suivante.

2 Un tri pour accélérer

Une méthode plus efficace serait de trier le tableau par ordre croissant tout en prenant la cellule du milieu dans le tableau trié. Cette méthode certes rapide requiert O(n ln n) opérations. Il existe une méthode optimale en temps linéaire O(n) pour trouver le médian d'un ensemble de n éléments. Cette partie a pour but d'en proposer une implémentation.

Une fonction annexe nécessaire pour cet algorithme consiste à savoir séparer en deux un ensemble de valeurs. Soit un tableau tab et un réel appelé pivot p = tab[i], il s'agit de réordonner les éléments du tableau en mettant en premier les éléments strictement plus petits que le pivot $tab_{<p}$, puis les éléments égaux au pivot p, et en dernier les éléments strictement plus grands $tab_{>p}$. Sur le tableau exemple, en prenant comme valeur de pivot 8 on obtiendra le tableau résultat suivant :

3	2	5	1	6	8	8	21	34	9	14

Notez que dans le résultat les nombres plus petits que le pivot 3; 2; 5; 1; 6 peuvent être dans n'importe quel ordre les uns par rapport aux autres.

Q3. Ecrire une fonction **partition(tab,indicePivot)** qui prend en paramètre un tableau d'entiers tab ainsi qu'un entier 0 < indiceP ivot < n-1. Soit p = tab[indiceP ivot]. La fonction devra réordonner les éléments de tab comme expliqué précédemment en prenant comme pivot le nombre p. La fonction retournera le nouvel indice de la case ou se trouve la valeur p.

Dans cette question, on suppose que les modifications effectuées par la fonction sur le tableau tab sont persistantes, même après l'appel de la fonction.

Remarquons que le [n/2]-ème élément dans l'ordre croissant d'un tableau de taille n est un élément médian du tableau considéré. Nous allons donc non pas programmer une méthode pour trouver le médian mais plus généralement pour trouver le k-ème élément d'un ensemble. Nous allons utiliser l'algorithme suivant :

On cherche le k-ème élément du tableau tab.

- Si k = 0 et taille(tab) = 1 alors renvoyer tab[0]
- Sinon, soit p = tab[a]. Partitionner le tableau tab en utilisant le pivot p en mettant en premier les éléments plus petits que p. Soit i l'indice de p dans le tableau résultant.
  - Si i > k chercher le k-ème élément dans tab[0..i] et renvoyer cet élément.
  - Si i = k renvoyer le pivot.
  - Si i < k chercher le (k-i-1)-ème élément dans tab[i+1..n-1] et renvoyer cet élément.

Q4. Ecrire une fonction **elementK(tab,k)** qui réalise l'algorithme de sélection du k-ème élément dans le tableau tab décrit précédemment et renvoie cet élément.

Q5. Supposons que dans l'algorithme précédent nous voulions rechercher le premier élément mais qu'à chaque étape le pivot choisi est le plus grand élément, quel est un ordre de grandeur du nombre d'opérations réalisées par votre fonction ?

L'algorithme précédent ne semble donc pas améliorer le calcul du médian. Le problème vient du fait que le pivot choisi peut être mauvais c'est-à-dire qu'à chaque étape un seul élément du tableau a été éliminé. En fait, si l'on peut choisir un pivot p dans tab tel qu'il y ait

au moins n/5 éléments plus petits et n/5 plus grands alors on peut montrer que l'algorithme précédent fonctionne optimalement en temps O(n).

Pour choisir un tel élément dans tab, on réalise l'algorithme choixPivot suivant où chaque étape sera illustrée en utilisant le tableau donné en introduction.

- On découpe le tableau en paquets de 5 éléments plus éventuellement un paquet plus petit.

  On calcule l'élément médian de chaque paquet.

3	2	5	8	1	34	21	6	9	14	8

  S'il n'y a qu'un paquet on renvoie son médian. Sinon on place ces éléments médians un nouveau tableau.

3	14	8

- On réalise choixPivot sur les médians précédents. Dans notre exemple on recommence donc les étapes précédentes.

Q6. Ecrire la fonction **choixPivot(tab)** qui réalise l'algorithme précédent et renvoie la valeur du pivot.