DM12

Dans ce problème, les programmes sont écrits en langage OCaml.

Dans le pire des cas, l'algorithme du tri rapide à une complexité en $O(n^2)$ alors que sa complexité en moyenne est en $O(n \log n)$. Ces résultats sont liés au choix du pivot. Différentes solutions existent pour améliorer la complexité dans le pire cas. Une première solution mélange les données initiales. Une deuxième solution consiste à mieux choisir le pivot pour obtenir une complexité en $O(n \log n)$ dans le pire des cas. ce sujet explore cette dernière possibilité.

Tri rapide

On rappelle le principe du tri rapide sur les tableaux. Soit t un tableau et $g < d$ des indices de ce tableau. On note $t[g : d]$ le sous-tableau de t d'indices compris entre les indices g inclus et d exclu. L'algorithme de tri partitionne d'abord une portion $t[g : d]$ du tableau selon une valeur p, appelée pivot, comparable aux éléments du tableau. À la fin de cette partition, on a échangé des cases du tableau pour qu'il existe deux indices i_s et i_e tels que :

- $\forall i \in [\![g, i_s[\![\quad t_i < p$
- $\forall i \in [\![i_s, i_e[\![\quad t_i = p$
- $\forall i \in [\![i_e, d[\![\quad t_i > p$

Contrairement à la version traditionnelle de l'algorithme, on partitionne ici en trois morceaux.

Question 1. Écrire une fonction `swap : 'a array -> int -> int -> unit` qui prend en entrée un tableau `t`, deux entiers `i` et `j` et qui échange, par effets de bords, les valeurs de `t.(i)` et `t.(j)`.

Question 2. Écrire une fonction `split3 : 'a array -> 'a -> int -> int -> int * int` qui prend en entrée un tableau `t`, un pivot `p`, deux indices `g` et `d`, qui effectue la partition décrite ci-dessus en modifiant le tableau `t` par effets de bords et qui renvoie les valeurs des indices `i_s` et `i_e`. Votre solution doit avoir une complexité linéaire.

Question 3.
□ **3.1.** Écrire une fonction `qsort1 : 'a array -> unit` qui trie par effets de bords un tableau et qui ne renvoie rien. Votre solution doit reposer sur le tri rapide et adopter le premier élément de la portion à trier comme pivot.
□ **3.2.** Quelle est sa complexité dans le pire des cas ?

Médiane d'un tableau

On cherche à améliorer la complexité du tri rapide dans le pire des cas en choisissant un pivot qui garantit de partitionner le tableau en deux morceaux de même taille, à un élément près. Pour cela, on cherche la *médiane* du tableau, c'est à dire le k-ème plus petit élément du tableau, où k est la moitié de la taille du tableau.

Question 4. Dans cette question, on suppose l'existence d'une fonction `median : 'a array -> int -> 'a` telle que `median t k` renvoie la valeur du `k`-ème plus petit élément de `t` avec une complexité $f(n)$ où n est la taille de `t`. On s'intéresse ici à la complexité d'une version du tri rapide qui utiliserait la fonction `median` pour choisir le pivot.
□ **4.1.** Quelle serait la complexité de cet algorithme de tri rapide si $f(n) = O(n)$?
□ **4.2.** Quelle serait la complexité de cet algorithme de tri rapide si $f(n) = O(n^2)$?

Question 5. Pour implémenter la fonction `median`, on utilise une approche qui s'inspire du tri rapide mais qui n'effectue qu'un seul appel récursif au lieu de deux.

- On copie `t` dans `t'` avec l'instruction let `t' = Array.copy t in ...`.
- On partitionne `t'` à l'aide de la fonction `split3`.
- après cette partition :
  - si $i_s = k$ ou $(i_s < k < i_e)$, alors `t'.(k)` est l'élément qu'on cherche ;
  - sinon, si $k < i_s$, alors l'élément qu'on recherche est dans la partie strictement à gauche de i_s ;
  - sinon, l'élément qu'on cherche est dans la partie à droite de i_e.

Dans ces deux derniers cas, on ne fait qu'un seul appel récursif dans le morceau de `t'` qui nous intéresse.
□ **5.1.** Implémenter une fonction `median` utilisant le principe précédent.
□ **5.2.** Quelle est sa complexité dans le pire des cas ?

Médiane des médianes

L'algorithme précédent a une complexité *linéaire en moyenne* mais on va maintenant s'intéresser à une méthode fournissant une complexité *linéaire dans le pire des cas*. Pour ce faire, on optimise le choix du pivot lors du calcul de la médiane. Pour un tableau t de taille n, le principe est le suivant.

- On regroupe les éléments de t en $\frac{n}{5}$ groupes de 5 éléments.
- On calcule (en temps constant) la médiane de chacun de ces groupes de 5 : $m_1, m_2, \dots, m_{\frac{n}{5}}$.
- On calcule récursivement la médiane de ces médianes.
- On utilise cette valeur comme pivot pour partitionner t.

Question 6.
□ **6.1.** Soit p le pivot obtenu, médiane des médianes. Montrer qu'au moins 30% des éléments de t sont inférieurs à p et qu'au moins 30% des éléments de t lui sont supérieurs.
□ **6.2.** Dans le pire des cas, quelle est alors la taille de la portion de t où s'effectue l'appel récursif ?
□ **6.3.** Notons $C(n)$ la complexité de l'algorithme `median` sur un tableau de taille n. Justifier que :

$$C(n) \leqslant C\left(\frac{1}{5}\cdot n\right) + C\left(\frac{7}{10}\cdot n\right) + \alpha \cdot n$$

où α est une constante positive qu'on n'essaiera pas d'exprimer.
□ **6.4.** Montrer que pour tout entier naturel n : $C(n) \leqslant 10 \cdot \alpha \cdot n$.

Question 7. Écrire une fonction `median5 : 'a array -> int -> int -> int` qui détermine la médiane des éléments `t.(g), t.(g+1), ..., t.(d)` et qui renvoie son indice .
Indication. En pratique, on n'utilise cette fonction que si $0 \leqslant d - g \leqslant 5$. On ne se préoccupe donc pas de sa complexité asymptotique. On peut par exemple trier la portion du tableau avec un algorithme de tri naïf (tri par sélection ou par insertion) et renvoyer l'indice du milieu de la portion.

Question 8. On fournit le code des deux fonctions mutuellement récursives suivantes.

```
let rec pivot t g d = match d-g < 5
    with
  | true -> median5 t g d
  | false ->
      let i = ref g in
      while !i <= d do
        let sd = min (!i+4) d in
        let med5 = median5 t !i sd in
        let mi = g + (!i-g)/5 in
        swap t med5 mi;
        i := !i+5
      done ;
      let mid = g + (d-g)/10 + 1 in
      let d' = g + (d-g)/5 in
      select t g d' mid
```

```
and select t g d k = match g = d with
  | true -> g
  | false ->
      let ip = pivot t g d in
      let p = t.(ip) in
      let i_s, i_e = split3 t p g d in
      match (i_s = k) || (i_s < k && k
          < i_e) with
        | true -> k
        | false when k < i_s -> select
            t g (i_s-1) k
        | _ -> select t i_e d k
```

□ **8.1.** Quel est le type des fonctions `pivot` et `select` ?
□ **8.2.** Expliquer le rôle de chacune de ces fonctions vis-à-vis de l'algorithme de la médiane des médianes.
□ **8.3.** Montrer que les fonctions `pivot` et `select` terminent.
□ **8.4.** Quelle est la complexité de la fonction `select` dans le pire des cas ?

Question 9. Écrire une fonction `qselect : 'a array -> int -> 'a` qui prend en entrée un tableau `t`, un entier `k` et qui renvoie le `k`-ième plus petit élément de `t`. Prendre garde à ne pas modifier `t` par effets de bords.

Question 10. Écrire une fonction `qsort2 : 'a array -> unit` qui trie par effets de bords un tableau pris en entrée. Votre solution doit reposer sur le tri rapide et choisir la médiane de la portion à trier comme pivot. Sa complexité dans le pire des cas doit être $O(n \log n)$.