DM12 (éléments de réponse)

Tri rapide

Question 1.

```
let swap t i j =
  let x = t.(i) in
  t.(i) <- t.(j) ;
  t.(j) <- x
;;
```

Question 2.

```
let split3 t p g d =
  let i_s = ref g in
  (* déplace les éléments < p au début *)
  for i = g to d-1 do
    if t.(i) < p then begin
      echange t i !i_s ;
      incr i_s
    end
  done ;
  (* déplace les éléments = p juste après *)
  let i_e = ref !i_s in
  for i = !i_s to d-1 do
    if t.(i) = p then begin
      echange t i !i_e ;
      incr i_e
    end
  done ;
  (!i_s, !i_e)
;;
```

Question 3.
□ **3.1.**

```
let qsort1 t =
  let rec aux g d = match g < d-1 with
    | false -> ()
    | true -> let ip, _ = split3 t t.(g) g d in
              aux g ip; aux (ip+1) d
  in
  aux 0 (Array.length t)
;;
```

□ **3.2.** Dans le pire des cas, si le tableau est déjà trié, une portion du tableau de taille n est partitionnée en une portion de taille 1 et une portion de taille $n-1$. La fonction `split3` ayant une complexité linéaire, la complexité du tri rapide vérifie donc : $C(n) = C(n-1) + O(n)$. D'où $C(n) = O\left(n^2\right)$.

Médiane d'un tableau

Question 4. L'utilisation de la fonction mediane permettrait de garantir de partitionner une portion du tableau de taille n en deux portions de taille $\frac{n}{2}$. Ainsi, si la fonction mediane est de complexité $f(n)$, la complexité du tri rapide vérifierait : $C(n) = 2C(n/2) + f(n)$.
□ **4.1.** Si $f(n) = O(n)$, le théorème maître donne : $C(n) = O(n \log n)$.
□ **4.2.** Si $f(n) = O\left(n^2\right)$, le théorème maître donne : $C(n) = O\left(n^2\right)$.

Question 5.
□ **5.1.**

```
let median t k =
  let t' = Array.copy t in
  let rec aux g d =
    let p = t'.(g) in
```

```
    let i_s, i_e = split3 t' p g d in
    match i_s = k || (i_s < k && k < i_e) with
      | true -> t'.(k)
      | false when k < i_s -> aux g (i_s-1)
      | false -> aux i_e d
  in aux 0 (Array.length t)
;;
```

□ **5.2.** Dans le pire des cas, (si le tableau est déjà trié), une portion du tableau de taille n est partitionnée en une portion de taille 1 et une portion de taille $n-1$, et l'appel récursif se fera sur la portion de taille $n-1$. La fonction partition_en_3 ayant une complexité linéaire, la complexité de la fonction mediane vérifie donc : $C(n) = C(n-1) + O(n)$. D'où $C(n) = O\left(n^2\right)$.

Médiane des médianes

Question 6.
□ **6.1.** Soit n la taille de t. Parmi les $\frac{n}{5}$ groupes, la moitié (donc $\frac{n}{10}$) ont leur médiane plus petite que p. De plus, pour chacun de ces $\frac{n}{10}$ groupes, deux éléments sont plus petits que leur médiane, donc 3 éléments de chacun de ces groupes sont plus petits que p. Ainsi, on a au moins $\frac{3}{10}n$ éléments de t inférieurs à p.

De manière analogue, on a au moins $\frac{3}{10}n$ éléments de t supérieurs à p.

□ **6.2.** Dans le pire des cas, on va donc partitionner une portion du tableau de taille n en une portion de taille $\frac{3}{10}n$ et une portion de taille $\frac{7}{10}n$, et l'appel récursif se fera dans le pire des cas sur la portion de taille $\frac{7}{10}n$.

□ **6.3.** Au final, la complexité de notre algorithme vérifiera la relation :

$$C(n) \leqslant \underbrace{C\left(\frac{1}{5} \times n\right)}_{\substack{\text{Calcul de la}\\ \text{médiane des}\\ \frac{n}{5}\text{ médianes}}} + \underbrace{C\left(\frac{7}{10} \times n\right)}_{\substack{\text{Recherche de l'élément}\\ \text{par récurrence sur un}\\ \text{morceau de la partition}}} + \underbrace{\alpha \times n}_{\substack{\frac{n}{5}\text{ mécul des}\\ +\text{ partition}}}$$

□ **6.4.** Montrons par récurrence sur n que $\forall n \in \mathbb{N}, C(n) \leqslant 10 \times \alpha \times n$.

- Initialisation : quitte à prendre un α assez grand, la propriété sera vraie pour les premières valeurs de n.
- Hérédité :

$$\begin{aligned} C(n) &\leqslant C\left(\frac{1}{5} \times n\right) + C\left(\frac{7}{10} \times n\right) + \alpha \times n \\ &\leqslant 10 \times \alpha \times \frac{n}{5} + 10 \times \alpha \times \frac{7}{10} \times n + \alpha \times n \\ &= (2+7+1) \times \alpha \times n \\ &= 10 \times \alpha \times n \end{aligned}$$

La complexité de cet algorithme est linéaire.

Question 7. La version suivante utilise le tri par sélection.

```
let median5 t g d =
  for i = g to d do
    let im = ref i in
    for j = i+1 to d do
      if t.(j) < t.(!im) then
        im := j
    done ;
    swap t i !im
  done ;
  (g+d)/2
;;
```

Question 8.
□ **8.1.**

```
val pivot : 'a array -> int -> int -> int = <fun>
val select : 'a array -> int -> int -> int -> int = <fun>
```

□ **8.2.** La fonction `pivot` parcourt la portion donnée du tableau par groupes de 5, et calcule la médiane de chacun des groupes à l'aide de `median5`. Elle déplace au fur et à mesure ces médianes au début de la portion du tableau et calcule la médiane de ces médianes grâce à l'appel à `select` (on renvoie l'indice de cet élément).

L'appel `select t g d k` renvoie le k-ème plus petit élément de `t[g:d]`. Pour cela, elle utilise la fonction pivot pour calculer l'indice de la médiane des médianes de `t[g:d]`, puis s'en sert comme pivot pour partitionner la portion avec `split3`. Au final, les éléments du tableau se situant entre les indices i_s et i_e sont à leur place finale (et sont tous égaux). Si $i_s = k$ ou $(i_s < k < i_e)$, le k-ème plus petit élément du tableau est désormais à l'indice k, on peut donc renvoyer k. Sinon :

- si $k < i_s$, l'élément recherché se trouve strictement avant l'indice i_s ;
- sinon, $i_e \leqslant k$, et l'élément recherché se trouve après l'indice i_e.

Dans les deux cas, on fait un appel récursif sur la portion du tableau adéquate.

□ **8.3.** Puisque les fonctions `pivot` et `select` s'appellent mutuellement, il suffit de montrer par exemple que `select` termine pour montrer du même coup que `pivot` termine.

Notons $n = d - g$. L'appel `select t g d k` effectue l'appel `pivot t g d` qui va lui-même effectuer un appel récursif à `select` sur une portion de taille $n' = d' - g < d - g = n$. Elle effectuera ensuite une autre appel récursif sur une portion de taille $\leqslant \frac{7}{10}n$ (d'après une question précédente). Tous les appels récursifs à `select` se font donc des portions du tableau strictement plus petites que n. En plus des appels récursifs, on effectue un appel à la fonction `split3` qui termine, et une boucle `while` qui termine car `i` augment strictement à chaque passage dans la boucle. Ainsi, la fonction `select` (et la fonction `pivot`) termine(nt).

□ **8.4.** L'appel à `spit3` ainsi que la boucle `while` s'effectuent en $O(n)$ (où $n = d - g$), car les appels à `median5` s'effectuent en $O(1)$ (morceaux de taille au plus 5). Le premier appel récursif à `select` s'effectue sur une portion de taille $\frac{n}{5}$ et le second appel récursif à `select` s'effectue sur une portion de taille au plus $\frac{7}{10}n$. Ainsi, la fonction `select` est de complexité linéaire.

Question 9.

```
let qselect t k =
  let t' = Array.copy t in
  let i = select t' 0 (Array.length t' - 1) k in
  t'.(i)
;;
```

Question 10.

```
let qsort2 t =
  let rec aux g d = match g < d-1 with
    | false -> ()
    | true -> let p = qselect t ((g+d)/2) in
              let i_s, _ = split3 t p g d in
              aux g i_s; aux (i_s+1) d
  in
  aux 0 (Array.length t)
;;
```