Exercice 1 (Le tri rapide) :

1.
```
let echange t i j =
  let x = t.(i) in
  t.(i) <- t.(j) ;
  t.(j) <- x
;;
```

2.
```
let partition_en_3 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)
;;
```

3. (a)
```
let tri_rapide1 t =
  let rec aux g d = match g < d-1 with
    | false -> ()
    | true -> let ip,_ = partition_en_3 t t.(g) g d in
      aux g ip ; aux (ip+1) d
  in aux 0 (Array.length t)
;;
```

(b) 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 `partition_en_3` ayant une complexité linéaire, la complexité du tri rapide vérifie donc : $C(n) = C(n-1) + O(n)$.
D'où $C(n) = O(n^2)$.

Exercice 2 (Calcul de la médiane d'un tableau) :

1. 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 : $C(n) = 2 \cdot C(n/2) + f(n)$.
   (a) Si $f(n) = O(n)$, le théorème maître nous donne (avec $\alpha = 1 = \beta$) : $C(n) = O(n \log n)$.
   (b) Si $f(n) = O(n^2)$, le théorème maître nous donne (avec $\alpha = 1$; $\beta = 2$) : $C(n) = O(n^2)$.

2. (a)
```
let mediane t k =
  let t' = Array.copy t in
  let rec aux g d =
    let p = t'.(g) in
    let i_s,i_e = partition_en_3 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)
;;
```

(b) 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 $C(n) = O(n^2)$.

Exercice 3 (L'algorithme de la médiane des médianes) :

1. (a) 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.

(b) 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 $\frac{7}{10}n$.

(c) Au final, la complexité de notre algorithme vérifiera la relation :

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

(d) Montrons par récurrence sur n que $\forall n \in \mathbb{N}, C(n) \leq 10\cdot\alpha\cdot n$.

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

$$\begin{aligned} C(n) &\leq C(\frac{1}{5}\cdot n) + C(\frac{7}{10}\cdot n) + \alpha\cdot n \\ &\leq 10\cdot\alpha\cdot\frac{n}{5} + 10\cdot\alpha\cdot\frac{7}{10}\cdot n + \alpha\cdot n \\ &= (2+7+1)\cdot\alpha\cdot n \\ &= 10\cdot\alpha\cdot n \end{aligned}$$

La complexité de cet algorithme sera donc bien **linéaire**.

2. Voici une version utilisant le tri par sélection :

```
let partition5 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 ;
    echange t i !im
  done ;
  (g+d)/2
;;
```

3. (a)
```
val pivot : 'a array -> int -> int -> int = <fun>
val select : 'a array -> int -> int -> int -> int = <fun>
```

(b) 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 `partition5`. 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).

La fonction `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 `partition_en_3`. 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), donc :

- 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, $i_e \leq k$, 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.

(c) 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` (ligne 18) qui va lui même effectuer un appel récursif à `select` sur une portion de taille $n' = d' - g < d - g = n$ (ligne 14). Elle effectuera ensuite une autre appel récursif (ligne 23 ou 24) sur une portion de taille $\leq \frac{7}{10}n$ (d'après la question 1b). Tous les appels récursifs à `select` se font donc bien sur des portions du tableau strictement plus petites que n.

En plus des appels récursifs, on effectue un appel à la fonction `partition_en_3` (ligne 20) qui termine, et une boucle `while` (lignes 5 à 11) qui termine car `i` augment strictement à chaque passage dans la boucle.

Ainsi, la fonction `select` (et la fonction `pivot`) termine(nt).

(d) L'appel à `partition_en_3`, ainsi que la boucle `while` des lignes 5 à 11, s'effectuent en $O(n)$ (où $n = d - g$), car les appels à `partition5` s'effectuent en $O(1)$ (morceaux de taille au plus 5). De plus, l'appel récursif à `select` de la ligne 14 s'effectue sur une portion de taille $\frac{n}{5}$, et l'appel récursif à `select` de la ligne 23 ou 24 s'effectue sur une portion de taille au plus $\frac{7}{10}n$.

Ainsi, d'après la question 1d, la fonction `select` est de complexité **linéaire**.

4.
```
let selection_rapide t k =
  let t' = Array.copy t in
  let i = select t' 0 (Array.length t' -1) k in
  t'.(i)
;;
```

5.
```
let tri_rapide2 t =
  let rec aux g d = match g<d-1 with
    | false -> ()
    | true ->
      let p = selection_rapide t ((g+d)/2) in
      let i_s, _ = partition_en_3 t p g d in
      aux g i_s ;
      aux (i_s+1) d
  in aux 0 (Array.length t)
;;
```