Nous avons vu en cours que l'algorithme du tri rapide à une complexité en $O(n^2)$ dans le pire des cas. On peut montrer que sa complexité en moyenne est en $O(n \log n)$.

Le but de ce DM est de mieux choisir le pivot pour obtenir une complexité en $O(n \log n)$ dans le pire des cas.

Exercice 1 (Le 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 $t[g : d]$ la portion de t comprise entre les indices g inclus et d exclu. Par exemple, si t est de taille n, la portion totale de t est $t[0 : n]$.

Pour cet algorithme de tri, on va commencer par **partitionner** une portion $t[g : d]$ de notre tableau selon un **pivot** (qui est une valeur p comparable aux éléments de notre 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 :

- $\forall g \leq i < i_s,\ t.(i) < p$;
- $\forall i_s \leq i < i_e,\ t.(i) = p$;
- $\forall i_e \leq i < d,\ t.(i) > p$.

Remarque : contrairement à la version vue en cours, on partitionne ici en 3 morceaux.

1. Écrire une fonction `echange` de type `'a array -> int -> int -> unit` qui prend en entrée un tableau `t` et deux entiers `i` et `j`, et qui échange par effets de bords les valeurs de `t.(i)` et `t.(j)`.
2. Écrire une fonction `partition_en_3` de type `'a array -> 'a -> int -> int -> int * int` qui prend en entrée un tableau `t`, un pivot `p`, et deux indices g et d, et qui effectue la partition décrite ci-dessus (en modifiant le tableau `t` par effets de bords) et renvoie les valeurs des indices (i_s, i_e).
   Votre algorithme devra avoir une complexité **linéaire**.
3. (a) Écrire une fonction `tri_rapide1` de type `'a array -> unit` qui trie par effets de bords un tableau pris en entrée et ne renvoie rien. Votre algorithme devra reposer sur le tri rapide et choisir l'élément le plus à gauche de la portion à trier comme pivot.
   (b) Quelle est la complexité de votre algorithme dans le pire des cas ?

Exercice 2 (Calcul de la médiane d'un tableau) : On cherche à améliorer la complexité du tri rapide dans le pire des cas. Pour cela, on va essayer de choisir un pivot garantissant de partitionner notre tableau en deux morceaux de même taille (à un élément près). Pour cela, on pourrait chercher 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.

1. Dans cette question, on suppose l'existence d'une fonction `mediane` de type `'a array -> int -> 'a` telle que `mediane 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 `mediane` pour choisir le pivot.
   (a) Quelle serait la complexité de cet algorithme de tri rapide si $f(n) = O(n)$?
   (b) Quelle serait la complexité de cet algorithme de tri rapide si $f(n) = O(n^2)$?

Pour implémenter la fonction `mediane`, on utilise une approche qui s'inspire du tri rapide (mais qui n'effectue qu'un seul appel récursif au lieu de deux) :

- on fait une copie `t'` de `t` avec l'instruction `let t' = Array.copy t in ...` ;
- on partitionne `t'` à l'aide de la fonction `partition_en_3` ;
- 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).

2. (a) Implémenter une fonction `mediane` utilisant le principe précédent.
   (b) Quelle est la complexité de votre programme dans le pire des cas ?

Exercice 3 (L'algorithme de la médiane des médianes) : On pourrait montrer que 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 cela, on va également optimiser le choix du pivot lors du calcul de la médiane. Le principe est le suivant (pour un tableau `t` de taille n) :

(i) on regroupe les éléments de `t` en $\frac{n}{5}$ groupes de 5 éléments ;

(ii) on calcule (en temps constant) la médiane de chacun de ces groupes de 5 : m_1, m_2, ..., $m_{\frac{n}{5}}$;

(iii) on calcule récursivement la médiane de ces médianes ;

(iv) on utilise cette valeur comme pivot pour partitionner `t`.

1. Avant d'implémenter l'algorithme, voyons pourquoi ce choix de pivot va améliorer la complexité du calcul de la médiane dans le pire cas.
   (a) Soit p le pivot obtenu (la 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` sont supérieurs à p.
   (b) Dans le pire des cas, quelle sera alors la taille de la portion de `t` où s'effectuera l'appel récursif ?
   (c) Notons $C(n)$ la complexité de l'algorithme `mediane` sur un tableau de taille n. Justifier que $C(n) \leq C(\frac{1}{5} \cdot n) + C(\frac{7}{10} \cdot n) + \alpha \cdot n$, où α est une constante qu'on n'essaiera pas d'exprimer.
   (d) Montrer que $\forall n \in \mathbb{N}, C(n) \leq 10 \cdot \alpha \cdot n$.
2. Écrire une fonction `partition5` de type `'a array -> int -> int -> int` qui cherche la médiane des éléments `t.(g)`, `t.(g+1)`, ..., `t.(d)`, et renvoie l'indice où il se trouve (on aura le droit de modifier cette portion du tableau `t` par effets de bords).
   Indication : en pratique, on n'utilisera cette fonction que si $0 \leq d - g \leq 5$, on ne se préoccupera donc pas de sa complexité asymptotique. On pourra 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.
3. On fournit le code des deux fonctions mutuellement récursives suivantes :

```
let rec pivot t g d = match d-g < 5 with
  | true -> partition5 t g d
  | false ->
    let i = ref g in
    while !i <= d do
      let sd = min (!i+4) d in
      let median5 = partition5 t !i sd in
      let mi = g + (!i-g)/5 in
      echange t median5 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 = partition_en_3 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
;;
```

   (a) Quel est le type des fonctions `pivot` et `select` ?

(b) Expliquer le rôle de chacune de ces fonctions vis-à-vis de l'algorithme de la médiane des médianes (sans prouver formellement leur correction).

(c) Montrer que les fonctions `pivot` et `select` terminent.

(d) Quelle est la complexité de la fonction `select` dans le pire des cas ?

4. Écrire une fonction `selection_rapide` de type `'a array -> int -> 'a` qui prend en entrée un tableau `t` et en entier k, et qui renvoie le k-ème plus petit élément du tableau `t`.
Attention : on prendra garde à ne **pas** modifier `t` par effets de bords.

5. Écrire une fonction `tri_rapide2` de type `'a array -> unit` qui trie par effets de bords un tableau pris en entrée et ne renvoie rien. Votre algorithme devra reposer sur le tri rapide et choisir la médiane de la portion à trier comme pivot.
Votre algorithme devra avoir une complexité en $O(n \log n)$ dans le pire des cas.