Les calculatrices, ordinateurs et documents de cours sont interdits.
Toutes les complexités seront exprimées avec la notation O(...) et **doivent être justifiées/prouvées**.
Vous avez le droit d'admettre une question pour passer à la suivante.
Les exercices sont indépendants et vous pouvez les traiter dans l'ordre que vous préférez.

I Minimum et maximum

Écrire une fonction `minmax` telle que, si `l` est une liste d'entiers, `minmax l` renvoie un couple `(mini, maxi)` où `mini` est le minimum de `l` et `maxi` son maximum.
Bonus : l'écrire en utilisant (à ± 1 près) $\frac{3n}{2}$ comparaisons (utilisations de `<` ou `<=`), où n est la taille de `l`.

II Tri rapide

1. Écrire une fonction `concat : 'a list -> 'a list -> 'a list` telle que `concat l1 l2` renvoie une liste composée des éléments de `l1` suivi des éléments de `l2`, sans utiliser `@`.
   Quelle est la complexité de `concat` ?

2. Écrire une fonction `partition` telle que, si `l` est une liste d'entiers et `p` un entier, `partition l p` renvoie un couple `(l1, l2)` où :
   - `l1` est une liste contenant les éléments de `l` inférieurs strictement à `p`
   - `l2` est une liste contenant les éléments de `l` supérieurs ou égaux à `p`

   Quelle est la complexité de `partition` ?

Le tri rapide d'une liste `l` consiste à :

- Choisir un élément (appelé pivot) de `l`. Ici on prendra le premier élément `p` de `l` comme pivot.
- Séparer les éléments de `l` autres que `p` en deux listes : la liste `l1` des éléments strictement inférieurs à `p` et la liste `l2` des éléments supérieurs à `p`.
- Trier récursivement `l1` et `l2` pour obtenir des listes triées `l1'` et `l2'`.
- Renvoyer la concaténation de `l1'`, `p` et `l2'`.

3. Écrire une fonction `quicksort : 'a list -> 'a list` triant une liste avec le tri rapide.

4. Quelle est la complexité de `quicksort l` pour une liste `l` de taille n déjà triée dans l'ordre croissant[1] ?

5. Quelle est la complexité de `quicksort` sur une liste de taille n quand la partition est toujours équilibrée dans les appels récursifs (les deux listes `l1` et `l2` sont de même taille)[2] ?

[1] On peut montrer qu'il s'agit du pire cas
[2] On peut montrer qu'il s'agit du meilleur cas

III Recherche par trichotomie

1. On considère deux entiers i et j tels que $0 \leq i \leq j$.
   Exprimer, en fonction de i et j, des entiers m_1 et m_2 qui partagent les entiers entre i et j en 3 ensembles de même taille (à $\pm$ 1 près). Plus précisément, m_1 et m_2 doivent vérifier :
   - $i \leq m_1 \leq m_2 \leq j$
   - Les trois ensembles suivants contiennent le même nombre d'entiers (à $\pm$ 1 près) :

   $$\{i, i+1, ..., m_1\},\ \{m_1+1, m_1+2, ..., m_2\},\ \{m_2+1, m_2+2, ..., j\}$$

2. Écrire une fonction `tricho` telle que `tricho t e` détermine si `e` appartient à un tableau trié `t`, en utilisant une méthode similaire à la recherche par dichotomie mais en découpant l'intervalle en 3 plutôt que 2.
3. Donner la complexité de `tricho` et comparer avec la recherche par dichotomie.

IV Méthode des deux pointeurs

Écrire une fonction `somme2 : int array -> int -> int*int` telle que, si `t` est un tableau trié de taille n, `somme2 t p` renvoie un couple `(i, j)` tel que `i` $\neq$ `j` et `t.(i) + t.(j)` = `p`. Si un tel couple n'existe pas, on renverra `(-1, -1)`.
`somme2 t p` doit être en complexité $O(n)$ et ne pas créer de nouvelle structure de donnée (pas de création de tableau, liste...)[3].

Indice : Utiliser deux références `i` et `j` valant initialement 0 et $n-1$. Que peut-on faire si `t.(i) + t.(j) < p` ? Et si `t.(i) + t.(j) > p` ?

V Élément majoritaire

Dans cet exercice, on veut trouver un élément strictement majoritaire dans un tableau de n entiers naturels, c'est à dire un élément apparaissant strictement plus de $\frac{n}{2}$ fois.

1. Écrire une fonction `occ : 'a -> 'a array -> int` telle que `occ e t` renvoie le nombre d'apparitions de `e` dans `t`.
   Par exemple, `occ 2 [|1; 2; 6; 2; 2; 8|]` doit renvoyer 3.
2. En déduire une fonction `maj` pour trouver un élément majoritaire dans un tableau. Si `t` n'a pas d'élément majoritaire, `maj t` renverra -1.
3. Quelle est la complexité de `maj t` sur un tableau `t` de taille n?

On considère maintenant la fonction suivante :

```
let vote t =
    let e = ref t.(0) in
    let k = ref 1 in
    for i = 0 to Array.length t - 1 do
        if t.(i) = !e then incr k else decr k;
        if !k = 0 then (e := t.(i); k := 1)
    done;
    !e
```

On rappelle que `incr k`/`decr k` augmente/diminue la valeur de la référence `k` de 1.

4. Supposons que le tableau `t` ait un élément strictement majoritaire `m`. Montrer que `vote t` renvoie `m`.
   Indice : considérer $c =$ `k` si `!e = m`, $c =$ `-k` sinon (c change donc au cours de l'algorithme).
5. En déduire une fonction `maj2` renvoyant un élément strictement majoritaire d'un tableau de taille n en complexité $O(n)$. On renverra -1 s'il n'y a pas d'élément strictement majoritaire.

[3] Autrement dit, la complexité en mémoire doit être $O(1)$