Important : Les programmes doivent être écrits en OCaml. Une grande importance sera apportée à la clarté, la lisibilité des raisonnements et des programmes.

Remarque : On pourra toujours librement utiliser une fonction demandée à une question précédente, même si cette question n'a pas été traitée.

1 Analyse de Programme

Exercice 1 : On considère la fonction suivante :

```
let f n =
  let s = ref 1 in
  let c = ref 0 in
  while !s <= n do
    incr c ;
    s := !s + 2 * !c + 1
  done ;
  !c
;;
```

1. Démontrer la terminaison de `f`.
2. Pour $n \in \mathbb{N}$, conjecturer une relation entre `f n` et `n`. En déduire une spécification de `f`.
3. Démontrer la correction de `f` relativement à cette spécification.
4. Déterminer la complexité de `f`.

2 Valeur majoritaire

Exercice 2 :

1. Écrire une fonction `valeur_majoritaire` prenant en argument un tableau `t` et renvoyant un élément de `t` dont le nombre d'occurrences est maximal.
   Par exemple :
   `valeur_majoritaire [|1;3;2;1;2;1|]` doit renvoyer 1.
   `valeur_majoritaire [|1;3;2;1;2|]` peut renvoyer 1 ou 2.
2. Démontrer la terminaison de cette fonction.
3. Déterminer la complexité de cette fonction.
4. Dans le cas d'une fonction itérative, donner pour chaque boucle utilisée un invariant de boucle permettant de déduire la correction de la fonction.
   Dans le cas d'une fonction récursive, donner la propriété à démontrer par récurrence.
   On ne demande pas le détail des démonstrations elles-mêmes.
5. On suppose qu'on dispose d'une fonction `trier` qui prend en argument un tableau `t` et renvoie un tableau **trié** contenant les mêmes éléments, et ayant une complexité dans le pire cas en $O(n \log n)$ (où n est la taille de `t`).
   En utilisant cette fonction, améliorer la complexité du calcul de la valeur majoritaire.

3 Recherche par dichotomie

Exercice 3 : Dans cet exercice, on s'intéresse à la recherche par dichotomie d'un élément dans un tableau trié, implémentée par la fonction suivante :

```
let dicho t x =
  let n = Array.length t in
  let g = ref 0 in
  let d = ref (n-1) in
  while !d - !g > 0 do
    let m = (!g + !d)/2 in
    if t.(m) >= x then d := m
    else g := m
  done ;
  t.(!g) = x
;;
```

1. Déterminer le type de la fonction `dicho`.
2. Pour quelle raison cette fonction ne commence-t-elle pas par appliquer la fonction `trier` de l'exercice précédent sur le tableau argument, ce qui permettrait de ne pas forcément supposer que le tableau soit initialement trié ?
3. Justifier que la proposition suivante est préservée à chaque passage dans la boucle while :

   $$P : x \text{ apparait dans } \texttt{t} \text{ si, et seulement si, il apparait entre les indices } g \text{ et } d \text{ inclus.}$$

4. La fonction `dicho` ne termine pas. L'illustrer sur un exemple.
5. Corriger la fonction pour qu'elle termine sur toute entrée tout en restant correcte.
6. Démontrer la terminaison et la correction de la fonction corrigée.
7. (a) Écrire une fonction `tricho`, ayant la même spécification que `dicho`, mais découpant l'espace de recherche en 3 plutôt qu'en 2 à chaque étape.
   Cette fonction devra être récursive (ou utiliser une fonction auxiliaire récursive).

   (b) Déterminer la complexité de `tricho`.