Exercice 1 (Fonctions sur les listes) :

1.
```
let rec iter f l = match l with
  | [] -> ()
  | t::q -> f t ; iter f q
;;
```

2.
```
let rec filter f l = match l with
  | [] -> []
  | t::q when f t -> t::(filter f q)
  | t::q -> filter f q
;;
```

3.
```
let rec append l1 l2 = match l1 with
    | [] -> l2
    | t::q -> t::(append q l2)
;;
```

4.
```
let rec flatten l = match l with
  | [] -> []
  | l1::q -> append l1 (flatten q)
;;
```

Exercice 2 (Fonction de Hofstadter) :

1. `h 0 = 0;` `h 1 = 1;` `h 2 = 1;` `h 3 = 2;` `h 4 = 3.`
2. Montrons par récurrence forte que $\forall n \in \mathbb{N}$, `h n` termine et $0 \leq$ `h n` $\leq n$.
   Initialisation : la propriété est vraie pour $n = 0$ d'après la question précédente.
   Hérédité : Soit $n > 0$. Supposons que la propriété soit vraie pour $n' < n$. Alors, par HR, `h(n-1)` termine, en renvoie un entier n' tel que $0 \leq n' \leq n - 1 < n$. Donc, par HR, `h(n')` termine, et renvoie un entier un entier n'' tel que $0 \leq n'' \leq n' \leq n - 1$. Donc $n \geq n - n'' \geq n - (n-1) = 1 \geq 0$. Ainsi, `h n` termine, et renvoie $n - n''$, qui est bien dans l'intervalle voulu.
   Ainsi, par principe de récurrence, la propriété est vraie pour tout $n \in \mathbb{N}$.

Problème : Élection par majorité

Exercice 3 (Méthode naïve) :

1. (a)
```
let nb t a =
  let n = Array.length t in
  let c = ref 0 in
  for i=0 to (n-1) do
    if t.(i) = a then incr c
  done ;
  !c
;;
```

   (b) La complexité de `nb` est $O(N)$.

2. (a)
```
let elu1 t =
  let n = Array.length t in
  (* il faut déjà chercher la valeur de k *)
  let k = ref 0 in
  for i=0 to (n-1) do
    k := max !k t.(i)
  done ;
  (* ensuite on teste tous les candidats possibles *)
  let elu = ref (-1) in
  for a=0 to !k do
    if nb t a > n/2 then elu := a
  done ;
  !elu
;;
```

Remarque : on pourrait remplacer la seconde boucle par une boucle while et en sortir dès qu'on a trouvé le candidat élu, mais ça ne changerait pas la complexité dans le pire cas.

(b) La première boucle se fait en $O(N)$, et la seconde en $O(kN)$.
D'où une complexité totale en $O(kN)$.

Exercice 4 (Diviser pour régner simple) :

1. Par **contraposée** : si a n'est pas élu par $T[0:m]$ ni par $T[m:N]$, alors a apparait au plus $m/2$ fois dans $T[0:m]$ et au plus $(N-m)/2$ fois dans $T[m:N]$, donc au plus $N/2$ fois au total.
Donc a n'est pas élu par T.

2.
```
let nb2 t a i j =
  let c = ref 0 in
  for k=i to (j-1) do
    if t.(k) = a then incr c
  done ;
  !c
;;
```

3. (a)
```
let rec dpr_simple t i j = match j-i with
  | 0 -> (-1,0)
  | 1 -> (t.(i),1)
  | _ ->
    let m = (i+j)/2 in
    let (ag,qg) = dpr_simple t i m in
    let (ad,qd) = dpr_simple t m j in
    let qg' = qg + nb2 t ag m j in
    let qd' = qd + nb2 t ad i m in
    if max qg' qd' <= (j-i)/2 then (-1,0) (* pas de candidat élu *)
    else if qg'>qd' then (ag,qg') (* candidat de gauche élu *)
    else (ad,qd') (* candidat de droite élu *)
;;
```

(b) La fonction `dpr_simple` effectue deux appels récursifs (lignes 6 et 7) sur des portions de `t` strictement plus petites. De plus, le reste du code termine, car il n'est constitué que d'opérations élémentaires et d'appels à `nb` qui termine sur toute entrée. Donc `dpr_simple` termine.

4.
```
let elu2 t = let (a,_) = dpr_simple t 0 (Array.length t) in a ;;
```

5. Les 2 appels récursifs se font sur tes portions de `t` de tailles deux fois plus petites, et le reste du code s'effectue en $O(N)$. Donc la complexité vérifie $C(N) = 2 \cdot C(N/2) + O(N)$.
   Ainsi, d'après le théorème maître, $C(N) = O(N \log N)$.
   Remarque : l'avantage de cette méthode est que la complexité ne dépend plus de k. Cependant, si $k << \log N$, la méthode naïve est plus efficace.

Exercice 5 (Diviser pour régner complexe) :

1. 3 est un postulant de $T =$ `[|1;2;3;4;3;2;3;3|]` pour $n = 5$. En effet :
   - On a bien $n = 5 > 4 = N/2$.
   - 3 apparait $4 \leq n$ fois dans T.
   - les entiers 1, 2 et 4 apparaissent moins de $3 = N - n$ fois dans T.
2. Si a est élu dans T, notons n le nombre de fois que a apparait dans T. Alors a est un postulant de T pour n.
3. Soit a un postulant de T pour n (on a donc $n > N/2$). Supposons que $b \neq a$ soit élu par T. Donc b apparait $n' > N/2$ fois dans T. Mais b n'apparait au plus que $N - n$ fois dans T, avec $N - n \leq N/2$: contradiction. Donc autre autre valeur que a ne peut être élue par T.
4. (a) Chaque élément du tableau `[|1;2;3;4|]` est un postulant pour $n = 3$.
   (b) le tableau `[|1;1;2;2|]` n'admet pas de postulant.
5. (a) Posons $n = l + \frac{m}{2} > \frac{m}{2} + \frac{m}{2} = N/2$.
   $T[m : N]$ n'a pas d'élu, donc toute valeur apparait au plus $m/2$ fois dans $T[m : N]$.
   De plus, a apparait au plus l fois dans $T[0 : m]$. Donc a apparait au plus n fois dans T.
   D'autre part, pour $b \neq a$, b apparait au plus $m - l$ fois dans $T[0 : m]$.
   Donc b apparait au plus $m - l + \frac{m}{2} = 2m - (l + \frac{m}{2}) = N - n$ fois dans T.
   Ainsi, a est un postulant de T pour n.

   (b) i. Posons $n = l + q > \frac{m}{2} + \frac{m}{2} = N/2$.
   a apparait bien au au plus n fois dans T.
   De plus, pour $c \neq a$, c apparait bien au plus $m - l + m - q = N - n$ fois dans T.
   Donc a est un postulant de T pour n.

   ii. Posons $n = m + q - l > m = N/2$.
   b apparait au plus $m - l$ fois dans $T[0 : m]$ et au plus q fois dans $T[m : N]$, donc au plus n fois dans T.
   a apparait au plus l fois dans $T[0 : m]$ et au plus $m - q$ fois dans $T[m : N]$, donc au plus $m - q + l = 2m - (m + q - l) = N - n$ fois dans T.
   Soit $c \notin \{a, b\}$. c apparait au plus $m - l$ fois dans $T[0 : m]$ et au plus $m - q$ fois dans $T[m : N]$, donc au plus $m - l + m - q = N - q - l \leq N - q - (m - l) = N - n$ fois dans T.
   (car $l \geq m - l$ par définition de l)
   Donc b est un postulant de T pour n.

   iii. a apparait au plus l fois dans $T[0 : m]$ et au plus $m - l$ fois dans $T[m : N]$, donc au plus m fois dans T. Donc a n'est pas élu. De manière symétrique, b n'est pas non élu. De plus, il est évident que $c \notin \{a, b\}$ n'est pas élu, car il n'est ni élu dans $T[0 : m]$ ni dans $T[m : N]$.
   Donc T n'admet pas d'élu.

6.
```
let rec postulant t i j = match j-i with
  | 0 -> (-1,0)
  | 1 -> (t.(i),1)
  | _ ->
    let m = (i+j)/2 in
    let (a,l) = postulant t i m in
    let (b,q) = postulant t m j in
    match (a,b,l,q) with
    | -1,-1,_,_ -> (-1,0) (* pas d'élu des deux côtés *)
    | _,-1,_,_ -> (a,l+m/2) (* Question 5a *)
    | -1,_,_,_ -> (b,q+m/2) (* symétrique de Question 5a *)
    | _ when a=b -> (a,l+q) (* Question 5bi *)
    | _ when q>l -> (b,m+q-l) (* Question 5bii *)
    | _ when l>q -> (a,m+l-q) (* symétrique Question 5bii *)
    | _ when l=q -> (-1,0) (* Question 5biii *)
;;
```

7.
```
let elu3 t =
  let n = Array.length t in
  let a,_ = postulant t 0 n in
  if a = -1 then -1 (* pas d'élu *)
  else (* on vérifie que le postulant est élu *)
  if nb t a > n/2 then a
  else -1
;;
```

8. La complexité de `postulant` vérifie $C(N) = 2 \cdot C(N/2) + O(1)$ (temps constant en dehors des 2 appels récursifs). Donc, d'après le théorème maître, $C(N) = O(N)$.
   De plus, on vérifie ensuite si a est élu en $O(N)$.
   D'où une complexité totale de `elu3` en $O(N)$.