Devoir surveillé n° 2
Éléments de correction

Problème 1

1.
```
let somme t i j =
  let s = ref 0 in
  for k = i to j do
    s := !s + t.(k)
  done;
  !s
;;
```

2.
```
let somme_max1 t =
  let n = Array.length t in
  let sMax = ref t.(0) in
  for i = 0 to n-1 do
    for j = i to n-1 do
      sMax := max !sMax (somme t i j)
    done;
  done;
  !sMax
;;
```

3. Évaluons la complexité en comptant le nombre d'additions effectuées par la fonction `somme_max`. Pour $0 \leqslant i \leqslant j \leqslant n-1$, l'appel `somme t i j` nécessite $j-i+1$ additions.
Par conséquent, le nombre d'additions réalisées lors de l'appel `somme_max1 t` est :

$$\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}(j-i+1) = \frac{n^3+3n^2+2n}{6}$$

Donc $\boxed{\text{la complexité de somme\_max1 t est en } O(n^3)}$ (et cette borne est atteinte).

4.
```
let somme_max2 t =
  let n = Array.length t in
  let sMax = ref t.(0) in
  for i = 0 to n-1 do
    let s = ref 0 in
    for j = i to n-1 do
      s := !s + t.(j);
      sMax := max !sMax !s
    done
  done;
  !sMax
;;
```

5. Chaque itération de la boucle en j s'effectue en temps $O(1)$, et le nombre d'itérations est $\sum_{i=0}^{n-1}(n-i) = \frac{n(n+1)}{2}$, donc $\boxed{\text{la complexité de somme\_max2 t est en } O(n^2)}$ (et cette borne est atteinte).

6. Remarquons que $p_{max}(t,g,d) \geqslant \max\{p_{max}(t,g,m), p_{max}(t,m+1,d), pc_{max}(t,g,m,d)\}$ puisque dans chacun des trois cas, les tranches considérées ont leurs extrémités dans $[\![g,d]\!]$.

Considérons la somme maximale $p_{max}(t,g,d)$ d'une tranche dont les indices sont compris entre g et d ; il existe alors deux entiers i et j tels que $g \leqslant i \leqslant j \leqslant d$ et tels que la somme de la tranche $[|t_i;\dots;t_j|]$ vaut $p_{max}(t,g,d)$.

- Si $j < m$, alors $s(t,i,j) \leqslant p_{max}(t,g,m)$, donc $p_{max}(t,g,d) = p_{max}(t,g,m)$;
- Si $i \leqslant m \leqslant j$, alors $p_{max}(t,g,d) = pc_{max}(t,g,m,d)$;
- Sinon, $m < i$, donc $p_{max}(t,g,d) = p_{max}(t,m+1,d)$.

Finalement, $\boxed{p_{max}(t,g,d) = \max\{p_{max}(t,g,m), p_{max}(t,m+1,d), pc_{max}(t,g,m,d)\}}$

7. On utilise deux boucles `for`

- La première pour calculer les valeurs des $s(t,m,k)$ pour $k \in [\![m+1,d]\!]$ et mettre à jour `sMax` si besoin ; à la fin de cette boucle, `sMax` contient la somme maximale d'une tranche débutant en m et dont l'indice de fin j appartient à $[\![m,d]\!]$.
- La seconde pour calculer les valeurs des $s(t,k,j)$ pour $k \in [\![g,m-1]\!]$ et mettre à jour `sMax` si besoin.

```
let smax_part_c t g m d =
  let s = ref t.(m) in
  let sMax = ref t.(m) in
  for k = m+1 to d do
    s := !s + t.(k);
    sMax := max !s !sMax
  done;
  s := !sMax;
```

```
  for k = m-1 downto g do
    s := !s + t.(k);
    sMax := max !s !sMax
  done;
  !sMax
;;
```

Le nombre d'additions effectuées par cette fonction est $(d-m)+(m-g) = d-g$.

8.
```
let rec smax_part t g d =
  if g = d
  then t.(g)
  else
    begin
      let m = (g+d)/2 in
      let s1 = smax_part t g m in
      let s2 = smax_part t (m+1) d in
      let s3 = smax_part_c t g m d in
      max s1 (max s2 s3)
    end
;;
```

9.
```
let somme_max3 t =
  let n = Array.length t in
  somme_max_part t 0 (n-1)
;;e
```

10. $C_1 = 0$ et pour tout $n \geqslant 2$, pour tous $d, g \in \mathbb{N}$ tels que $d-g+1 = n$, si $m = \left\lfloor \dfrac{g+d}{2} \right\rfloor$,
   - L'appel `smax_part t g m` effectue $C_{\lceil n/2 \rceil}$ additions ;
   - L'appel `smax_part t (m+1) d` effectue $C_{\lfloor n/2 \rfloor}$ additions ;
   - L'appel `smax_part_c t g m d` effectue $n-1$ additions.

   Par conséquent, $\boxed{C_n = C_{\lceil n/2 \rceil} + C_{\lfloor n/2 \rfloor} + n - 1}$.

   En utilisant la même méthode que pour la complexité du tri fusion, on montre que $\boxed{\text{la complexité de } \texttt{somme\_max3 t} \text{ est en } O(n \ln n)}$

11. Tout d'abord,
   - ou bien une tranche de somme maximale se terminant en k ne contient que t_k, et dans ce cas $f_{max}(t,k) = t_k$;
   - ou bien une telle tranche privée de t_k est une tranche de somme maximale se terminant en $k-1$, et dans ce cas $f_{max}(t,k) = t_k + f_{max}(t,k-1)$.

   Enfin, une tranche de somme maximale dont les indices sont inférieurs ou égaux à k peut :
   - soit contenir l'élément d'indice k, et dans ce cas $g_{max}(t,k) = f_{max}(t,k)$;
   - soit ne pas contenir l'élément d'indice k, et dans ce cas $g_{max}(t,k) = g_{max}(t,k-1)$.

   Finalement,

$$\boxed{g_{max}(t,k) = \max\{t_k, t_k + f_{max}(t,k-1), g_{max}(t,k-1)\}}$$

12.
```
let somme_max4 t =
  let n = Array.length t in
  let f = ref t.(0) in
  let g = ref t.(0) in
  for k = 1 to n-1 do
    f := max t.(k) (!f + t.(k))  ;
    g := max !g !f
  done;
  !g
;;
```

13. Chaque itération de la boucle s'effectue en $O(1)$, et il y a $n-1$ itérations, donc $\boxed{\text{la complexité de cette fonction est linéaire}}$.

Problème 2

1.
```
let rec card x =
  match x with
  | [] -> 0
  | t::q -> 1 + card q
;;
```

   En dehors d'un éventuel appel récursif, on effectue des opérations en temps $O(1)$. On compte donc le nombre d'appels.

   Or le nombre total d'appels C_n pour une liste de longueur n vérifie $C_0 = 1$ et pour tout $n \in \mathbb{N}^n$, $C_n = 1 + C_{n-1}$, donc pour tout $n \in \mathbb{N}$, $C_n = n+1$.

   $\boxed{\text{La complexité en temps de cette fonction est linéaire}}$.

2.
```
let rec nPetits p x =
  match x with
  | [] -> 0
  | t::q -> (if t < p then 1 else 0) + nPetits p q
;;
```
Comme dans la question précédente,
$\boxed{\text{la complexité en temps de cette fonction est linéaire}}$.

3.
```
let rec partitionP p x =
  match x with
  | [] -> []
  | t::q -> if t < p
            then t::(partitionP p q)
            else partitionP p q
;;
let rec partitionG p x =
  match x with
  | [] -> []
  | t::q -> if t > p
            then t::(partitionG p q)
            else partitionG p q
;;
```
$\boxed{\text{La complexité en temps de ces deux fonctions est linéaire}}$.

4. En utilisant le premier élément de la liste comme pivot p, on obtient trois sous-listes constituées respectivement
   - des éléments strictement plus petits que p ;
   - de l'élément p ;
   - des éléments strictement plus grands que p.

   Soit c le nombre d'éléments strictement plus petits que p. Si $c \geqslant k$, l'élément de rang k se trouve dans la première sous-liste, au rang k ; si $c = k-1$, l'élément de rang k est p ; et si $c < k-1$, l'élément de rang k se trouve dans la dernière sous-liste, au rang $k-c-1$.
```
let rec elementDeRang k x =
  match x with
  | [] -> failwith "Liste␣Vide"
  | p::q -> let c = nPetits p q in
            if c = k-1
            then p
            else
              if c >= k
              then elementDeRang k (partitionP p x)
              else elementDeRang (k-c-1) (partitionG p x)
;;
```

5. Lors d'un appel à `elementDeRang` pour une liste de longueur n, on effectue au plus un appel récursif, pour une liste de longueur au plus $n-1$. Donc le nombre total d'appels est majoré par n.

   De plus, lors de chaque appel, les opérations réalisées en dehors de l'éventuel appel récursif ont une complexité linéaire en la taille $|\ell|$ de la liste, donc majorée par $\alpha|\ell| + \beta$ où α et β sont deux constantes, et par conséquent majorée par $\alpha n + \beta$ (puisque $|\ell| \leqslant n$.

   Finalement, la complexité de la fonction est majorée par $\alpha n^2 + \beta n$, donc $M(n) = O(n^2)$.

   Enfin, dans le cas où la longueur de la liste ne diminue que d'un à chaque appel récursif (c'est par exemple le cas si on cherche l'élément de rang 1 dans une liste triée dans l'ordre décroissant), le nombre total d'appels à `nPetits` sera $\displaystyle\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$ (en comptant les appels récursifs), donc $\boxed{\text{la complexité dans le pire des cas } M(n) \text{ est quadratique}}$.

6. On procède par récurrence forte. Soit $c \in \mathbb{R}^+$.
   - $T(0) = 0$ donc $T(0) \leqslant c.0$.
   - Soit $n \in \mathbb{N}^*$. Supposons que pour tout $k \in [\![0, n-1]\!]$, $T(k) \leqslant ck$.
     Pour tout $i \in [\![1, n]\!]$, $i-1 \in [\![0, n-1]\!]$ et $n-i \in [\![0, n-1]\!]$ donc $T(i-1) \leqslant c(i-1)$ et $T(n-i) \leqslant c(n-i)$.
     Par conséquent, $\max\{T(i-1), T(n-i)\} \leqslant \max\{c(i-1), c(n-i)\}$.

     Si n est pair, il existe $p \in \mathbb{N}^*$ tel que $n = 2p$ et
$$T(n) \leqslant \ell n + \frac{1}{n}\sum_{i=1}^{p} c(2p-i) + \frac{1}{n}\sum_{i=p+1}^{2p} c(i-1)$$
$$T(n) \leqslant \ell n + \frac{c}{n}\sum_{j=p}^{2p-1} (j+j)$$
$$T(n) \leqslant \ell n + \frac{c}{p}\left(\frac{2p(2p-1)}{2} - \frac{p(p-1)}{2}\right)$$

$$T(n) \leqslant \ell n + c(\frac{3p-1}{2})$$
$$T(n) \leqslant \left(\ell + \frac{3c}{4}\right) n.$$

Si n est impair, il existe $p \in \mathbb{N}$ tel que $n = 2p+1$ et

$$T(n) \leqslant \ell n + \frac{1}{n}\sum_{i=1}^{p} c(2p+1-i) + \frac{cp}{n} + \frac{1}{n}\sum_{i=p+2}^{2p+1} c(i-1)$$
$$T(n) \leqslant \ell n + \frac{cp}{n} + \frac{c}{n}\sum_{j=p+1}^{2p}(j+j)$$
$$T(n) \leqslant \ell n + \frac{cp}{n} + \frac{2c}{n}\left(\frac{2p(2p+1)}{2} - \frac{p(p+1)}{2}\right)$$
$$T(n) \leqslant \ell n + \frac{c}{n}\left(3p^2+2p\right)$$
$$T(n) \leqslant \ell n + \frac{c}{4n}\left(3n^2-2n-1\right)$$
$$T(n) \leqslant \left(\ell + \frac{3c}{4}\right) n$$

Or $\ell + \dfrac{3c}{4} = c \Leftrightarrow c = 4\ell$, donc pour $c = 4\ell$, la propriété est héréditaire.

Finalement, pour tout $n \in \mathbb{N}$, $T(n) \leqslant cn$ avec $c = 4\ell$.

7.
```
let rec medians x =
  match x with
  | t1::t2::t3::t4::t5::q
    -> (elementDeRang 3 [t1; t2; t3; t4; t5])::(medians q)
  | _ -> []
;;
```

La complexité temporelle de cette fonction est linéaire.

8.
```
let rec elementDeRangBis k x =
  if card x < 5
  then elementDeRang k x
  else
    begin
      let y = medians x in
      let p = elementDeRangBis ((card y + 1)/2) y in
      let pp  = partitionP p x in
      let c = card pp in
      print_int c;
      if c = k-1
      then p
      else
        if c >= k
        then elementDeRangBis k pp
        else elementDeRangBis (k-c-1) (partitionG p x)
    end
;;
```

9. Dans le cas où la liste comporte au plus 4 éléments, le temps maximum est borné. Dans le cas où la liste comporte au moins 5 éléments, on effectue des opérations de complexité en temps linéaire (appels à `medians`, `card`, `partitionP`, `partitionG`) ou borné, et au plus deux appels récursifs :
   - l'un pour calculer un médian de y, dont la longueur est $\left\lfloor \dfrac{n}{5} \right\rfloor$:
   - l'autre appliqué soit à `partitionP p x`, soit à `partitionG p x`.

   Il nous faut donc majorer les longueurs des listes renvoyées par ses deux appels (on supposera M' croissante). D'après le calcul de p, y comporte $\left\lfloor \dfrac{\lfloor n/5 \rfloor - 1}{2} \right\rfloor$ éléments strictement plus petits que p et $\left\lceil \dfrac{\lfloor n/5 \rfloor - 1}{2} \right\rceil$ éléments strictement plus grands.

   Or chaque élément de y est le médian d'un paquet de cinq éléments de x.

   Par conséquent,
   - $\left\lceil \dfrac{\lfloor n/5 \rfloor - 1}{2} \right\rceil + 1$ paquets ont un médian plus grand ou égal à p, donc seuls 2 de leurs éléments peuvent être strictement inférieurs à p ;
   - les $\left\lfloor \dfrac{\lfloor n/5 \rfloor - 1}{2} \right\rfloor$ autres paquets ont au plus 5 éléments strictement inférieurs à p ;
   - enfin, au plus 4 éléments ne sont dans aucun paquet.

   Le nombre d'éléments de `partitionP p x` est donc majoré par

$$2\left\lceil \frac{\lfloor n/5 \rfloor + 1}{2} \right\rceil + 5\left\lfloor \frac{\lfloor n/5 \rfloor - 1}{2} \right\rfloor + 4 \leqslant 7\left\lfloor \frac{n}{10} \right\rfloor + 6$$

   Impossible d'obtenir la majoration demandée ; prendre par exemple la liste `[9; 8; 7; 6; 5; 4; 3; 2; 1]` *pour laquelle il y aura bien 6 éléments et non 4...*

   De même, le nombre d'éléments de `partitionG p x` est donc majoré par

$$5\left\lceil \frac{\lfloor n/5 \rfloor - 1}{2} \right\rceil + 2\left\lfloor \frac{\lfloor n/5 \rfloor + 1}{2} \right\rfloor + 4 \leqslant 7\left\lfloor \frac{n}{10} \right\rfloor + 6$$

On en déduit qu'il existe ℓ' tel que

$$\boxed{M'(n) \leqslant \ell' n + M'\left(\left\lfloor \frac{n}{5} \right\rfloor\right) + M'\left(7\left\lfloor \frac{n}{10} \right\rfloor + 6\right)}$$

$\left\lfloor \frac{n}{5} \right\rfloor + 7\left\lfloor \frac{n}{10} \right\rfloor + 4 \leqslant \frac{9n+40}{10} \leqslant \frac{19n}{20} < n$ pour $n \geqslant 80$.
Soit c_0' une constante telle que pour tout $n \in [\![0, 80]\!]$, $M'(n) \leqslant c_0' n$.
Soit $c' \in \mathbb{R}$ tel que $c' \geqslant c_0$.

- Pour tout $k \in [\![0, 80]\!]$, $M'(n) \leqslant c'n$.
- Soit $n \geqslant 81$, supposons que pour tout $k \in [\![0, n-1]\!]$, $M'(n) \leqslant c'n$.
  Alors $\left\lfloor \frac{n}{5} \right\rfloor \in [\![0, n-1]\!]$ et $7\left\lfloor \frac{n}{10} \right\rfloor + 4 \in [\![0, n-1]\!]$ donc

$$M'(n) \leqslant \ell' n + c'\left\lfloor \frac{n}{5} \right\rfloor + c'\left(7\left\lfloor \frac{n}{10} \right\rfloor + 4\right) \leqslant (\ell' + \frac{19c'}{20})n$$

  Donc en choisissant $c' \geqslant 20\ell'$, la propriété est héréditaire.

On en déduit alors que $\boxed{M'(n) \leqslant c'n}$

10. En regroupant les éléments par groupes de trois, on obtiendrait

$$M'(n) \leqslant \ell' n + M'\left(\left\lfloor \frac{n}{3} \right\rfloor\right) + M'\left(4\left\lfloor \frac{n}{6} \right\rfloor + 3\right)$$

On remarque tout d'abord qu'il n'est pas possible d'obtenir une majoration comme dans la question précédente, car si $n \in 6\mathbb{Z}$, $\left\lfloor \frac{n}{3} \right\rfloor + 4\left\lfloor \frac{n}{6} \right\rfloor + 3 = n + 3$.

Par ailleurs, on vérifie que le nombre m maximal d'éléments dans `partitionG p x` vérifie $m \geqslant \left\lfloor 4\frac{n}{6} \right\rfloor$.
En supposant M' croissante, on en déduit qu'il existe une constante ℓ'' telle que

$$M'(n) \geqslant \ell'' n + M'\left(\left\lfloor \frac{n}{3} \right\rfloor\right) + M'\left(4\left\lfloor \frac{n}{6} \right\rfloor\right)$$

On montre ensuite par récurrence forte sur k que pour tout $k \in \mathbb{N}$, pour tout $q \in \mathbb{N}^*$, $M'(q6^k) \geqslant \ell'' q(k+1)6^k$.

- Pour tout $q \in \mathbb{N}^*$, $M'(q) \geqslant \ell'' q$.
- Soit $k \in \mathbb{N}^*$, supposons que pour tout $j \in [\![0, k-1]\!]$, pour tout $q \in \mathbb{N}^*$, $M'(q6^j) \geqslant \ell'' q(j+1)6^j$.
  Alors pour tout $q \in \mathbb{N}^*$,

$$M'(q6^k) \geqslant \ell''(q6^k) + M'\left(2q6^{k-1}\right) + M'\left(4q6^{k-1}\right)$$
$$M'(q6^k) \geqslant \ell'' q6^k + \ell'' 2qk.6^{k-1} + \ell'' 4qk.6^{k-1}$$
$$M'(q6^k) \geqslant \ell'' q(k+1)6^k$$

On peut en déduire que $\frac{M'(6^k)}{6^k} \underset{k \to +\infty}{\longrightarrow} +\infty$.

$\boxed{\text{La complexité de cet algorithme n'est pas linéaire}}$.