Sauf indication contraire, il ne vous est pas demandé de prouver vos algorithmes. Néanmoins, il sera tenu compte de la qualité des explications accompagnant ceux-ci.

Problème 1 - Sous-somme maximale

Si s est une suite finie d'entiers relatifs $s = a_0, ..., a_{n-1}$ de longueur n, on appelle somme de s, notée $\sigma(s)$, la somme de tous les éléments de s :

$$\sigma(s) = \sum_{i=0}^{n-1} a_i$$

Si la suite est la suite vide (*i.e.* de longueur 0), alors on pose $\sigma(s) = 0$.

On appelle dans ce problème sous-suite de s toute suite constituée d'éléments *consécutifs* de s :

si $s = 1, 5, -3, -8, 2, 6, 7, 1$, alors $s' = 5, -3, -8, 2$ est une sous-suite alors que $s'' = 5, -8, 2$ n'en est pas une.

On dira qu'une sous-suite de s est cofinale si elle est de la forme $a_k, ..., a_{n-1}$ (où $k \in \{0, ..., n-1\}$) ou si elle est vide. De même, on dira qu'elle est co-initiale si elle est de la forme $a_0, ..., a_k$ (où $k \in \{0, ..., n-1\}$) ou si elle est vide.

L'objectif du problème est de construire un algorithme de paramètre une suite finie d'entiers relatifs par la méthode « diviser pour régner », qui détermine une sous-suite de somme maximale. En particulier, si la suite ne contient que des termes strictement négatifs, cet algorithme retourne la suite vide ayant pour somme 0.

Par exemple, si $s = -3, 5, -1, 4, -3, -6, 8$, alors une sous-suite de somme maximale est $s' = 5, -1, 4$. De même, si la suite est vide, alors la sous-suite de somme maximale est la suite vide de somme 0.

Partie 1 - Algorithme naïf

Donnez le principe (pas de pseudo-code) d'un algorithme élémentaire qui résout ce problème. Donnez l'ordre de grandeur de sa complexité en fonction de la longueur n de la suite.

Partie 2 - Principe de l'algorithme évolué

Soit s une suite d'entiers relatifs de longueur n. On la partage en deux sous-suites s', s'' de longueurs à peu prés égales à $n/2$.

Soit s_m une sous-suite de s de somme maximale. Trois cas se présentent :
- s_m est une sous-suite de s' ;
- s_m est une sous-suite de s'' ;
- s_m est « à cheval » sur s' et s'' : le premier élément de s_m appartient à s' et son dernier élément appartient à s''.

Les deux premiers cas sont accessibles par appel récursif de l'algorithme. On suppose donc qu'on est dans le dernier cas.

Question 1) Montrez que la sous-suite $s' \cap s_m$ (définition évidente : la sous-suite de s_m qui est une sous-suite de s') est une sous-suite de s', de somme maximale parmi toutes les sous-suites cofinales de s'.

Que peut-on dire de $s'' \cap s_m$?

Question 2) Donnez le principe d'un algorithme récursif qui utilise la méthode « diviser pour régner » et qui répond au problème posé : vous indiquerez précisément les trois étapes *division, règne, fusion* et les cas de base.

Partie 3 - Représentation informatique

On représente une suite d'entiers relatifs par une liste de type `int list`. La fonction à construire calculera un couple (ℓ', x) tel que ℓ' soit une sous-suite de ℓ de somme maximale et x sa somme.

Question 1) Écrivez une fonction `scinder l` de type `'a list -> 'a list * 'a list`, qui partage une liste en deux sous-liste de longueurs à peu prés égales. Les sous-listes sont vraiment des sous-listes : il faut garder les éléments de la liste initiale dans le même ordre !

Précisez sa complexité.

Question 2) Écrivez une fonction `pgsi l` (« plus grande somme initiale ») de type `int list -> int list * int`, qui donne un couple `l', x` tel que `l'` soit une sous-suite co-initiale de sous-maximale et `x` sa somme (remarque : dans le cas particulier d'une liste d'entiers tous strictement négatifs, elle rendra le couple `[],0`).

Précisez sa complexité.

Question 3) Écrivez une fonction `pgsf l` (« plus grand somme finale ») de type `int list -> int list * int`, qui donne un couple `l'', y` tel que `l''` soit une sous-somme cofinale de somme maximale et `y` sa somme (même remarque).

Précisez sa complexité.

Question 4) Concluez en écrivant la fonction récursive `pgs l`, de type `int list -> int list * int`, qui répond au problème.

Question 5) Justifiez rapidement sa terminaison et donnez sa complexité.

Devoir non surveillé 3 - Corrigé

Problème 1

Partie 1

Pour chaque élément a_i de la suite, on calcule toutes les sommes des sous-suites qui commencent par a_i. Comme il y a n éléments, on a au total $\frac{n(n-1)}{2}$ itérations à effectuer pour calculer toutes les sommes. À chaque itération, on compare la somme calculée avec une mémoire et on retient la meilleure des deux. Donc on obtient un algorithme de complexité quadratique.

Partie 2

Question 1) On note $s' = a_0, ..., a_k$ où $k = n/2$ par exemple (notation de CAML pour la partie entière de $\frac{n}{2}$). On suppose donc que $s_m = a_i, ...a_j$ où i, j sont deux indices tels que $i < n/2 < j$. Donc $s' \cap s_m = a_i, ..., a_k$.

Si cette sous-suite n'est pas de somme maximale parmi les sous-suites cofinales de s', alors il existe un autre indice i' tel que la somme $a_{i'} + ... + a_k$ soit strictement supérieure à $a_i + ... + a_k$.

Soit alors $s_M = a_{i'}, ..., a_j$. On a donc

$\sigma(s_M) = a_{i'} + ... + a_j = (a_{i'} + ... + a_k) + (a_{k+1} + ... + a_j) > (a_i + ... + a_k) + (a_{k+1} + ... + a_j) = \sigma(s_m)$.

Ceci contredit la définition de s_m, puisqu'on a trouvé une sous-suite de somme strictement plus grande que s_m.

Donc $s' \cap s_m$ est de somme maximale parmi les sous-suites cofinales à s'.

De même, $s'' \cap s_m$ est de somme maximale parmi les sous-suites co-initiales à s''.

Question 2) Cas de bases : si la suite est de longueur 0, alors la sous-suite de somme maximale est la suite vide ; si la suite est de longueur 1, alors soit elle n'a qu'un élément strictement négatif, auquel cas la sous-somme maximale est nulle, soit elle n'a qu'un élément positif, auquel cas la sous-somme maximale est cet élément.

Sinon :

- *division* : on coupe la suite s de longueur n en deux sous-suites s', s'' de longueur $n/2$ et $(n+1)/2$
- *règne* : on détermine récursivement les sous-suites de somme maximale de s' et s'', notées s'_m et s''_m
- *fusion* : on détermine la sous-suite cofinale de s' de somme maximale, notée s'_0 et la sous-suite de s'' de somme maximale, notée s''_0 ; on concatène ces deux suites en une suite s''', de somme $\sigma(s'_0) + \sigma(s''_0)$; alors la sous-suite de s de somme maximale est soit s'_m, soit s''_m, soit s'''.

Partie 3

Question 1)

```
let scinder l =
  let n= list_length l in
  let rec scin (p,q) k =
    if k = n / 2 then (p,q)
    else
      let a = hd q in
      scin (a :: p, tl q) (k+1)
  in
  let (p,q) = scin ([],l) 0 in
  (rev p,q);;

scinder [1;2;3;4;5;6;7;8;9;10];;
```

La fonction interne a une complexité qui dépend de k : $C(k) = C(k+1) + O(1)$ (à chaque appel récursif, il y a quelques opérations élémentaires — une comparaison, un appel à la fonction hd et tl, un appel au constructeur :: et une addition). Donc $C(k) = O(n/2 - k)$, donc le coût de l'appel à la fonction scinder est $C(0) + O(n/2)$ (appel à la fonction rev), ce qui donne un coût global en $O(n)$.

Question 2)

```
let rec pgsi l =
  match l with
  | [] -> [],0
  | a :: q -> let (l',x) = pgsi q in
    if a + x >=0 then (a :: l', a + x)
    else ([], 0);;

pgsi [-1;-2;-3];;
```

Là encore, de manière évidente, la complexité est linéaire (un parcours de la liste et des opérations élémentaires à chaque appel récursif).

Question 3)

```
let pgsf l =
 let k = rev l in
 let (k', x) = pgsi k in
 (rev k', x);;

pgsf [2;-10;3;5;-1];;
```

Pareil : complexité linéaire (la fonction rev est de complexité linéaire).

Question 4)

```
let rec pgs l =
  match l with
  | [] -> [], 0
  | [a] -> if a > 0 then (l, a) else ([], 0)
  | _ -> (* cas general *)
 (* division *)
    let gauche, droite = scinder l in
 (* regne *)
    let (g, x) = pgs gauche and (h, y) = pgs droite in
 (* fusion *)
    (* calcul des sous-sommes maximales co-... dans les deux sous-tableaux *)
     (* plus grande somme finale de la partie gauche du tableau *)
    let (g1, x1) = pgsf gauche in
     (* plus grande somme initiale de la partie droite du tableau *)
    let (h1, y1) = pgsi droite in
         (* calcul de la somme de la sous-suite a cheval *)
        let z = x1 + y1 in
      (* choix de la plus grande sous-somme *)
    let pgs_retenue = max z (max x y) in
    if pgs_retenue = z then
      (g1 @ h1), z
    else if pgs_retenue = x then
      g, x
    else h, y;;

pgs [-3;5;-1;4;-3;-6;8];;
```

Question 5) En dehors des cas de base, la longueur des listes des appels récursifs est strictement inférieure à celle de la liste initiale, donc comme $\mathbb{N}$ est un ensemble bien fondé, on en déduit que l'algorithme termine.

L'étape de division est de complexité linéaire.

L'étape de règne est constituée de deux appels récursifs.

L'étape de fusion est de complexité linéaire (deux parcours de deux listes et à chaque appel récursif, quelques sommes et comparaisons, soit $O(n)$ opérations auxquelles on ajoute les quelques dernières comparaisons).

D'après le th. du cours, on a donc une complexité $C(n)$ qui vérifie la relation $C(n) = qC(n/2) + O(n^p)$ où $q = 2 = 2^\alpha$, $\alpha = 1$ et $p = 1$, donc comme $p = \alpha$, on a $C(n) = O(n^\alpha \log_2 n) = O(n \log_2 n)$.