Question 3

Question 3.3

(a) Montrer qu'on peut se ramener à placer les s_i dans l'ordre croissant, c'est-à-dire tous ceux de la première colonne sont inférieurs à tous ceux de la deuxième, tous ceux de la deuxième sont inférieurs à tous ceux de la troisième et ainsi de suite.

Considérons un partitionnement P de somme S (la fonction à minimiser). Cette somme ne change pas si nous réarrangeons les colonnes de manière différente, par exemple par nombre décroissant de rectangles. Donc si S est minimale nous pouvons supposer que pour le partitionnement P, la première colonne a plus de rectangles que la deuxième, qui en a plus que la troisième... Suposons maintenant que les rectangles ne soient pas dans l'ordre croissant, c'est-à-dire qu'il existe un rectangle s dans la colonne i et un rectangle s' dans la colonne j avec $s > s'$ et $i < j$. Soit r_i (resp. r_j) le nombre de rectangles de la colonne i (resp. j). Nous avons donc $r_i \geq r_j$. En inversant s et s' nous obtenons un nouveau partitionnement de somme $S' = S - r_i(s-s') + r_j(s-s')$. Si $r_i = r_j$ alors $S = S'$ et si $r_i > r_j$ alors $S' > S$, ce qui contredit la minimalité de S.

La propriété est bien démontrée.

(b) On commence par trier les s_i par ordre croissant.

On crée un (demi) tableau T dont la case d'abscisse c et d'ordonnée $k \geq c$ contient $T[c,k] = (S,r)$ où S est la somme minimale des demi-périmètres d'un partitionnement par colonne d'un rectangle de hauteur 1 et de largeur $\Sigma_{1 \leq i \leq k} s_i$ par s_1, ..., s_k et r le nombre de rectangles de la dernière colonne. On remplit le tableau de la manière suivante :

Si $c = 1$ renvoyer $T[1, k] = (1 + k\Sigma_{1 \leq i \leq k} s_i, k)$.
Pour c allant de 2 à n
    Renvoyer $T[c, c] = (k + \Sigma_{1 \leq i \leq k} s_i, 1)$.
    Pour k allant de $(c + 1)$ à n
        Pour i allant de 1 à $\inf(k - 1, k - c + 1)$
            lire S dans $T[c - 1, k - i]$
            et calculer $S_i = S + 1 + i(s_k + ... + s_{k-i+1})$
            (périmètre minimal si la dernière colonne
            contient i rectangles).
        Choisir i_0 tels que S_{i_0} soit minimal parmi les S_i
            et renvoyer $T[c, k] = (S_{i_0}, i_0)$.
    Fin Pour
Fin Pour

Pour calculer le plus petit S pour s_1, ..., s_n il faut donc regarder le contenu des cases $T[1, n]...T[n, n]$ et choisir le nombre de colonnes c tel que S soit minimal dans $T[c, n]$. Pour retrouver le partitionnement correspondant il suffit alors de lire le nombre de rectangles r de la dernière colonne dans $T[c, n]$, puis le nombre de rectangles de l'avant dernière colonne dans $T[c - 1, n - r]$ et ainsi de suite.

Pour montrer que le résultat obtenu est correct il suffit de remarquer que si S est minimal pour le partitionnement de $s_1, ..., s_n$ en c colonnes avec r rectangles dans la dernière colonne alors en enlevant la dernière colonne on obtient un partitionnement optimal de $s_1, ..., s_{n-k}$ en $c - 1$ colonnes.

Question 4

Question 4.1

Le coût est linéaire. On montre la correction par récurrence sur $p \leq n$: quand on a fini la boucle $i = p$ dans l'algorithme alors thisSum contient la valeur de la plus grande coupe de $A[1...p]$ finissant en $A[p]$ et maxSum contient la valeur de la plus grande coupe de $A[1...p]$.

C'est vrai pour $p = 1$. Supposons que ce soit vrai pour un entier p et soit C la valeur de la plus grande coupe de $A[1...p]$ et C' la valeur de la plus grande coupe de $A[1...p]$ finissant en $A[p]$. Alors la valeur de la plus grande coupe finissant en $A[p + 1]$ est $S = \max(C' + A[p + 1], 0)$ et la valeur de la plus grande coupe de $A[1...p + 1]$ est $\max(S, C)$. C'est bien ce que calcule l'algorithme et la propriété est donc vérifiée pour tout p.

Question 4.2

Considérons le tableau $T[0,...,n]$ où $T[0]=0$ et pour tout entier i supérieur ou égal à 1 on a $T[i]=A[1]+...+A[i]$. La somme des éléments d'une coupe est de la forme $T[m]-T[l]$ avec $m \geq l$. On cherche la plus grande valeur négative, donc le plus petit écart négatif entre deux éléments du tableau. Si la plus grande coupe négative est $T[m]-T[l]$ avec $m \geq l$, alors il n'y a pas d'autre élément de T strictement entre $T[m]$ et $T[l]$. Montrons le. Si $m=l$ c'est évident. S'il existe p tel que $T[p] \in]T[m];T[l][$ alors soit $p>l$ et $T[m]-T[l]<T[p]-T[l]<0$, soit $p<m$ et $T[m]-T[l]<T[m]-T[p]<0$. Dans les deux cas cela contredit la maximalité de $T[m]-T[l]$.

L'algorithme est donc le suivant : on pose $T[0]=0$, $T[1]=A[1]$ et pour tout entier i supérieur ou égal à 1 on pose $T[i+1]=T[i]+A[i+1]$. On crée donc le T en temps linéaire. Puis on trie le tableau T par ordre croissant avec un tri en $n \log n$ en gardant trace de l'indice d'origine dans T. On calcule ensuite en temps linéaire l'ensemble des différences de deux éléments consécutifs du tableau trié quand leurs indices d'origine sont en ordre croissant et en ne gardant que les négatifs. On cherche leur maximum en temps linéaire : c'est la somme cherchée. Cet algorithme est finalement en $n \log n$.

Mais c'est aussi une borne inférieure ! En effet savoir si dans un tableau (contenant des éléments d'un moule commutatif ordonné) il existe deux éléments égaux est de complexité au moins $n \log n$ (cf. article bidule). Or il existe une réduction en temps linéaire de ce problème au nôtre, c'est à dire que si on connait un algorithme en temps $f(n)$ pour notre problème alors on en connait un en temps $\max(\mathcal{O}(n), f(n))$ pour l'existence de deux éléments égaux. Montrons le.

Soit $T[1,...,n]$ un tableau. On veut savoir s'il existe deux éléments identiques dans ce tableau. On considère ce tableau comme un tableau des sommes partielles et on construit en temps linéaire le tableau associé $A[1]=T[1]$ et pour si $1 \leq i \leq n$ alors $A[i+1]=T[i+1]-T[i]$. On applique alors l'algorithme de recherche de la plus grande coupe négative à A ainsi qu'à $-A$. En prenant la plus grande des deux, c'est-à-dire la plus petite en valeur absolue, on obtient la plus petite valeur absolue non nulle ε de la différence entre deux éléments du tableau T auquel on a ajouté l'élément 0. Le problème est qu'on s'interdit les coupes nulles, donc on trouve les éléments les plus proches mais on ne détecte pas les éléments égaux ! On va donc perturber un peu le système T et construire le tableau T' de la manière suivante. On commence par ajouter (en temps linéaire) 2ε aux éléments positifs de T et -2ε aux éléments négatifs (ceci pour éviter la comparaison d'un élément de T avec 0, ou autrement dit une coupe de A commençant par l'indice 1). Puis pour tout entier i entre 1 et n on ajoute $\varepsilon/2^i$ à l'élément d'indice i (encore une opération qui peut se faire en temps linéaire). Le tableau T' a alors la propriété suivante : tous ses éléments sont de valeur absolue strictement plus grande que ε et $|T[i] - T[j]| = 0$ si et seulement si $|T'[i] - T'[j]| < \varepsilon/2$. On construit alors (comme précédemment A pour T) le tableau A' dont T' est le tableau des sommes partielles et on lui applique l'algorithme de recherche de la coupe de plus petite valeur absolue c. Alors $c \leq \varepsilon/2$ si et seulement s'il y a deux éléments identiques dans T.

Question 4.3

Le calcul se fait en temps linéaire. Soit P la valeur maximale du produit d'une coupe. On commence par repérer les éléments nuls de A. S'il n'y en a pas alors on regarde les éléments négatifs. S'il y en a un nombre pair alors M est le produit de tous les éléments du tableau. S'il y en a un nombre impair alors soit i l'indice du premier et j l'indice du dernier : M est le maximum du produit des coupes $A[1..j[$ et $A[i+1..n+1[$. S'il y a des éléments nuls dans A alors on découpe le tableau en sous-tableaux maximaux ne contenant pas d'élément nul et auxquels on applique récursivement l'algorithme.