Correction du DSI

Correction.
1) La complexité algébrique de l'algorithme de Horner est $O(n)$. On en déduit l'algorithme naïf suivant : pour i entre 0 et $n-1$, on évalue P en u_i via l'algorithme de Horner, et on renvoie $(P(u_0), \cdots, P(u_{n-1}))$. On effectue donc une boucle de n étapes, et à l'étape i on effectue un calcul en $O(n)$ par Horner pour le calcul de $P(u_i)$, ce qui donne une complexité algébrique totale $O(n^2)$, c'est-à-dire quadratique.
2) On a vu en cours que la transformée de Fourier rapide (FFT) permettait de calculer les $(P(u_0), \cdots, P(u_{n-1}))$ avec une complexité $O(n \log n)$ lorsque $u_i = \omega^i$, pour i une racine primitive n-ième de l'unité, puisqu'on est dans les hypothèses où n est une puissance de 2, et que le corps dans lequel on travaille est de caractéristique différente de 2, et contient les racines primitives n-ièmes de l'unité.
3) Pour $k = 3$:
- pour $i = 0$, on a $M_{i,j} = (X - u_j)$ pour j entre 0 et 7 ;
- pour $i = 1$, on a $M_{i,j} = (X - u_{2j})(X - u_{2j+1})$, pour j entre 0 et 3 ;
- pour $i = 2$, on a $M_{i,j} = (X - u_{4j})(X - u_{4j+1})(X - u_{4j+2})(X - u_{4j+3})$, pour j entre 0 et 1.

4) On calcule $M_{0,j} = \prod_{l=0}^{0}(X - u_{j+l}) = X - u_j$ car $2^0 = 1$. Cela montre la propriété (1).

Pour la propriété (2), on remarque que $M_{i,j}$ est défini comme un produit de 2^i polynômes qui sont tous de degré 1, et donc $M_{i,j}$ est de degré 2^i, le nombre de polynômes de degré 1 intervenants dans le produit.

Il reste à montrer la propriété (3) : on a

$$M_{i+1,j} = \prod_{l=0}^{2^{i+1}-1}(X - u_{2^{i+1}j+l}) = \prod_{l=0}^{2^i-1}(X - u_{2^{i+1}j+l}) \prod_{l=2^i}^{2^{i+1}-1}(X - u_{2^{i+1}j+l})$$

ce qui donne, en écrivant que $2^{i+1}j = (2^i)(2j)$ et en effectuant le changement de variable $k = l - 2^i$ dans le second produit :

$$M_{i+1,j} = \prod_{l=0}^{2^i-1}(X - u_{2^i(2j)+l}) \prod_{k=0}^{2^i-1}(X - u_{2^i(2j+1)+k}),$$

soit $M_{i+1,j} = M_{i,2j}M_{i,2j+1}$.
5) L'algorithme proposé est une conséquence directe des propriétés (1) et (3) de la question précédente : la première boucle (étape numérotée 1. de l'algorithme) calcule les $M_{0,j}$, pour j allant de 0 à $2^k - 1$, par la propriété (1) de la question 4. Puis, comme on sait par (3) que $M_{i+1,j} = M_{i,2j}M_{i,2j+1}$ pour i entre 0 et $k-2$ et j entre 0 et $2^{k-i-1} - 1$, on peut calculer les $M_{i+1,j}$ en connaissant tous les $M_{i,j}$, ce que fait la 2-ième étape de l'algorithme : à la fin de la i-ième étape de la boucle

sur i de cette 2-ième étape, on aura calculé les $M_{i+1,j}$. Finalement, l'algorithme sort la liste, pour i allant de 0 à $k-1$, des listes des $M_{i,j}$, pour j entre 0 et $2^{k-i}-1$.

Il reste à calculer la complexité de cet algorithme. L'étape 1 de l'algorithme est une boucle de n étapes qui construit les $M_{0,j}$, ce qui nécessite donc n opérations. La deuxième étape est une boucle en $k-1$ étapes, où $k=\log_2(n)$, où à la i-ième étape on effectue 2^{k-i-1} produits de deux polynômes de degrés 2^i. Chaque produit se fait donc en $O(i*2^i)$ opérations, et on effectue donc $O(2^{k-i-1}*i*2^i)=O(i*2^{k-1})$ opérations à la i-ième étape. Le coût total de l'étape 2 de l'algorithme est donc $O(k^2 2^{k-1})=O(n(\log n)^2)$.

Au final, on effectue $n+O(n(\log n)^2)=O(n(\log n)^2)$ opérations, ce qui donne une complexité algébrique totale $O(n(\log n)^2)$.

6

```
def PolyMij(n,u):
    M0j=[x-u[j] for j in range(n)]
    M=[M0j]
    k=valuation(n,2)
    for i in range(k-1):
        Mi=[(M[i][2*j])*(M[i][2*j+1]) for j in range(2^(k-i-1))]
        M=M+[Mi]
    return M
```

7) On sait que $M_{k-1,0}=\prod_{l=0}^{2^{k-1}-1}(X-u_l)$, de sorte que $M_{k-1,0}(u_l)=0$ pour l entre 0 et $2^{k-1}-1=n/2-1$. De même, $M_{k-1,1}=\prod_{l=0}^{2^{k-1}-1}(X-u_{2^{k-1}+l})$ et donc $M_{k-1,1}(u_l)=0$ pour l entre $2^{k-1}=n/2$ et $2^k-1=n-1$.

En effectuant la division euclidienne de P par $M_{k-1,0}$, on trouve que $P=Q_0M_{k-1,0}+P_0$, et en évaluant en u_l, l entre 0 et $n/2-1$, on trouve que $P(u_l)=P_0(u_l)$ comme u_l est racine de $M_{k-1,0}$. Le même argument montre que $P(u_l)=P_1(u_l)$ pour u_l entre $n/2$ et $n-1$, en effectuant cette fois la division euclidienne de P par $M_{k-1,1}$.

8) La question précédente nous montre que la liste $(P(u_0),\cdots,P(u_{n-1}))$ coïncide avec la concaténation des listes $(P_0(u_0),\cdots,P_0(u_{n/2-1}))$ et $(P_1(u_{n/2}),\cdots,P_1(u_n))$. On en déduit donc l'algorithme récursif suivant, qui consiste à réappliquer cette stratégie aux listes $(P_0(u_0),\cdots,P_0(u_{n/2-1}))$ et $(P_1(u_{n/2}),\cdots,P_1(u_n))$:

```
def Multieval(n,P,M):
    if n==1:
        return [P]
    k=valuation(n,2)
    m=n//2
    P0= P % (M[k-1][0])
    P1= P % (M[k-1][1])
    M0=[]
    M1=[]
    for i in range(0,k-1):
        M0i=[[M[i][j] for j in range(2^(k-i-1))]]
```

```
        M1i=[[M[i][j] for j in range(2^(k-i-1),2^(k-i))]]
        M0=M0+M0i
        M1=M1+M1i
    return Multieval(m,P0,M0)+Multieval(m,P1,M1)
```

On fera bien attention qu'il faut modifier l'ensemble des M_{ij} : on fait l'appel récursif Multieval$(n/2, P_0, M_0)$ avec M_0 qui est la liste des M_{ij} correspondants seulement aux u_l pour l entre 0 et $n/2-1$ (et de même, l'appel récursif Multieval$(n/2, P_1, M_1)$ se fait avec la liste M_1 des M_{ij} qui correspondent aux u_l pour l entre $n/2$ et $n-1$), c'est-à-dire que M_0 est construite à partir de M en enlevant la dernière liste (qui est $(M_{k-1,0}, M_{k-1,1})$) de M, et en ne gardant dans chaque autre liste de M que la première moitié des termes. De même, M_1 est construite à partir de M en enlevant la dernière liste, et en ne gardant dans chaque autre liste de M que la deuxième moitié des termes.

9) Calculons la complexité de notre fonction Multieval en fonction de n. Cette fonction effectue deux appels récursifs Multieval$(n/2, P_0, M_0)$ et Multieval$(n/2, P_1, M_1)$, avec les degrés de P_0 et de P_1 qui sont strictement inférieurs à $n/2$, c'est-à-dire effectue deux appels récursifs sur des instances de Multieval de taille $n/2$. Il reste à calculer la complexité des calculs de P_0, P_1, M_0 et M_1. Le calcul de P_0 et P_1 est une division euclidienne de P par $M_{k-1,0}$ et $M_{k-1,1}$ respectivement, se qui se fait donc avec une complexité $O(n \log n)$. La création de M_0 et M_1 se fait en parcourant les $M_{i,j}$, et donc également en $n \log n$ (on vérifie directement que le nombre des $M_{i,j}$ est $k2^k = \log_2(n) * n$).

On en déduit que la complexité $T(n)$ de Multieval est égale à $2T(n/2) + O(n \log n)$, le terme $2T(n/2)$ à cause des deux appels récursifs sur des instances de taille $n/2$, et le terme $O(n \log n)$ correspondant au nombre d'opérations nécessaires pour créer ces instances sur lesquels on fait les appels récursifs.

10) Notons $f(n)$ pour le terme $O(n \log n)$ dans l'expression $T(n) = 2T(n/2) + O(n \log n)$. On sait qu'il existe une constance C, indépendante de n, telle que pour tout $n \geqslant 1$, $f(n) \leqslant C \cdot n \log n$. On écrit $n = 2^k$. Par récurrence, on a

$$T(n) \leqslant f(n) + 2f(n/2) + 2^2 f(n/2^2) + \cdots + 2^{k-1} f(2) + 2^k T(1).$$

On en déduit que

$$T(n) \leqslant C \sum_{i=0}^{k-1} (2^i (n/2^i) \log(n/2^i)) + 2^k T(1)$$

soit $T(n) \leqslant C \sum_{i=0}^{k-1} n \log(n/2^i)) + 2^k T(1)$. Comme on a $k = \log_2 n$ et que $\log(n/2^i) \leqslant \log n$, on en déduit que $T(n) \leqslant C \cdot n(\log n)^2 + nT(1)$, et donc que $T(n) = O(n(\log n)^2)$.

11) La syntaxe de l'algorithme en Sage est celle proposée ci-dessus.

12) L'application successive des algorithmes POLYMIJ et MULTIEVAL permet donc de répondre au problème initial d'évaluation simultanée en n éléments de K, pour un polynôme à coefficients dans K dont le degré est $< n$, et a une complexité $O(n(\log n)^2)$ (on effectue successivement deux algorithmes dont la complexité de

chacun est $O(n(\log n)^2)$), ce qui est bien un algorithme rapide. Mieux, si jamais on doit évaluer plusieurs polynômes de degrés inférieurs à n en les mêmes n points, on peut réutiliser le calcul des $M_{i,j}$ fourni par l'algorithme POLYMIJ.