Correction du DSI

Correction.
1)

(1) Pour $k \geqslant 1$ notons Q_k et S_k respectivement les valeurs de S et Q à la fin du k-ième tour de boucle. On note S_0 et Q_0 les valeurs initiales de S et Q. On va démontrer par récurrence que $Q_k = (X+u)^k$ et $S_k = \sum_{j=0}^{k} a_j (X+u)^j$.

Lorsqu'on initialise les valeurs de S et Q, cela correspond à Q_0 et S_0 et on a bien $Q_0 = 1 = (X+u)^0$ et $S_0 = a_0$.

Soit maintenant $k \geqslant 1$ tel qu'on ait $Q_{k-1} = (X+u)^{k-1}$ et $S = \sum_{j=0}^{k-1} a_j (X+u)^j$. Lors de l'exécution du k-ième tour de boucle, on obtient par définition de l'algorithme $Q_k = Q_{k-1} \cdot (X+u)$ et $S_k = S_{k-1} + a_k Q_k$. Il vient donc $Q_k = (X+u)^k$ puis $S_k = S_{k-1} + a_k (X+u)^k = \sum_{j=0}^{k} a_j (X+u)^j$.

La propriété est donc démontrée par récurrence. On sort de la boucle après l'exécution du $(n-1)$-ième tour de boucle, et on a donc à la sortie $S = S_{n-1} = \sum_{j=0}^{n-1} a_j (X+u)^j = P(X+u)$. NAIF renvoie donc ce qu'il faut.

(2) On conserve les notations de la question précédente. L'algorithme effectue $(n-1)$ tours de boucle, indicés par un entier $k \in \{1, \cdots, n-1\}$, et à l'étape k de la boucle on effectue les opérations suivantes :
— la multiplication de Q_{k-1}, polynôme de degré $k-1$ par $(X+u)$ qui est de degré 1. On sait que ce calcul s'effectue en $O(k)$ (car $(X+u)$ est de degré 1) ;
— le calcul de $a_k Q_k$, ce qui se fait en $O(k)$ car Q_k est de degré k ;
— l'addition de S_{k-1} et de $a_k Q_k$, ce qui se fait en $O(k)$ car c'est une addition de deux polynômes dont le degré est au plus k.

La complexité (algébrique) de l'algorithme est donc $C(n) = O(\sum_{k=1}^{n-1} k) = O(n^2)$ car $\sum_{k=1}^{n-1} k = \frac{n(n-1)}{2}$.

(3) On choisit ici de travailler dans un anneau de polynômes, par exemple à coefficients réels, en posant

```
Rx.<x>=PolynomialRing(RR).
```

On pouvait aussi faire le choix de travailler avec des listes et reprendre des fonctions de multiplication de polynômes écrites dans des TD précédents.

```
def NAIF(P,u):
    n=P.degree()
    Coeffs=P.list()
    S=Coeffs[0]
    Q=1
    for k in range(1,n+1):
        Q=Q*(x+u)
```

```
            S=(Coeffs[k]*Q)+S
        return S
```

2)

(1) On propose un algorithme récursif utilisant (E) et qui, prenant en entrée n (une puissance de 2), P (de degré $< n$) et u doit renvoyer $P(X+u)$ et $(X+u)^n$. En particulier, lorsque $n = 1$, l'algorithme doit renvoyer P (qui est simplement un coefficient constant car P est de degré < 1) et $X + u$. Pour $n \geqslant 2$, on va s'appuyer sur l'identité (E) : on écrit $P = P_0 + X^{n/2}P_1$ (où P_1 et P_0 sont respectivement le quotient et le reste de la division euclidienne de P par $X^{n/2}$. Si on sait calculer $Q(X+u)$ pour n'importe quel polynôme de degré $< n/2$, alors on peut calculer $P_0(X+u)$ et $P_1(X+u)$. De plus, si on connait $R = (X+u)^{n/2}$ alors $(X+u)^n = R^2$ et $P(X+u) = P_0(X+u) + R^2P_1(X+u)$. Autrement dit, l'algorithme récursif suivant permet effectivement de renvoyer $P(X+u)$ et $(X+u)^n$ (on l'écrit ici directement avec la syntaxe Sage, ce qui répond au passage à la dernière question du sujet) :

```
def SMART(n,P,u):
    if n==1:
        return [P,(x+u)]
    m=n//2
    P0 = P % (x^m)
    P1 = (P-P0)//(x^m)
    Q0=SMART(m,P0,u)
    Q1=SMART(m,P1,u)
    return [Q0[0]+Q0[1]*Q1[0],Q0[1]^2]
```

(2) Notons $T(n)$ la complexité de notre algorithme. Pour calculer $P(X+u)$ et $(X+u)^n$, l'algorithme effectue deux appels récursifs de taille $n/2$, l'un sur P_0 et l'autre sur P_1, où $P = P_0 + X^{n/2}P_1$. Chacun de ces appels a donc une complexité $T(n/2)$. En plus de ces appels, l'algorithme effectue les calculs suivants :

— Le calcul de P_0 et P_1, qui s'effectue en $O(n)$;
— les deux produits $(X+u)^{n/2} \cdot (X+u)^{n/2}$ et $(X+u)^{n/2} \cdot P_1(X+u)$ qui s'effectuent chacun avec une complexité $M(n/2)$ par définition de la fonction M ;
— la somme $P_0(X+u) + (X+u)^n P_1(X+u)$, où toutes les quantités ont été calculées précédemment.

En rassemblant ces quantités, on obtient bien la formule donnée par la question, à savoir $T(n) = 2T(n/2) + 2M(n/2) + O(n)$.

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

$$T(n) \leqslant 2T(n/2) + 2M(n/2) + Cn.$$

On en déduit que

$$T(n/2) \leqslant 2T(n/4) + 2M(n/4) + C(n/2),$$

et plus généralement, que

$$T(n/2^j) \leqslant 2T(n/2^j) + 2M(n/2^j) + C(n/2^j).$$

En réinjectant ces identités successivement et en sommant, on obtient

$$T(n) \leqslant 2^k T(1) + 2\sum_{j=1}^{k} 2^j M(n/2^j) + C\sum_{j=0}^{k} 2^j (n/2^j).$$

De l'identité $2M(n/2) \leqslant M(n)$ on déduit que pour tout $j \leqslant k$, $M(n/2^j) \leqslant 2^{-j} M(n)$. En utilisant ce résultat et le fait que $n = 2^k$, on obtient :

$$T(n) \leqslant 2^k T(1) + kM(n) + C(k+1)n$$

soit $T(n) = O(\log(n)(M(n) + n))$.