Le but de ce TD est d'étudier un algorithme évolué de multiplications des polynômes.

I Version naturelle

Nous reprenons le travail effectué en PTSI sur les polynômes.

1.1 Encodage

Un polynôme $P(X) = a_0 + a_1X + \cdots + a_nX^n$ est encodé en python par la liste $[a_0, \ldots, a_n]$ de longueur n + 1.

Exercice 1
On notera deg(P) le degré d'un polynôme non nul et $l(P)$ la longueur de sa liste en python.
Pour P, Q deux polynômes, donner un lien entre $l(P)$ et deg(P) puis exprimer $l(P \times Q)$ en fonction de $l(P)$ et $l(Q)$.
Quel est l'indice dans la liste de P du coefficient de X^i ?

1.2 Opérations sur les polynômes

Exercice 2
Lire le code présent dans le fichier **polynomes.py** (la correction partielle du TD de PTSI) et compléter la fonction `soustrait`.

Exercice 3
Créer une fonction `multiplie(P, Q)` qui retourne la liste correspondant au polynôme produit.
On pourra développer à la main un produit de polynômes pour trouver l'algorithme et/ou trouver une expression théorique sous forme de somme double.

Exercice 4
Exprimer le nombre d'additions et le nombre de multiplications entre entiers que l'algorithme précédent effectue en fonction de $l(P)$ et $l(Q)$.
Même question pour l'algorithme d'addition présent dans la feuille python.

II Diviser pour régner

2.1 Principe de l'algorithme

On remarque grâce à l'exercice précédent que l'addition est beaucoup moins coûteuse que la multiplication (et ceci vaut aussi pour les entiers).

On considère le problème suivant : multiplier deux polynômes A, B de longueur 2n (degré 2n - 1).
Ecrivons $A = Q_AX^n + R_A$ et $B = Q_BX^n + R_B$ les divisions euclidienne par X^n.
Les polynômes R_A, R_B sont représentés par la première moitié des listes pour A et B respectivement, et les quotients sont représentés par les secondes moitiés.
Il faut maintenant reconstruire le résultat de $A \times B$ en fonction de Q_A, R_A, Q_B, R_B. L'algorithme naïf revient à :

$$AB = R_AR_B + X^n(R_AQ_B + R_BQ_A) + X^{2n}Q_AQ_B$$

Ainsi multiplier deux polynômes de taille 2n revient à effectuer 4 multiplications sur des polynômes de taille n (et on a bien $(2n)^2 = 4 \times n^2$).

L'idée maîtresse de l'algorithme efficace est de remarquer que

$$R_AQ_B + R_BQ_A = (R_A + Q_A)(R_B + Q_B) - R_AR_B - Q_AQ_B$$

Ainsi si on note $P_1 = R_AR_B, P_2 = Q_AQ_B$ et $P_3 = (R_A + Q_A)(R_B + Q_B)$ on obtient

$$AB = P_1 + X^n(P_3 - P_1 - P_2) + X^{2n}P_2$$

Le tour de magie est ici : au prix de 2 soustractions et 1 addition supplémentaires, on peut effectuer seulement 3 multiplications. On utilisera évidemment l'algorithme efficace pour effectuer ces 3 multiplications, et nous voici en face d'un algorithme récursif.

Exercice 5
Quel est le cas d'arrêt de la récursion ? Que vaut alors le produit ?

2.2 Au travail !

Exercice 6
Implémenter l'algorithme précédent en deux fonctions :

- Une fonction récursive qui prend deux polynômes de même longueur et calcule leurs produit.
- Une fonction `mult` non récursive qui prend deux polynômes quelconques et utilise la fonction récursive pour calculer leurs produits.

Exercice 7
L'ordre de grandeur du nombre d'opérations pour multiplier deux polynômes de longueur n est $Kn^{\log_2(3)}$ (où K est une constante) et non plus n^2. Nous allons tenter de le vérifier.

1. Créer une fonction `poly_alea(n, coeff_max)` qui retourne un polynôme de longueur n à coefficients aléatoires dans $[\![0, coeff\_max]\!]$.
2. Dans une console IPython, la commande

```
%timeit mult(P, Q)
```

évalue le temps mis par une commande pour s'exécuter.

Vérifier que l'ordre de grandeur est bien celui annoncé. On pourra comparer (plusieurs fois) le rapport entre les temps d'exécution quand on double la longueur.

Exercice 8
Créer une fonction `timeit(func, n)` qui évalue le temps d'exécution de la fonction `func` pour multiplier deux polynômes de longueur n. On pourra utiliser

```
import time

t0 = time.perf_counter()
...
duree = time.perf_counter() - t0
```

Utiliser cette fonction pour tracer le temps d'exécution en fonction de la longueur n pour nos deux fonctions de multiplication.

Quel est lintérêt de tout ceci ? La multiplication des entiers (bien plus courante, avons le) est tout à fait similaire à celle des polynômes (étudier l'algorithme de primaire, il s'agit tout simplement de développer le produit...). Si nous arrivons à une implémentation meilleure de notre algorithme (voire, maintenant que la porte est ouverte, à trouver un algorithme encore meilleur), nous pourrons gagner en efficacité sur les multiplications entre entiers "grands".
La cryptographie (qui protège vos échange de données sur internet, par exemple) requiert très souvent des calculs (dont la multiplication) sur des entiers pouvant aller jusqu'à 3000 bits de longueur, et serait difficilement praticable sur téléphone portable sans ces "bons" algorithmes.

2.3 Autres applications du principe de diviser pour régner

Le principe général mis en oeuvre dans l'algorithme précédent est le suivant :

1. on "coupe" le problème en deux.
2. on résout les deux moitiés du problème
3. on reconstruit la solution du problème de départ.

D'autres algorithmes connus utilisent ce principe, on peut les implémenter de manière naturelle sous forme de fonctions récursives :

- recherche dans une liste triée.
- recherche d'un point d'annulation d'une fonction.
- calcul de a^n sous la forme $(a^{\frac{n}{2}})^2$ si n est pair et $a \times a^{n-1}$ sinon