21. Complexité en temps d'un algorithme

Exercice 1. On considère les algorithmes suivants :

Fonction: pow(x, n)
Début
  $l \leftarrow [\,]$
  Pour k **allant de** 1 **à** n **faire**
    $l \leftarrow l + [x^k]$
  FinPour
  Renvoyer l
Fin

Fonction: fastpow(x, n)
Début
  $l \leftarrow [\,]$
  $p \leftarrow 1$
  Pour k **allant de** 1 **à** n **faire**
    $p \leftarrow xp$
    $l \leftarrow l + [p]$
  FinPour
  Renvoyer l
Fin

1. Déterminer la sortie de chacun des algorithmes.
2. Déterminer le nombre de multiplications effectuées dans chacun des algorithmes.

Exercice 2. On considère la suite de Fibonacci définie par $\begin{cases} u_0 &= 1 \\ u_1 &= 1 \\ u_{n+2} &= u_{n+1} + u_n \ , \ n \in \mathbb{N} \end{cases}$.

1. Déterminer le nombre d'additions nécessaires pour calculer les n premiers termes de cette suite.
2. Écrire un algorithme permettant d'obtenir la liste des n premiers termes de la suite de Fibonacci.

Exercice 3.

On représente un vecteur de $\mathbb{R}^n$ par la liste U de ses coordonnées et une matrice de $\mathcal{M}_n(\mathbb{R})$ par la liste M de ses lignes.

1. Évaluer le nombre d'additions et de multiplications nécessaires pour calculer le produit MU.
2. Écrire un algorithme permettant de calculer le produit MU.

Exercice 4. On considère un polynôme de degré n, $P(X) = a_0 + a_1X + a_2X^2 + \cdots + a_nX^n$.

1. Montrer qu'il faut en principe pour x donné $\frac{n(n+1)}{2}$ multiplications pour calculer $P(x)$.
2. Montrer en remarquant que $P(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + a_nx)))$ qu'on peut calculer $P(x)$ avec n multiplications.
3. Écrire les algorithmes précédents permettant de calculer $P(x)$, le polynôme P étant représenté par la liste de ses coefficients.