

## EA4 – Éléments d'algorithmique

### TD n° 1

#### Exercice 1 : Des limites du progrès...

Il y a 25 ans, un ordinateur faisait dix millions d'opérations par seconde et implémentait un algorithme de tri demandant  $50 \times n \times \log_{10} n$  opérations pour un tableau d'entrée de taille  $n$ . On souhaite le comparer à un ordinateur 100 fois plus rapide, mais sur lequel tourne un (moins bon) algorithme de tri demandant  $n^2$  opérations. Quels sont les temps de calcul de chacun pour une entrée de taille  $n = 10^6$  ? et  $n = 10^7$  ?

#### Exercice 2 : Évaluation de polynômes

1. Proposer un algorithme qui, étant donné un polynôme  $P$  et une valeur  $x$ , calcule  $P(x)$ . On supposera que  $P$  est décrit par un tableau contenant, en case d'indice  $i$ , le coefficient du monôme de degré  $i$ .
2. Quelle est la complexité de cet algorithme si les opérations considérées comme élémentaires sont les additions et multiplications de réels ?
3. Montrer que l'algorithme suivant résout le même problème :

```
def horner(P, x) :
    res = 0
    for coeff in P[::-1] :      # parcours du tableau à l'envers
        res = res * x + coeff
    return res
```

Quelle est sa complexité ?

#### Exercice 3 : Produit de polynômes

1. Décrire un algorithme de calcul du produit de deux polynômes. Quelle est sa complexité ?

On se limite maintenant au cas des polynômes  $P$  dont le degré est de la forme  $2^k - 1$ , et pour tout tel polynôme  $P$ , on note  $P^{(0)}$  et  $P^{(1)}$  les polynômes de degré  $2^{k-1} - 1$  tels que :

$$P = P^{(0)} + P^{(1)} \cdot X^{2^{k-1}}.$$

2. Décrire un algorithme récursif permettant de calculer le produit de deux polynômes  $P$  et  $Q$  de même degré  $2^k - 1$  à l'aide des polynômes  $P^{(0)}$ ,  $P^{(1)}$ ,  $Q^{(0)}$  et  $Q^{(1)}$ . Quelle est sa complexité ?
3. Montrer que l'algorithme précédent peut être modifié pour faire seulement 3 appels récursifs au lieu de 4 (considérer les polynômes  $P^{(0)} + P^{(1)}$  et  $Q^{(0)} + Q^{(1)}$ ). Prouver la correction de ce nouvel algorithme (dû à Karatsuba). Quelle en est la complexité ?

**Exercice 4 : Nombres premiers**

1. Proposer un algorithme (le plus naïf possible) permettant de déterminer si un entier est premier. Quelle est sa complexité? Comment peut-on l'améliorer?

On considère maintenant l'algorithme suivant :

```
def eratosthene(n) :  
    tab = [False, False] + [True] * (n - 1)  
    for i in range(2, n + 1) :  
        if tab[i] :  
            k = 2 * i  
            while k <= n :  
                tab[k] = False  
                k = k + i  
    return tab
```

2. Que représente le tableau calculé par `eratosthene(n)` ? Justifier.
3. Quelle est la complexité de ce calcul ?