Informatique PCSI
Corrigé TP 10b : analyse d'un programme

Exercice 1

À chaque passage dans la boucle, nous avons deux affectations avec une addition et une division entière et le nombre n perd un chiffre dans son écriture décimale. Le nombre de passages dans la boucle est donc égal au nombre de chiffres dans l'écriture décimale de n, soit $\texttt{Ent}(\log_{10}(n)+1)$ où `Ent` désigne la partie entière. La complexité est en $\mathcal{O}(\log_{10} n)$, soit logarithmique.

Exercice 2

1. Il y a n passages dans la boucle (`k` prend toutes les valeurs de 1 à n), et à chaque passage nous comptons une multiplication et une affectation (`p = 2 * p`) ; le total est donc de $2n$ opérations et le niveau de complexité est en $\mathcal{O}(n)$.
2. Pour prouver la terminaison nous considérons la suite des valeurs de n. C'est une suite d'entiers positifs strictement décroissante qui atteint 0 après un nombre fini d'itérations (le nombre de chiffres dans l'écriture binaire de n puisque chaque division par 2 retire un chiffre à ce nombre).

   Montrons que l'expression $p \times b^n$ est un invariant de la boucle. Considérons l'expression $p \times b^n$ avant un passage dans la boucle avec $n = 2q + r$. Lors du passage dans la boucle, les nouvelles valeurs de p, b et n sont $p \times b^r$, b^2, et q. Or $p \times b^r \times (b^2)^q = p \times b^{2q+r} = p \times b^n$.

   Avant l'entrée dans la boucle, $pb^n = 2^n$. Donc en sortie de boucle, puisque la valeur de la variable `n` est 0, on obtient pour valeur de `p` : $p = 2^n$.

   À chaque passage dans la boucle, quatre affectations et cinq opérations sont effectuées. Ce nombre est constant et il y a k passages dans la boucle avec k égal à la partie entière de $(1 + \log_2 n)$. La complexité est donc logarithmique.

Exercice 3

1. La boucle interne est utilisée pour calculer les puissances de x et la boucle externe pour multiplier le résultat obtenu par le coefficient correspondant et l'ajouter à la valeur courante de $P(x)$ affectée à la variable `p`.

   La variable `i` prend n valeurs. Pour chaque valeur de `i`, nous comptons une affectation avant la boucle interne puis une multiplication, une addition et une affectation après cette boucle. Pour chaque valeur de `i`, la variable `j` prend `i` valeurs et il y a donc `i` passages dans cette boucle avec à chaque passage un nombre d'opérations constant. Le nombre total de passages dans la boucle interne est $1 + 2 + \ldots + n$, soit de l'ordre de n^2. Le niveau de complexité est donc en $\mathcal{O}(n^2)$.

2. Un programme correspondant à l'algorithme de Hörner est le suivant :

```
def horner(x, a):
    n = len(a)
    p = a[n-1]
    for i in range(1, n):
        p = p * x + a[n-1-i]
    return p
```

   La variable `i` prend $n-1$ valeurs. À chaque passage dans la boucle le nombre d'opérations effectuées est constant. Le niveau de complexité est donc en $\mathcal{O}(n)$.

   L'algorithme de Hörner apparaît clairement comme le plus efficace.