

## TD 3

## Exercices d'application

D'après IPT, Éditions Vuibert.

## Savoirs et compétences :

- Alg – C15 : Récursivité : avantages et inconvénients.

## Exercice 1

Soit l'algorithme suivant :

## ■ Python

```
def mult(n, p):
    if p == 0:
        return 0
    else :
        return n+mult(n,p-1)
```

**Question 1** Énoncer un variant de boucle et montrer la terminaison de l'algorithme.

**Question 2** Énoncer un invariant de boucle et montrer la correction de l'algorithme.

**Correction** • Soit  $\mathcal{P}$  la propriété d'invariance : à l'itération  $p$ , on a  $mult(n, p) = n \cdot p$ .

- À l'instant 0, on a : d'une part :  $\forall n, n \cdot 0 = 0$ . D'autre part,  $mult(n, 0)$  renvoie 0. La propriété de récurrence est vraie.
- À l'instant  $p$ , on considère la propriété de récurrence est vraie à l'instant  $p$  :  $mult(n, p) = n \cdot p$ .
- À l'instant  $p + 1$ , on applique l'algorithme :  $p$  étant différent de 0, l'algorithme retourne  $mult(n, p + 1) = n + mult(n, p)$ . D'après la propriété de récurrence, on a donc  $mult(n, p + 1) = n + n \cdot p = n(p + 1)$ . La propriété est donc vraie au rang  $p + 1$ .
- L'algorithme calcule donc le produit  $np$ .

**Question 3** Donner et justifier la complexité temporelle de la fonction `mult`.

**Correction** On note  $C(p)$  le nombre d'appels récursifs :  $C(p) = 1 + C(p - 1) = 1 + 1 + C(p - 2) = p + T(0)$ . On a donc  $C(p) = \mathcal{O}(p)$ . La complexité temporelle est linéaire.

**Question 4** Donner et justifier la complexité spatiale de la fonction `mult`.

**Correction** On stocke une valeur à chaque appel récursif. Si ce stockage est à coût constant, étant donné qu'il y a  $n$  appels récursifs, la complexité spatiale est en  $\mathcal{O}(n)$ .

## Exercice 2

Soit l'algorithme suivant :

### ■ Python

```
def puiss(x, n):
    if n == 0:
        return 1
    else :
        return x*puiss(x,n-1)
```

**Question 1** Énoncer un variant de boucle et montrer la terminaison de l'algorithme.

**Question 2** Énoncer un invariant de boucle et montrer la correction de l'algorithme.

**Correction**

- Soit  $\mathcal{P}$  la propriété d'invariance : à l'itération  $p$ , on a  $puiss(x, n) = x^n$ .
- À l'instant 0, on a : d'une part :  $\forall x > 0, x^0 = 1$ . D'autre part,  $puiss(x, 0)$  renvoie 1. La propriété de récurrence est vraie.
- À l'instant  $n$ , on considère la propriété de récurrence est vraie et à l'instant  $p : puiss(x, n) = x^n$ .
- À l'instant  $n + 1$ , on applique l'algorithme :  $n$  étant différent de 0, l'algorithme retourne  $puiss(x, n + 1) = x * mult(x, n)$ . D'après la propriété de récurrence, on a donc  $puiss(x, n + 1) = x * x^n = x^{n+1}$ . La propriété est donc vraie au rang  $n + 1$ .
- L'algorithme calcule donc le produit  $x^n$ .

**Question 3** Donner et justifier la complexité temporelle de la fonction `puiss`.

**Correction** On note  $C(p)$  le nombre d'appels récurrents :  $C(p) = 1 + C(p - 1) = 1 + 1 + C(p - 2) = p + T(0)$ . On a donc  $C(p) = \mathcal{O}(p)$ . La complexité temporelle est linéaire.

**Question 4** Donner et justifier la complexité spatiale de la fonction `puiss`.

**Correction** On stocke une valeur à chaque appel récursif. Si ce stockage est à coût constant, étant donné qu'il y a  $n$  appels récurrents, la complexité spatiale est en  $\mathcal{O}(n)$ .

## Exercice 3

Soit l'algorithme suivant :

### ■ Python

```
def rechercheDichoRec(x, l):
    n=len(l)
    if n == 0:
        return False
    elif x<l[n//2] :
        return rechercheDichoRec(x, l[0:n//2])
    elif :
        return rechercheDichoRec(x, l[n//2:n])
    else :
        return True
```

**Question 1** Donner et justifier la complexité temporelle de la fonction `rechercheDichoRec`.

**Correction** Définissons le coût temporel comme le nombre d'appel récursif. Dans le pire des cas, l'élément n'est pas dans la liste. On cherche  $p$  le nombre de fois que  $n$  est divisible par 2. On cherche donc  $p$  tel que  $n \left(\frac{1}{2}\right)^p > 1 \Leftrightarrow p \ln\left(\frac{1}{2}\right) > \ln\left(\frac{1}{n}\right) \Leftrightarrow p > \frac{\ln\left(\frac{1}{n}\right)}{\ln\left(\frac{1}{2}\right)} = \frac{-\ln(n)}{-\ln(2)} = \frac{\ln(n)}{\ln(2)}$ . La complexité est logarithmique.

**Question 2** Donner et justifier la complexité spatiale de la fonction `rechercheDichoRec`.

**Correction** On stocke une liste à chaque appel récursif. On note  $n$  le coût de stockage d'une liste de taille  $n$ . Soit une liste 1 de taille  $n = 2^k$  avec  $k \in \mathbb{N}$ . À chaque itération, on stocke une liste de taille  $n/2$ . On a donc  $C(n) = n + \frac{n}{2} + \frac{n}{4} + \dots + 1$ .

Par conséquent,  $C(n) = 2^k + \frac{2^k}{2} + \frac{2^k}{4} + \dots + 1 = 2^k + 2^{k-1} + 2^{k-2} + \dots + 1$ . S'agissant de la somme des termes d'une suite géométrique :  $C(n) = 1 + 2^1 + \dots + 2^k = \frac{1 - 2^{k+1}}{1 - 2} = 2n + 1$ . La complexité spatiale est donc linéaire.