Correction des exercices

24 février 2022

1 Exercice 1

```python
def fibonacci(n):
    i = 2
    a = [-1]*(n+1)
    a[0] = 1
    a[1] = 1
    while i <= n:
        a[i] = a[i-1] + a[i-2]
        i = i+1
    return a[-1]
```

FIGURE 1 – Réponse à la question 1, code en O(n) en temps et espace pour avoir le n-ième nombre de fibonacci

```python
def fibo_rec(n):
    if n==0 or n==1:
        return 1
    else:
        return fibo_rec(n-1) + fibo_rec(n-2)
```

FIGURE 2 – Réponse à la question 2

En étudiant la figure 2, on se rend compte que la complexité spatiale et temporelle est de l'ordre de $\mathcal{O}(\varphi^n)$ où φ est le nombre d'or, c'est un complexité exponentielle qui peut être ramenée à une complexité linéaire à l'aide de la programmation dynamique, un paradigme d'algorithmie, (hors programme d'IPT). L'étude de la complexité d'un programme récursif est aussi hors programme.

```python
def fibo(n):
    a = [1,1]
    i = 2
    while i <= n:
        a[i%2] = a[0] + a[1]
    return a[n%2]
```

Cette fonction fait le même travail que la première mais n'utilise qu'un tableau de taille 2 donc est en $\mathcal{O}(1)$ en espace, on peut faire ça car chaque F_i ne nécessite que F_{i-1} et F_{i-2} pour être calculé.

2 Exercice 2

— Le code A. est une fonction "classique" qui réimplémente $a * b$ en $\mathcal{O}(n)$ en temps et $\mathcal{O}(1)$ en espace

— Le code B. est une fonction "récursive" non terminale, c'est l'algorithme dit *d'exponentiation rapide* il est en $\mathcal{O}(\log_2(n))$ en temps et en espace (dans le pire cas)

— Le code C. est une fonction "récursive" terminale, c'est l'algorithme d'euclide elle calcule le pgcd de a et b il est en $\mathcal{O}(n)$ si on considère la division euclidienne constante en temps, et en $\mathcal{O}1$ en espace