


## EA4 – Éléments d’algorithmique

### TP n° 2 : autour de la suite de Fibonacci

**Modalités de rendu :** À chaque TP, vous devrez rendre les exercices marqués par un symbole . Le rendu de l’exercice  $K$  du TP  $N$  doit être inclus dans le fichier `tpN_exK.py`, à télécharger depuis la section « Énoncés de TP ». Vous devez remplir les zones marquées par le commentaire À COMPLÉTER dans ce fichier. Vous devez également remplir les zones marquées par COMMENTAIRES - question ... où les pointillés sont remplacés par le numéro de la question dans le sujet. Ces questions portent sur l’interprétation des courbes obtenues.

Ne modifiez pas les autres fonctions du fichier, sauf demande explicite de l’énoncé. Chaque fichier contient une fonction `main` qui teste les fonctions que vous devez programmer et qui vous affiche un score donné par le nombre de tests passés avec succès. Pour passer ces tests, vous devez exécuter le programme écrit.

En fin de TP, vous devez déposer les fichiers modifiés (maximum 3) sur Moodle. Vous avez ensuite quelques jours supplémentaires pour les compléter (date à voir avec votre chargé de TP).

**Version de Python :** attention, utilisez exclusivement une version 3.x.x de Python (pour connaître la version de la commande `python`, tapez dans le shell `python --version`).

**Dépendances :** Pour ce TP vous aurez besoin de la bibliothèque graphique `matplotlib`.<sup>1</sup>. N’oubliez pas également de télécharger le fichier `ea4lib.py`. En cas d’affichage étranges, changez les variables de configuration au début de ce fichier.

#### Exercice 1 : Fibonacci revisité


Dans le fichier `tp2_ex1.py`, vous trouverez le code des fonctions `fibonacci_1`, `fibonacci_2` et `fibonacci_3` vues en cours.

1.  Lancez une première fois l’exécution du fichier fourni.

Une première fenêtre graphique apparaît montrant les courbes des temps d’exécution des trois algorithmes en fonction de l’indice  $n$  du nombre d’éléments de la suite calculés avec en plus en noir pointillé, la courbe de la fonction témoin  $n \mapsto n \times \left(\frac{1+\sqrt{5}}{2}\right)^n$  renormalisée.

Fermez la première fenêtre. Une deuxième fenêtre apparaît montrant les courbes du temps d’exécution de `fibonacci_3` et de la fonction témoin  $n \mapsto n^2$  renormalisée.

Commentez les courbes obtenues.

2.  Modifier les fonctions `fibonacci_i_adds` pour  $i \in \llbracket 1, 3 \rrbracket$  pour qu’elles renvoient, comme second composant du résultat, le nombre d’additions d’entiers de la suite faites lors d’un appel à chacune d’entre elles.

---

1. Pour l’obtenir avec votre gestionnaire de paquets : `apt-get install python3-matplotlib` (remplacer `apt-get` par `yum`, `brew`, ... bref votre gestionnaire de paquets si jamais ce n’est pas `apt`). Si pb avec mac voir : <https://matplotlib.org/stable/users/installing/index.html> avec `python3` au lieu de `python`

Le fichier fourni effectue ensuite dans l'ordre :

- des tests de ces fonctions,
- l'affichage des courbes des nombres d'additions d'entiers obtenus, d'abord pour les trois algorithmes, puis seulement pour `fibonacci_2` et `fibonacci_3`.

Ces différentes mesures sur le nombre d'additions d'entiers reflètent-elles les temps d'exécution ?

3. En mode interactif, appeler la fonction `courbes_adds` avec différentes valeurs des paramètres.
4. ✎ Écrire la fonction `nbOfBits(i)` qui calcule le nombre de bits nécessaires pour coder la valeur de son paramètre entier.
5. ✎ Modifier les fonctions `fibonacci_i_bits` pour  $i \in \llbracket 1, 3 \rrbracket$  pour qu'elles renvoient, comme second composant du résultat, le nombre d'opérations élémentaires sur les bits faites lors d'un appel à chacune d'entre elles.

Rappel : une addition entre entiers dont le résultat est codé sur  $n$  bits nécessite  $n$  opérations sur les bits.

Les courbes obtenues reflètent-elles les conclusions du cours ? Elles s'affichent avec également une courbe témoin en trait noir pointillé.

## Exercice 2 : Fibonacci par produit de matrices

Pour la fonction `fibonacci_4` vue en cours, on utilise l'identité :

$$\forall n \geq 1, \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n.$$

1. ✎ Compléter la fonction `puissance_matrice_2_2` pour qu'elle calcule la puissance  $n$ -ième d'une matrice  $2 \times 2$  **en utilisant l'algorithme d'exponentiation binaire**. Vous pourrez utiliser la fonction `produit_matrice_2_2` définie dans le fichier `tp2_ex2.py` qui calcule le produit de deux matrices de dimension  $2 \times 2$ .
2. ✎ Lancez une première fois l'exécution du fichier `tp2_ex2.py`. Des fenêtres graphiques apparaissent montrant d'abord les courbes des temps d'exécution de `fibonacci_3` (de l'exercice 1) et `fibonacci_4` (vue en cours), puis uniquement `fibonacci_4` avec en plus en noir pointillé, la courbe de la fonction témoin  $n \mapsto n^{\log_2(3)}$  renormalisée.

Commentez les courbes obtenues.

3. ✎ Compléter les fonctions `puissance_matrice_2_2_ops`, `produit_matrice_2_2_ops` et `fibonacci_4_ops` pour compter les opérations arithmétiques sur des entiers (multiplication, addition, modulo, division) effectuées à chaque appel.

Que pensez-vous des courbes obtenues ?

4. ✎ Compléter les fonctions `puissance_matrice_2_2_bits`, `produit_matrice_2_2_bits` et `fibonacci_4_bits` pour compter les opérations élémentaires sur les bits, en supposant qu'une multiplication entre deux entiers sur  $m$  et  $n$  bits utilise :
  - a. la multiplication naïve (avec le paramètre `compte = decompte_naif`),
  - b. la multiplication par l'algorithme de Karatsuba (avec le paramètre `compte = decompte_karatsuba`).

Conclure sur les courbes obtenues.