Complexité - TP 1

Faucheron Alcide

Claux Clarence





# Introduction

Ce Tp, divisé en trois exercices, a pour but d’analyser le temps d’exécution de différents algorithmes selon leurs paramètres, leur construction, le nombres d’itérations, …

Nous verrons au cours de ce rapport différents algorithmes associés aux tableaux des temps d’exécutions et leurs courbes.

Tout le code pour la réalisation de ce TP a été réalisé en python3, le readMe contient toutes les instructions pour exécuter ce dernier. La machine utilisée pour compiler et exécuter le code est un macbook pro m1 de 2021 de 8go ram. Ce pc est réputé pour sa puissance de calcul grâce à son processeur arm 64bits. On peut donc s’attendre à de très bons résultats.

# Exercice 1 – Hanoi

Cet exercice a pour but d’étudier le temps d’exécution du problème des tours de hanoi pour différents n donnés.

## Pseudo-code :

Le code est réparti en deux fonctions :

1. Hanoi\_tour, la fonction récursive permettant de résoudre le problème des tours de hanoi selon n, les trois emplacements des tours A, B et C et l’occurrence:

*DEFINE FUNCTION hanoi\_tour(n, A, B, C, occurence):*

*IF n EQUALS 1:*

*RETURN occurence*

*END IF*

*RETURN hanoi\_tour(n - 1, A, C, B, occurence + 1) + hanoi\_tour(n - 1, C, B, A, occurence + 1)*

*END FUNCTION*

1. Hanoi donnant les paramètres nécessaires

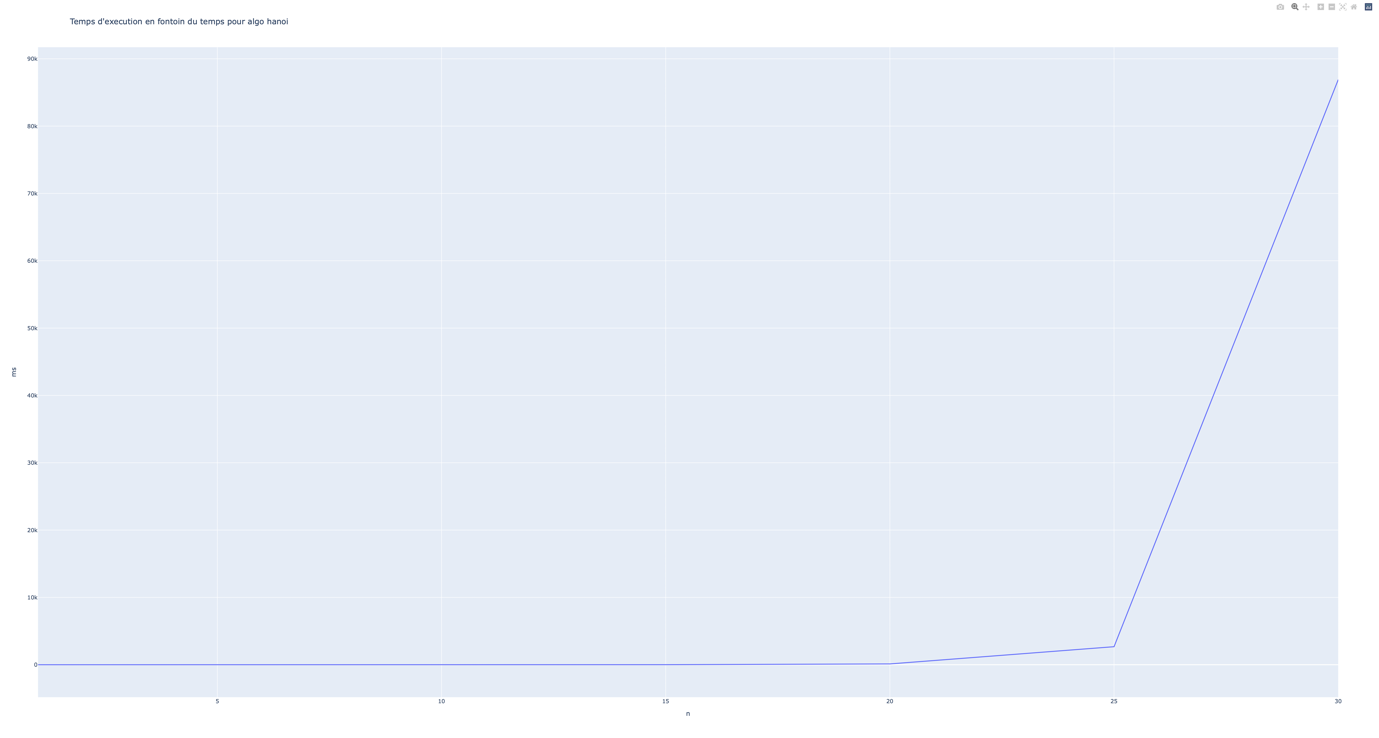
*DEFINE FUNCTION hanoi(n):*

*RETURN hanoi\_tour(n, 'A', 'B', 'C', 0)*

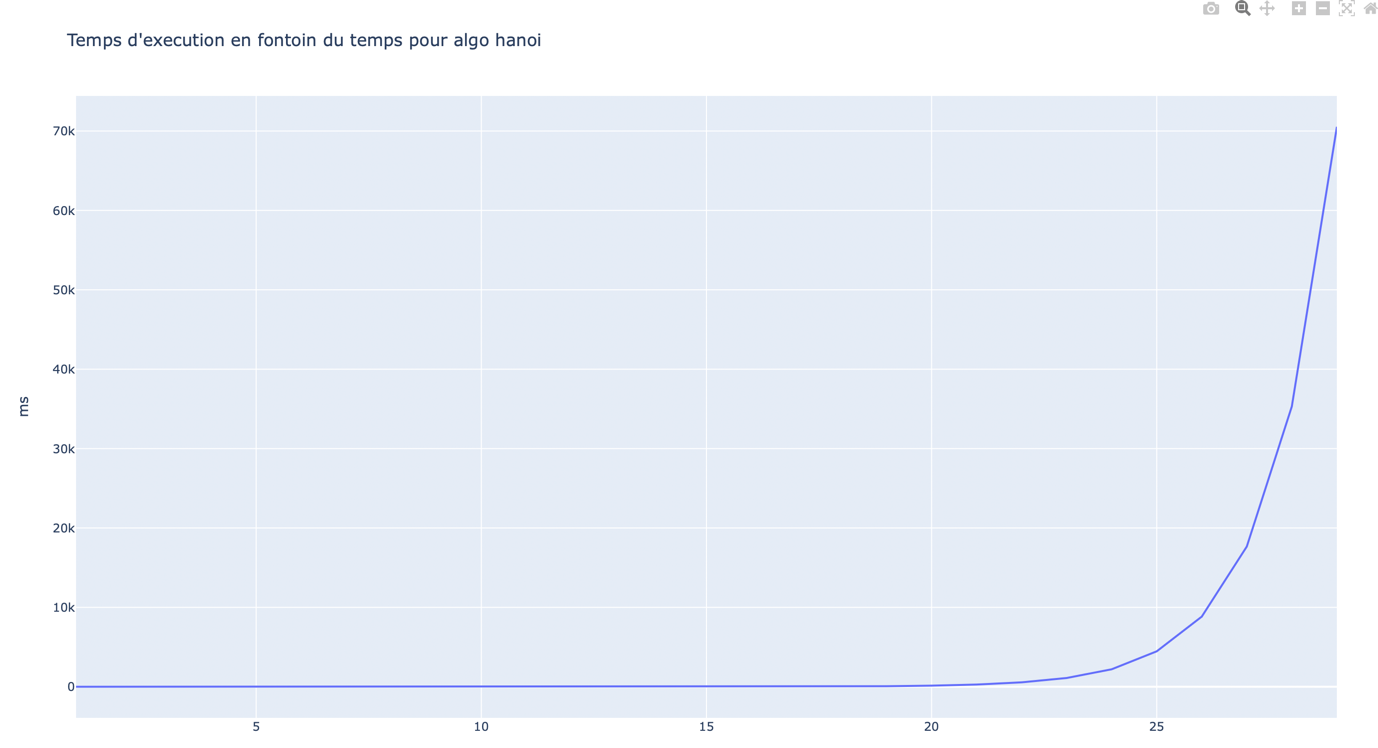
Nous exécutons cet algorithme pour différentes valeurs de n :

|  |  |  |
| --- | --- | --- |
| **N** | **Temps exécution (ms)** | **Temps exécution (s)** |
| 1 | 0,009 | 0 |
| 2 | 0,004 | 0 |
| 5 | 0,011 | 0 |
| 10 | 0,262 | 0 |
| 15 | 7,772 | 0,007 |
| 20 | 120,081 | 0,120 |
| 25 | 2676 | 2,68 |
| 30 | 86892 | 86,9 |
|  |  |  |

Et voici la courbe du temps d’exécution en fonction du temps (disponible au format html interactif dans le dossier source du rendu) :



On peut observer une augmentation exponentielle du temps d’exécution en fonction du nombre n qui augmente. Voici la courbe avec plus de détails pour des n allant de 1 à 30 avec un pas de 1 :



# Exercice 2

Cet exercice a pour but d’étudier le temps d’exécution d’une suite données avec un algorithme itératif et récursif.

## Pseudo-code

### Suite iterative

*DEFINE FUNCTION suite\_interative(n) :*

*SET res TO []*

*FOR i IN range(n + 1):*

*IF i EQUALS 0 or i EQUALS 1:*

*res.append(1)*

*ELSE:*

*res.append(res[i - 1] + res[i - 2])*

*RETURN res[-1]*

### Suite récursive

*DEFINE FUNCTION suite\_recursive(n):*

*IF n EQUALS 0 or n EQUALS 1:*

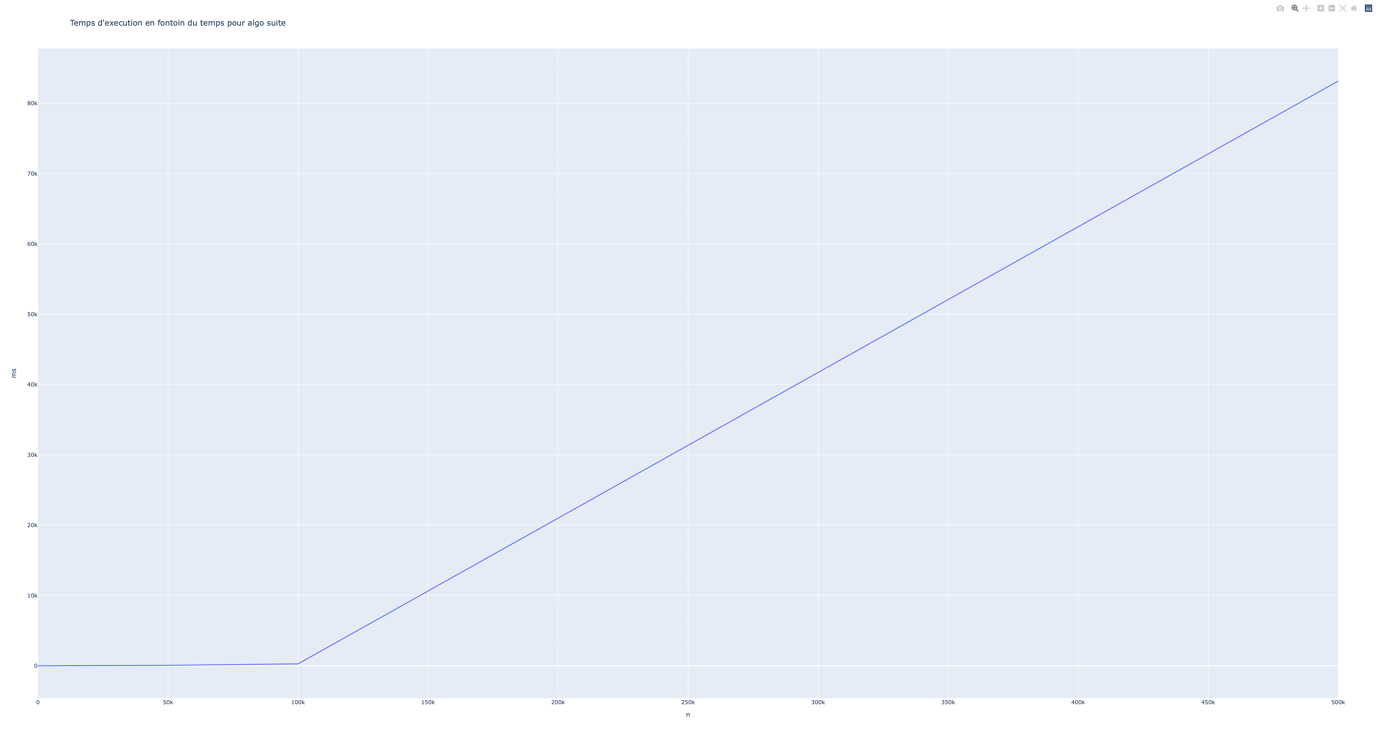
*RETURN 1*

*ELSE:*

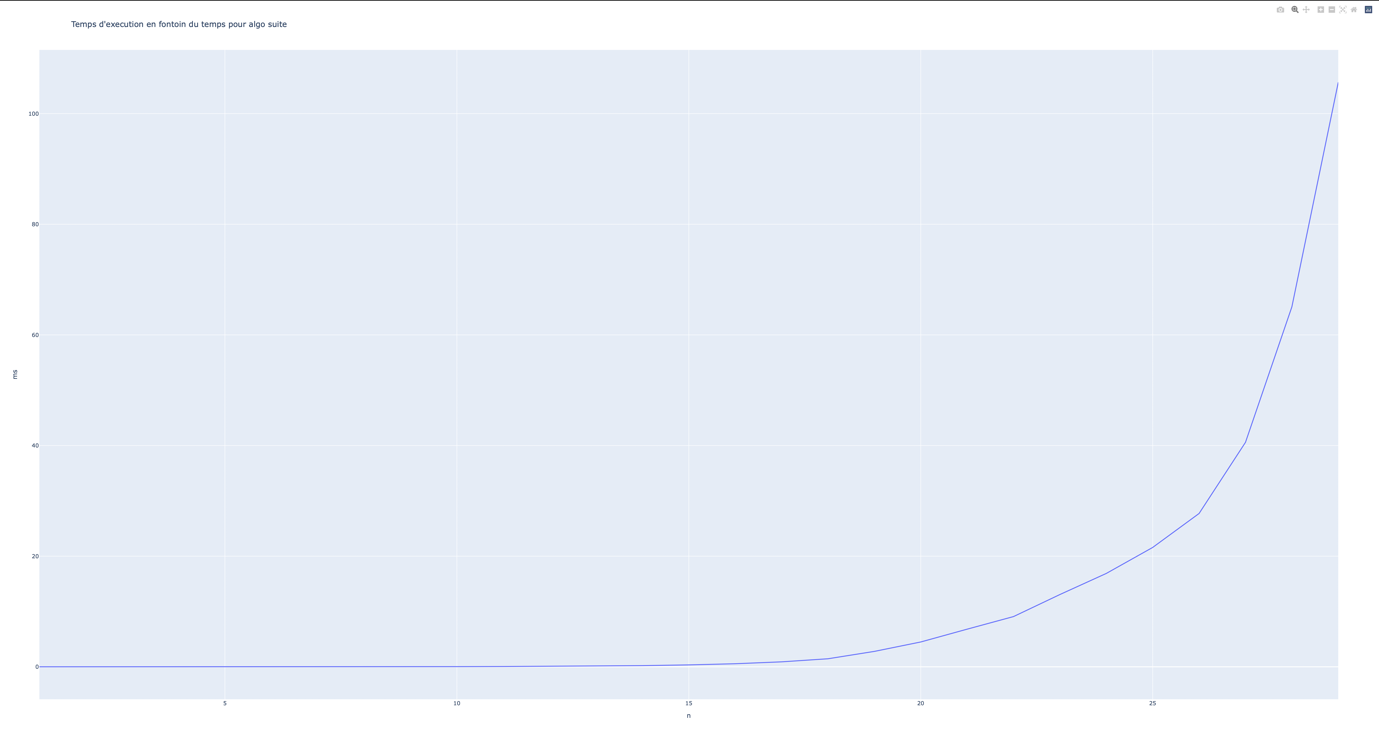
*RETURN suite\_recursive(n - 1) + suite\_recursive(n - 2)*

Temps d’exécution pour différentes valeurs de n :

|  |  |  |
| --- | --- | --- |
| **N** | **TPS exécu itérative (ms)** | **TPS exécu récursive (ms)** |
| 1 | 0,14 | 0,008 |
| 100 | 0,064 | inf |
| 500 | 0,284 | inf |
| 1000 | 0,61 | inf |
| 5000 | 5,78 | inf |
| 10000 | 5,90 | inf |
| 15000 | 14,338 | inf |
| 20000 | 20,536 | inf |
| 35000 | 41,787 | inf |
| 50000 | 62,48 | inf |
| 100000 | 270 | inf |
| 500000 | 83150 | inf |

**Courbe des données fonction itérative :**

**Courbe des données de la fonction récursive :**



On remarque que la courbe de la fonction récursive est exponentielle, elle nécessite trop de temps pour calculer la suite avec un grand n contrairement à la fonction itérative qui se rapproche plus d’une fonction affine au fur et à mesure que n augmente ce qui permet de calculer avec un temps beaucoup moindre pour un grand n.

# Exercice 3

Cet exercice a pour but de déterminer la complexité du crible d’Erastosthène.

## Pseudo-code

Fonction creation\_tableau\_entiers permettant de créer un tableau de n entiers allant de 1 à n :

*DEFINE FUNCTION creation\_tableau\_entiers(n):*

*SET tableau TO []*

*FOR i IN range(1, n):*

*tableau.append(i*

*del(tableau[0])*

*RETURN tableau*

Fonction eratosthene\_crible permettant de déterminer la liste des entiers premiers allant de 1 à n

*DEFINE FUNCTION eratosthene\_crible(tableau, n):*

*L1 FOR i IN range(2, n):*

*L2 if(tableau[i] != 1):*

*L3 FOR j IN range(i + 1, n):*

*L4 if(tableau[j] % tableau[i] EQUALS 0):*

*L5 SET tableau[j] TO 1*

*RETURN tableau*

Calcul de la complexité de la fonction *eratosthene\_crible :*

|  |  |
| --- | --- |
| **Ligne** | **Cout** |
| L1 | (n-2) + 1 |
| L2 | (n-2) |
| L3 |  |
| L4 |  |
| L5 |  |

Soit T(n) la complexité de la fonction,

### Courbes générées par le programme :