SAÉ S1.02

Comparaison d'approches algorithmiques

Berne Léa, Malosse Audran

Table des matières

[1. Vérification et description 1](#_Toc124450071)

[1.1. Algo X 1](#_Toc124450072)

[1.1.1. Analyse théorique 1](#_Toc124450073)

[1.1.2. Analyse pratique 1](#_Toc124450074)

[1.1.3. Description de la complexité temporelle et spatiale 1](#_Toc124450075)

[1.2. Algo Y 1](#_Toc124450076)

[1.2.1. Analyse théorique 1](#_Toc124450077)

[1.2.2. Analyse pratique 2](#_Toc124450078)

[1.2.3. Description de la complexité temporelle et spatiale 2](#_Toc124450079)

[1.3. Algo Z 2](#_Toc124450080)

[1.3.1. Analyse théorique 2](#_Toc124450081)

[1.3.2. Analyse pratique 3](#_Toc124450082)

[1.3.3. Description de la complexité temporelle et spatiale 3](#_Toc124450083)

[2. Comparaison 3](#_Toc124450084)

[3. Ecriture d’un algorithme de tri 3](#_Toc124450085)

[4. Contribution individuelle au projet 4](#_Toc124450086)

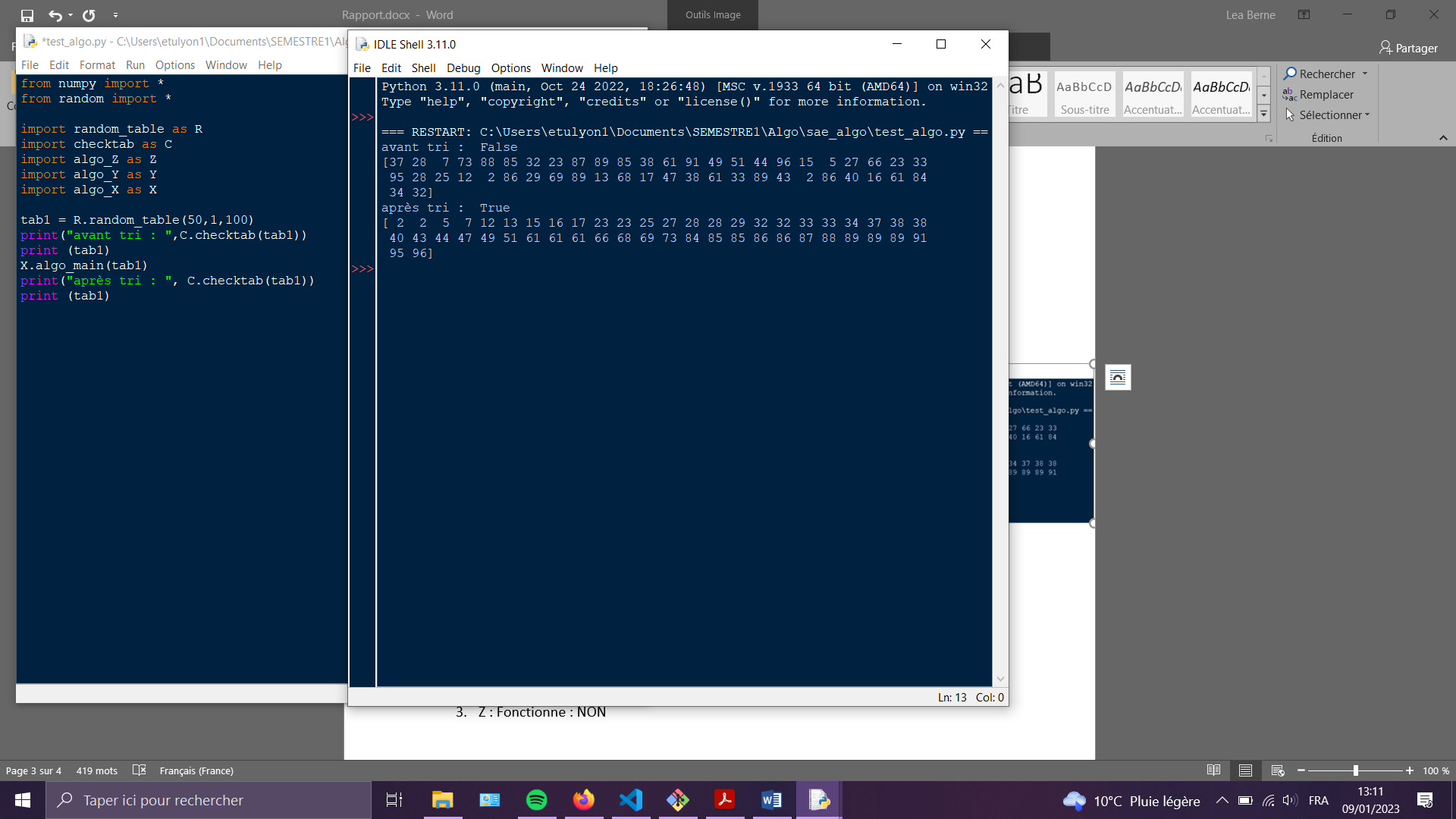
# Vérification et description

## Algo X

### Analyse théorique

L’algorithme X respecte la post-condition (trie un tableau par ordre croissant). La variable i est initialisé à 1. On va donc parcourir le tableau à partir de la seconde valeur de celui-ci. On affecte à la variable j la valeur de i-1, soit l’indice dans le tableau précédent celui de i. Dans la première boucle, on affecte la valeur à l’indice i dans une variable tmp. Puis on affecte la valeur du tableau à l’indice j à tab[j+1] tant que la valeur à indice j est supérieur à la variable tmp (précédemment affecté à la valeur d’indice i) et que j est supérieur ou égale à 0. Et on décrémenter j de 1. Cela a pour effet de faire descendre la valeur à trier dans le tableau pour qu’elle se retrouve à sa place. En sortie de boucle, il faut donc replacer la valeur stockée dans tmp à sa nouvelle place qui est l’indice j+1. On incrémente i et on recommence. Il s’agit d’un algorithme de type tri insertion.

### Analyse pratique

Voilà ce qu’il se passe quand on trie un tableau de 50 entiers compris entre 1 et 100 :

### Description de la complexité temporelle et spatiale

* Complexité Temporelle : O(n²) car une première boucle parcoure le tableau n fois, et dans cette boucle, une autre boucle parcoure le tableau n fois.
* Complexité Spatiale :
  + En place : OUI puisqu'on ne crée pas de nouveau tableau.
  + Stable : NON car on vérifie que la valeur à classer (stockée dans tmp) est strictement inferieure à la valeur comparée. Au moment où tmp sera comparé à une valeur identique à elle-même, la boucle continuera de tourner.

## Algo Y

### Analyse théorique

L’algorithme Y respecte la post-condition (trie un tableau par ordre croissant).

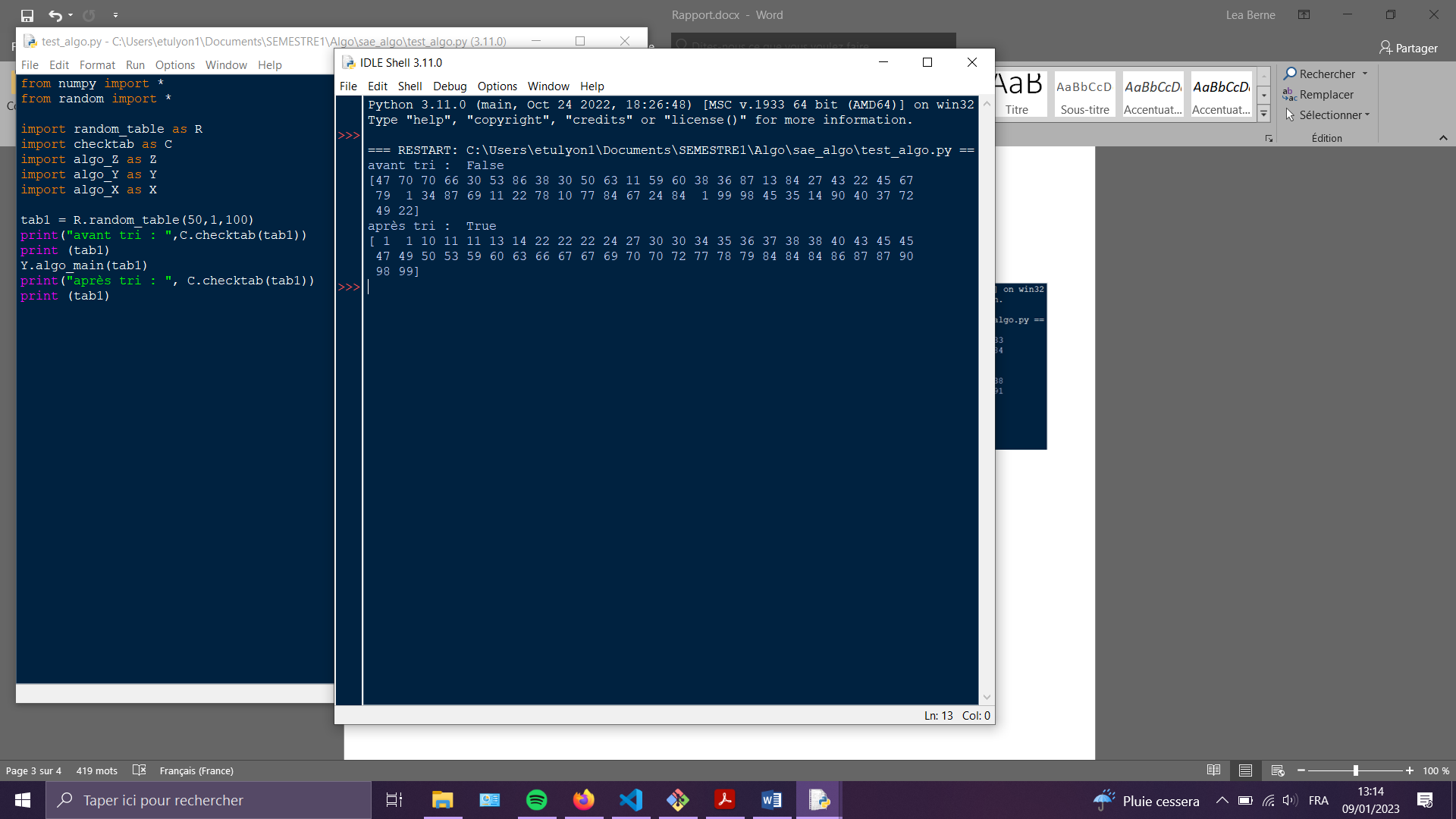
Cet algorithme est composé de trois fonctions qui s’appellent entre elles. C’est la fonction algo\_main qui prends en argument le tableau et appelle la fonction algo\_b avec, en paramètre, le tableau à trier, 0 (qui est l’indice minimum) et len(tab)-1 (qui est l’indice maximum). En vérité, algo\_main sert simplement à appeler algo\_b avec les paramètres nécessaires à un tri fusion.

Algo\_b est un algorithme récursif, et utilise une méthode « diviser pour régner », c’est-à-dire qu’il divise le tableau qu’il prend en paramètre en deux parties. Pour cela, on teste d’abord si i (indice minimum) est égal à j (indice maximum), ce qui entrainerait l’arrêt de la fonction (car un tableau qui n’aurait qu’un élément serait déjà trié). En cela, l’algorithme est similaire au tri fusion, mais contrairement au tri fusion vu en cours, nos trois algorithmes produisent un tri qui est en place : cela vient de l’appel à la fonction algo\_a, qui remplace l’algorithme de fusion que l’on a vu en cours.

L’algo\_a ne fait pas appel à la fonction zeros, et ne prends pas en paramètre le milieu du tableau comme le fait la version de la fusion vue en cours. Par contre, l’algo\_a cherche lui-même le milieu du tableau qui lui a été donné en paramètre pour effectuer un tri par insertion à gauche du milieu.

Par ailleurs, une autre différence entre algo\_b + algo\_a et tri\_fusion + fusion vu en cours est que algo\_b effectue un tri du tableau si le tableau est de deux éléments en permutant les voisins qui ne seraient pas triés. L’algorithme test si le tableau à trier n’a que deux éléments, et ce n’est que si le tableau à trier comporte plus de deux éléments qu’algo\_b fait appel à algo\_a : trier de cette manière permet de ne pas appeler algo\_a dans le cas d’un tableau à deux éléments, ce qui permet sûrement d’optimiser l’algorithme.

### Analyse pratique

Voilà ce qu’il se passe quand on trie un tableau de 50 entiers compris entre 1 et 100 :

### Description de la complexité temporelle et spatiale

* Complexité Temporelle : O(n\*log(n)) car
* Complexité Spatiale :
  + En place : OUI puisqu'on ne crée pas de nouveau tableau.
  + Stable : NON puisque l'algorithme déplace les éléments vers la bonne position, donc il peut modifier l'ordre des éléments à valeurs égales.

## Algo Z

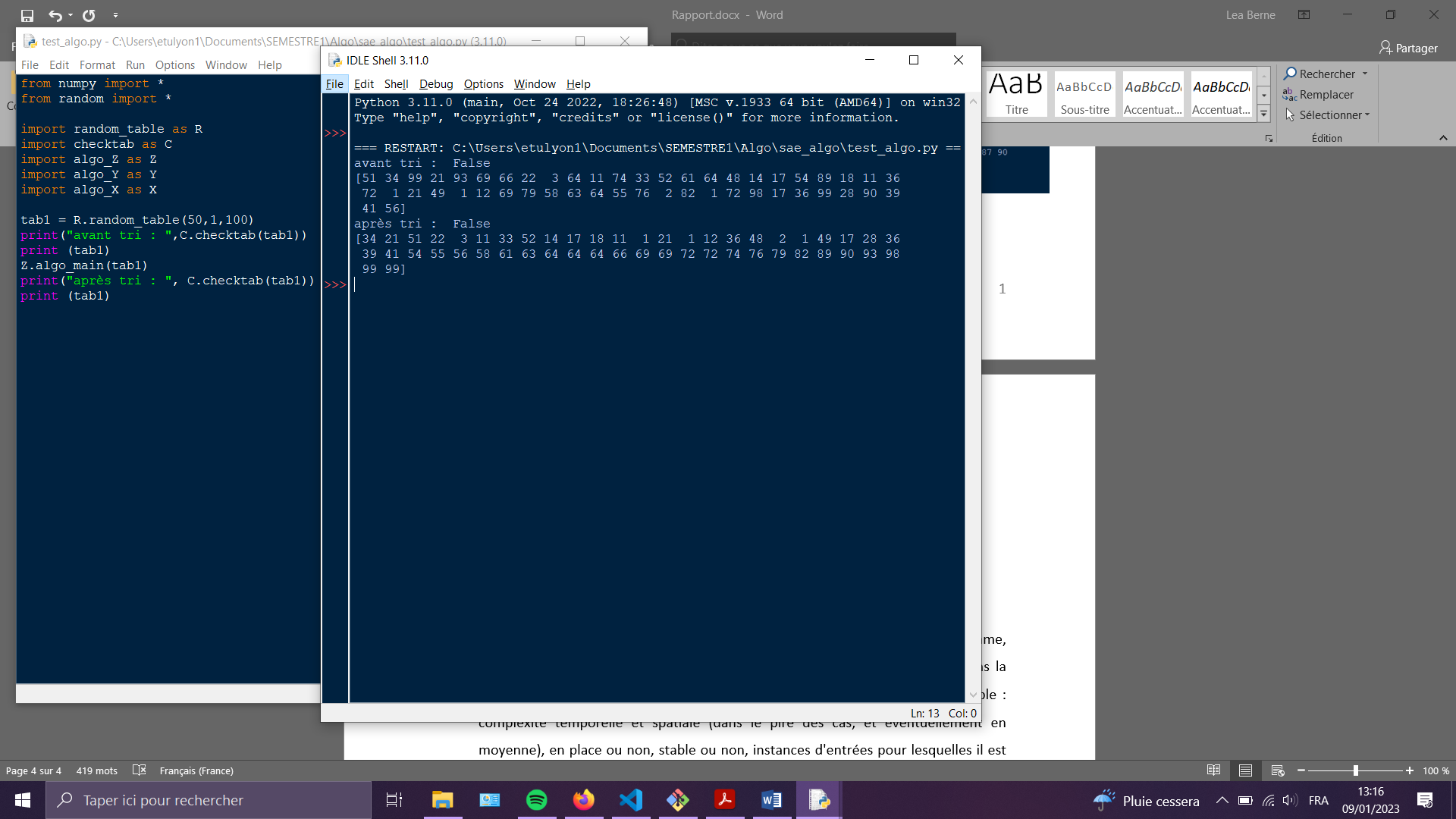
### Analyse théorique

L’algorithme Z ne respecte la post-condition (trie un tableau par ordre croissant). En réalité, l’algorithme compare deux valeurs voisines dans le tableau (celle d’indice j et celle d’indice j+1) et les inverse si la valeur de tab[j] est plus grande que celle de tab[j+1]. Ainsi, à la fin du premier tour de la première boucle while, la plus grande valeur du tableau parcouru aura remonté à droite, et donc la partie de droite sera la partie triée du tableau, ce qui explique que le deuxième while s’arrête à len(tab)-1-i.

Le problème, c’est qu’à la fin du premier tour de la deuxième boucle while, i est incrémenté, or la valeur de j dépend de celle de i. Comme j est initialisé à i, la valeur d’indice 0 ne sera comparée à son voisin de droite qu'une seule fois au court de l’exécution, celle d'indice 1 que deux fois etc. Pour généraliser, la valeur d’indice n ne sera comparée à son voisin de droite que n+1 fois, et s’il se trouve que la valeur d’indice n+1 est plus grande que lui mais que la suivante (n+2) est plus petite, alors la valeur d’indice n restera où elle est malgré tout. La partie de gauche ne sera donc pas triée.

Pour corriger cela, il faut initialiser j à 0 pour reparcourir toute la partie gauche du tableau, puisqu’elle n'est pas triée. Il s’agit en fait d’un tri à bulle.

### Analyse pratique

Voilà ce qu’il se passe quand on trie un tableau de 50 entiers compris entre 1 et 100 :

### Description de la complexité temporelle et spatiale

* Temporelle : O(n²) car la fonction va comparer n fois le tableau en entier.
* Spatiale :
  + En place : OUI, puisqu'on ne crée pas de nouveau tableau.
  + Stable : NON, car en échangeant deux valeurs, on peut inverser l'ordre dans lequel ils apparaissaient.

/ !\ 🡪 penser à ajouter des captures pour temps min & temps max + moyenne POUR TOUS LES ALGOS

# Comparaison

# Ecriture d’un algorithme de tri

* Écrire votre propre algorithme de tri. Vous donnerez sa description, et le comparerez aux trois algorithmes reçus. Vous pourrez par exemple vous inspirer de l'un des algorithmes que vous avez reçu en essayant de l'améliorer, ou bien d'un autre algorithme : vu en cours, ou trouvé dans la littérature. Vous l'écrirez en Python, et vous n'utiliserez pas un algorithme de tri déjà fourni par une librairie. L'évaluation de cette tâche ne repose pas sur le fait de produire un algorithme "meilleur" que ceux que vous aurez reçus, mais surtout sur l'explication de votre solution et ses différences par rapport aux autres algorithmes.

# Contribution individuelle au projet

J’ai fait l’analyse des algorithmes et la description algo Z + algo Y et Audran a fait description algo X + les courbes, process\_time et time\_to sort. Audran a fait la comparaison des algos et j’ai écrit l’algo de tri. Nous avons fait checktab et random\_table + modification algo Z ensemble.

Notre groupe a très bien fonctionné parce que même si on a beaucoup séparé des tâches, on s’est toujours expliqué ce qu’on faisait, on travaillait en même temps, l’un à coté de l’autre pour qu’on soit toujours au courant de ce que faisait l’autre. C’était très appréciable parce qu’on avançait très efficacement.