Rapport SAE 2.02

16/06/2023 - Nathan Pagnucco

I. Outils utilisés

- Pour le temps d'éxecution, le temps moyen sur 1000 essais sur un texte de 1198 caractères avec un ordre de 62 caractères, calculé avec la classe Chrono disponible dans analyse, pour l'utiliser il faut la déplacer dans exercice
- Pour la qualité du code, nous avons utilisé Codacy que nous avons connecté à notre repository github
- Pour la lisibilité du code, nous avons regardé le code en tant qu'évaluateur et lui avons donné une note sur de 1 à 5 avec 1 pour du code illisible et 5 pour du bon code facile à maintenir.
- Pour la complexité du code, on regarde les boucles qui prennent le plus de temps
- Pour la mémoire utilisée, on utilise Runtime.totalMemory() et Runtime.freeMemory() pour avoir la mémoire utilisée

II. Rapport sur les algorithmes

A. Simplicité meilleur

Algorithme 29

Notation :

Passe vos tests supplémentaires plus complets	20

Note : 20

Classement : 1/3

- Lisibilité ⅘ :

Le code est assez facile à comprendre mais les noms de variables à une ou deux lettres ne facilite pas la lecture.

- Qualité du code : A
- Temps d'éxecution : 0.138754 ms

Algorithme 35

Notation :

Fonctionne mais ne passe pas les tests fournis initialement	10

Note : 10

Précision : Ne passe pas les tests car il ignore les chiffres

Classement : 2/3

- Lisibilité ⅘ :

Le code est assez facile à comprendre grâce à la javadoc mais les noms de variables à une lettre ne facilitent pas la lecture.

- Qualité du code : C
- Temps d'éxecution : 0.21782 m

Algorithme 56

Notation :

Fonctionne mais ne passe pas les tests fournis initialement	10
Non respect de la nomenclature précise (Classe ne s'appelle pas Exercice)	-1

Note : 9

Précision : Ne passe pas les tests car il ne rajoute pas les mots qui ne sont pas dans l'ordre.

⚠️Suspicion de chat gpt : Car si on lui donne le sujet du readme il produit un code similaire avec la même erreur d'oublier les mots qui ne sont pas dans l'ordre.

Classement : 3/3

- Lisibilité 5/5 :

Le code est simple à lire et toutes les variables sont bien nommées.

- Qualité du code : B

- Temps d’éxecution : 0.07206ms

B. Simplicité pire

Algorithme 24

Notation :

Fonctionne mais ne passe pas les tests fournis initialement	10
Non respect de la nomenclature précise (Classe ne s’appelle pas Exercice)	-1

Note : 9

Précision : Ne passe pas les tests car il ignore les chiffres .

Classement : 1/2

- Lisibilité : ⅛ :

Le code est en une ligne et le nom des variables est long et dur à reconnaître.

- Qualité du code D
- Temps d’éxecution : 0.103102ms

Remarque

Algorithme 66

Notation :

Passe tous les tests fournis initialement	18
Non respect de la nomenclature précise (Classe ne s’appelle pas Exercice)	-1

Note : 17

Classement : 2/2

- Lisibilité ⅋ :

Le code est toujours bien indenté mais les noms de variables et les commentaires rendent le code plus dur à lire.

- Qualité du code C
- Temps d'éxecution : 0.202175ms

C. Efficacité meilleur

Algorithme 57

Notation :

Fonctionne mais ne passe pas les tests fournis initialement (Si on le répare)	10
Non respect de la consigne sur les méhodes de java.util (pour efficacité)	-1
Ne compile pas	/2

Note : (4,5) arrondi à 5

⚠️HORS CONCOURS

Remarque : Ne compile pas car le package est situé après les imports ce qui est extrêmement bizarre car ça ne peut fonctionner nul part quel que soit l'IDE

Classement : 2/2

- Lisibilité ⅋:

Les variables sont bien nommées pour la plupart mais l'utilisation d'un comparateur avec un lambda ne rend pas le code très lisible

- Qualité du code A
- Temps d'éxecution : 0.153797ms
- Compléxité : O(n+nlog(n)) avec n le nombre de mots, car on utilise une boucle et un tri rapide avec sort pour trier.

Remarque : Car il utilise java.util.*

Algorithme 20

Notation :

Fonctionne mais ne passe pas les tests fournis initialement	10

Note : 10

Remarque : Je ne sais pas pourquoi mais ne marche pas pour le premier test Il fait beau aujourd'hui comme en aout.

Classement : 1/2

- Lisibilité ¾ :

Pour du code en c l'utilisation des commentaires rend le code plus lisible et la javadoc sur le code C permet de comprendre une partie du code, mais une autre partie n'est pas expliqué et reste dure à lire.

- Qualité du code D
- Temps d'execution : 0.201000 ms
- Compléxité : O(m+n+nlog(n)) Avec m taille de l'ordre et n nombre de mots car

D. Efficacité pire

Algorithme 61

Notation :

Passe tous les tests fournis initialement	18
Non respect de la nomenclature précise (Classe ne s'appelle pas Exercice)	-1

Note : 17

Remarque : Ne passe pas les tests supplémentaires car il trie les mots uniquement par leurs initiales

Classement : 1/2

- Lisibilité ⅗ :

Les variables à un mot ne sont pas les plus pratiques pour lire.

- Qualité du code B
- Temps d'éxecution : 0.550359 ms
- Compléxité O(o+m*n+m*n) Avec o le nombre de caractères, m la taille de l'ordre et n le nombre de mots, car on a une première boucle qui parcourt toute la chaine, puis des doubles boucles

Algorithme 64

Notation :

Fonctionne mais ne passe pas les tests fournis initialement	10
Non respect de la nomenclature précise (Classe ne s'appelle pas Exercice)	-1

Note : 9

Remarque : Le code ne marche pas car il essayer de trouver des mots entre plusieurs caractères spéciaux collés et ignore certains caractères spéciaux

Classement : 2/2

- Lisibilité : ⅗

Le nom et l'utilisations de certaines fonctions n'aide pas à comprendre ce qu'elles font

- Qualité du code B
- Temps d'execution :0.48327 ms
- Complexité : O(n + k^2 + m * k) avec n est la taille du texte, k est le nombre de mots dans le texte et m est la taille du mot le plus long.

E. Sobriété meilleur

Algorithme 4

Notation :

Passe tous les tests fournis initialement	18

Note : 18

Remarque : Ne passe pas les tests supplémentaires car il trie les mots uniquement par leurs initiales et ne prend en compte que certains caractères spéciaux

Classement :

- Lisibilité ⅘ :

Facile à comprendre mais certaines variables à une lettre et certains noms de variables rendre plus difficile la lecture.

- Qualité du code B
- Temps d'éxecution 0.197902 ms
- Mémoire utilisée : 572 688 octets

Algorithme 10

Notation :

Fonctionne mais ne passe pas les tests fournis initialement	10

Note : 10

Remarque : Lève une exception sur tous les tests -> Surtout ceux avec des chiffres mais marche sur certains textes. Même si il marchait utilise Character.LowerCase -> ce qui entraînerait d'autres problèmes

Classement :

- Lisibilité 5/5:

Le code est aéré et lisible même avec une variable à une lettre

- Qualité du code B
- Temps d'éxecution 0.581418 ms
- Mémoire utilisée : 650 952 octets

F. Sobriété pire

Algorithme 32

Notation:

Fonctionne mais ne passe pas les tests fournis initialement (Quand réparé)	10
Non respect de la nomenclature précise (Classe ne s'appelle pas Exercice)	-1
Ne compile pas	/2

Note : (4,5) arrondi à 5

⚠️HORS CONCOURS

Remarque : Le code ne compile pas car le package n'est pas valide

Classement : 2/2

- Lisibilité ⅛ :

Le code est illisible car il n'est que sur quelques lignes

- Qualité du code D
- Temps d'execution : 0.340653 ms
- Mémoire utilisée : 651 200 octets

Algorithme 66

Notation :

Passe tous les tests fournis initialement	18
Non respect de la nomenclature précise (Classe ne s'appelle pas Exercice)	-1

Note : 17

Remarque : Ne passe pas les tests supplémentaires car ne trie les mots que par leurs initiales. Et le code ne prend pas en compte tous les caractères spéciaux

Classement : 1/2

- Lisibilité ٪:

Les variables sont toutes bien nommées, mais l'omniprésence des commentaires n'aide pas à lire.

- Qualité du code E
- Temps d'éxecution : 0.265982 ms
- Mémoire utilisée : 663 200 octets