Cahier d'évaluation : SAÉ 2.02

SAÉ 2.02 : Exploration algorithmique d'un problème

Fait par : *Zachary Ivars*

Table of Contents

CONTEXTE **3**

L'EVALUATION **3**

1. EFFICACITE 4
   - *1.1. Efficacite04* *4*
   - *1.2. Efficacite12* *4*
   - *1.3. Efficacite44* *5*
   - *1.4. Efficacite46* *6*
   - *1.5. Efficacite57* *6*
2. SIMPLICITE 7
   - *2.1. Simplicite45* *7*
   - *2.2. Simplicite48* *7*
   - *2.3. Simplicite57* *8*
   - *2.4. Simplicite61* *8*
   - *2.5. Simplicite68* *8*
3. SOBRIETE 9
   - *3.1. Sobriete11* *9*
   - *3.2. Sobriete37* *9*
   - *3.3. Sobriete50* *9*
   - *3.4. Sobriete60* *10*

CLASSEMENT **10**

Contexte

L'objectif de cette SAÉ est d'approfondir la réflexion sur l'approche algorithmique des problèmes vus lors des phases de développement.

Il consiste à :

- Participer à une sorte de « concours » de codage pour fournir des algorithmes différents
- Lire, comprendre et évaluer le code d'autrui
- Comparer les algorithmes sur un critère précis
- Justifier de manière objective ses comparaisons et son classement

Cette SAÉ se déroule en deux phases : codage de trois catégories algorithmiques différentes (efficacité, sobriété et simplicité), et l'évaluation de plusieurs algorithmes appartenant à ces catégories. Ces deux phases sont **individuelles**. Nous avons des contraintes à respecter, tel que le respect des noms de fonctions, il doit passer les tests JUnit…

L'évaluation

20% de la note finale est consacré au codage de nos propres algorithmes, et le 80% restant est l'évaluation des autres algorithmes, avec des justificatifs pertinent et le passage de tests.

Nous avons utilisé **Codacy** pour comparer et évaluer la qualité du code, ainsi que la sécurité, les **test JUnit fourni** pour voir si l'algorithme répondais aux contraintes, puis un site web(shunnarski.github.io/BigO.html) qui fournit la complexité du code insérer. Je n'ai pas fait une comparaison de la comparaison de consommation énergétique ou consommation matériel car ceci n'est pas une évaluation globale. Selon les composants de la machine, leur puissance, les applications en cours, l'environnement de développement utilisé, …, les résultats peuvent varier. Je me suis donc contenter d'évaluer la qualité du code, et le passage des test JUnit et la comparaison de complexité.

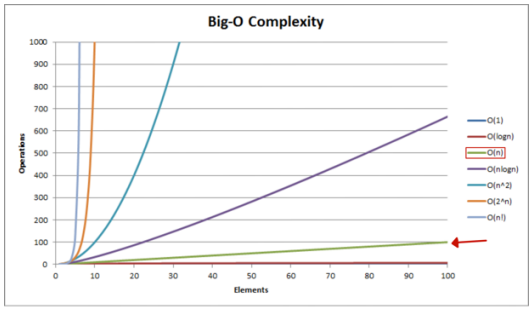
1. Efficacité

1.1. Efficacite04

Classement : 5/5

Évaluation :

- **<u>Évaluation de tests JUnit :</u>** 10/20
- **<u>Lisibilité :</u>** Utilisation de ressources non-vu en cours, sans commentaire pour expliquer le fonctionnement [3/5]
- **<u>Qualité :</u>** B (réaffectation dans une boucle) [4/5]
- **<u>Temps d'exécution (des tests) :</u>** 3ms (mise en commentaire de 3 tests qui échouent) [3/5]
- **<u>Efficacité :</u>** Majorité des fonctions ont une complexité de O(N) [4/5]



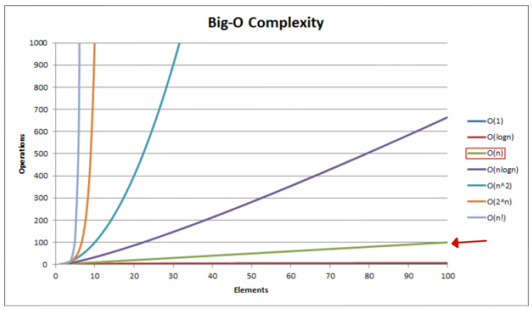
Note globale : 11,5

1.2. Efficacite12

Classement 2/5

Évaluation :

- **<u>Évaluation de tests JUnit :</u>** 18/20
- **<u>Lisibilité :</u>** Utilisation de ressources non-vu en cours, sans commentaire pour tout le programme, non-respect des noms des fonctions [3/5]
- **<u>Qualité :</u>** B (réaffectation dans une boucle) [4/5]
- **<u>Temps d'exécution (des tests) :</u>** 5ms [4.5/5]
- **<u>Efficacité :</u>** Majorité des fonctions ont une complexité de O(N) [4/5]



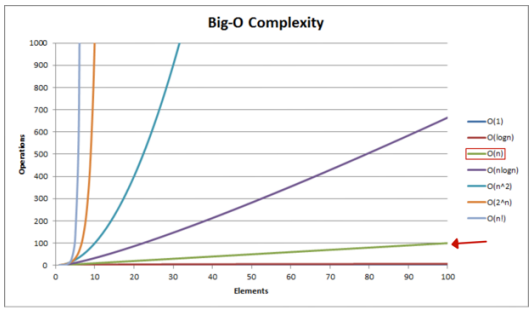
Note globale : 17

1.3. Efficacite44

Classement : 1/5

Évaluation :

- **<u>Évaluation de tests JUnit :</u>** 18/20
- **<u>Lisibilité :</u>** Utilisation de ressources non-vu en cours, commentaires pertinents mêmes s'ils restent très peu [4/5]
- **<u>Qualité :</u>** B (2 réaffectations dans des boucle) [3/5]
- **<u>Temps d'exécution (des tests) :</u>** 7ms [4/5]
- **<u>Efficacité :</u>** Toutes les fonctions ont une complexité de O(N) [5/5]
-



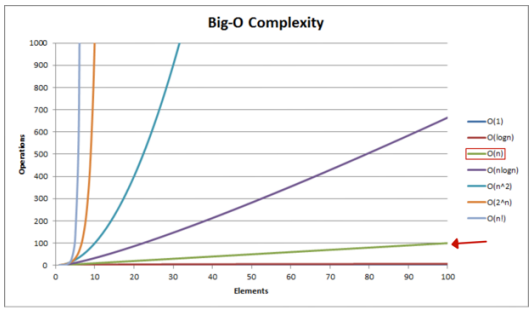
Note globale : 17

1.4. Efficacite46

Classement : 3/5

Évaluation :

- **<u>Évaluation de tests JUnit :</u>** 18/20
- **<u>Lisibilité :</u>** Utilisation de ressources non-vu en cours, commentaires pertinents mêmes s'ils restent très peu [4/5]
- **<u>Qualité :</u>** A [5/5]
- **<u>Temps d'exécution (des tests) :</u>** 61ms [3/5]
- **<u>Efficacité :</u>** Majorité des fonctions ont une complexité de O(N), avec possibilité de 2 boucles à l'infinie [3/5]
-



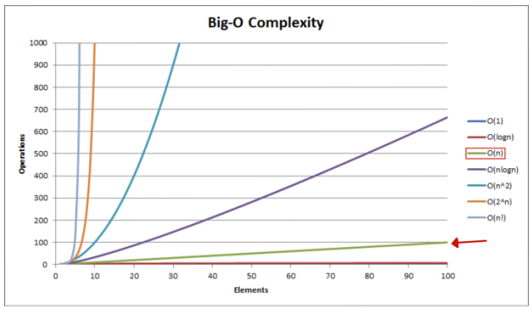
Note globale : 17

1.5. Efficacite57

Classement : 4/5

Évaluation :

- **<u>Évaluation de tests JUnit :</u>** 5/20
- **<u>Lisibilité :</u>** Utilisation de ressources non-vu en cours, commentaires pertinents [5/5]
- **<u>Qualité :</u>** B (réaffectation dans une boucle) [4/5]
- **<u>Temps d'exécution (des tests) :</u>** 10ms [4/5]
- **<u>Efficacité :</u>** Tous les fonctions ont une complexité de O(N) [4/5]



Note globale : 11

2. Simplicité

2.1. Simplicite45

Classement : 3/5

Évaluation :

- **<u>Évaluation de tests JUnit :</u>** 18/20
- **<u>Lisibilité :</u>** Code noyé par la répétition des commentaires, une majorité étant peu pertinent [3/5]
- **<u>Qualité :</u>** C (réaffectations dans des boucle) [2/5]
- **<u>Temps d'exécution (des tests) :</u>** 13ms [4/5]
- **<u>Efficacité :</u>** Majorité des fonctions ont une complexité de O(N) (possibilité de 2 boucles à l'infinie) [3/5]

Note globale : 15

2.2. Simplicite48

Classement : 2/5

Évaluation :

- **<u>Évaluation de tests JUnit :</u>** 18/20
- **<u>Lisibilité :</u>** Pas de commentaire, mais séparation claire des éléments du code [3/5]
- **<u>Qualité :</u>** C (comparaison entre String avec '==') [3/5]
- **<u>Temps d'exécution (des tests) :</u>** 10ms [4/5]
- **<u>Efficacité :</u>** O(1) [5/5]

Note globale : 17

2.3. Simplicite57

Classement : 5/5

Évaluation :

- **Évaluation de tests JUnit :** 5/20 (non-fonctionnement des unRLE)
- **Lisibilité :** Javadoc et commentaires pertinents tout le long du code [3/5]
- **Qualité :** A (1 réaffectation) [5/5]
- **Temps d'exécution (des tests) :** 10ms [2/5] (échoue de 11 tests)
- **Efficacité :** O(1) [5/5]

Note globale : 10

2.4. Simplicite61

Classement : 4/5

Évaluation :

- **Évaluation de tests JUnit :** 10/20
- **Lisibilité :** Commentaire qui guide l'utilisateur [3/5]
- **Qualité :** B (variable i qui n'est pas utilisé) [4/5]
- **Temps d'exécution (des tests) :** 1ms [4/5] (échoue de 2 tests)
- **Efficacité :** O(N) [4/5]

Note globale : 13

2.5. Simplicite68

Classement : 1/5

Évaluation :

- **Évaluation de tests JUnit :** 18/20
- **Lisibilité :** Javadoc et commentaires de qualité et très pertinent tout le long [5/5]
- **Qualité :** A [5/5]
- **Temps d'exécution (des tests) :** 6ms [4/5]
- **Efficacité :** O(N) (possibilité de 2 boucles infinies) [3/5]

Note globale : 18

3. Sobriété

3.1. Sobriete11

Classement : 3/4

Évaluation :

- **Évaluation de tests JUnit :** 18/20
- **Lisibilité :** Pas de commentaire (notation subjectif car je ne suis pas le meilleur en C) [1.5/6.5]
- **Qualité :** A (utilisation de printf au lieu de sprint_f) [5/6.5]
- **Temps d'exécution (des tests) :**
- **Efficacité :** O(N) [5.5/6.5]

Note globale : 15.5

3.2. Sobriete37

Classement : 4/4

Évaluation :

- **Évaluation de tests JUnit :** 10/20
- **Lisibilité :** Des commentaires pertinents [5/6.5]
- **Qualité :** B (pas de condition d'arrêt pour les valeurs de type String) [5/6.5]
- **Temps d'exécution (des tests) :**
- **Efficacité :** $O(N^3)$[2/6.5]

Note globale : 11

3.3. Sobriete50

Classement : 2/4

Évaluation :

- **Évaluation de tests JUnit :** 18/20
- **Lisibilité :** Code en ligne, ce qui le rends peu lisible [2.5/5]
- **Qualité :** D [2.5/5]
- **Temps d'exécution (des tests) :** 12ms [4/5]
- **Efficacité :** O(1) ou boucle infinie [4/5]

Note globale : 16

3.4. Sobriete60

Classement : 1/4

Évaluation :

- **Évaluation de tests JUnit :** 18/20
- **Lisibilité :** Code bien structuré, manque de commentaires [4/5]
- **Qualité :** B [3/5]
- **Temps d'exécution (des tests) :** 2ms [5/5]
- **Efficacité :** O(1) [5/5]

Note globale : 18

Classement

Voici le classement final des catégories de programmes :

Efficacité :

- 1/5 : efficacite44.java
- 2/5 : efficacite12.java
- 3/5 : efficacite46.java
- 4/5 : efficacite57.java
- 5/5 : efficacite04.java

Simplicité :

- 1/5 : simplicite68.java
- 2/5 : simplicite48.java
- 3/5 : simplicite45.java
- 4/5 : simplicite61.py
- 5/5 : simplicite57.java

Sobriété :

- 1/4 : sobriete60.java
- 2/4 : sobriete50.java
- 3/4 : sobriete11.c
- 4/4 : sobriete37.c