

SAE 2.02 : Exploration algorithmique d'un problème

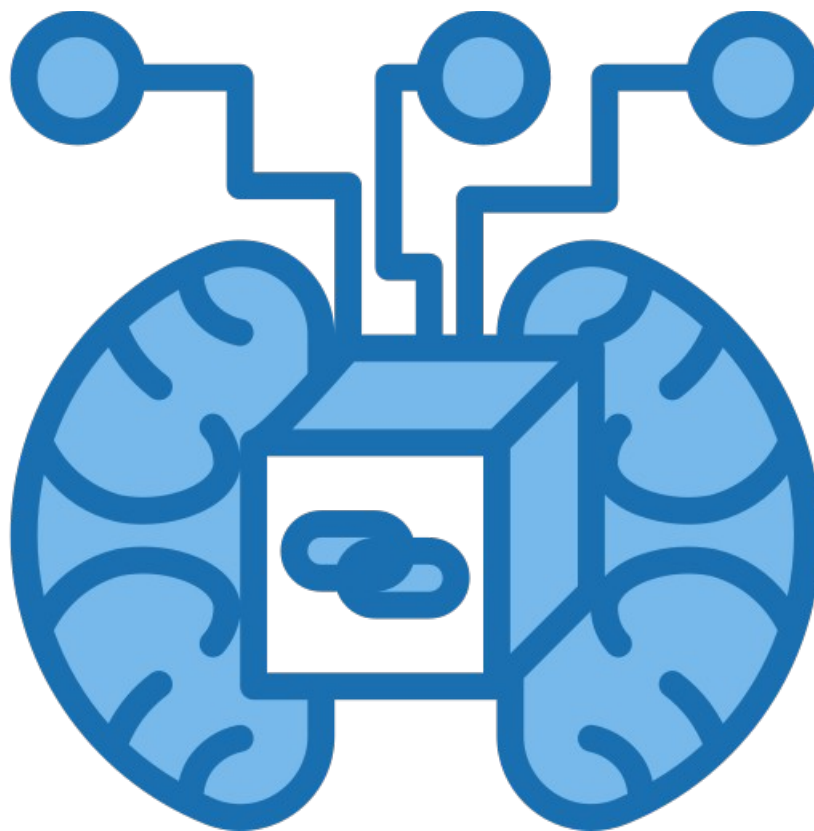


Table des matières

SAE 2.02 : Exploration algorithmique d'un problème.....	1
1) Présentation des outils d'évaluation.....	3
2) Classement des algorithmes par petite catégorie.....	3
2.1) Simplicité-Meilleur.....	4
2.2) Simplicité-Pire.....	4
2.3) Efficacité-Meilleur.....	5
2.4) Efficacité-Pire.....	5
2.5) Sobriété-Meilleur.....	5
2.6) Sobriété-Pire.....	5
3) Classement des algorithmes par grande catégorie.....	6
3.1) Simplicité.....	6
3.2) Efficacité.....	6
3.3) Sobriété numérique.....	7
3) Conclusion.....	7

1) Présentation des outils d'évaluation

Pour évaluer les algorithmes qui m'ont été donnés je vais utiliser des outils donnés et en libre service comme par exemple Codacy mais également mon esprit critique pour la lisibilité du code et sa capacité à être compris par le plus grand nombre. J'utiliserai également les tests JUnit donnés avec en plus un test ajoutés par Bastien Record et moi même. Enfin, j'utiliserai une classe réalisée par mes soins pour calculer le temps de calcul d'un programme ainsi que les ressources en mémoire utilisées pendant le processus.

2) Classement des algorithmes par petite catégorie

Dans un premier temps, nous allons procéder à classer les algorithmes par petite partie à savoir meilleur et pire.

Voici la liste des algorithmes qui m'étaient attribués :

- 3-sobrietemeilleur
- 47-sobrietemeilleur
- 58-sobrietemeilleur
- 12-sobriete-pire
- 48-sobrietePire
- 7-efficacite-meilleur
- 40-efficacite-meilleur
- 36-efficacite-pire
- 49-efficacite-pire
- 23-simplicite-meilleur
- 51-simplicite-meilleur
- 47-simplicite-pire
- 56-simplicite-pire

Quand l'emoji « 🤖 » apparaît devant le classement cela signifie que je suspecte ce code d'avoir été rédigé ou partiellement rédigé par ChatGPT.

Je vous invite à regarder attentivement le barème modifié présent sur github dans le dossier 'rapport' pour comprendre comment ont été notés les algorithmes.

Il est à noter que pour mesurer l'efficacité et la sobriété, j'ai dû comparer les algorithmes entre eux pour dénoter les points positifs relatifs à ces catégories, ces critères étant extrêmement dépendant de la machine qui les exécute. Il n'y a donc pas réellement de temps d'exécutions ou d'utilisation de mémoire « parfaits ».

2.1) Simplicité-Meilleur

🏆 23-simplicite-meilleur : Non respect de la nomenclature : - 1. Assez commenté, noms de variables explicites, les tests ne passent pas mais compile, 18% d'issues sur Codacy ce qui est passable. TOTAL : 9/20

🏆 3-simplicite-meilleur : Non respect de la nomenclature : - 1. Les tests ne passent pas mais compile. Très bien commenté, noms de variables clairs. Score codacy de 14 %. TOTAL : 9/20.

🏆 51-simplicite-meilleur : Non respect de la nomenclature : - 1. Très commenté, noms de variables très explicites cependant le main compile mais pas les tests donc : 5 et hors-concours. Score Codacy : 17 %. TOTAL : 4/20.

2.2) Simplicité-Pire

🏆 47-simplicite-pire : Non respect de la nomenclature : -1. Aucun commentaire : - 2, noms de variables absolument pas explicites - 2. Les tests ne passent pas mais compile. 12% d'issues sur Codacy. TOTAL : 5/20.

🏆 56-simplicite-pire : Respect de la nomenclature mais non respect des conventions de nommage sur la classe : - 1. Aucun commentaire - 2. Nommage de variables pas du tout explicites : - 2. Tests passent pas mais compile. Codacy 13% passable. TOTAL : 5/20.

2.3) Efficacité-Meilleur

Leboucher Antoine 1B

🏆 40-efficacite-meilleur : Utilisation de la méthode startWith() (clairement possible de faire sans) : - 1. Tests passent tous sauf notre tests. Très bon temps d'exécution. TOTAL : 17.5/20

🏆 7-efficacite-meilleur : Non respect de la nomenclature : -1, retour erroné, renvoie une liste de caractère sans aucun tri au lieu de renvoyer une liste de mot. Très bon temps d'exécution. TOTAL : 4/20

2.4) Efficacité-Pire

🏆🤖 36-efficacite-pire : Peu de méthode réutilisée (bonne chose), bon temps d'exécution, tous les tests passent y compris le dernier rajouté par nos soins. TOTAL : 20/20

🏆 49-efficacite-pire : Non respect de la nomenclature : -1, utilisation de split() - 1 (alors qu'elle était explicitement interdite) Aucun test ne passe mais compile. Main livré avec la classe - 1. Bon temps d'exécution. TOTAL : 7/20

2.5) Sobriété-Meilleur

🏆 58-sobriete-meilleur : Les tests initiaux passent mais pas nos tests. Très bon temps d'exécution mais utilisation de mémoire améliorable. TOTAL: 18/20

🏆 47-sobriete-meilleur : Les tests ne passent pas mais compile. Temps d'exécution entre 1 et 0.3 ms pas terrible : - 1. Utilisation mémoire satisfaisante. TOTAL : 9/20

2.6) Sobriété-Pire

🏆 12-sobriete-pire : Les tests fournis par le sujet passent tous, pas nos tests, Temps d'exécution pas mal et utilisation de mémoire pareil (entre 900 ko et 1 mo). TOTAL : 18/20

🏆🤖 48-sobriete-pire : Non respect de la nomenclature : - 1. Les import sont manquants : - 1. Ne passe pas les tests mais compile, temps d'exécution très bien, mémoire pareille. TOTAL : 8/20.

3) Classement des algorithmes par grande catégorie

Dans un deuxième temps pour comparer de manière plus poussée on va comparer les algorithmes par grande catégorie à savoir Simplicité/Efficacité/Sobriété Numérique pour voir si certains algorithmes mériteraient d'être qualifiés de meilleur ou de pire. Les notes et les commentaires restent les mêmes et c'est pour cette raison que je les ai effacé dans cette partie par souci de lisibilité.

3.1) Simplicité

 23-simplicite-meilleur


 3-simplicite-meilleur

 47-simplicite-pire

4ème) 56-simplicite-pire

5ème) 51-simplicite-meilleur. Se rapproche plus d'un simplicite-pire.

3.2) Efficacité

 36-efficacite-pire. Se rapproche plus d'un efficacite-meilleur

 40-efficacite-meilleur

 49-efficacite-pire

4ème) 7-efficacite-meilleur. Se rapproche plus d'un efficacite-pire.

3.3) Sobriété numérique

🏆 12-sobriete-pire. Se rapproche plus d'un sobriété-meilleur.

🏆 58-sobriete-meilleur

🏆 47-sobriete-meilleur. Se rapproche plus d'un sobriété-pire.

4ème 🤖) 48-sobriete-pire

3) Conclusion

Dans cette SAE nous avons pu apprendre tout d'abord à développer des algorithmes servant un but final précis. Puis nous avons évalué les algorithmes d'autres développeurs à l'aide d'outils pour comprendre les enjeux d'un bon algorithme ainsi que les différentes choses qui font la qualité d'un algorithme à travers les 3 catégories proposées :

- la simplicité
- la sobriété
- l'efficacité.

Tout cela a contribué à nous montrer les bonnes pratiques du développement informatique sur plusieurs plans.