S2.02
Exploration algorithmique d'un problème

Phase 2
Évaluation des algorithmes

Sommaire

Évaluation

Grille d'évaluation

Problème	Sanction(s)
Ne compile pas	Note finale divisé par 2 & Hors concours
Ne respect de l'anonymat	-1
Non-respect de la consigne sur les méthodes de java.util (pour efficacité)	-1 & Hors concours
Ne fonctionne pas (retour erroné, ou pas du bon type attendu)	5
Fonctionne mais ne passe pas les tests fournis initialement	10
Passe tous les tests fournis initialement	18
Passe vos tests supplémentaires plus complets	20

Correction des algorithmes

Algorithme	Problème(s)	Sanction(s)	Note finale
19simplicite.py	Fonctionne mais une seule fonction a été implémenté RLE renommé RLE avec itérations : Absent unRLE : Absent unRLE avec itérations : Absent	-10 -1 -1 -1 -1	6
54simplicite.java	Fonctionne mais ne passe pas tout les tests RLE avec itérations : Renommé unRLE : Absent unRLE avec itérations : Absent	-10 -1 -1 -1	7
41simplicite.java	Aucun	0	20
37simplicite.py	Aucun	0	20
46simplicite.java	Aucun	0	20
34efficacite.c	Aucun	0	20
36efficacite.java	Aucun	0	20

Algorithme	Problème(s)	Sanction(s)	Note finale
43efficacite.java	Aucun	0	20
08efficacite.java	Aucun	0	20
62efficacite.java	Aucun	0	20
59sobriete.java	Aucun	0	20
17sobriete.c	Aucun	0	20
11sobriete.c	Aucun	0	20
19sobriete.py	Une seule fonction passe le testPython RLE renommé RLE avec itérations : Absent unRLE : Absent unRLE avec itérations : Absent	-10 -1 -1 -1 -1	6

Classements

Simplicité

Algorithme	Justifications	Classement final
46simplicite.java	Ce code utilise des noms de variables et de méthodes descriptifs, ce qui améliore la lisibilité. Il est bien structuré avec des boucles et des conditions claires et contient des commentaires utiles. Note Codacy : A	1
37simplicite.py	Le code est simple à comprendre grâce à l'utilisation de noms de variables descriptifs, bien que parfois trop longs, et à une structure de boucle claire. Il contient des commentaires détaillés qui expliquent le fonctionnement du code, ce qui aide à la lisibilité. Note Codacy : B	2
41simplicite.java	Ce code utilise des noms de variables et de méthodes descriptifs et bien organisés. Les commentaires sont présents et aident à comprendre le code. Note Codacy : C	3
19simplicite.py	Le code est clair et utilise des noms de variables descriptifs, avec une structure logique simple à suivre. Cependant, il manque des commentaires explicatifs et il manque trois fonctions. Note Codacy : B	4
54simplicite.java	Bien que le code utilise des noms de variables et de méthodes descriptifs et soit bien structuré, il manque des commentaires explicatifs. De plus, il ne contient que deux fonctions (RLE et RLEIteration) Note Codacy : B	5

Efficacité

Algorithme	Justifications	Classement final
43efficacite.java	Sa gestion des opérations avec un contrôle strict des itérations et l'utilisation de StringBuilder pour éviter la concaténation coûteuse offre une bonne complexité algorithmique de O(n).	1
34efficacite.c	Offre la meilleure vitesse d'exécution brute grâce à l'usage direct de la mémoire système, mais la complexité algorithmique est aussi O(n). Toutefois, la gestion manuelle de la mémoire en C pourrait introduire des inefficacités et des risques d'erreur	2
62efficacite.java	Même si ce code utilise efficacement StringBuilder avec des initialisations optimisées pour éviter les conversions inutiles, il fait plusieurs passes sur la chaîne pour gérer les comptages, ce qui pourrait potentiellement augmenter la complexité dans certains cas, bien que généralement il opère en O(n).	3
36efficacite.java	La complexité reste O(n) car chaque caractère est traité une fois, mais il est légèrement moins optimisé pour les itérations multiples	4
08efficacite.java	L'utilisation de concaténations de chaînes simples augmente le coût des opérations de chaînes, et bien qu'il opère en O(n), les méthodes employées sont moins efficaces que celles utilisant StringBuilder.	5

Sobriété

Algorithme	Mémoire allouée	Classement Final
17sobriete.c	Ce code gère efficacement la mémoire avec des allocations et libérations appropriées et utilise des techniques de gestion directe qui réduisent la surcharge. Il est le plus sobre en termes de gestion de la mémoire et des ressources CPU, malgré les coûts associés aux itérations.	1
59sobriete.java	Bien que Java soit plus gourmand en ressources que C, ce code utilise StringBuilder efficacement, ce qui minimise les coûts de création de nouvelles chaînes et la surcharge de la mémoire.	2
11sobriete.c	Ce code est similaire au 17sobriete.c mais avec moins d'optimisation dans la gestion des itérations et l'utilisation de sprintf, qui peut être moins efficace que les opérations directes en C.	3
19sobriete.py	Ce code Python est le moins sobre numériquement à cause de l'utilisation répétée de concaténations de chaînes, ce qui entraîne une consommation excessive de la mémoire.	4