

SAE 2.02 : Rapport d'évaluation des Algorithmes

Table des matières

Grille d'évaluation des algorithmes.....	3
<i>Simplicité-Meilleur.....</i>	<i>3</i>
<i>Simplicité-Pire.....</i>	<i>5</i>
<i>Efficacité-Meilleur.....</i>	<i>6</i>
<i>Efficacité-Pire.....</i>	<i>7</i>
<i>Sobriété-Meilleur.....</i>	<i>8</i>
<i>Sobriété-Pire.....</i>	<i>9</i>
Lisibilité du code.....	10
<i>Simplicité-Meilleur.....</i>	<i>10</i>
<i>Simplicité-Pire.....</i>	<i>12</i>
<i>Efficacité-Meilleur.....</i>	<i>13</i>
<i>Efficacité-Pire.....</i>	<i>14</i>
<i>Sobriété-Meilleur.....</i>	<i>15</i>
<i>Sobriété-Pire.....</i>	<i>16</i>
Qualité du code.....	17
<i>Simplicité-Meilleur.....</i>	<i>17</i>
<i>Simplicité-Pire.....</i>	<i>19</i>
<i>Efficacité-Meilleur.....</i>	<i>19</i>
<i>Efficacité-Pire.....</i>	<i>20</i>
<i>Sobriété-Meilleur.....</i>	<i>21</i>
<i>Sobriété-Pire.....</i>	<i>22</i>
Efficacité / Temps d'exécution.....	23
<i>Tableau de la rapidité des différents algorithmes.....</i>	<i>24</i>
Sobriété numérique.....	25
Classement par catégorie.....	26
<i>Simplicité-Meilleur.....</i>	<i>26</i>
<i>Simplicité-Pire.....</i>	<i>26</i>
<i>Efficacité-Meilleur.....</i>	<i>27</i>
<i>Efficacité-Pire.....</i>	<i>27</i>
<i>Sobriété-Meilleur.....</i>	<i>27</i>
<i>Sobriété-Pire.....</i>	<i>28</i>
Classement toutes catégories confondues.....	29

Grille d'évaluation des algorithmes

Simplicité-Meilleur

1	Problème	Sanction		30-Main
2	Ne compile pas	note finale divisée par 2	Et hors concours	Compile en changeant le nom de la Fonction et le type en paramètre
3	Ne correspond pas à une classe Exercice (cas des main ou des zip)	note finale divisée par 2		-1
4	Non respect de la nomenclature précise	-1		
5	Non respect de l'anonymat	-1		
6	Non respect de la consigne sur les méthodes de <code>java.util</code> (pour efficacité)	-1	Et hors concours	Deux tests ne fonctionnent pas
7	Passe tous les tests fournis initialement	18		
8	Passe vos tests supplémentaires plus complets	20		
9	Fonctionne mais ne passe pas les tests fournis initialement	10		Deux tests ne fonctionnent pas
10	Ne fonctionne pas (retour <code>erronné</code> , ou pas du bon type attendu)	5		On doit changer le type des paramètres
11	TOTAL (/20)			2/20

Figure 1: 30-Main(Simplicité-meilleur)

1	Problème	Sanction		37-simplicité-meilleur
2	Ne compile pas	note finale divisée par 2	Et hors concours	Compile
3	Ne correspond pas à une classe Exercice (cas des main ou des zip)	note finale divisée par 2		
4	Non respect de la nomenclature précise	-1		
5	Non respect de l'anonymat	-1		
6	Non respect de la consigne sur les méthodes de <code>java.util</code> (pour efficacité)	-1	Et hors concours	
7	Passe tous les tests fournis initialement	18		Deux tests ne fonctionnent pas
8	Passe vos tests supplémentaires plus complets	20		Deux tests ne fonctionnent pas
9	Fonctionne mais ne passe pas les tests fournis initialement	10		Deux tests ne fonctionnent pas
10	Ne fonctionne pas (retour <code>eronné</code> , ou pas du bon type attendu)	5		
11	TOTAL (/20)			12/20

Figure 2: 37-Simplicité-meilleur

1	Problème	Sanction		58-simplicité-meilleure
2	Ne compile pas	note finale divisée par 2	Et hors concours	Compile
3	Ne correspond pas à une classe Exercice (cas des main ou des zip)	note finale divisée par 2		
4	Non respect de la nomenclature précise	-1		
5	Non respect de l'anonymat	-1		
6	Non respect de la consigne sur les méthodes de <code>java.util</code> (pour efficacité)	-1	Et hors concours	
7	Passe tous les tests fournis initialement	18		Passe tous les tests
8	Passe vos tests supplémentaires plus complets	20		Passe tous les tests
9	Fonctionne mais ne passe pas les tests fournis initialement	10		Passe tous les tests
10	Ne fonctionne pas (retour <code>eronné</code> , ou pas du bon type attendu)	5		
11	TOTAL (/20)			20/20

Figure 3: 58-simplicité-meilleure

Simplicité-Pire

1	Problème	Sanction		12-simplicite-pire
2	Ne compile pas	note finale divisée par 2	Et hors concours	Compile
3	Ne correspond pas à une classe Exercice (cas des main ou des zip)	note finale divisée par 2		
4	Non respect de la nomenclature précise	-1		
5	Non respect de l'anonymat	-1		
6	Non respect de la consigne sur les méthodes de java.util (pour efficacité)	-1	Et hors concours	
7	Passe tous les tests fournis initialement	18		Passe tous les tests
8	Passe vos tests supplémentaires plus complets	20		Passe tous les tests
9	Fonctionne mais ne passe pas les tests fournis initialement	10		Passe tous les tests
10	Ne fonctionne pas (retour erroné, ou pas du bon type attendu)	5		
11	TOTAL (/20)			20/20

Figure 4: 12-Simplicité pire

1	Problème	Sanction		48-simplicitePire
2	Ne compile pas	note finale divisée par 2	Et hors concours	Compile
3	Ne correspond pas à une classe Exercice (cas des main ou des zip)	note finale divisée par 2		
4	Non respect de la nomenclature précise	-1		
5	Non respect de l'anonymat	-1		
6	Non respect de la consigne sur les méthodes de java.util (pour efficacité)	-1	Et hors concours	
7	Passe tous les tests fournis initialement	18		Un des tests ne fonctionne pas
8	Passe vos tests supplémentaires plus complets	20		Aucun test ne fonctionne
9	Fonctionne mais ne passe pas les tests fournis initialement	10		
10	Ne fonctionne pas (retour erroné, ou pas du bon type attendu)	5		
11	TOTAL (/20)			16/20

Figure 5: 48-simplicitePire

Efficacité-Meilleur

1	Problème	Sanction		25-efficacite-meilleur
2	Ne compile pas	note finale divisée par 2	Et hors concours	Compile
3	Ne correspond pas à une classe Exercice (cas des main ou des zip)	note finale divisée par 2		
4	Non respect de la nomenclature précise	-1		
5	Non respect de l'anonymat	-1		
6	Non respect de la consigne sur les méthodes de java.util (pour efficacité)	-1	Et hors concours	
7	Passe tous les tests fournis initialement	18		Passe tous les tests
8	Passe vos tests supplémentaires plus complets	20		Un test ne fonctionne pas (Liste de majuscules vide)
9	Fonctionne mais ne passe pas les tests fournis initialement	10		Passe tous les tests
10	Ne fonctionne pas (retour erroné, ou pas du bon type attendu)	5		
11	TOTAL (/20)			19/20

Figure 6: 25-efficacite-meilleur

1	Problème	Sanction		61-EfficaciteMeilleur
2	Ne compile pas	note finale divisée par 2	Et hors concours	Compile
3	Ne correspond pas à une classe Exercice (cas des main ou des zip)	note finale divisée par 2		
4	Non respect de la nomenclature précise	-1		
5	Non respect de l'anonymat	-1		
6	Non respect de la consigne sur les méthodes de java.util (pour efficacité)	-1	Et hors concours	
7	Passe tous les tests fournis initialement	18		Passe tous les tests
8	Passe vos tests supplémentaires plus complets	20		Passe tous les tests
9	Fonctionne mais ne passe pas les tests fournis initialement	10		Passe tous les tests
10	Ne fonctionne pas (retour erroné, ou pas du bon type attendu)	5		
11	TOTAL (/20)			20/20

Figure 7: 61-EfficaciteMeilleur

Efficacité-Pire

1	Problème	Sanction		9-efficacite-pire
2	Ne compile pas	note finale divisée par 2	Et hors concours	Compile
3	Ne correspond pas à une classe Exercice (cas des main ou des zip)	note finale divisée par 2		
4	Non respect de la nomenclature précise	-1		
5	Non respect de l'anonymat	-1		
6	Non respect de la consigne sur les méthodes de java.util (pour efficacité)	-1	Et hors concours	
7	Passe tous les tests fournis initialement	18		Passe tous les tests
8	Passe vos tests supplémentaires plus complets	20		Passe tous les tests
9	Fonctionne mais ne passe pas les tests fournis initialement	10		Passe tous les tests
10	Ne fonctionne pas (retour erroné, ou pas du bon type attendu)	5		
11	TOTAL (/20)			20 /20

Figure 8: 9-efficacite-pire

1	Problème	Sanction		63-efficacite-pire2
2	Ne compile pas	note finale divisée par 2	Et hors concours	Compile
3	Ne correspond pas à une classe Exercice (cas des main ou des zip)	note finale divisée par 2		
4	Non respect de la nomenclature précise	-1		
5	Non respect de l'anonymat	-1		
6	Non respect de la consigne sur les méthodes de java.util (pour efficacité)	-1	Et hors concours	
7	Passe tous les tests fournis initialement	18		Deux tests ne fonctionnent pas
8	Passe vos tests supplémentaires plus complets	20		Deux tests ne fonctionnent pas
9	Fonctionne mais ne passe pas les tests fournis initialement	10		
10	Ne fonctionne pas (retour erroné, ou pas du bon type attendu)	5		
11	TOTAL (/20)			13/20

Figure 9: 63-efficacite-pire2

Sobriété-Meilleur

1	Problème	Sanction		16-Sobriete-Meilleur
2	Ne compile pas	note finale divisée par 2	Et hors concours	Compile
3	Ne correspond pas à une classe Exercice (cas des main ou des zip)	note finale divisée par 2		
4	Non respect de la nomenclature précise	-1		
5	Non respect de l'anonymat	-1		
6	Non respect de la consigne sur les méthodes de <code>java.util</code> (pour efficacité)	-1	Et hors concours	
7	Passe tous les tests fournis initialement	18		Ne passe aucun test
8	Passe vos tests supplémentaires plus complets	20		
9	Fonctionne mais ne passe pas les tests fournis initialement	10		
10	Ne fonctionne pas (retour <code>errené</code> , ou pas du bon type attendu)	5		
11	TOTAL (/20)			10/20

Figure 10: 16-Sobriete-Meilleur

1	Problème	Sanction		44-sobriete_meilleur
2	Ne compile pas	note finale divisée par 2	Et hors concours	Compile
3	Ne correspond pas à une classe Exercice (cas des main ou des zip)	note finale divisée par 2		
4	Non respect de la nomenclature précise	-1		
5	Non respect de l'anonymat	-1		
6	Non respect de la consigne sur les méthodes de <code>java.util</code> (pour efficacité)	-1	Et hors concours	
7	Passe tous les tests fournis initialement	18		Passe tous les tests
8	Passe vos tests supplémentaires plus complets	20		Passe tous les tests
9	Fonctionne mais ne passe pas les tests fournis initialement	10		Passe tous les tests
10	Ne fonctionne pas (retour <code>errené</code> , ou pas du bon type attendu)	5		
11	TOTAL (/20)			20/20

Figure 11: 44-sobriete_meilleur

Sobriété-Pire

1	Problème	Sanction		41-sobriete-pire
2	Ne compile pas	note finale divisée par 2	Et hors concours	Compile (le code commenté seulement?)
3	Ne correspond pas à une classe Exercice (cas des main ou des zip)	note finale divisée par 2		
4	Non respect de la nomenclature précise	-1		
5	Non respect de l'anonymat	-1		-1
6	Non respect de la consigne sur les méthodes de <code>java.util</code> (pour efficacité)	-1	Et hors concours	
7	Passe tous les tests fournis initialement	18		
8	Passe vos tests supplémentaires plus complets	20		Un test ne fonctionne pas (Liste de majuscules vide)
9	Fonctionne mais ne passe pas les tests fournis initialement	10		
10	Ne fonctionne pas (retour <code>eronné</code> , ou pas du bon type attendu)	5		
11	TOTAL (/20)			18/20

Figure 12: 41-sobriete-pire

1	Problème	Sanction		43-sobriete-pire
2	Ne compile pas	note finale divisée par 2	Et hors concours	Compile
3	Ne correspond pas à une classe Exercice (cas des main ou des zip)	note finale divisée par 2		
4	Non respect de la nomenclature précise	-1		
5	Non respect de l'anonymat	-1		
6	Non respect de la consigne sur les méthodes de <code>java.util</code> (pour efficacité)	-1	Et hors concours	
7	Passe tous les tests fournis initialement	18		Passe tous les tests
8	Passe vos tests supplémentaires plus complets	20		Passe tous les tests
9	Fonctionne mais ne passe pas les tests fournis initialement	10		Passe tous les tests
10	Ne fonctionne pas (retour <code>eronné</code> , ou pas du bon type attendu)	5		
11	TOTAL (/20)			20/20

Figure 13: 43-sobriete-pire

Lisibilité du code

Simplicité-Meilleur

30-Main(Simplicité meilleur)

- L'algorithme ne présente aucun commentaire/javadoc
- Il n'y a aucun saut de ligne dans le code
- Cependant les noms des variables sont assez cohérents et rendent l'algorithme assez lisible pour une personne qualifiée dans le langage Java.

J'en déduis que l'algorithme est plutôt simple à lire en terme de code mais manque de documentation, ce qui le rend moins lisible pour un débutant ou collaborateur qui ne voudrait pas avoir à comprendre lui-même le code.

37-Simplicité-meilleur

- L'algorithme ne présente aucun commentaire/javadoc mais contrairement au premier le codeur l'a rendu plus lisible grâce à des sauts de lignes.
- Les noms des variables sont bien transparentes et le code est bien indenté
- Création d'une autre classe au sein de la classe rend le code moins « simple ».
- Utilisation de beaucoup de méthodes « complexes » telles que « `getOrDefault` », « `split` », « `sort` »...

Selon moi, cet algorithme est partiellement lisible, voire difficile à lire pour une personne non qualifiée à cause de la classe `CompareurOrdreFixe implements Comparator<String>`.

58-simplicite-meilleure

- L'algorithme ne présente aucun commentaire/javadoc.
- Très bonne indentation du code avec en plus des sauts de lignes qui permettent d'aérer le code.
- Les conditions « if » sont très compréhensibles et accessibles à tous

Cet algorithme est un très bon algorithme en termes de simplicité et ne présente pas de difficulté à la lecture.

Simplicité-Pire

12-Simplicité pire

- L'algorithme ne présente aucun commentaire/javadoc.
- les noms de variables restent compréhensibles
- en revanche le code est bien difficile à comprendre à cause de toutes les conditions et en particulier de la fonction lambda dans la variable « comparator ».

Selon moi, cet algorithme est a bien sa place dans la catégorie de pire simplicité même si il n'est pas illisible pour autant.

48-simplicitePire

- Le code ne présente pas de javadoc mais quasiment toutes les lignes de code sont commentées pour expliquer ce que fait l'algo (contradiction avec la catégorie pire simplicité?)
- nom des variables assez compréhensifs même si ce n'est pas optimal
- Beaucoup de conditions « if » qui rendent le code lourd à comprendre
- Ajout d'une méthode « compare » personnalisée, je n'en comprend pas l'utilité

En conclusion, ce code est assez incohérent mais le codeur a fait l'effort de commenter, code assez surchargé d'infos entre toutes les conditions, les boucles, les tests et commentaires.

Efficacité-Meilleur

25-efficacite-meilleur

- Utilisation de plusieurs méthodes auxiliaires afin de simplifier le code
- De très bonnes javadocs sur les 5 méthodes auxiliaires développées qui permettent de se passer de commentaires au sein du code
- Code bien indenté et des noms de variables clairs.

L'algorithme est bien lisible grâce aux bons commentaires de l'auteur, de plus, il n'utilise pas de méthodes ou de manière de coder difficile à la compréhension.

61-EfficaciteMeilleur

- Le code ne présente pas de javadoc mais est bien commenté (parfois trop de commentaires non nécessaires cependant)
- Le code est long (70 lignes) mais dispose de sauts de lignes et est très lisible et compréhensible.

Pour résumer, ce code, grâce aux nombreux commentaires pertinents du développeur, est très lisible pour des développeurs de tous niveaux, dommage qu'il ne dispose pas de javadoc.

Efficacité-Pire

9-efficacite-pire

- Le code ne dispose pas de javadoc mais est bien commenté par le développeur
- le code manque d'aération, il aurait pu être plus lisible avec des sauts de lignes et une séparation plus nette des blocs
- les noms des variables sont assez explicites

L'algorithme est plutôt lisible.

63-efficacite-pire2

- Le code est commenté mais les méthodes n'ont pas de javadoc
- Parcours de boucles à l'envers puis boucle imbriquée à l'endroit rend le code assez obscur
- Ajout d'une méthode HungerGames ? Peut être enlevée mais respecte le fait que le code est inefficace.

Ce code est difficile à lire malgré quelques commentaires.

Sobriété-Meilleur

16-Sobriete-Meilleur

- l'algorithme est pourvu de commentaires placés judicieusement juste avant chaque boucle afin d'expliquer leur rôle.
- aucun commentaire ou java doc n'est fournis
- les noms des variables sont descriptifs et significatifs.

L'algorithme est très lisible.

44-sobriete meilleur

- Javadoc absente et faibles commentaires au sein du code.
- Les noms des variables sont bons et cohérents
- Utilisation de boucles et conditions biens compréhensibles, dommage que le code ne soit pas assez commenté pour guider un peu plus le lecteur.

Pour conclure, cet algorithme est assez lisible dans le code en général car il n'utilise pas de modules ou de techniques difficiles à comprendre mais manque toutefois de commentaires/javadoc.

Sobriété-Pire

41-sobriete-pire

- Une partie du code est placée en commentaire alors que c'est la partie qui compile et fait fonctionner l'algorithme ?
- Pas de javadoc sur le code non-commenté mais présente sur le code commenté (laissé exprès par le codeur?)
- Le code qui compile n'est presque pas commenté par le développeur et utilise des noms de variables pas assez transparentes.

Cet algorithme est donc difficilement lisible mais il semblerait que cela soit fait de manière volontaire par son développeur.

43-sobriete-pire

- Pas de javadoc ni de commentaires
- Le code est bien aéré cependant avec des sauts de lignes et une bonne indentation
- Les noms de variables sont explicites et permettent une meilleure compréhension
- Les boucles et conditions sont facilement compréhensibles pour un lecteur extérieur.

Ce dernier algorithme est donc facilement lisible, des commentaires auraient pu être un plus.

Qualité du code

Pour tester la qualité des codes, nous utilisons l'outil « [Codacy](#) ».

Simplicité-Meilleur

30-Main(Simplicité meilleur)

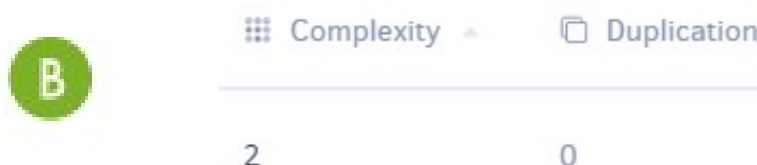
Classement « C » par Codacy :

- Eviter les `java.IO.*` lors des imports : Using the `'.*'` form of import should be avoided
- Eviter d'avoir plus de 80 caractères par ligne de code (répété plusieurs fois)




37-Simplicité-meilleur

Classement « B » donné par Codacy : mêmes remarques que pour l'algorithme n°30



58-simplicite-meilleure

Classement « B » : certaines conditions « if » pourraient être combinées au lieu d'utiliser des boucles if imbriquées. These nested if statements could be combined

	Complexity ▲	Duplication
	14	0 clones

Simplicité-Pire

12-Simplicité pire : PAS D'EVALUATION CODACY ?

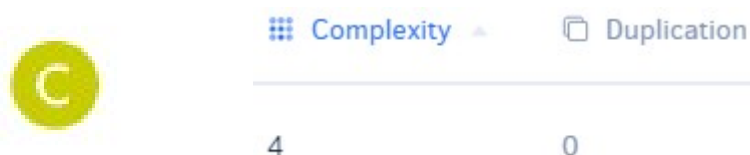
48-simplicitéPire :PAS D'EVALUATION CODACY ?

Efficacité-Meilleur

25-efficacité-meilleur

Classement « C » :

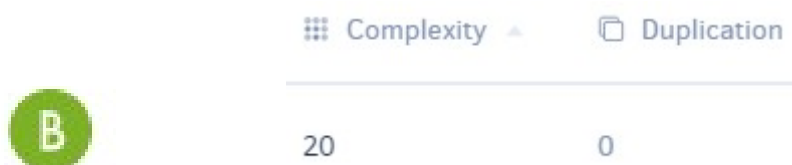
- Déclaration de plusieurs variables sur la même ligne (répété)
- Lignes de plus de 80 caractères (répété)



61-EfficacitéMeilleur

Classement « B » :

- Complexité trop élevée et plusieurs lignes de plus de 80 caractères



Efficacité-Pire

9-efficacite-pire

Classement « B » :

- Plusieurs lignes de plus de 80 caractères



Complexity ▲

Duplication

8

0

63-efficacite-pire2

Classement « C » :

- Beaucoup de lignes de plus de 80 caractères
- Import non utilisé



Complexity ▲

Duplication

7

0

Sobriété-Meilleur

16-Sobriete-Meilleur

Classement « A »

A	Complexity ▲	Duplication
	10	0

44-sobriete meilleur

Classement « B »

- Oubli du package, import avec « * » : Using the '.*' form of import should be avoided

B	Complexity ▲	Duplication
	9	0

Sobriété-Pire

41-sobriete-pire

Classement « B » :

- Beaucoup de lignes de plus de 80 caractères



Complexity ▲

Duplication

8

0

43-sobriete-pire

Classement « C » :

- Utilisation de == au lieu de « .equals » pour comparer des String
- code pas assez aéré :

'import' should be separated from previous line. / 'CLASS_DEF' should be separated from previous line.



Complexity ▲

Duplication

9

0

Efficacité / Temps d'exécution

Pour tester l'efficacité des codes, nous allons rajouter dans la classe main(), celle qui lance les différents algorithmes, les lignes suivantes :

```
double startTime = System.currentTimeMillis();  
(...)  
double endTime = System.currentTimeMillis();  
System.out.println((endTime-startTime)/1000+"s");
```

Ces lignes nous permettent d'obtenir le temps de calcul d'un algorithme, on pourra donc ajouter un gros texte et voir si le temps augmente de beaucoup ou non.

Nous utiliserons plusieurs taille de texte : celui de base à 5 mots, un texte à 100 mots et un à 1000 mots.

5 mots : "exemple de texte a classer"

100 mots : "Lete etait enfin arrive apportant avec lui le doux parfum des fleurs et le chant melodieux des oiseaux Les rayons du soleil caressaient delicatement la peau rechauffant les coeurs et chassant les soucis Les journees se remplissaient d'aventures et de rires tandis que les soirees invitaient a la detente et a la contemplation Les vacances tant attendues permettaient de sevader du quotidien et de decouvrir de nouveaux horizons Les plages accueillaien les baigneurs avides de fraicheur tandis que les montagnes offraient des panoramas a couper le souffle Cetait le moment parfait pour se ressourcer et creer des souvenirs inoubliables"

1000 mots : "Le soleil se levait lentement a l'horizon baignant le paysage d'une douce lueur doree Les oiseaux commencaient a chanter leur symphonie matinale annoncant ainsi le debut d'une nouvelle journee Dans cette petite ville tranquille la vie s' eveillait peu a peu Les rues se remplissaient de passants presses se rendant a leurs occupations quotidiennes Les commerçants ouvraient leurs boutiques et disposaient leurs produits avec soin Les étales se remplissaient de couleurs et d'odeurs alléchantes Les clients déambulaient dans les rues s'arrêtant çà et là pour admirer les étalages et effectuer leurs achats Au parc de nombreux enfants s'amusaient dans l'aire de jeux Les rires et les cris joyeux résonnaient dans l'air Les parents les observaient avec tendresse tout en échangeant des sourires complices Les étudiants se pressaient vers l'université pour assister à leurs cours Les salles de classe se remplissaient d'une foule studieuse prête à apprendre Les professeurs partageaient leur savoir avec passion et les étudiants prenaient des notes attentivement Dans les cafés et les restaurants les gens prenaient leur petit-déjeuner ou dégustaient tranquillement leur repas Des conversations animées se mêlaient aux arômes du café fraîchement moulu et des plats savoureux Certains se rendaient au travail en bus en train ou à vélo Les rues se remplissaient de voitures qui avançaient lentement dans le flot de la circulation Les klaxons retentissaient de temps en temps rompant le calme matinal Les parcs et les jardins étaient des havres de paix où les gens venaient se détendre et profiter de la nature Certains faisaient leur jogging matinal d'autres se promenaient en respirant l'air frais et en admirant les fleurs colorées qui s'épanouissaient À la bibliothèque les étudiants et les amateurs de lecture se plongeaient dans les pages de livres et de revues Le silence régnait dans cet espace dédié à la connaissance où chacun pouvait s'évader à travers les mots et les histoires Le soir venu les restaurants et les bars s'animaient de conversations animées et de rires joyeux Les amis se retrouvaient pour partager un repas ou prendre un verre ensemble dans une ambiance conviviale et chaleureuse Les étoiles scintillaient dans le ciel nocturne offrant un spectacle enchanteur Les couples se promenaient main dans la main profitant de la quiétude de la nuit Les lumières des lampadaires donnaient une ambiance romantique à la ville endormie Puis la nuit s'installait peu à peu enveloppant la ville d'un voile sombre Les rues se vidaient et le calme revenait Les habitants s'endormaient paisiblement se préparant ainsi à une nouvelle journée qui les attendait au réveil"

Tableau de la rapidité des différents algorithmes

Algorithmes	X5 mots	X100 mots	X1000 mots	Unité : secondes
9- efficacite-pire	0,001	0,004	0,009	
12- simplicite-pire	0,007	0,009	0,015	
16- Sobriete-Meilleur	0,0 ?	0,002	0,003	
25- efficacite-meilleur	0,001	0,006	0,011	
30-Main	0,009	0,12	0,55	
37- simplicite-meilleur	0,002	0,002	0,008	
41- sobriete-pire	0,001	0,003	0,018	
43- sobriete-pire	0,005	0,008	0,011	
44- sobriete_ meilleur	0,001	0,002	0,006	
48- simplicitePire	0,001	0,005	0,009	
58- simplicite-meilleure	0,004	0,008	0,012	
61- EfficaciteMeilleur	0,001	0,004	0,016	
63- efficacite-pire2	0,5 à 1s	7,5	35	(en spammant la touche entrée)

Sobriété numérique

Cette étape consiste à donner une mesure de la consommation en ressources d'un algorithme, afin de réaliser cette mesure, nous devions utiliser l'outil « *JoularJX* » mais à cause de problèmes avec l'installation de l'outil, il m'a été impossible de réaliser ces mesures, je m'en excuse.

Classement par catégorie

Simplicité-Meilleur

1 – 58-simplicite-meilleure : Cet algorithme est très lisible par tous et répond très bien aux attentes du sujet, il passe tous les tests fournis initialement et aussi les tests additionnels

2 – 37-Simplicité-meilleur : L'algorithme n°37 n'est pas simple à la lecture, de plus il ne passe qu'un tiers des tests fournis initialement ainsi que ceux additionnels. Je décide de le mettre en deuxième place car il est moins fonctionnel que le 58 mais reste supérieur au 30.

3 - 30-Main(Simplicité meilleur) : A la dernière place de cette catégorie « Simplicité-meilleur » on retrouve l'algorithme n°30 : Le sujet a certainement mal été compris par le développeur : j'ai dû changer le type d'un paramètre de la méthode « solution » pour rendre ce programme possible, de plus, il ne respecte pas la nomenclature et ne passe qu'un tiers de tous les tests également. Il est donc logique de le placer à la dernière place de ce classement.

Simplicité-Pire

1 - 12-Simplicité pire : Cet algorithme n'est pas le plus lisible selon moi mais il semble qu'il soit le plus efficace de cette catégorie car il passe tous les tests fournis, qu'ils soient bonus (ceux que j'ai créés) ou les tests fournis de base. Il me semble donc logique qu'un algorithme, même si il est difficile à la lecture, soit classé premier car il répond tout de même aux besoins de ce sujet.

2 – 48-simplicitePire : L'algorithme 48 est un peu plus compréhensible que le précédent mais ne respecte pas bien la consigne qui est de passer tous les tests pour trier la chaîne de caractères avec un certain ordre donné. Selon moi, il est cohérent de le placer en seconde position car il est moins efficace que l'algo 12.

Efficacité-Meilleur

1 – 61-EfficaciteMeilleur : Je décide de placer l’algo n°61 à la première place de cette catégorie car malgré le fait que sa complexité soit trop élevée, il respecte tous les tests et évite donc de devoir passer derrière l’algorithme en cas d’erreur ce qui fait au final gagner plus de temps que ce qu’il en perd à cause de sa complexité.

2 – 25-efficacite-meilleur : Comme énoncé ci-dessus, cet algorithme est certes plus efficace en termes de complexité mais il ne réussit pas tous les tests qui lui sont donnés, c’est pour cela que j’ai décidé de le placer à la seconde position de cette catégorie.

Efficacité-Pire

1 – 9-efficacite-pire : L’algo n°9 passe avec succès tous les tests donnés ainsi que ceux « bonus », il est donc cohérent de le placer à la première place de cette catégorie même si les temps et la complexité de l’algo sont assez contradictoires avec la catégorie dans laquelle il est placé.

2 – 63-efficacite-pire2 : Le format de cet algorithme est assez original et mérite une mention honorable pour son imagination, cependant il ne réussit pas à passer tous les tests, il est donc dommage pour son développeur qu’il ne fonctionne pas « parfaitement » car il respectait très bien cette catégorie d’« efficacité-pire ».

Sobriété-Meilleur

1 – 44-sobriete_meilleur : Cet algo est l’un des plus simples à comprendre parmi tous ceux vus auparavant. De plus, sa complexité est assez faibles quand on la compare avec les autres algorithmes et en particulier l’algo n°16. Ce code passe tous les tests et affiche la meilleure rapidité parmi tous les algos (cf Tableau de la rapidité des différents algorithmes p24).

2 – 16-Sobriete-Meilleur : Malgré le fait que le code soit classé « A » par Codacy, il ne passe aucun des tests, il est donc évident de le placer à la seconde place du classement de cette catégorie.

Sobriété-Pire

1 – 43-sobriete-pire : L’algo n°43 passe avec succès tous les tests en plus de n’être pas compliqué à comprendre à la lecture, c’est pour ces raisons que j’ai décidé de le placer à la première place de cette catégorie.

2 – 41-sobriete-pire : Ce deuxième algorithme de la catégorie « sobriété-pire » est assez incohérent et bien plus compliqué à comprendre que le n°43, en plus de cela, il ne passe pas tous les tests fournis et il est donc moins fiable que son concurrent.

Classement toutes catégories confondues

En plus de faire le classement des algorithmes par catégorie, je décide de faire un classement qui ne prend pas en compte les catégories de chacun des algos. Voici mon classement personnel :

3 – A la troisième position de ce classement nous retrouvons l’algorithme n°9 faisant partie de la catégorie « efficacité-pire », cet algorithme a passé avec succès tous les tests et a l’une des meilleures complexité de tous les algos à évaluer. Cependant, je décide de le placer ici car il ne respecte pas totalement la catégorie « efficacité-pire » selon moi car le but de cette catégorie était d’avoir la pire complexité.

2 – A la deuxième place se situe l’algorithme n°43 de la catégorie « Sobriété-pire ». En effet, cet algo mérite très bien sa place de second car il passe tous les tests avec succès et affiche une bonne complexité équivalente à son prédécesseur.

1 – Cette première place est occupée par l’algo numéro 44 de la catégorie « Sobriété-meilleur », j’ai décidé de placer cet algorithme à la première place car il est celui qui respecte le mieux sa catégorie car il est très simple à comprendre et pour autant assure l’une des meilleures complexité.

Mentions honorables aux algorithmes n°58 et 61 qui passaient également tous les tests fournis avec succès mais qui malheureusement affichaient une complexité trop élevée comparée à leurs « rivaux ».