Sommaire

Objectif

Ce document à pour objectif de rendre compte de l'analyse de 11 algorithmes de différents langages en fonction de trois catégories et de critères spécifiques.
Ces algorithmes ont tous le même objectif, qui est de transformer une chaine de charactère donnée en enlevant tous les espaces simples compris dedans.

Voici la grille de notation utilisée :

Cas	Points
Ne compile/s'exécute pas	0
Ne passe pas les tests fournis ou a raté certains cas	1
Meilleur algo de la catégorie	5
2nd algo	4
3e algo	3
4e algo	2

En cas d'ex-aequo, la même note peut être attribuée.

Paramètres de tests

Pré-requis

- Compilateur Java, C ou python

Reproductibilité

Java :

1. Ouvrir Eclipse
2. Importer les fichiers d'algorithmes dans des packages correspondants
3. Mettre en place un main avec les imports des autres fichiers
4. Appliquer les tests nécessaires

C :

1. Consulter le site « replit.com »
2. Ouvrir un « Repl » (repository) avec le langage correspondant
3. Ajouter les fichiers à tester
4. Mettre en place les éléments pour lancer la méthode à tester
5. Appliquer les tests nécessaires

Python :

1. Ouvrir le site « JupyterHub »
2. Importer les fichiers d'algorithmes
3. Mettre en place un main pour appeler les méthodes
4. Appliquer les tests nécessaires

Références

Eclipse

https://replit.com

https://jupyterhub.iut-blagnac.fr/

https://www.codacy.com pour les mesures de qualité

La méthode `System.nanoTime()` pour les mesures de temps pour le Java

La méthode clock() de la librairie <time.h> pour les mesures de temps pour le C

La méthode time() de la librairie time pour les mesures de temps pour le Python

Catégorie « Efficacité »

Efficacité-10.java

Lisibilité du code

Aucun commentaire, laissant le testeur comprendre par lui-même, mais n'utilise pas de méthodes ni de librairies complexes.

Qualité du code

Le site Codacy donne un score de B, avec comme problèmes :

-Le nom de classe « Erase1 » ne suit pas les conventions d'écriture

-Il n'y a pas de package

Efficacité

Formule de complexité : 5 + n(2a + b + 3) + c

Avec : n nb de chars

a nb d'espaces

b nb de char après un espace

c booleen vérifiant si le dernier char est un espace

Ordre de grandeur : O(n)

Sobriété numérique

Le String fourni

1 char [taille du String fourni]

2 String

2 int

Temps d'exécution

Les temps sont des moyennes sur 5 essais identiques.

«» 701 280 nano secondes

«06 07 65 19 70 »	808 540 nano secondes
«666, the number of the beast»	890 820 nano secondes
«Cou cou J M B»	896 760 nano secondes

Efficacité-43.java

Lisibilité du code

Quelques commentaires clarifiant des détails importants pour la compréhension, tel que le lien entre '32' et le caractère ASCII « espace ».

Qualité du code

Le site Codacy donne un score de B, avec comme problème :

-Le nom de classe « Erase2 » ne suit pas les conventions d'écriture

Efficacité

Formule de complexité : 1 + n(a(b + c) + d)

Avec : n nb de chars

a nb d'espaces

b nb d'espaces après un espace

c nb d'espaces avant un espace

d nb de char différents d'espace

Ordre de grandeur : O(n)

Sobriété numérique

Le String fourni

1 String

1 int

Temps d'exécution

Les temps sont des moyennes sur 5 essais identiques.

«»	1 004 980 nano secondes
«06 07 65 19 70 »	914 140 nano secondes
«666, the number of the beast»	821 240 nano secondes
«Cou cou J M B»	901 760 nano secondes

Efficacité-75.java

Lisibilité du code

Plein de commentaires explicatifs, rendant la compréhension facile.

Qualité du code

Le site Codacy donne un score de A, avec comme problème :

-Le nom de classe « Eraser » ne suit pas les conventions d'écriture

Efficacité

Formule de complexité : 2 + n(a(1 + 2b + 2c) + d)

Avec : n nb de chars

a nb d'espaces

b nb de char après un espace

c booleen vérifiant si le dernier char est un espace

Ordre de grandeur : O(n)

Sobriété numérique

Le String fourni

1 String

3 int

Temps d'exécution

Les temps sont des moyennes sur 5 essais identiques.

«»	770 180 nano secondes
«06 07 65 19 70 »	930 500 nano secondes
«666, the number of the beast»	850 300 nano secondes
«Cou cou J M B»	865 760 nano secondes

Efficacité-121.c

Lisibilité du code

Aucun commentaire, laissant le testeur comprendre par lui-même, mais n'utilise pas de méthodes ni de librairies complexes.

Qualité du code

Le site Codacy donne un score de B, avec comme problème :

-Ne prends pas en compte les strings ne terminant pas par `'\0'`

Efficacité

Formule de complexité : n(a(2) + 1) + 1

Avec : n nb de chars

a nb de chars différents d'un espace simple(avec un espace à côté)

Ordre de grandeur : O(n)

Sobriété numérique

Le char* fourni

1 char* (taille du char* fourni)

2 int

1 free(char*)

Temps d'exécution

Les temps sont des moyennes sur 5 essais identiques.

«»	28 000 nano secondes
«06 07 65 19 70 »	31 800 nano secondes
«666, the number of the beast»	25 800 nano secondes
«Cou cou J M B»	35 200 nano secondes

Résultat efficacité

Le code 10.java à une vitesse moyenne et à une formule de complexité correcte.

Le code 43.java à une vitesse plus lente et à une bonne formule de complexité.

Le code 75.java à une vitesse moyenne et à une mauvaise formule de complexité.

Le code 121.C à une vitesse très élevé et possède une très bonne formule de complexité.

Catégorie « Simplicité »

Comme les algorithmes comportent très peu de commentaires, il est plus difficile de comprendre les différentes étapes des codes.

Simplicité-14.java

Lisibilité du code

Le code comporte des if else bien indentés, aidant à la compréhension générale.

Qualité du code

Le site Codacy donne un score de B, avec comme problème :

-Le nom de classe « Eraser » ne suit pas les conventions d'écriture

Efficacité

Formule de complexité : 2 + n(a + b + c + 1)

Avec : n nb de chars

a nb de char différent d'espace

b nb de char avant un espace

c nb de char après un espace

Ordre de grandeur : O(n)

Sobriété numérique

Le String fourni

1 StringBuilder

1 int

Temps d'exécution

Les temps sont des moyennes sur 5 essais identiques.

«»	787 160 nano secondes
«06 07 65 19 70 »	752 360 nano secondes
«666, the number of the beast»	998 800 nano secondes
«Cou cou J M B»	1 041 180 nano secondes

Simplicité-39.java

Lisibilité du code

Le code comporte beaucoup de if indentés les uns dans les autres avec peu de else, rendant la compréhension moins agréable et plus difficile.

Qualité du code

Le site Codacy donne un score de A, avec comme problème :

-Le nom de classe « solutionCategorie1 » ne suit pas les conventions d'écriture

Efficacité

Formule de complexité : a(2 + n(1 + b(c(1 +b(2d + 1))) + e))

Avec : a booleen vérifiant si le String fourni est vide

n nb de chars

b nb d'espaces

c i!=taille-1

d message.charAt(i)==32 && i!=taille-1

e nb de char différent d'espaces

Ordre de grandeur : O(n)

Sobriété numérique

Le String fourni

1 String

4 int

Temps d'exécution

Les temps sont des moyennes sur 5 essais identiques.

«»	714 080 nano secondes
«06 07 65 19 70 »	842 220 nano secondes
«666, the number of the beast»	854 360 nano secondes
«Cou cou J M B»	992 600 nano secondes

Simplicité-55.java

Lisibilité du code

Le code comporte des if indentés les uns dans les autres, mais avec quelques else pour compenser, rendant la compréhension un peu plus longue.

Qualité du code

Le site Codacy donne un score de A, avec comme problème :

-Le nom de classe « Eraser » ne suit pas les conventions d'écriture

Efficacité

Formule de complexité : $4 + a + 1 + n(b(c(2d) + 2e + 1) + f) + g$

Avec : a str.indexOf(' ') < 0

n nb de chars

b str.charAt(i) == 32

c i == str.length()-1

d !wasSpace

e !wasSpace && str.charAt(i+1) != 32

f !b

g end < str.length()-1

Ordre de grandeur : O(n)

Sobriété numérique

Le String fourni

1 String

4 int

Temps d'exécution

Les temps sont des moyennes sur 5 essais identiques.

«»	705 600 nano secondes
«06 07 65 19 70 »	1 291 320 nano secondes
«666, the number of the beast»	1 108 320 nano secondes
«Cou cou J M B»	1 390 100 nano secondes

Simplicité-169.py

Lisibilité du code

Le code est concis, clair et ne comporte qu'une boucle munie d'un seul if, très lisible et donc compréhensible.

Qualité du code

Le site Codacy donne un score de B, avec comme problème :

-1 ligne vide nécessaire entre le sommaire/résumé et la description

Efficacité

Formule de complexité : 2 + n(a)

Avec : n nb de chars

a nb d'espaces simples

Ordre de grandeur : O(n)

Sobriété numérique

Le String fourni

1 list de String

1 int

1 char

Temps d'exécution

Les temps sont des moyennes sur 5 essais identiques.

«»	221 395 nano secondes
«06 07 65 19 70 »	287 293 nano secondes

«666, the number of the beast» 348 615 nano secondes

«Cou cou J M B» 336 694 nano secondes

Résultat simplicité

Le code 14.java est correctement lisible et compréhensible.

Le code 39.java est difficilement lisible car il a beaucoup de if indentés les uns dans les autres ainsi que peu de else pour éviter cette surcharge.

Le code 55.java est confus avec assez de if et de else.

Le code 169.py est très lisible et compréhensible.

Catégorie « Sobriété »

Sobriété- 68.java

Lisibilité du code

Le code n'est pas commenté et utilise des méthodes que je ne connais pas, il n'est donc pas très lisible.

Qualité du code

Le site Codacy donne un score de A, avec comme problème :

-Le nom de classe « Eraser4 » ne suit pas les conventions d'écriture

Efficacité

Dut à la mauvaise lisibilité du code et l'utilisation de méthodes inconnues, je ne suis pas capable de calculer la formule de complexité de cet algorithme.

Sobriété numérique

Le String fourni

2 ArrayList de String

1 String

3 int

Temps d'exécution

Les temps sont des moyennes sur 5 essais identiques.

«» 874 780 nano secondes

«06 07 65 19 70 »	1 201 760 nano secondes
«666, the number of the beast»	1 080 780 nano secondes
«Cou cou J M B»	1 316 400 nano secondes

Sobriété-76.c

Lisibilité du code

Le code n'est pas commenté et utilise des méthodes que je ne connais pas, il n'est donc pas très lisible.

Qualité du code

Le site Codacy donne un score de A, avec comme problème :

-La variable compteur est assignée une valeur qui n'est jamais utilisé

Efficacité

Dut à la mauvaise lisibilité du code et l'utilisation de méthodes inconnues, je ne suis pas capable de calculer la formule de complexité de cet algorithme.

Sobriété numérique

Le char* fourni

1 char*

4 int

Temps d'exécution

Les temps sont des moyennes sur 5 essais identiques.

«»	13 000 nano secondes
«06 07 65 19 70 »	22 600 nano secondes
«666, the number of the beast»	23 600 nano secondes
«Cou cou J M B»	16 400 nano secondes

Sobriété-113.java

Lisibilité du code

Le code n'est pas commenté et utilise des méthodes que je ne connais pas, il n'est donc pas très lisible.

Qualité du code

Le site Codacy donne un score de A, avec comme problème :

-Le nom de classe « Eraser » ne suit pas les conventions d'écriture

Efficacité

Dut à la mauvaise lisibilité du code et l'utilisation de méthodes inconnues, je ne suis pas capable de calculer la formule de complexité de cet algorithme.

Sobriété numérique

Le String fourni

1 StringBuilder

3 int

Temps d'exécution

Les temps sont des moyennes sur 5 essais identiques.

«»	971 200 nano secondes
«06 07 65 19 70 »	782 820 nano secondes
«666, the number of the beast»	738 460 nano secondes
«Cou cou J M B»	799 160 nano secondes

Résultat sobriété

Le code 68.java est assez lent et utilise beaucoup de variables lourdes.

Le code 76.c est très rapide et utilise pas de variables lourdes.

Le code 113.java à une vitesse moyenne et utilise une variable lourde.

Classements

Catégorie	Algorithme	Position	Note
Efficacité	10.java	3	3
	43.java	2	4
	75.java	4	2
	121.c	1	5
Simplicité	14.java	2	4
	39.java	4	2
	55.java	3	3
	169.py	1	5
Sobriété	68.java	3	3
	76.c	1	5
	113.java	2	4