Rubén
Longèque
Groupe 4B

SAÉ 2.02

Rapport d’évaluation des algorithmes

Sommaire

Préambule :

Avant d'effectuer les différentes analyses de code, je tiens à préciser la grille de notation des algorithmes :

Notation	Explication
0 pt	Ne compile (ou ne s'exécute) même pas
1 pt	Ne passe pas les tests fournis dans le dépôt ou a raté un cas (donc ne marche pas dans tous les cas)
2 pts	4ème algo
3 pts	3ème algo
4 pts	2nd algo
5 pts	Meilleur algo de la catégorie

Catégorie Simplicité :

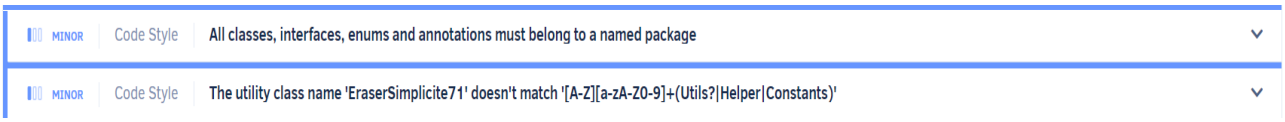
Fichier simplicite-71.java :

Lisibilité du code :

Le code est lisible, on comprend parfaitement ce qu'il fait, malgré l'absence de commentaire ou de javadoc, il est à la portée de beaucoup de monde dans sa compréhension.

Qualité du code :

Pour m'aider dans cette analyse, je vais utiliser l'outil open source de mesure de qualité de code Codacy.



On peut voir que hormis des erreurs de nomination (qui sont de mon ressort), le code ne possède pas d'erreur qui aggraverait sa qualité. Donc le code est bon pour cela.

Efficacité/Compilation du code:

Cependant, bien que le code ne soit pas mal « écrit », celui-ci ne passe pas tous les tests à effectuer.

```
for(int i=0 ; i<str.length(); i++) {
    if(str.charAt(i) != ' ' || (str.charAt(i+1)==' '|| str.charAt(i-1)==' ')){
        res+= str.charAt(i);

    }
```

En effet, la gestion des conditions (voir ci-dessus) pose problème quand au fonctionnement du code. Car celui-ci, en voulant vérifier le caractère précédent ou suivant, risque très fortement de sortir de la limite de taille de la chaîne de caractère. Ce qui est problématique.

En ce qui concerne l'efficacité, dans une classe de test, j'ai pu obtenir le nombre de tour qu'il effectue pour un texte de 100 000 caractères, il effectue 98 999 tours de boucle. Ce qui correspond quasiment à un complexité algorithmique de O(n).

Sobriété numérique :

En utilisant l'outil joular, on peut obtenir la consommation électrique de notre programme en joules. Ainsi dès lors que le programme est terminé nous pouvons obtenir sa consommation.

Si on utilise joular pendant 1 minute, la consommation de ce programme est de 63,12 joules.

```
C:\Users\ruben\Desktop\saeAlgo\saeAlgoTest\src>Program consumed 63,12 joules
```

Temps d'exécution :

En ce qui concerne le temps d'exécution, dans une classe de test, j'ai pu obtenir le temps moyen en ms qui enlève les espaces non nécessaires dans un texte de 100 000 caractères, il effectue la manipulation 100 fois. Ce qui donne un temps moyen de 881,99 ms.

```
Temps que ça prend : 858.0 ms
Temps que ça prend : 857.0 ms
Temps que ça prend : 874.0 ms
881.99 ms
```

Fichier simplicite-85.java :

Lisibilité du code :

Bien que le code soit imposant, la présence de commentaires permet amplement d'améliorer sa lisibilité, notamment en plaçant les commentaires avant chaque « partie » du code permettant d'expliquer ce que le code fait à cet endroit là.

Qualité du code :

En utilisant Codacy, j'ai pu déceler quelques problèmes qui nuisent à la qualité du code.

MEDIUM | Error Prone | The method 'erase(String)' has an NPath complexity of 480, current threshold is 200

```
4    public static String erase(String chaine) {
5        String chaineSansEspace = "";
6
7            //Cas si la saisie utilisateur se compose seulement d'espace ou est vide !
8            if (chaine.length() < 2)
9                return "";
```

Tout d'abord le code aurait un souci de complexité, on analysera un peu plus précisément cela après, celui-ci serait plus conséquent que la normal sur ce point là.

Enfin la non utilisation de la méthode *.equals()* lors d'une comparaison entre deux chaînes, ce qui va nuire à la qualité du code

MEDIUM Error Prone **Use equals() to compare object references.**

MEDIUM Error Prone **Use equals() to compare strings instead of '==' or '!='**

```
10              if (chaine.trim() == "" && chaine.charAt(0) == ' ' && chaine.charAt(1) == ' ') {
11                      return chaine;
12              }
13
14              //Compte le nombre d'espace(s) à la fin de la chaine de caractère
15              int longueur = 0;
16              int c = chaine.length();
17              while (chaine.charAt(c-1) == ' ') {
18                      c--;
19                      longueur++;
20              }
21
22              //Remplace les espaces à la fin de la chaine
```

Puis, bien que ce soit un erreur de style, on remarque que le développeur n'a pas réutilisé l'objet en paramètre pour être celui que l'on renvoie, mais un autre objet crée au sein de la méthode.

MEDIUM Code Style **Avoid reassigning parameters such as 'chaine'**

```
23                  chaine = chaine.replaceFirst("\\s++$", "");
24
25                  //Ne remplace que les blancs simples
26                  int j = 0;
27                  for (int i = 0; i < chaine.length(); i++) {
28                          if (i != chaine.length()) {
29                              if (chaine.charAt(i) != ' ') {
30                                  chaineSansEspace += chaine.charAt(i);
31                              } else if (chaine.charAt(i) == ' ' && chaine.charAt(i + 1) == ' ' && j == 0) {
32                                  chaineSansEspace += chaine.charAt(i);
33                                  chaineSansEspace += chaine.charAt(i+1);
34                                  j++;
35                              } else if (chaine.charAt(i) == ' ' && chaine.charAt(i + 1) == ' '){
36                                  chaineSansEspace += chaine.charAt(i+1);
37                              }
38                          } else {
39                                  if (chaine.charAt(i) != ' ') {
40                                  chaineSansEspace += chaine.charAt(i);
41                          }
42                  }
43              }
44                  // Rajoute les espaces (s'il y en avait) à la fin de la chaine de caractère modifiée
45                  if (longueur >= 2) {
46                          for (int k = 0; k != longueur; k++ ) {
47                                  chaineSansEspace += " ";
48                          }
49                  }
50                          return chaineSansEspace;
51      }
52  }
```

Efficacité/Compilation du code:

Cependant le code ne passe pas tous les tests proposés.

En effet, le code réduit de un espace, lorsqu'il y a deux espaces, ce qui donne ce résultat là par exemple :

En terme d'efficacité, en utilisant une classe de test différentes j'ai pu obtenir le nombre de tour effectué pour un texte de 100 000 caractères, nous sommes dans ce cas là sur 98 987 tours ce qui est quasiment de l'ordre de O(n).

Sobriété numérique :

En utilisant l'outil joular, on peut obtenir la consommation électrique de notre programme en joules. Ainsi dès lors que le programme est terminé nous pouvons obtenir sa consommation.

Si on utilise joular pendant 1 minute, la consommation de ce programme est de 49,26 joules.

```
C:\Users\ruben\Desktop\saeAlgo\saeAlgoTest\src>Program consumed 49,26 joules
```

Temps d'exécution :

En ce qui concerne le temps d'exécution, dans une classe de test, j'ai essayé d'obtenir le temps moyen en ms qui enlève les espaces non nécessaires dans un texte de 100 000 caractères, mais il s'avère qu'il n'arrive pas à passer le test interne de ma méthode. Je ne peux donc relever de temps d'éxécution.

Fichier simplicite-103.java :

Lisibilité du code :

Le code manque de lisibilité notamment dans la compréhension des différents types utilisés, où du rôle de chaque variable, quelques commentaires aurait suffit pour aider à une meilleure compréhension du code.

Qualité du code :

En utilisant Codacy, on observe qu'il n'y a aucun problème concernant le code, si ce n'est le nom du fichier mais cela est de mon ressort. Un code très bien écrit qui ne vient pas réduire sa qualité.

Efficacité/Compilation du code:

Il existe un cas où le code ne passe pas le test. Celui-ci en effet réduit les endroits contenant 3 espaces à 2 espaces, ce qui n'était pas attendus. Cela donne cela par exemple :

Concernant l'efficacité, en calculant le nombre de tours effectué pour un texte de 100 000 caractères, j'ai obtenu 98 998 tour. Ce qui est quasiment de l'ordre de O(n).

Sobriété numérique :

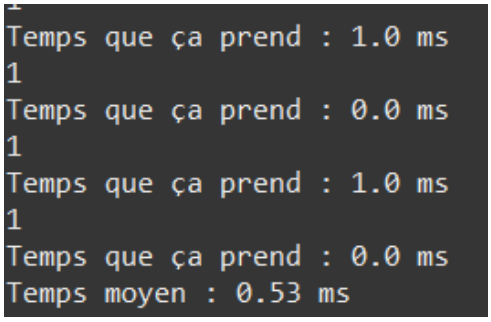
En utilisant l'outil joular, on peut obtenir la consommation électrique de notre programme en joules. Ainsi dès lors que le programme est terminé nous pouvons obtenir sa consommation.

Si on utilise joular pendant 1 minute, la consommation de ce programme est de 43,43 joules.

```
C:\Users\ruben\Desktop\saeAlgo\saeAlgoTest\src>Program consumed 43,43 joules
```

Temps d’exécution :

En ce qui concerne le temps d’exécution, dans une classe de test, j’ai pu obtenir le temps moyen en ms qui enlève les espaces non nécessaires dans un texte de 100 000 caractères, il effectue la manipulation 100 fois. Ce qui donne un temps moyen de 0,53 ms. Ce qui est extrêmement rapide.



Résultat :

En fin de compte, voici la note que je donne à ce programme. Bien qu’il n’a pas réussi un test, il est beaucoup trop performant pour ne pas saluer la performance. Ce qui lui fais la note de 2,5/5.

Fichier simplicite-111.java :

Lisibilité du code :

Le code manque de lisibilité notamment dans la compréhension des différents types utilisés, où du rôle de chaque variable, quelques commentaires aurait suffit pour aider à une meilleure compréhension du code.

Qualité du code :

En utilisant Codacy, on observe qu’il n’y a aucun problème concernant le code, si ce n’est le nom du fichier mais cela est de mon ressort. Un code très bien écrit qui ne vient pas réduire sa qualité.

Efficacité/Compilation du code:

Le code passe tous les tests proposés, il n’y a donc aucun problème lié à cela.

Concernant l’efficacité, en calculant le nombre de tours effectué pour un texte de 100 000 caractères, j’ai obtenu 100 000 tours. Ce qui est de l’ordre de O(n).

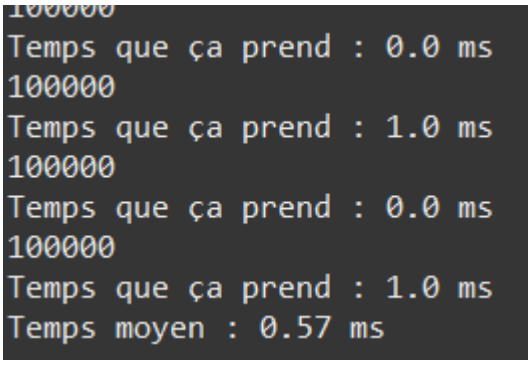
Sobriété numérique :

En utilisant l’outil joular, on peut obtenir la consommation électrique de notre programme en joules. Ainsi dès lors que le programme est terminé nous pouvons obtenir sa consommation.

Si on utilise joular pendant 1 minute, la consommation de ce programme est de 58,58 joules.

Temps d’exécution :

En ce qui concerne le temps d’exécution, dans une classe de test, j’ai pu obtenir le temps moyen en ms qui enlève les espaces non nécessaires dans un texte de 100 000 caractères, il effectue la manipulation 100 fois. Ce qui donne un temps moyen de 0,57 ms. Ce qui est extrêmement rapide.



Catégorie Efficacité :

Fichier efficacite-23.py :

Lisibilité du code :

Le code n’étant pas fini, il est hors concours.

Qualité du code :

Le code n’étant pas fini, il est hors concours.

Efficacité/Compilation du code:

Le code n’étant pas fini, il est hors concours.

Sobriété numérique :

Le code n’étant pas fini, il est hors concours.

Temps d’exécution :

Le code n’étant pas fini, il est hors concours.

Fichier efficacite-64.py :

Lisibilité du code :

Le code manque de lisibilité notamment sur la gestion de ces conditions :

```
if (char == " " and wordslength == 1) or (char == " "
  and listwords[max(i-1, int(wordslength > 1))] != " "
  and listwords[min(i+1, wordslength-int(wordslength > 1)*2)] != " "):
    del listwords[i]
```

Il aurait été intéressant de mettre un peu plus de commentaires pour expliquer le fonctionnement de certaines parties du code.

Qualité du code :

En utilisant Codacy, on observe qu'il y a surtout des problème de *code style* concernant notamment des espaces ou de l'indentation. Ce qui pourrait peut être nuire à la qualité du code :

```
1  # return string without spaces
2  def erase(cc):
   MINOR | Code Style | 1 blank line required between summary line and description (found 0) (D205)
3      """
4      Removes only single spaces from given string.
5      :param text : string to edit
6      :return : edited string
7      """
   MINOR | Code Style | Trailing whitespace
8      listwords = list(cc)
9      for i, char in enumerate(listwords):
10         wordslength = len(listwords)
11         if (char == " " and wordslength == 1) or (char == " "
   MINOR | Code Style | Trailing whitespace
12           and listwords[max(i-1, int(wordslength > 1))] != " "
13           and listwords[min(i+1, wordslength-int(wordslength > 1)*2)] != " "):
14             del listwords[i]
15     return "".join(char for char in listwords)
```

Efficacité/Compilation du code:

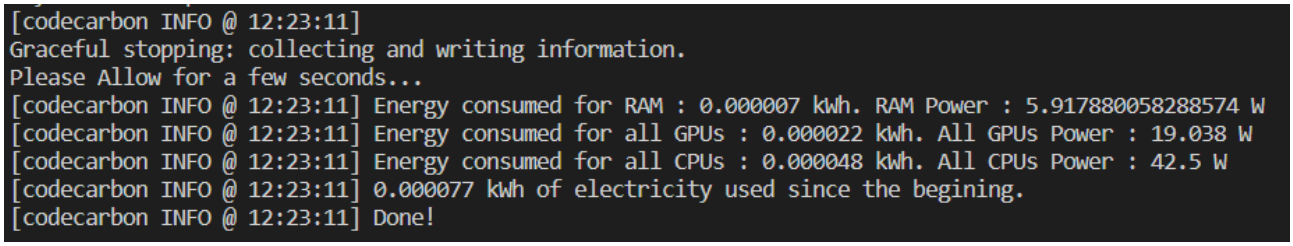
Le code passe toutesLe code passe tous les tests proposés, il n'y a donc aucun problème lié à cela.

Concernant l'efficacité, en calculant le nombre de tours effectué pour un texte de 100 000 caractères, j'ai obtenu 100 000 tours. Ce qui est de l'ordre de O(n).

Sobriété numérique :

Pour calculer la sobriété numérique d'un fichier python, j'ai utilisé codecarbon. En l'important, on peut utilisé différente méthode dont *track_emission*. Cela nous permet d'obtenir la consommation électrique (en W et kWh) de la RAM, du/des GPU, du/des CPU.

J'ai placé cette méthode dans une classe de test qui enlève les espaces non nécessaires dans un texte de 100 000 caractères, il effectue la manipulation 100 fois. (classe utilisé pour calculer le temps d'exécution), et j'ai obtenu le résultat suivant :



Temps d'exécution :

En ce qui concerne le temps d'exécution, dans une classe de test, j'ai pu obtenir le temps moyen en ms qui enlève les espaces non nécessaires dans un texte de 100 000 caractères, il effectue la manipulation 100 fois. Ce qui donne un temps moyen de 0,039 ms. Ce qui est vraiment rapide.

Fichier efficacite-65.java :

Lisibilité du code :

Le code manque un peu de lisibilité notamment dans la compréhension des différents types utilisés, mais vu qu'il y a une sorte de javadoc expliquant le fonctionnement du programme, cela permet de mieux comprendre son fonctionnement..

Qualité du code :

En utilisant Codacy, on observe qu'il n'y a aucun problème concernant le code, si ce n'est le nom du fichier mais cela est de mon ressort. Un code très bien écrit qui ne vient pas réduire sa qualité.

Efficacité/Compilation du code:

Le code passe tous les tests proposés, il n'y a donc aucun problème lié à cela.

Concernant l'efficacité, en calculant le nombre de tours effectué pour un texte de 100 000 caractères, j'ai obtenu 100 000 tours. Ce qui est de l'ordre de O(n).

Sobriété numérique :

En utilisant l'outil joular, on peut obtenir la consommation électrique de notre programme en joules. Ainsi dès lors que le programme est terminé nous pouvons obtenir sa consommation.

Si on utilise joular pendant 1 minute, la consommation de ce programme est de 58,58 joules.

Temps d'exécution :

En ce qui concerne le temps d'exécution, dans une classe de test, j'ai pu obtenir le temps moyen en ms qui enlève les espaces non nécessaires dans un texte de 100 000 caractères, il effectue la manipulation 100 fois. Ce qui donne un temps moyen de 5,12ms. Ce qui est très efficace.

```
100000
Temps que ça prend : 4.0 ms
100000
Temps que ça prend : 5.0 ms
100000
Temps que ça prend : 5.0 ms
100000
Temps que ça prend : 5.0 ms
Temps moyen : 5.12 ms
```

Fichier efficacite-101.py :

Lisibilité du code :

Bien que le code ne soit que sur une seule ligne, il est loin d'être clair car utilisant une librairie python peut connu finalement.

Qualité du code :

En utilisant Codacy, on observe qu'il n'y a aucun problème concernant le code. C'est un code très bien écrit qui ne vient pas entraver sa propre qualité.

Efficacité/Compilation du code:

Le code passe tous les tests proposés, il n'y a donc aucun problème lié à cela.

Concernant l'efficacité, en calculant le nombre de tours effectué pour un texte de 100 000 caractères, j'ai obtenu 1 tour. Ce qui est de l'ordre de O(1).

Sobriété numérique :

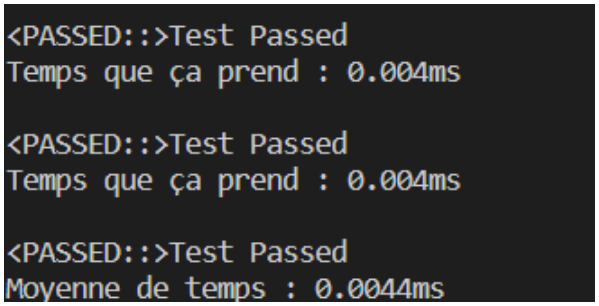
Pour calculer la sobriété numérique d'un fichier python, j'ai utilisé codecarbon. En l'important, on peut utilisé différente méthode dont *track_emission*. Cela nous permet d'obtenir la consommation électrique (en W et kWh) de la RAM, du/des GPU, du/des CPU.

J'ai placé cette méthode dans une classe de test qui enlève les espaces non nécessaires dans un texte de 100 000 caractères, il effectue la manipulation 100 fois. (classe utilisé pour calculer le temps d'exécution), et j'ai obtenu le résultat suivant :

```
[codecarbon INFO @ 15:18:45]
Graceful stopping: collecting and writing information.
Please Allow for a few seconds...
[codecarbon INFO @ 15:18:45] Energy consumed for RAM : 0.000011 kWh. RAM Power : 5.917880058288574 W
[codecarbon INFO @ 15:18:45] Energy consumed for all GPUs : 0.000009 kWh. All GPUs Power : 4.735 W
[codecarbon INFO @ 15:18:45] Energy consumed for all CPUs : 0.000081 kWh. All CPUs Power : 42.5 W
[codecarbon INFO @ 15:18:45] 0.000101 kWh of electricity used since the begining.
[codecarbon INFO @ 15:18:45] Done!
```

Temps d’exécution :

En ce qui concerne le temps d’exécution, dans une classe de test, j’ai pu obtenir le temps moyen en ms qui enlève les espaces non nécessaires dans un texte de 100 000 caractères, il effectue la manipulation 100 fois. Ce qui donne un temps moyen de 0,0044 ms.



Catégorie Sobriété :

Fichier sobriete-19.py :

Lisibilité du code :

Le code n'est pas spécialement compliqué à comprendre à première vue, de plus il faut mentioner la présence de commentaire sous forme d'une petite histoire. C'est sympathique ça change des habituels commentaires, peut être plus efficace mais un peu plus barbant.

Qualité du code :

En utilisant Codacy, on observe qu'il y a surtout des problème de *code style* concernant notamment des espaces ou de l'indentation. Ce qui pourrait peut être nuire à la qualité du code :

MINOR | Code Style | **No blank lines allowed before function docstring (found 1) (D201)**

```
5     '''
6         Author       : xxx Anonymized by JMB xxx
7         Version      : 1.1
8         Date         : 08-06-2022
```

MINOR | Code Style | **Trailing whitespace**

```
9
10        Description  : Takes a string parameter and remove simple spaces, not double, inside of it.
11
12        Return       : String
13    '''
14
```

MINOR | Code Style | **Trailing whitespace**

```
15    # Reflexion
16        # A String is an array of char type variables in its primary form.
```

MINOR | Code Style | **Trailing whitespace**

```
17        #
18        # Adept and fascinated of recursive algorithms, I've made one for this project !
19        #
20        # The reasonning is purely primitive but very efficient because it doesn't include a lot of
21        # It's based on array accesses.
22        #
23        # I will train to through the algorithm to explain his functionning with a little game.
24        #
25        # In this game you are a char, the first one of the queue, and you don't know who are behin
26        # If you are a space char, you are in love with another space char, every moment of your li
```

MINOR | Code Style | **Trailing whitespace**

```
27        # Be sure to never lose your partner of life !
28
29        #   Who am I ?
30    if message == "": # Am I even real ? No...
31        return message # So I just return myself : the void
32
33        #   So I know that I'm not the void, am I space ?
34    if message[0] != ' ': # Surely not, you are discussing to the wrong person
35        return message[0] + erase(message[1:]) # I enter the Valallah and the purge continues !
36
37        #   Yeah it's me ! The famous and beautiful space char ;)
38    else:
```

Efficacité/Compilation du code:

Il est impossible de lancer le code car il renvoie toujours des erreurs liés à une récursion trop importante donc incapable de renvoyer quoique ce soit. Il est donc hors-concours.

Sobriété numérique :

Il est impossible de lancer le code car il renvoie toujours des erreurs liés à une récursion trop importante donc incapable de renvoyer quoique ce soit. Il est donc hors-concours.

Temps d'exécution :

Il est impossible de lancer le code car il renvoie toujours des erreurs liés à une récursion trop importante donc incapable de renvoyer quoique ce soit. Il est donc hors-concours.

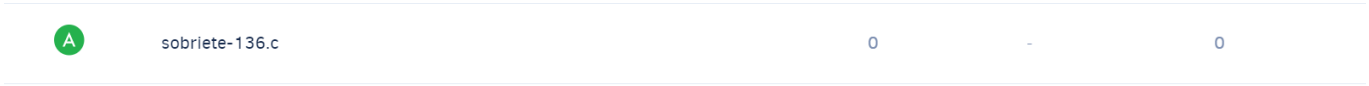
Fichier sobriete-136.c :

Lisibilité du code :

Le code manque en lisibilité, il aurait été préférable de mettre plus de commentaire afin d'y mettre plus de clareté.

Qualité du code :

En utilisant Codacy, on observe qu'il n'y a aucun problème concernant le code. C'est un code très bien écrit qui ne vient pas entraver sa propre qualité.



Efficacité/Compilation du code:

Le code passe tous les tests proposés, il n'y a donc aucun problème lié à cela.

Pour l'efficacité, J'ai utilisé un compteur qu'on incrémente de un chaque fois qu'on rentre dans une boucle for ou une boucle if :

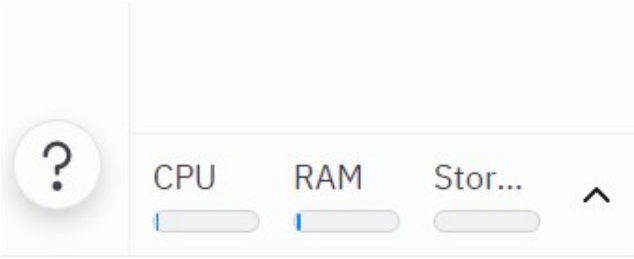
Caractères en entrée	5	10	100	1000	5000
Compteur	10	20	200	2000	10000

Grâce au tableau on peut voir qu'il y a un peu plus de 2 fois le nombre d'itération de boucle par rapport au nombre de caractère en entrée

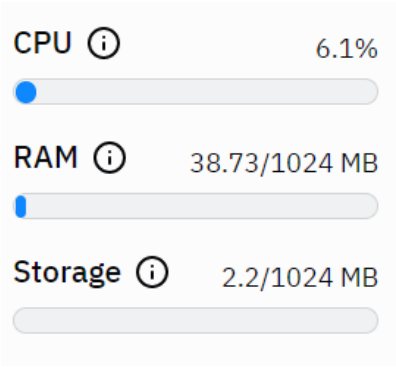
Cependant, lorsqu'on observe les temps d'exécution du programme, on remarque que les temps sont très linéaire. On pourrait donc dire que la complexité algorithmique de cet algo est de O(nlog(n)).

Sobriété numérique :

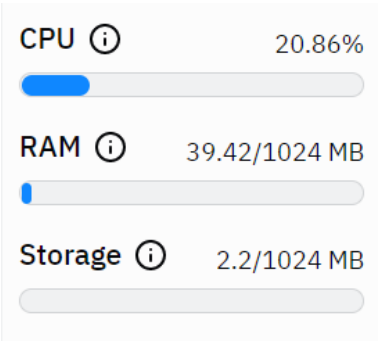
Pour évaluer la sobriété numérique d'un algorithme en C, j'utilise dans Replit l'onglet dédié lorsqu'on code un programme. (en bas à gauche de l'écran) :



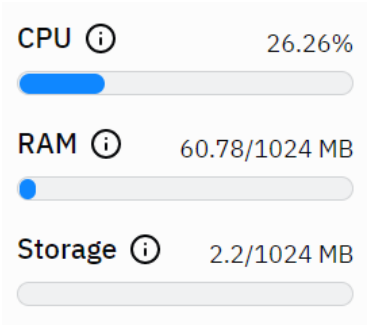
Avec une chaîne en entré de 5 caractères :



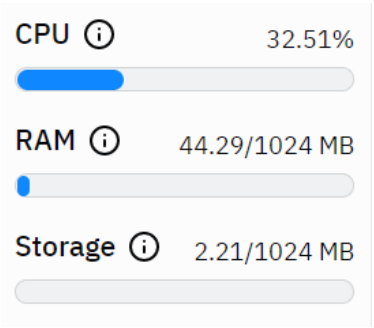
Avec une chaîne en entré de 10 caractères :



Avec une chaîne en entré de 100 caractères :



Avec une chaîne en entré de 1000 caractères :



Temps d'exécution :

Nombre de donnée en entrée	5	10	100	200	400
Temps	50µs	52µs	64µs	67µs	66µs

Fichier sobriete-162.py :

Lisibilité du code :

Le code est un peu compréhensible à première vue mais des commentaires serait le bienvenu pour comprendre encore mieux où la logique du code, en tout cas son fonctonnement.

Qualité du code :

En utilisant Codacy, on observe qu'il n'y a aucun problème concernant le code. C'est un code très bien écrit qui ne vient pas entraver sa propre qualité.

Efficacité/Compilation du code:

Le code passe tous les tests proposés, il n'y a donc aucun problème lié à cela.

Concernant l'efficacité, en calculant le nombre de tours effectué pour un texte de 100 000 caractères, j'ai obtenu 101 000 tours. Ce qui est un peu plus que O(n).

Sobriété numérique :

Pour calculer la sobriété numérique d'un fichier python, j'ai utilisé codecarbon. En l'important, on peut utilisé différente méthode dont *track_emission*. Cela nous permet d'obtenir la consommation électrique (en W et kWh) de la RAM, du/des GPU, du/des CPU.

J'ai placé cette méthode dans une classe de test qui enlève les espaces non nécessaires dans un texte de 100 000 caractères, il effectue la manipulation 100 fois. (classe utilisé pour calculer le temps d'exécution), et j'ai obtenu le résultat suivant :

```
[codecarbon INFO @ 16:22:09]
Graceful stopping: collecting and writing information.
Please Allow for a few seconds...
[codecarbon INFO @ 16:22:09] Energy consumed for RAM : 0.000007 kWh. RAM Power : 5.917880058288574 W
[codecarbon INFO @ 16:22:09] Energy consumed for all GPUs : 0.000009 kWh. All GPUs Power : 7.6000000000000005 W
[codecarbon INFO @ 16:22:09] Energy consumed for all CPUs : 0.000049 kWh. All CPUs Power : 42.5 W
[codecarbon INFO @ 16:22:09] 0.000064 kWh of electricity used since the begining.
[codecarbon INFO @ 16:22:09] Done!
```

Temps d’exécution :

En ce qui concerne le temps d’exécution, dans une classe de test, j’ai pu obtenir le temps moyen en ms qui enlève les espaces non nécessaires dans un texte de 100 000 caractères, il effectue la manipulation 100 fois. Ce qui donne un temps moyen de 0,038 ms.

```
<PASSED::>Test Passed
Temps que ça prend : 0.032ms

<PASSED::>Test Passed
Temps que ça prend : 0.04ms

<PASSED::>Test Passed
Moyenne de temps : 0.038ms
```