
# Alignement de séquences

LU3IN003

Léa Cohen-Solal
Lysa Rudy
*Dirigé par Olivier Spanjaard*

# Table des matières

Le code des taches A, B, C et D se trouve dans les fichiers *tache_A.py*, *tache_B.py*, *tache_C.py* et *tache_D.py* respectivement . Les tests des fonctions sont dans les fichiers *essai_A.py*, *essai_B.py*, *essai_C.py* et *essai_D.py*

# 1 Le problème d'alignement des séquences

## 1.1 Alignement de deux mots

### Question 1

Montrons que si ( $\overline{x},\overline{y}$) et ($\overline{u},\overline{v}$) sont des alignements de (x,y) et (u,v), alors ($\overline{x}.\overline{u}$ , $\overline{y}.\overline{v}$) est un alignement de (x.u, y.v).
($\overline{x},\overline{y}$) = ($x_1.....x_k, y_1.....y_k$) un alignement de $(x,y) = (x'_1....x'_{k'}, y'_1....y'_{k''})$
($\overline{u},\overline{v}$) = ($u_1.....u_n, v_1.....y_n$) un alignement de $(u,v) = (u'_1....u'_{n'}, v'_1....v'_{n''})$ avec n',n" ≤ n et k',k"≤ k.
(x.u) = $x'_1....x'_{k'}.u'_1....u'_{n'}$ avec |(x.u)|= n'+k'
On obtient x en supprimant tous les gaps de $\overline{x}$ et u en supprimant tous les gaps de $\overline{u}$. Comme on le remarque x.u s'obtient de la même manière.
Soient $\overline{X} = (\overline{x}.\overline{u})$ et $\overline{Y} = (\overline{y}.\overline{v})$

— Montrons que $\pi(\overline{x}.\overline{u}) = x.u$ et $\pi(\overline{y}.\overline{v}) = y.v$
On sait que $\pi(\overline{x}) = x$, $\pi(\overline{y}) = y$ , $\pi(\overline{u}) = u$ et $\pi(\overline{v}) = v$
Par définition $\pi(a.b) = \pi(a).\pi(b)$
$\Longrightarrow$ $\pi(\overline{x}.\overline{u}) = \pi(\overline{x}).\pi(\overline{u}) = x.u$ et $\pi(\overline{y}.\overline{v}) = \pi(\overline{y}).\pi(\overline{v}) = y.v$

— Montrons que $|\overline{X}| = |\overline{Y}|$ c'est-à-dire que $|(\overline{x}.\overline{u})| = |(\overline{y}.\overline{v})|$
$(\overline{x}.\overline{u}) = x_1.....x_k u_1....u_n$ et $(\overline{y}.\overline{v}) = y_1.....y_k v_1....v_n$
Comme ( $\overline{x}$,$\overline{y}$) et ( $\overline{u}$, $\overline{v}$) sont respectivement des alignements de (x,y) et (u,v), $|\overline{x}| = |\overline{y}|$ et $|\overline{u}| = |\overline{v}|$
Donc $|(\overline{x}.\overline{u})| = |\overline{x}| + |\overline{u}| = k + n = |\overline{y}| + |\overline{v}| = |(\overline{y}.\overline{v})|$
Donc $|\overline{X}| = |\overline{Y}|$

— Montrons que $\forall i \in \{1,....,n+k\}$ $\overline{X_i} \neq \_ \; ou \; \overline{Y_i} \neq \_$
On sait que
$\forall i \in \{1,....,k\}$ $\overline{x_i} \neq \_ \; ou \; \overline{y_i} \neq \_$
$\forall i \in \{1,....,n\}$ $\overline{u_i} \neq \_ \; ou \; \overline{v_i} \neq \_$
Supposons qu'il y ait un j $\in \{1,...,n+k\}$ tel que $\overline{X_j} = \_ et \overline{Y_j} = \_$
- Si j <= k $\overline{X_j} = \overline{x_j} = \_$ et $\overline{Y_j} = \overline{y_j} = \_$ $\Longrightarrow$ $\overline{y_j} = \_$ et $\overline{x_j} = \_$ donc ($\overline{x},\overline{y}$) n'est pas un alignement de (x,y)
- Si j>k $\overline{X_j} = \overline{u_{j-k}} = \_$ et $\overline{Y_j} = \overline{v_{j-k}} = \_$ $\Longrightarrow$ $\overline{u_{j-k}} = \_$ et $\overline{v_{j-k}} = \_$ donc ($\overline{u},\overline{v}$) n'est pas un alignement de (u,v)

$\Longrightarrow$ Contradiction.
Donc $\forall i \in \{1,....,n+k\}\ \overline{X_i} \neq \_\ ou\ \overline{Y_i} \neq \_$

On en conclut donc que ($\overline{x}.\overline{u}$ , $\overline{y}.\overline{v}$) est un alignement de (x.u, y.v)

**Question 2**

Soient |x| = n et |y| = m et $(\overline{x},\overline{y})$ un alignement de (x,y).
Par définition de l'alignement $|\overline{x}| = |x| + k$ et $|\overline{y}| = m + k'$ avec k et k'>=0 où k est le nombre de gaps ajouté à x et k' le nombre de gaps ajoutés à y.
donc $|\overline{y}| >= m$ et $|\overline{x}| >= n$ Or $|\overline{y}| = |\overline{x}|$ Donc $|\overline{x}|$ >= max(n,m)

Montrons que le nombre de gaps ne peut pas dépasser min(n,m).

*Supposons ici que n<=m ( Le raisonnement est le même si $m \leq n$)* :
max(n,m) $\leq |\overline{x}| \leq n + min(n,m) = 2n$
max(n,m) $\leq |\overline{y}| \leq m + min(n,m) = n + m$ max( 2n, n+m) = n+m.
Donc $|\overline{x}| = |\overline{y}| \leq n + m$ .
En effet supposons que $|\overline{x}| = n + m + 1$.
$\overline{x}$ compte donc m+1 gaps $\{g_1,....,g_{m+1}\}$ où $g_i$ est la position du i-ième gap dans $\overline{x}$.
$\overline{y}$ compte donc n+1 gaps $\{g_1,....,g_{n+1}\}$ où $g_i$ est la position du i-ième gap dans $\overline{y}$
$\overline{y}$ a plus de gaps que $\overline{x}$ n'a de lettre $\Longrightarrow \exists i \in \{1,....,n+m+1\}$ tel que $\overline{y_i} = \_$ et $\overline{x_i} = \_ \Longrightarrow (\overline{x},\overline{y})$ n'est pas un alignement de (x,y).
Donc $|\overline{x}| = |\overline{y}| \leq n + m$ .

Si $|\overline{x}| = n + m$
$\overline{x}$ compte donc m gaps $\{g_1,....,g_m\}$
$\overline{y}$ compte donc n gaps $\{g_1,....,g_n\}$
Pour chaque lettre l de x $\{l_{x1},....l_{xn}\}$ il est possible de placer la lettre $l_{xi}$ dans $\overline{x}$ et le gap $g_i$ dans $\overline{y}$ au même indice.
Par ce processus les lettres de $\overline{x}$ seront placées aux mêmes indices que les gaps de $\overline{y}$ et les lettres de $\overline{y}$ seront placées aux mêmes indices que les gaps de $\overline{x}$. La longueur maximale d'un alignement de (x,y) est donc n+m.

# 2 Algorithmes pour l'alignement de séquences

## 2.1 Méthode naïve par énumération

### Question 3

Nous cherchons le nombre de mots obtenus en ajoutant k gaps à un mot de n lettres. On obtient alors un mot de n+k lettres. Pour approcher le problème d'une autre manière on pourrait se demander : Pour k $\in \{1, ....., n+k\}$, combien de combinaisons différentes de k chiffres pouvons nous créer ? En prenant en compte le fait que la combinaison $\{1, 3, 7\}$ est la même que $\{3, 1, 7\}$ nous cherchons donc le nombre de combinaisons de k éléments parmi n+k éléments, soit :
$\binom{n+k}{k}$

### Question 4

Une fois que l'on a ajouté k gaps à x pour obtenir $\overline{x}$, on obtient l'égalité :
n + k = m + k' où k' est le nombre de gaps que l'on rajoute à y. Les gaps de $\overline{x}$ ne doivent pas être au même endroit que les gaps de de $\overline{y}$. Ainsi les k' gaps peuvent se situer seulement sur les n emplacements restants. Alors, il y a $\binom{n}{k'}$ façons d'insérer les k' gaps dans y pour satisfaire cette contrainte. On a : $\binom{n}{k'} = \binom{n}{n+k-m}$.
On observe que pour k gaps, on obtient $\binom{n+k}{k}$ x $\binom{n}{k'}$ alignements possibles pour (x,y). Enfin, d'après la question 2, au maximum, $\overline{x}$ compte m gaps (k=m) et $\overline{y}$ compte n gaps (k'=n). On obtient donc le nombre d'alignements possibles de (x,y) : $\sum_{k=0}^{m}(\binom{n+k}{k} * \binom{n}{n+k-m})$.
Si n=15 et m=10, on obtient que le nombre d'alignements possibles de (x,y) est : 298 199 265

### Question 5

En supposant que le nombre d'alignements possibles d'un couple de mots (x,y) de longueurs respectives n et m est $\sum_{k=0}^{m}(\binom{n+k}{k} * \binom{n}{n+k-m})$, la complexité temporelle pour énumérer tous les alignements de (x,y) est donc de $\sum_{k=0}^{m}(\binom{n+k}{k} * \binom{n}{n+k-m})$

$$\binom{n+k}{k} * \binom{n}{n+k-m} = \frac{(n+k)n!}{k!n!(n+k-m!(m-k)!} = \frac{(n+k)!}{k!(n+k-m!(m-k)!} \leq (n+k)! \leq (n+m)!$$

Donc $\sum_{k=0}^{m}(\binom{n+k}{k} * \binom{n}{n+k-m}) \leq \sum_{k=0}^{m}(n+m)! = m * (n+m)!$

La complexité est donc de $O(m \times (n+m)!)$.

**Question 6**

Comme nous avons besoin de stocker toutes les distances d'édition avant de trouver la plus petite. Pour chaque alignement trouvé et donc pour chaque distance trouvée on doit stocker x ( O(n)) et y(O(m)). Comme vu à la question 5, le nombre maximal d'alignements possible est en O(m*(n+m) !). La complexité spatiale totale est donc :

$$\text{nombre d'alignements possible de (x,y)} * \text{taille de x et y} = O(m * (m+m)! * (n+m))$$

Les complexités spatiales et temporelles de cette méthode se rapprochent de la complexité exponentielle, elle n'est donc pas optimale.

**Tache A**

*TEST DE DIST_NAIF*

Les fonctions $DIST\_NAIF$ et $DIST\_NAIF\_REC$ se trouvent dans le fichier $Tache\_A.py$ Après vérification, on obtient bien les valeurs de retour 10, 8 et 2 quand on applique dist_naif sur *Inst_0000010_44.adn*, *Inst_0000010_7.adn* et *Inst_0000010_8.adn*

```
 Pour Inst_0000010_7.adn  Dist = 8

Pour Inst_0000010_44.adn  Dist = 10

Pour Inst_0000010_8.adn  Dist = 2
```

*TEMPS MAX DE DIST_NAIF*

En effectuant des tests sur plusieurs instances, on s'aperçoit qu'après |x| = 12, on dépasse la minute. On remarque tout de même que lorsque |y| se rapproche de |x|, les temps d'exécutions s'allongent considérablement comme on peut remarquer entre les fichiers 12_13 (et 12_32 ) et 12_56 .

```
Instances_genome/Inst_0000010_44.adn 0.03739595413208008
Instances_genome/Inst_0000010_7.adn 7.484240770339966
Instances_genome/Inst_0000010_8.adn 2.916300058364868
Instances_genome/Inst_0000012_13.adn 13.385396957397461
Instances_genome/Inst_0000012_32.adn 13.467100143432617
Instances_genome/Inst_0000012_56.adn 96.79302024841309
```

*CONSOMMATION MEMOIRE DE DIST_NAIF*

Nous avons mesuré la quantité de mémoire utilisée par la fonction DIST_NAIF pour les 5 premières instances ( les quantités sont en octets ou bytes).
Pour cela nous avons utiliser le module *tracemalloc*.

```
Inst_0000010_44.adn Temps = 0.0508267879486084 consommation mémoire 19329

Inst_0000010_7.adn Temps = 10.213174819946289 consommation mémoire 27069

Inst_0000010_8.adn Temps = 4.136791229248047 consommation mémoire 29832

Inst_0000012_13.adn Temps = 20.95844578742981 consommation mémoire 29967

Inst_0000012_32.adn Temps = 21.820443868637085 consommation mémoire 30623

Inst_0000012_56.adn Temps = 172.80903887748718 consommation mémoire 30757
```

# 3 Programmation dynamique

## 3.1 Calcul de la distance d'édition par programmation dynamique

**Question 7**

— Si $\overline{u}_l = \_$ , par définition de l'alignement on ne peut pas avoir $\overline{v}_l = \_$ donc $\overline{v}_l$ est le dernier caractère de y soit $y_j$. Donc si $\overline{u}_l = \_$ , $\overline{v}_l = y_j$.
— Si $\overline{v}_l = \_$ , de la même manière on aura $\overline{u}_l = x_i$.
— Si $\overline{u}_l \neq \_$ et $\overline{v}_l \neq \_$ , il n'y a qu'une seule possibilité, $\overline{u}_l = x_i$ et $\overline{v}_l = y_j$

**Question 8**

$$C(\overline{u},\overline{v}) = \begin{cases} c\_ins + c(\overline{u}_{[1....l-1]}, \overline{v}_{[1....l-1]}) & \text{si } \overline{u}_l = \_ \\ c\_del + c(\overline{u}_{[1....l-1]}, \overline{v}_{[1....l-1]}) & \text{si } \overline{v}_l = \_ \\ c\_sub(\overline{u_l}, \overline{v_l}) + c(\overline{u}_{[1....l-1]}, \overline{v}_{[1....l-1]}) & \text{si } \overline{v}_l \neq \_ \textbf{ et } \overline{u}_l \neq \_ \end{cases}$$

**Question 9**

On a D(i,j) = d ( $x_{[1...i]}, y_{[1...j]}$) = min{ C($\overline{x_{[1...i]}}, \overline{y_{[1...j]}}$)) | ($\overline{x_{[1...i]}}, \overline{y_{[1...j]}}$)} est un alignement de $(x\_[1...i], y\_[1...j])$}. Nous avons déterminé dans la question précédente la formule récursive permettant de calculer la distance d'édition de ( $\overline{x}, \overline{y}$). Sachant que pour (x,y) il y a plusieurs manières d'obtenir ( $\overline{x}, \overline{y}$), on souhaite trouver l'alignement ayant la distance d'édition la plus faible. Prenons un exemple : Soit $x_{[1,2,3,4]}$ = ATGA et $y_{[1,2,3]}$ = GGC Pour trouver d($x_{[1,2,3,4]}$, $y_{[1,2,3]}$) on prend la dernière lettre de chaque mot ( ici A et C) et on les confronte à un choix :

— Soit on place un gap à la fin de $\overline{x}$ donc $\overline{x_l} = \_$ et dans ce cas la , $\overline{y_l} = y_j = y_3 = C$. D'après q8 on a $C(\overline{u},\overline{v}) = c(\overline{x_l},\overline{y_l}) + c(\overline{u}_{[1....l-1]}, \overline{v}_{[1....l-1]}) = c(\_, C) + c(\overline{u}_{[1....l-1]}, \overline{v}_{[1....l-1]}) = c\_ins + c(\overline{u}_{[1....l-1]}, \overline{v}_{[1....l-1]})$. Donc la lettre $y_3$ a été utilisé, on doit continuer le processus avec les mots $x_{[1,2,3,4]}$ = ATGA et $y_{[1,2]}$ = GG. Plus formellement si on place un gap à la fin de $\overline{x}$ on aura distance d'édition($(x_{[1,...,i]}, y_{[1,...,j-1]}) = c_ins$ + distance d'édition $(x_{[1,...,i]}, y_{[1,...,j-1]})$
— Soit on place un gap à la fin de $\overline{y}$ donc $\overline{y_l} = \_$ et $\overline{x_l} = x_i = A$ dans ce cas la , d'après q8 on a $C(\overline{u},\overline{v}) = c(\overline{x_l},\overline{y_l}) + c(\overline{u}_{[1....l-1]}, \overline{v}_{[1....l-1]}) = c(A, \_) + c(\overline{u}_{[1....l-1]}, \overline{v}_{[1....l-1]}) = c\_del + c(\overline{u}_{[1....l-1]}, \overline{v}_{[1....l-1]})$ De la même manière on aura distance d'édition($x_{[1,...,i]}, y_{[1,...,j]}$) = $c_{del}$ + distance d'édition $(x_{[1,...,i-1]}, y_{[1,...,j]})$
— Soit on ne place aucun gap à la fin de $\overline{x}$ et $\overline{y}$ et dans ce cas la on aura $C(\overline{u},\overline{v}) = c(\overline{x_l},\overline{y_l}) + c(\overline{u}_{[1....l-1]}, \overline{v}_{[1....l-1]}) = c(A, C) + c(\overline{u}_{[1....l-1]}, \overline{v}_{[1....l-1]}) = c\_sub(A, C) + c(\overline{u}_{[1....l-1]}, \overline{v}_{[1....l-1]})$
$\Longrightarrow$ distance d'édition $(x_{[1,...,i]}, y_{[1,...,j]}) = c_{sub}(x_i, y_j)$+ distance d'édition $(x_{[1,...,i-1]}, y_{[1,...,j-1]})$

Nous pouvons donc en déduire une formule de D(i,j) :

$$\text{D(i,j) = min ( } c_{ins} + D(i, j-1) \text{ , } c_{del} + D(i-1, j) \text{ , } c_{sub}(x_i, y_j) + D(i-1, j-1)\text{)}$$

En prenant i' = i-1 et j' = j-1 on retrouve :

$$\text{D(i,j) = min ( } c_{ins} + D(i, j') \text{ , } c_{del} + D(i', j) \text{ , } c_{sub}(x_i, y_j) + D(i', j')\text{)}$$

**Question 10**

D(0,0) = $d(x_{[1\ldots0]}, y_{[1\ldots0]}) = d(\emptyset, \emptyset) = 0$.

En effet l'alignement de 2 mots vides est unique car, si on a x et y vide, la seule chose que l'on peut faire pour obtenir $\overline{x}$ et $\overline{y}$ est d'ajouter des gaps. Cependant si l'on ajoute un gap à x et y on obtiendra $\overline{x_1} = \overline{y_1} = \_$ ce qui ne peut pas correspondre à un alignement de (x,y). Le seul alignement possible de (x,y) est donc $\overline{x} = \overline{y} = \emptyset$ La distance d'édition est donc nulle.

**Question 11**

D(0,j) = $d(\emptyset, y_{[1\ldots j]})$. Il n'y a qu'une seule façon d'obtenir un alignement de (x,y) avec x le mot vide. Il faudra rajouter j gap dans x et 0 gap dans y. En effet pour que $(\overline{x}, \overline{y})$ puisse être un alignement de (x,y) il faut que $|\overline{x}| = |\overline{y}| >= |y|$. Comme |y| = j, $\overline{x}$ doit au moins avoir j gaps. Si on rajoute j+1 gap à x, on devra rajouter un gap à y et on aura donc forcement un $i \in \{1, \ldots j+1\}$ pour lequel $\overline{x_i} = \overline{y_i}$. Donc on a $\overline{x}$ composé de j gaps et $\overline{y} = y_{[1\ldots j]}$

On a donc :

D(0,j) = $C(\overline{x}, \overline{y_{[1,\ldots,j]}}) = \sum_{k=1}^{j} c(\overline{x}_k, \overline{y}_k) = \sum_{k=1}^{j} c(\_, y_k) = \sum_{k=1}^{j} c_{ins} = \sum_{k=1}^{j} 2 = 2j$

**Question 12**

```
DIST_1
    Entrée : x, y et une matrice T vide de taille (n+1)*(m+1)
    Sortie : distance d'édition de (x,y)
    n = |x|
    m = |y|
    T = [n+1][m+1]  On crée un tableau de n+1 lignes et m+1 colonnes qu'on rempli de 0
    For i from 0 to n :
        T[i][0] = c_del * i
    For j from  0 to m :
        T[0][j] = c_ins * j


    For i from  1 to n :
        For j from  1 to m :
        T[i,j] = min(T[i][j-1] + c_ins , T[i-1][j] + c_del, T[i-1][j-1] + c_sub(x[i-1],y[j-1])

    retourner T[n][m]
```

**Question 13**

Dans $DIST\_1$, on alloue une matrice de n + 1 lignes et m + 1 colonnes. La taille totale est donc de n+1 $*$ m+1 cases. La complexité spatiale de $DIST\_1$ est $O(n*m)$

**Question 14**

L'algorithme se compose de :

- une boucle de n itérations permettant de donner l valeur D(i,0) pour i allant de 0 à n
- une boucle de m itérations permettant de donner l valeur D(0,j) pour j allant de 0 à m
- 2 boucles imbriquées ( une à n itérations et l'autre à m itérations) contenant chacune des opérations élémentaires en O(1).

Donc la complexité totale est de O(n + m + n $*$ m ) = $O(n*m)$

## 3.2 Calcul d'un alignement optimal par programmation dynamique

**Question 15**

Montrons que :
Si D(i,j) = D(i-1, j-1) + $c_{sub}(x_i, y_j)$ alors $\forall(\overline{s},\overline{t}) \in$ Al(i-1,j-1) , $(\overline{s}.x_i, \overline{t}.y_j) \in Al(i,j)$
On suppose que D(i,j) = D(i-1, j-1) + $c_{sub}(x_i, y_j)$ et soit $(\overline{s},\overline{t}) \in Al(i-1, j-1)$.
Montrons que $(\overline{s}.x_i, \overline{t}.y_j) \in Al(i,j)$ :
$(\overline{s}.x_i, \overline{t}.y_j) \in Al(i,j)$ si il respecte les 2 conditions suivantes :

— $(\overline{s}.x_i, \overline{t}.y_j)$ est un alignement de $(x_{[1,...i]}, y_{[1,...j]})$
— $C(\overline{s}.x_i, \overline{t}.y_j) = d(x_{[1,...i]}, y_{[1,...j]})$

Montrons que $(\overline{s}.x_i, \overline{t}.y_j)$ est un alignement de $(x_{[1,...i]}, y_{[1,...j]})$ :
— $\Pi(\overline{s}.x_i) = \Pi(\overline{s}).\Pi(x_i)$ et $\Pi(\overline{t}.y_j) = \Pi(\overline{t}).\Pi(y_j)$
Comme $x_i$ et $y_j \in \sum$ alors $\Pi(x_i) = x_i$ et $\Pi(y_j) = y_j$.
On a $(\overline{s}, \overline{t})$ est un alignement de $(x_{[1,...i-1]}, y_{[1,...j-1]})$ alors $\Pi(\overline{s}) = x_{[1,...i-1]}$ et $\Pi(\overline{t}) = y_{[1,...j-1]}$
On a donc bien $\Pi(\overline{s}.x_i) = x_{[1,...i-1]}.x_i = x_{[1,...i]}$ et $\Pi(\overline{t}.y_j) = y_{[1,...j-1]}.y_j = y_{[1,...j]}$
— $|\Pi(\overline{s}.x_i)| = |\overline{s}| + |x_i| = 1 + |\overline{s}|$
$|\Pi(\overline{t}.y_j)| = |\overline{t}| + |y_j| = 1 + |\overline{t}|$
Comme $(\overline{s}, \overline{t})$ est un alignement de $(x_{[1,...i-1]}, y_{[1,...j-1]})$ $|\overline{s}| = |\overline{t}| = k$
On a donc $|\Pi(\overline{s}.x_i)| = |\overline{s}| + 1 = |\overline{t}| + 1 = |\Pi(\overline{t}.y_j)|$.
— Pour faciliter la notation on écrira $(\overline{s}.x_i, \overline{t}.y_j) = (\overline{u}, \overline{v})$.
Comme $(\overline{s}, \overline{t})$ est un alignement de $(x_{[1,...i-1]}, y_{[1,...j-1]})$ on a
$\forall i \in [1, ...|\overline{t}|]$ $\overline{x}_i = \_$ ou $\overline{y}_i = \_$
Soit k $\in [1, ...|\overline{t}| + 1]$.
- Si k $\leq |\overline{t}|$ alors $\overline{u}_k = \overline{s}_k$ et $\overline{v}_k = \overline{t}_k$. Donc k $\leq |\overline{t}| \implies \overline{u_k} = \_$ ou $\overline{v_k} = \_$
- Si k $= |\overline{t}| + 1$ alors $\overline{u}_k = x_i \neq \_$ et $\overline{v}_k = y_j \neq \_$
On a donc bien $\forall i \in [1, ...|\overline{t}.y_j|]$ $\overline{u}_i = \_$ ou $\overline{v}_i = \_$
$(\overline{s}.x_i, \overline{t}.y_j)$ est donc bien un alignement de $(x_{[1,...i]}, y_{[1,...j]})$

Montrons que $C(\overline{s}.x_i, \overline{t}.y_j) = d(x_{[1,...i]}, y_{[1,...j]})$
On rappelle qu'ici on notera $(\overline{s}.x_i, \overline{t}.y_j) = (\overline{u}, \overline{v})$. Soit $|\overline{u}|$ = n

$$C(\overline{s}.x_i, \overline{t}.y_j) = C(\overline{u}, \overline{v}) = \sum_{k=1}^{n} c(\overline{u}_k, \overline{v}_k) = c(\overline{u}_n, \overline{v}_n) + C(\overline{u}_{[1,...,n-1]}, \overline{v}_{[1,...,n-1]})$$

Ici on a $(\overline{u}_n, \overline{v}_n) = (x_i, y_j)$, avec $x_i \neq \_ et y_j \neq \_$, donc $c(\overline{u}_n, \overline{v}_n) = c_{sub}(\overline{u}_n, \overline{v}_n)$
$C(\overline{u}, \overline{v}) = c_{sub}(x_i, y_j) + C(\overline{u}_{[1,...,n-1]}, \overline{v}_{[1,...,n-1]}) = c_{sub}(x_i, y_j) + C(\overline{s}, \overline{t})$.
$(\overline{s}, \overline{t}) \in Al(i-1, j-1)$ donc $C(\overline{s}, \overline{t}) = d(x_{[1,...i-1]}, y_{[1,...j-1]}) = D(i-1, j-1)$
$C(\overline{u}, \overline{v}) = c_{sub}(x_i, y_j) + D(i-1, j-1)$.
Or $D(i-1, j-1) + c_{sub}(x_i, y_j) = D(i, j)$, donc $C(\overline{u}, \overline{v}) = D(i, j) = d(x_{[1,...i]}, y_{[1,...j]})$
On a donc bien montré que :
Si D(i,j) = D(i-1, j-1) + $c_{sub}(x_i, y_j)$ alors $\forall (\overline{s}, \overline{t}) \in$ Al(i-1,j-1) , $(\overline{s}.x_i, \overline{t}.y_j) \in Al(i, j)$

### Question 16

```
SOL_1
    Entrée x, y et T la matrice contenant les valeurs de D
    Sortie : alignement optimal de (x,y)
    i ← |x|
    j ←|y|
    s ← ""
    t ← ""

    while i>0 and  j>0
        if T[i,j] == T[i-1,j-1] + c_sub(x[i-1],y[j-1])
            s ← x[i-1]+s
            t ← y[j-1]+t
            i ← i-1
            j ← j-1
        elif T[i,j] == T[i,j-1] + c_ins
            s ← "_"+s
            t ← y[j-1]+t
            j ← j-1
        elif T[i,j] == T[i-1,j] + c_del
            s ← x[i-1]+s
            t ← "_"+t
            i ← i-1

    while i>0
        if T[i,j] == T[i-1,j] + c_del
            s ← x[i-1]+s
            t ← "_"+t
        i ← i-1

    while j>0
        if T[i,j] == T[i,j-1] + c_ins :
            s ← "_"+s
            t ← y[j-1]+t
        j ← j-1
    retourner (s,t)
```

### Question 17

Pour résoudre le problème ALI avec les ressources que nous avons à notre disposition, nous devons d'abord construire la matrice des valeurs de D grâce à la fonction $DIST\_1$ puis construire l'alignement associé avec la fonction $SOL\_1$

Soit x et y de taille respective n et m.

- **Complexité de DIST_1 = O(n.m)**
- **Complexité de SOL_1** :
  La fonction $SOL\_1$ est composée de 3 boucles :
  - La première boucle fait au plus *c1 = min(n,m)* itérations.
  - Si |x| > |y| :
  On passe seulement par la 2e boucle de complexité *c2 = n -c1*
  - Si |x| < |y| :
  On passe seulement par la 3e boucle de complexité *c2 = n -c1*

La complexité totale est de c1 + c2 + Complexité de $DIST\_1$ = O(n.m +n) = **O**($n^2$) car $m \leq n$.

**Question 18**

Comme vu dans la question 13, la complexité spatiale de $DIST\_1$ est O($n * m$) et dans $SOL\_1$ on utilise la mémoire du tableau T de taille $(n+1)*(m+1)$ et les mots x et y donc la complexité spatiale de $SOL\_1$ est $O(n*m+n+m)$ La complexité totale est $O(2*n*m+n+m) = O(n*m)$

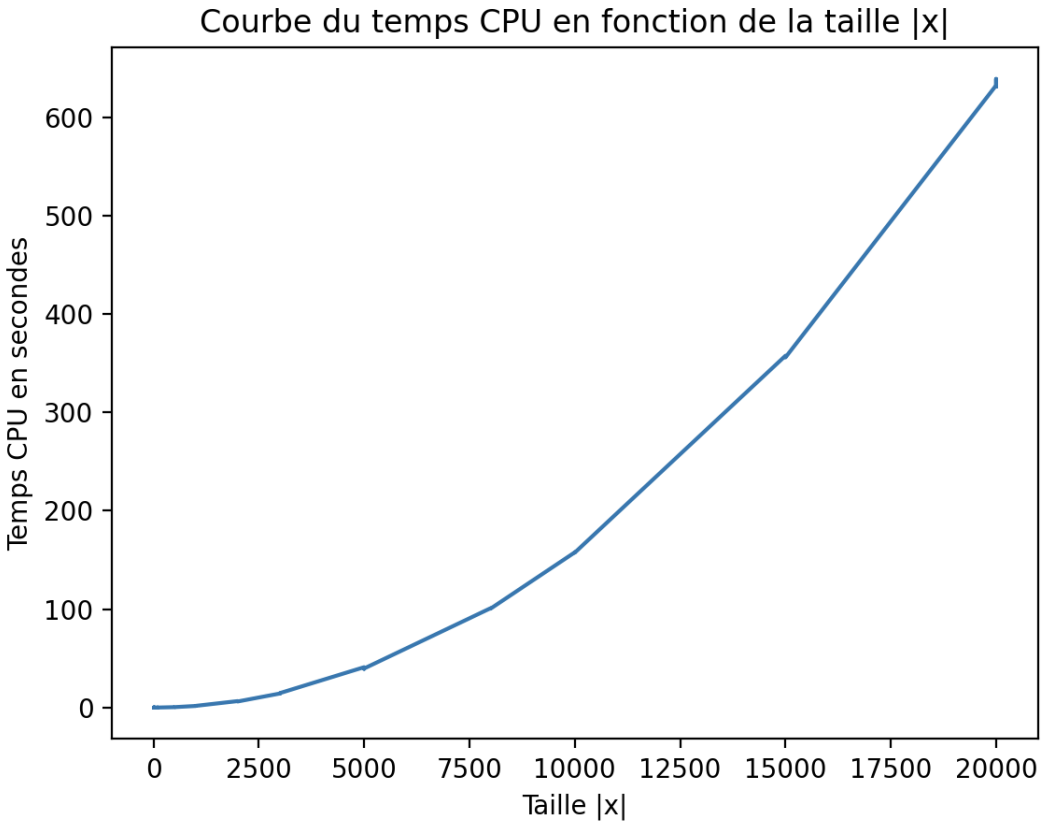
**Tache B**

*TEST DE PROG_DYN*

```
PROG_DYN de Inst_0000010_44.adn = DIST = 10 alignement = ( TATATGAGTC , TAT_T___T_
PROG_DYN de Inst_0000010_7.adn = DIST = 8 alignement = ( TGGGTGCTAT , GGGGTTCTAT
PROG_DYN de Inst_0000010_8.adn = DIST = 2 alignement = ( AACTGTCTTT , AACTGT_TTT
PROG_DYN de Inst_0000012_13.adn = DIST = 9 alignement = ( CTGGAAAGTGCG , CT_G_AACTG_G
PROG_DYN de Inst_0000012_32.adn = DIST = 6 alignement = ( CCATTGATTTTC , _CATTGA_TTT_
PROG_DYN de Inst_0000012_56.adn = DIST = 9 alignement = ( GCTTAACTAACG , GCTAAACT_ACT
PROG_DYN de Inst_0000013_45.adn = DIST = 2 alignement = ( CGGGGGGTAGCAA , CGGGGGGT_GCAA
PROG_DYN de Inst_0000013_56.adn = DIST = 9 alignement = ( GCTTAACTAAC_GA , GCATAACT__CAGA
PROG_DYN de Inst_0000013_89.adn = DIST = 8 alignement = ( TCCGCAAGCGTGT , TCGGCAA_CGTCT
PROG_DYN de Inst_0000014_23.adn = DIST = 15 alignement = ( CCCCACAGGGCTAG , __AGACA_GGC_AG
```

*CALCUL DU TEMPS CPU DE* $PROG\_DYN$

Après calcul, nous nous limitons aux mots x de longueur au plus 20 000 car après ceux-ci le calcul de prog_dyn(x,y) prend plus de 10 minutes.
À l'aide des instances fournies, on obtient la courbe suivante :




On observe que la courbe correspond bien à la complexité théorique trouvée à la question 17 : $O(n^2)$

***CONSOMMATION MEMOIRE DE*** $PROG\_DYN$

On a mesuré la quantité de mémoire utilisée pour l'instance $Inst\_0015000\_3.adn$ et on obtient 244077 octets. Pour des petites instance on obtient environ 13000 octets.

```
Consommation memoire pour Instances_genome/Inst_0000013_45.adn : 19329
```

# 4 Amélioration de la complexité spatiale du calcul de la distance

### Question 19

On a T[i,j] = min(T[i,j-1] + $c_{ins}$,T[i-1,j]+$c_{del}$, T[i-1,j-1] + $c_{sub}(x_i, y_j)$).
Pour avoir T[i,j] il faut avoir accès a T[i-1,j], T[i-1,j-1] et T[i,j-1] c'est-à-dire aux lignes i-1 et i et aux colonnes j-1 et j.

### Question 20

```
DIST_2
    Entrée : x et y
    Sortie : distance d'édition de (x,y)
    n = |x|
    m = |y|
    T = [2][m+1]        On crée un tableau de 2 lignes et de m+1 colonnes
    For i from  0 to 1 :
        T[i][0] = c_del*i
    For i from 0 to m :
        T[0][j] = c_ins * j

    For i from 1 to n :
        For j from 1 to m :
            T[1][j]  = min(T[0][j] + c_del , T[1][j-1] + c_ins, T[0][j-1] + c_sub(x[i-1],y[j-1])
        For k from 0 to m :
            T[0][k] = T[1][k]       La deuxième ligne du tableau devient la premiere ligne

    retourner T
```

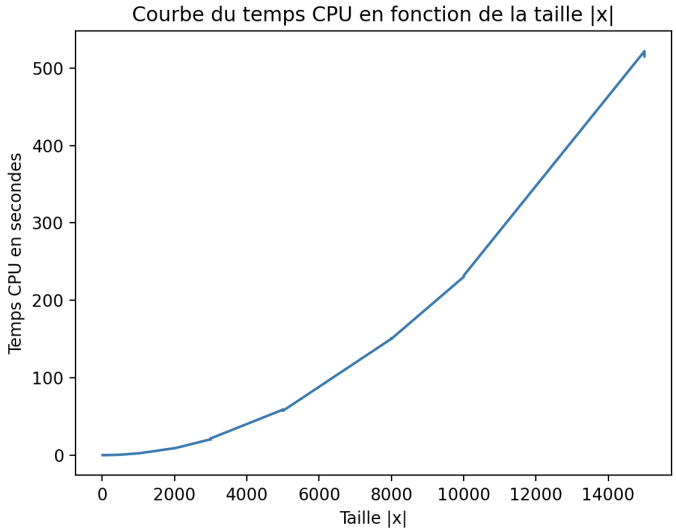
**Tache C**

*TEST de DIST_2*

```
DIST_2 pour l'instance Inst_0000010_44.adn : 10
DIST_2 pour l'instance Inst_0000010_7.adn : 8
DIST_2 pour l'instance Inst_0000010_8.adn : 2
DIST_2 pour l'instance Inst_0000012_13.adn : 9
DIST_2 pour l'instance Inst_0000012_32.adn : 6
DIST_2 pour l'instance Inst_0000012_56.adn : 9
DIST_2 pour l'instance Inst_0000013_45.adn : 2
DIST_2 pour l'instance Inst_0000013_56.adn : 9
DIST_2 pour l'instance Inst_0000013_89.adn : 8
DIST_2 pour l'instance Inst_0000014_23.adn : 15
DIST_2 pour l'instance Inst_0000014_7.adn : 8
DIST_2 pour l'instance Inst_0000014_83.adn : 8
DIST_2 pour l'instance Inst_0000015_2.adn : 8
DIST_2 pour l'instance Inst_0000015_4.adn : 23
DIST_2 pour l'instance Inst_0000015_76.adn : 11
DIST_2 pour l'instance Inst_0000020_17.adn : 14
DIST_2 pour l'instance Inst_0000020_32.adn : 8
```

*COURBE DE TEMPS CPU DE DIST_2*

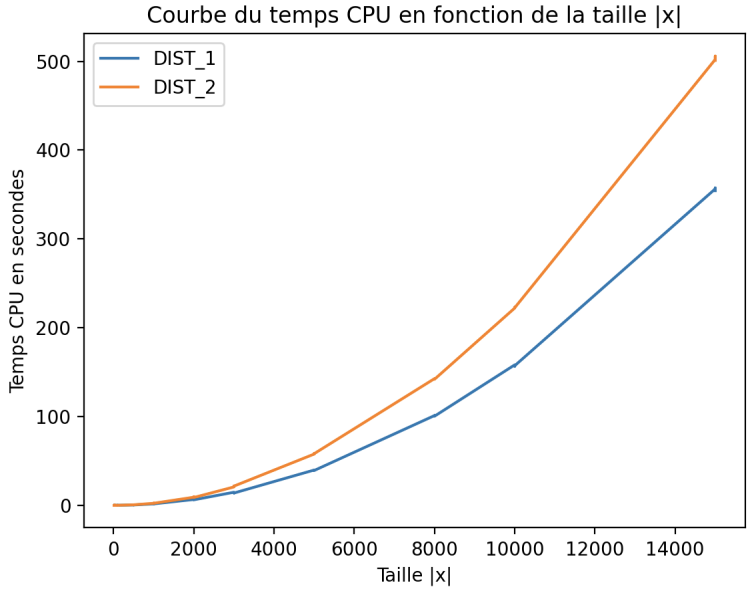
Après calcul, nous nous limitons aux mots x de longueur au plus 15 000 car après ceux-ci le calcul de DIST_2(x,y) prend plus de 10 minutes.
À l'aide des instances fournies on obtient la courbe de temps CPU de DIST_2 suivante :




La complexité théorique nous donne : O (2 + m + 1 + n ∗ ( m + m )) = O(n ∗ m).
Donc DIST_2 est en O($n^2$) car $m \leq n$, ce que l'on obtient bien sur la courbe.

*COMPARAISON* $DIST\_1$ *et* $DIST\_2$

Si l'on compare les performances de $DIST\_1$ et $DIST\_2$, on remarque que $DIST\_1$ est plus rapide $DIST\_2$ comme représenté sur le graphique ci-dessous :




*CONSOMMATION MEMOIRE DE* $DIST\_2$

On a mesuré la quantité de mémoire utilisée pour l'instance $Inst\_0015000\_3.adn$ et on obtient 244077 octets. Pour des petites instance on obtient environ 13000 octets.

```
Consommation memoire pour Instances_genome/Inst_0000013_45.adn : 19329

Consommation mémoire de Inst_0015000_3.adn 244077
```

# 5 Amélioration de la complexité spatiale du calcul d'un alignement optimal par la méthode "diviser pour régner"

### Question 21

```
Mot_gaps
    Entrée : k un entier
    Sortie : mot composé de k gaps
    s = mot vide
    For i from  1 to k
        s = s + "_"
    retourner s
```

### Question 22

Nous avons implémenté 2 fonctions auxiliaires :

- **teste_lettre(x,l)**
  Cette fonction teste si la lettre l existe dans x. Si c'est le cas elle renvoie l'indice de la première occurrence de l dans x, sinon elle renvoie -1
- **teste_complémentaire(x,l)**
  Cette fonction teste si il existe une lettre complémentaire à l dans x. Si c'est le cas elle renvoie la première occurrence de celle ci, sinon elle renvoie -1

```
teste_lettre
        Entrée x un mot et l une lettre
        Sortie : retourne l'indice de la lettre l dans x, retourne -1 si l n'est pas dans x
    for i from 0 to |x|
        if x[i] =l
            retourner i
    retourner -1

teste_complémentaire
        Entrée : un mot x et une lettre l
        Sortie : retourne l'indice de la lettre l dans x, retourne -1 si le complémentaire de l n'est pas dans x
    for i from 0 to n
        if (x[i] in {'A','T'} and l in {'A','T'}) or (x[i] in {'C','G'} and l in {'C','G'})
            retourner i
    retourner -1
```

```
align_lettre_mot
    Entré : x et y
    Sortie : alignement optimal de (x,y)


    lettre ← teste_lettre(y,x[0])
    longueur  ← |y|
    res  ← mot_gaps(longueur)
    complémentaire  ←  teste_complémentaire(y,x[0])

    if lettre >= 0         il y a une lettre similaire entre x et y
        res[lettre]  ← x[0]
        retourner (res,y)

    if complementaire >= 0
        res[complémentaire]  ← x[0]
        return (res,y)

    res[0]  ←  x[0]             On place la lettre x[0] à la premiere place dans res
    return (res,y)
```

**Question 23**

Soient $(x^1, y^1)$ = (BAL , RO) et $(x^2, y^2)$ = (LON , ND).
On remarque facilement que :
1. Un alignement optimal de $(x^1, y^1)$ est $(BAL, RO\_)$ avec une distance d'édition de 13. On le notera $(\overline{s}, \overline{t})$.
2. Un alignement optimal de $(x^2, y^2)$ est $(LON\_ , \_.\_ND)$ avec une distance d'édition de 9. On le notera $(\overline{u}, \overline{v})$.
Montrons que $(\overline{s}.\overline{u}, \overline{t}.\overline{v})$ n'est pas un alignement optimal de (x,y)
$(\overline{s}.\overline{u}, \overline{t}.\overline{v}) = (BALLON.\_, RO.\_.\_.\_.ND)$ avec une distance d'édition de 22.
Or si on prend $\overline{n} = BALLON.\_$ et $\overline{m} = R.\_.\_.\_.OND$ , on a bien $(\overline{n}, \overline{m})$ un alignement de (x,y) avec une distance d'édition de 17 < 22.
Donc $(\overline{s}.\overline{u}, \overline{t}.\overline{v})$ n'est pas un alignement minimal.

## Question 24

```
sol_2(x,y)
        Entrée : x et y
        Sortie : alignement optimal de (x,y)
        n ← |x|
        m ← |y|
        if m==0
            s ← mot_gaps(n)
            retourner (x,s)
        if n==1 and m>0
            s ← align_lettre_mot(x,y)
            retourner  s

        i ← partie entière inférieur (n/2)
        j ← coupure(x,y)

        (x1,y1) ← sol_2(x[0:i],y[0:j])
        (x2,y2)  ← sol_2(x[i:n+1],y[j:m+1])

        retourner tuple( map( lambda i,j : i+j, (x1,y1) , (x2,y2) ))
```

## Question 25

```
Coupure
Entrée : x et y
Sortie : indice j associé à i.

    n ← |x|
    m ← |y|
    i ← partie entière inférieure de (n/2)
    T ← np.zeros((2,m+1),int)
    I ← np.zeros((2,m+1),int)
    for i2 from 0 to 1
        T[i2][0]  ←  c_del*i2
        I[i2][0]  ←  0
    for j2 from 0 to m  :
        T[0][j2]  ← c_ins * j2
        I[0][j2]  ← j2

    for k from 1 to n :
        for j from 0 to m :
            if j==0 :
                T[1][j]  ← T[0][j] + c_del
            else:
                T[1][j]   ←  min(T[0][j] + c_del , T[1][j-1] + c_ins, T[0][j-1] + c_sub(x[k-1],y[j-1]))

            if k>i :
                if j==0 :
                    I[1][j]  ←  I[0][j]
                else :
                    if T[1][j] == T[0][j] + c_del :
                        I[1][j]  ←  I[0][j]
                    elif T[1][j] == T[1][j-1] + c_ins :
                        I[1][j]  ←  I[1][j-1]
                    elif T[1][j] == T[0][j-1] + c_sub(x[k-1],y[j-1]) :
                        I[1][j]  ←  I[0][j-1]

        for k2 from 0 to m  :
            if k> i :
                I[0][k2]  ←  I[1][k2]
            T[0][k2]  ← T[1][k2]
    retourner  I[1][m]
```

**Question 26**

Dans $coupure(x, y)$, on alloue seulement deux tableaux de deux lignes et m + 1 colonnes, un tableau pour D(i,j) et un tableau pour calculer l'indice j* associé à i*. Donc la complexité spatiale de $coupure(x, y)$ est O(2 ∗ (2 ∗ m + 1)) = O(m)

**Question 27**

Dans l'éxécution de $SOL\_2$ :

- Si y est le mot vide alors on retourne $mot\_gaps(n)$ donc O(n)
- Si x est un mot de longueur 1, on retourne $align\_mot\_lettre(x, y)$ en O(m)
- Sinon on fait appel à coupure , donc avec une complexité spatiale en O(m).
  On a l'inégalité suivante :

$$\frac{n}{2^k} \geq 1$$

  En effet pour chaque appel récursif on divise le mot x par 2 jusqu'à ce que |x| = 1. On a donc $k \leq log_2(n)$. Il y a au plus $2 * log_2(n)$ appels récursifs de $SOL\_2$ ( car on appelle 2 fois $SOL\_2$ à chaque fois). Chaque appel recursif appelle $coupure$ en O(m) et alloue 2 tableaux de tailles 2*(m+1) donc chaque appel est en O(m + 2*(m+1) ) = O(m).
  La complexité spatiale de $SOL\_2$ est $C(n, m) = \begin{cases} O(n) \text{ si } |y| = 0 \\ O(m) \text{ si } |x| = 1 \\ O(m * log_2(n)) \text{ sinon} \end{cases}$

**Question 28**

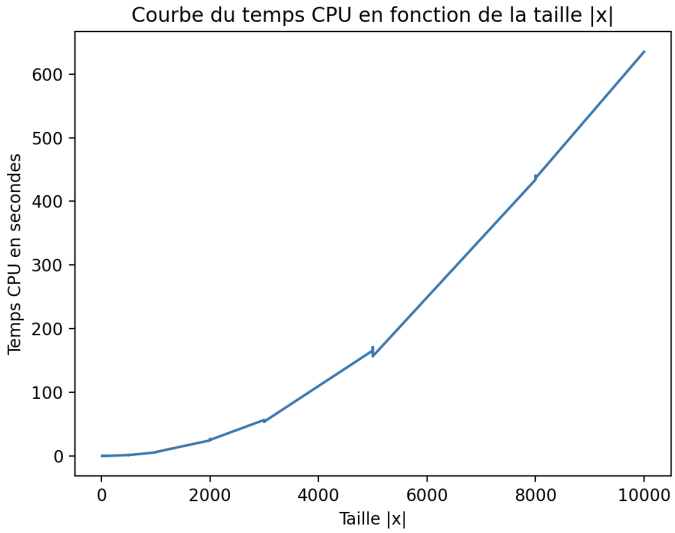
- Les instructions des lignes 1 à 5 sont en O(1)
- La 1e boucle est en O(1), la 2e en O(m) avec des instructions en O(1)
- La 3e boucle est en O(n*m) avec des instructions en O(1)
- La dernière boucle est en O(m) avec des instructions en O(1)

La complexité totale est de O(m + n*m) = $O(n * m)$

**Tache D**

*TEMPS CPU $SOL\_2$*

Après calcul, nous nous limitons aux mots x de longueur au plus 10 000 car après ceux-ci le calcul de $SOL\_2$ prend plus de 10 minutes.
À l'aide des instances fournies, on obtient la courbe suivante :




*CONSOMMATION MÉMOIRE* $SOL\_2$

Nous avons mesuré la consommation mémoire de la fonction $SOL\_2$ sur :

— Une instance de petite taille ( 12 caractères ) : Environ 19 000 octets

```
Consommation memoire pour Instances_genome/Inst_0000013_45.adn : 19329
```

— Des instances de grande taille ( 500 caractères ) : Environ 45 000 octets

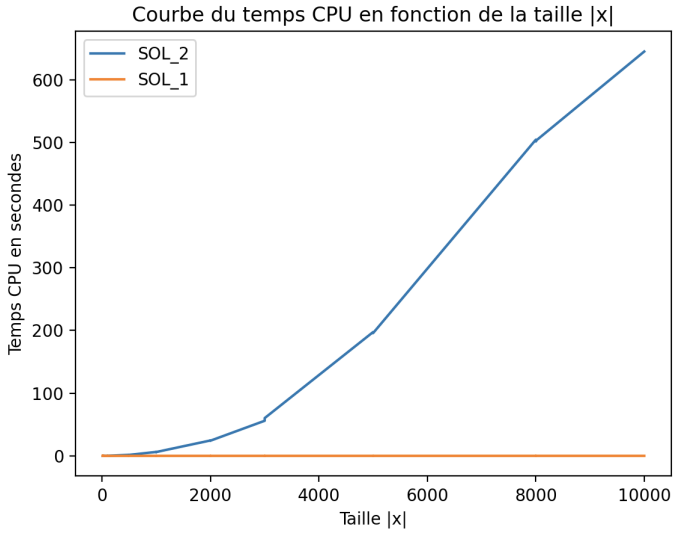
```
Consommation memoire pour Instances_genome/Inst_0000500_8.adn : 44884
```

— Des instances de grande taille ( 500 caractères )

```
Consommation memoire pour Instances_genome/Inst_0002000_3.adn : 139821
```

**Question 29**

On compare l'efficacité temporelle de $SOL\_1$ avec $SOL\_2$ en effectuant des mesures de temps et en traçant une courbe du temps CPU pour chaque fonction.
Pour des instances identiques, on obtient :




On observe que, bien que la complexité spatiale soit améliorée dans $SOL\_2$ par rapport à $SOL\_1$ car on ne calcule pas une matrice entière, la complexité temporelle est énormément dégradée. On voit sur les courbes ci-dessus que $SOL\_1$ est bien plus rapide que $SOL\_2$ . Il serait plus intéressant de comparer $PROG\_DYN$ et $SOL\_2$ car les deux fonctions construisent la matrice des valeurs D(i,j). On obtient les courbes suivantes :