NSI Terminale S4 : Algorithmique : Boyer-Moore-Horspool et Tri par fusion

Trouver les occurrences d'une chaîne de caractères (motif ou mot) dans un texte est un problème qui se rencontre fréquemment dans les éditeurs de texte. Les algorithmes de recherche de chaîne de caractères sont également utilisés pour trouver une séquence ADN par exemple ou pour rechercher des pages web correspondantes à une recherche Internet.
Exemple : on souhaite rechercher le mot exo dans le texte : « Bonjour Je fais l'exo de NSI »

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Texte	B	o	n	j	o	u	r		J	e		f	a	i	s		l	'	e	x	o		d	e		N	S	l
mot	e	x	o																									

I) Boyer-Moore-Horspool :

1. Programmation naïve :

a) Principe :

On compare le B et le e s'il n'y a pas de correspondance, on décale d'un caractère

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Texte	B	o	n	j	o	u	r		J	e		f	a	i	s		l	'	e	x	o		d	e		N	S	l
mot	e	x	o																									

On compare le o et le e s'il n'y a pas de correspondance, on décale d'un caractère

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Texte	B	o	n	j	o	u	r		J	e		f	a	i	s		l	'	e	x	o		d	e		N	S	l
mot		e	x	o																								

On compare le n et le e s'il n'y a pas de correspondance, on décale d'un caractère

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Texte	B	o	n	j	o	u	r		J	e		f	a	i	s		l	'	e	x	o		d	e		N	S	l
mot			e	x	o																							

……..
On fait de même jusqu'au premier e
On compare le « l'espace » et le x s'il n'y a pas de correspondance, on décale d'un caractère

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Texte	B	o	n	j	o	u	r		J	e		f	a	i	s		l	'	e	x	o		d	e		N	S	l
mot										e	x	o																

On compare le « l'espace » et le e s'il n'y a pas de correspondance, on décale d'un caractère

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Texte	B	o	n	j	o	u	r		J	e		f	a	i	s		l	'	e	x	o		d	e		N	S	l
mot											e	x	o															

On compare le « l'espace » et le x s'il n'y a pas de correspondance, on décale d'un caractère

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Texte	B	o	n	j	o	u	r		J	e		f	a	i	s		l	'	e	x	o		d	e		N	S	l
mot												e	x	o														

……..
On fait de même jusqu'au prochain e

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Texte	B	o	n	j	o	u	r		J	e		f	a	i	s		l	'	e	x	o		d	e		N	S	l
mot																			e	x	o							

On compare le x et le x, comme il y a correspondance, on compare le o et le o. Il y a trois correspondances correspondant à la taille du mot donc nous avons trouvé une occurrence.
…..
On se décale d'un caractère jusqu'au prochain e.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Texte	B	o	n	j	o	u	r		J	e		f	a	i	s		l	'	e	x	o		d	e		N	S	l
mot																								e	x	o		

On compare le « l'espace » et le e s'il n'y a pas de correspondance, on décale d'un caractère

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Texte	B	o	n	j	o	u	r		J	e		f	a	i	s		l	'	e	x	o		d	e		N	S	l
mot																									e	x	o	

On compare le N et le e s'il n'y a pas de correspondance, on arrête la recherche et on renvoie 1 occurrences

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Texte	B	o	n	j	o	u	r		J	e		f	a	i	s		l	'	e	x	o		d	e		N	S	l
mot																										e	x	o

b) Algorithme:

L'algorithme de cette méthode naïve est le suivant :

Algorithm 1 : recherche_naive(motif,texte)

```
1: n ⟵ longueur du texte
2: m ⟵ longueur du motif
3: compteur_occurrences ⟵ 0
4: Pour s de 0 à n − m faire
5:    Si motif[0···m] == texte[s·····s+m] alors
6:       compteur_occurrences ⟵ compteur_occurrences + 1
7:    Fin Si
8: Fin Pour
9: Renvoyer compteur_occurrences
```

c) Codage :

Écrire une fonction recherche_naive(motif,texte) avec en paramètres des strings motif et texte et qui renvoie le nombre d'occurrences du motif dans le texte.
On pourra aussi proposer un affichage du décalage de chaque occurrence trouvée. Une assertion vérifiera que le texte est de longueur supérieure ou égale au mot.
Tester avec le texte : Bonjour, je fais l'exo de NSI et le motif exo (donnés par l'utilisateur grâce à deux appels input).
Sauver sous Boyer_Moore.py et capturer le résultat.

```
*** Console de processus distant Réinitialisée ***
L'ocurrence  1  à été trouvée au  19 ième caractère
En tout, il y a :  1  occurence(s)
```

2. Algorithme de Boyer Moore Horspool :

L'algorithme de Boyer-Moore-Horspool ou Horspool est un algorithme de recherche de sous-chaîne publié en 1980 par Nigel Horspool, un professeur à l'université de Victoria au Canada. Il consiste en une simplification de l'algorithme de Boyer-Moore qui ne garde que la première table de saut.

a) Principe :

L'algorithme de Boyer-Moore effectue la vérification, c'est-à-dire qu'il tente d'établir la correspondance entre le motif et le texte à une certaine position, à l'envers.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Texte	B	o	n	j	o	u	r		J	e		f	a	i	s		l	'	e	x	o		d	e		N	S	l
mot	e	x	o																									

On va comparer le o et le n. Comme le n n'est pas dans le motif, on peut faire un saut de la taille du motif (ici 3 caractères)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Texte	B	o	n	j	o	u	r		J	e		f	a	i	s		l	'	e	x	o		d	e		N	S	l
mot				e	x	o																						

On va comparer le o et le u. Comme le u n'est pas dans le motif, on peut faire un saut de la taille du motif (ici 3 caractères)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Texte	B	o	n	j	o	u	r		J	e		f	a	i	s		l	'	e	x	o		d	e		N	S	l
mot							e	x	o																			

On va comparer le o et le J. Comme le J n'est pas dans le motif, on peut faire un saut de la taille du motif (ici 3 caractères)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Texte	B	o	n	j	o	u	r		J	e		f	a	i	s		l	'	e	x	o		d	e		N	S	l
mot								e	x	o																		

On va comparer le o et le f. Comme le f n'est pas dans le motif, on peut faire un saut de la taille du motif (ici 3 caractères)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Texte	B	o	n	j	o	u	r		J	e		f	a	i	s		l	'	e	x	o		d	e		N	S	l
mot											e	x	o															

On va comparer le o et le s. Comme le s n'est pas dans le motif, on peut faire un saut de la taille du motif (ici 3 caractères)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Texte	B	o	n	j	o	u	r		J	e		f	a	i	s		l	‘	e	x	o		d	e		N	S	l
mot																e	x	o										

On va comparer le o et le ‘. Comme le ‘ n’est pas dans le motif, on peut faire un saut de la taille du motif (ici 3 caractères)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Texte	B	o	n	j	o	u	r		J	e		f	a	i	s		l	‘	e	x	o		d	e		N	S	l
mot																			e	x	o							

On va comparer le o et le o. Comme il y a correspondance, on va comparer le x et le x et comme il y a encore correspondance, on va comparer le e et le e (là on a 3 correspondances qui correspondent à la taille du motif) on va donc ajouter 1 au compteur d’occurrences puis faire un saut de trois caractères.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Texte	B	o	n	j	o	u	r		J	e		f	a	i	s		l	‘	e	x	o		d	e		N	S	l
mot																						e	x	o				

On va comparer le o et le e. Ici e est dans le motif donc on ne peut pas décaler de 3 caractères mais seulement faire correspondre le 2 e (longueur motif - indice du e dans le motif + 1 :3-0-1=2) soit un décalage de 2 caractères.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Texte	B	o	n	j	o	u	r		J	e		f	a	i	s		l	‘	e	x	o		d	e		N	S	l
mot																								e	x	o		

On va comparer le o et le N. Comme le N n’est pas dans le motif, on peut faire un saut de la taille du motif (ici 3 caractères) mais 25+3>27 donc on peut arrêter de chercher, il ne peut plus y avoir d’occurrences.

On se rend compte qu’avec cette méthode, on fait des sauts plus grands qu’avec la méthode naïve. Ce principe de fonctionnement explique pourquoi plus le motif est long, et plus l’algorithme est efficace pour le trouver.

b) Cas particulier lorsque certaines cases coïncident :

i	0	1	2	3	4	5	6	7	8	9	10	11
Texte	C	T	G	T	G	A	G	C	A	T	G	A
mot	C	T	G	C	G	A						

Des cas plus complexes peuvent se produire lorsque certaines lettres coïncident, car le saut doit être calculé par rapport à l’indice de la lettre du motif. Par exemple si les deux lettres G et A coïncident, on teste celle en 4e position qui ne coïncide pas puisqu’on lit un T au lieu d’un C. Le T est dans le motif, on va effectuer un saut de 4 (longueur motif - indice du T dans le motif - 1 :6-1-1=4)-2 (on a deux caractères qui sont bon)=2.

i	0	1	2	3	4	5	6	7	8	9	10	11
Texte	C	T	G	T	G	A	G	C	A	T	G	A
mot			C	T	G	C	G	A				

On va superposer les deux premiers C : 7-5=2 donc faire un décalage de 2 caractères possible puisque 7+2=9<=11

i	0	1	2	3	4	5	6	7	8	9	10	11
Texte	C	T	G	T	G	A	G	C	A	T	G	A
mot					C	T	G	C	G	A		

7

On va superposer les deux premiers T: 9-5=4 donc faire un décalage de 4 caractères mais impossible puisque 9+4=13>11 donc pas d’occurrence dans cet exemple.

Pour trouver ce saut pour chaque lettre, on remarque qu’il suffit de regarder l’indice du motif (numéroté de 0 à len(motif)-1) et de calculer len(motif)-imotif-1. Si une lettre apparait plusieurs fois on prendra la dernière valeur lors du parcours du motif (elle correspondra à la plus petite). Ceci correspondra à la table des sauts qu’il faudra faire en prétraitement.

c) Codage de la table des sauts :

On va construire ce que l'on nomme une table des sauts, pour chaque caractère du motif. Celle table traduit en fait l'écart minimal entre une lettre du motif et la fin du motif. La dernière lettre du mot est traitée à part, elle renvoie un écart maximal si elle n'est pas présente ailleurs dans le mot.

Exemples :

Motif	CTGCGA			
Lettres	A et les autres lettres	G	C	T
Distance à la fin du motif	6	1	2	4

Motif	ACTGCGA				
Lettres	Les autres lettres	G	C	T	A
Distance à la fin du motif	7	1	2	4	6

Coder la fonction suivante table_sauts(motif) d'après l'algorithme suivant :
Affecter à la variable table un dictionnaire vide
Affecter à la variable i la valeur 0
Pour chaque lettre du motif sans le dernier caractère :
    Affecter à la valeur de la clé lettre du dictionnaire la taille du motif – i- 1
    Incrémenter i de 1
Renvoyer le dictionnaire table.
Tester avec les motifs exo, CTGCGA et ACTGCGA.
Sauver sous Boyer_Moore.py et capturer le résultat.

```
Table de sauts pour le motif exo: {'x': 1, 'e': 2}
Table de sauts pour le motif CTGCGA: {'T': 4, 'C': 2, 'G': 1}
Table de sauts pour le motif ACTGCGA: {'T': 4, 'A': 6, 'C': 2, 'G': 1}
```

d) Codage de la fonction Boyer-Moore-Horspool :

Soit le code suivant de la fonction boyer_moore_horspool qui renvoie une liste, des positions du motif dans le texte.

```
def boyer_moore_horspool(texte, motif):
    """In : texte et motif des chaines de caractères
    Out : une liste de la position des occurrences du motif dans le texte"""
    assert len(texte)>= len(motif)
    n = len(texte)#
    m = len(motif)
    positions = []#
    tbSauts = table_sauts(motif)#
    i=0
    trouve=False #
    while(i<=n-m):#
        for j in range (m-1,-1,-1):#
            trouve=True
            if(texte[i+j]!=motif[j]):#
                if(texte[i+j] in tbSauts and tbSauts[texte[i+j]]<=j): #
                    i+=tbSauts[texte[i+j]]#
                else:
                    i+=j+1#
                trouve=False#
                break
        if(trouve):#
            positions.append(i)#
            i=i+1#
            trouve=False #
    return positions
```

Coder celle-ci et compléter tous les commentaires après le # (surligné en jaune ci-dessus).
Tester avec le texte : Bonjour, je fais l'exo de NSI et le motif exo (donnés par l'utilisateur grâce à deux appels input).
Sauver sous Boyer_Moore.py et capturer le résultat.

```
*** Console de processus distant Réinitialisée ***
[19]
```

3. Efficacité de l'algorithme de Boyer Moore Horspool :

a) Calcul de comparaisons effectuées par la méthode naïve :

Modifier la fonction recherche_naive(motif,texte) afin d'afficher le nombre de comparaisons lors de l'appel de celle-ci.
Tester avec le texte : Bonjour, je fais l'exo de NSI et le motif exo (donnés par l'utilisateur grâce à deux appels input).
Sauver sous Boyer_Moore.py et capturer le résultat.

b) Calcul de comparaisons effectuées par la méthode Boyer Moore Horspool :

Modifier la fonction boyer_moore_horspool(texte,motif) afin d'afficher le nombre de comparaisons lors de l'appel de celle-ci.
Tester avec le texte : Bonjour, je fais l'exo de NSI et le motif exo (donnés par l'utilisateur grâce à deux appels input).
Sauver sous Boyer_Moore.py et capturer le résultat.

```
*** Console de processus distant Réinitialisée ***
L'ocurrence  1  à été trouvée au  19 ième caractère
Avec la méthode naïve, on a fait : 27 comparaison(s)
Avec la méthode Boyer Moore Horspool, on a fait : 13 comparaison(s)
```

c) Comparaison des deux algorithmes avec un texte plus long :

On va comparer les deux algorithmes sur un texte plus long en faisant varier le motif. Pour cela, on va utiliser le fichier texte : Texte_comparaison.txt qui correspond aux poèmes générés en php de la séance 11 de première année au format txt.
Pour lire le fichier et mettre dans la variable texte, coder les deux instructions suivantes :

```
fichier_texte=open("Texte_comparaison.txt","r")
texte=fichier_texte.read()
```

Tester les motifs suivants : e, lorsque, alors et « en vain cherchent leurs mots» puis afficher le nombre d'occurrences et de comparaison pour chaque méthode.
Sauver sous Boyer_Moore.py et capturer le résultat.

```
*** Console de processus distant Réinitialisée ***
Avec la méthode naïve, on a fait : 10599 comparaison(s)
En tout, il y a :  1187  occurence(s)(méthode naive) du motif : e
Avec la méthode Boyer Moore Horspool, on a fait : 10599 comparaison(s)
En tout, il y a :  1187  occurence(s)(Boyer Moore Horspool) du motif e
>>>
*** Console de processus distant Réinitialisée ***
Avec la méthode naïve, on a fait : 10593 comparaison(s)
En tout, il y a :  9  occurence(s)(méthode naive) du motif : lorsque
Avec la méthode Boyer Moore Horspool, on a fait : 2196 comparaison(s)
En tout, il y a :  9  occurence(s)(Boyer Moore Horspool) du motif lorsque
*** Console de processus distant Réinitialisée ***
Avec la méthode naïve, on a fait : 10595 comparaison(s)
En tout, il y a :  3  occurence(s)(méthode naive) du motif : alors
Avec la méthode Boyer Moore Horspool, on a fait : 2647 comparaison(s)
En tout, il y a :  3  occurence(s)(Boyer Moore Horspool) du motif alors
>>>
*** Console de processus distant Réinitialisée ***
Avec la méthode naïve, on a fait : 10572 comparaison(s)
En tout, il y a :  3  occurence(s)(méthode naive) du motif : en vain cherchent leurs mots
Avec la méthode Boyer Moore Horspool, on a fait : 1001 comparaison(s)
En tout, il y a :  3  occurence(s)(Boyer Moore Horspool) du motif en vain cherchent leurs mots
```

Conclure sur l'efficacité de l'algorithme suivant la taille du motif.

II) Tri par fusion :

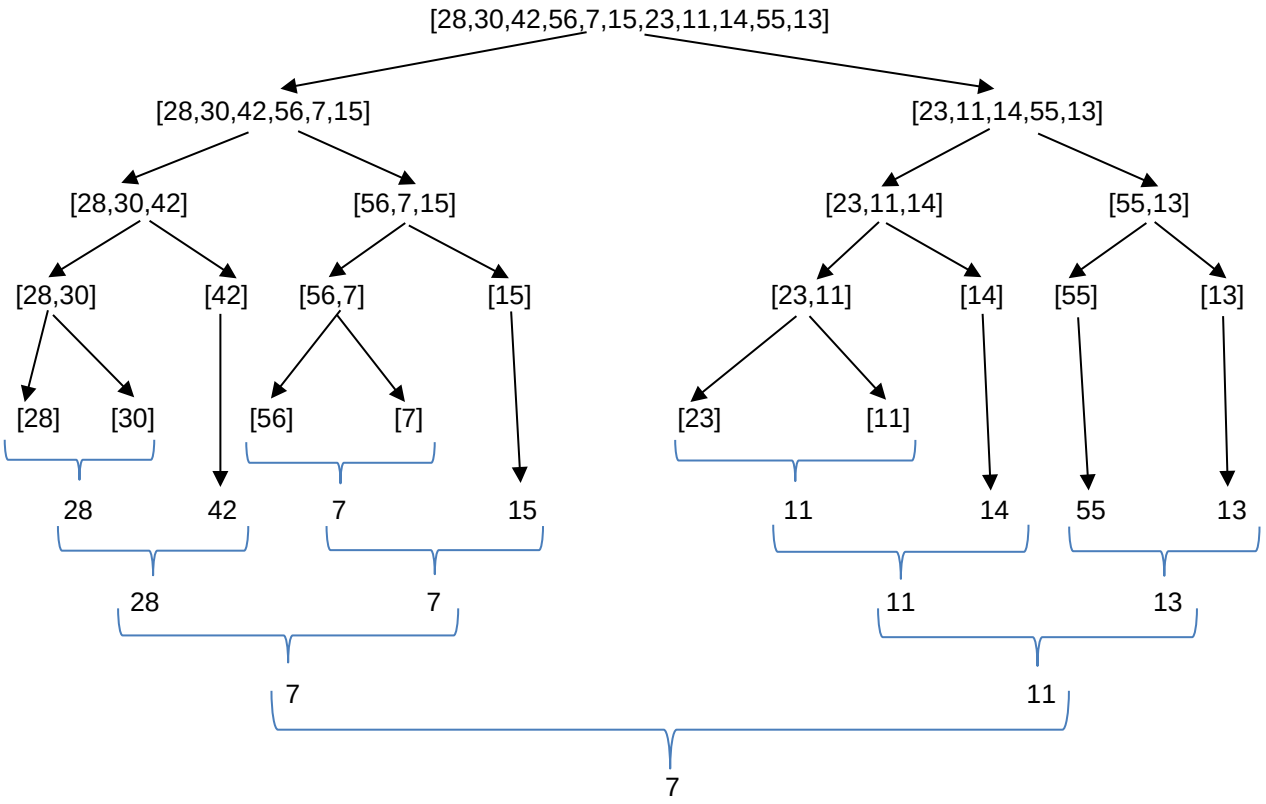
1. Diviser pour régner :

a) Principe :

La méthode diviser pour régner se décompose en trois phases :

- Diviser : on divise les données initiales en plusieurs sous-parties.
- Régner : on résout récursivement chacun des sous-problèmes (ou on le résout directement si la taille est assez petite)
- Combiner : on combine les résultats obtenus pour obtenir un résultat au problème initial.

b) Exemple d'utilisation pour la recherche du minimum d'une liste :



c) Implémentation :

Si on regarde ce schéma de principe :

- On divise la liste en deux sous-listes jusqu'à n'avoir au maximum que deux éléments.
- On calcule récursivement le minimum de chaque sous-liste.
- On arrête la récursion lorsque les listes n'ont plus qu'un seul élément
- On retourne le plus petit des deux éléments.

On peut résumer cela par l'algorithme suivant :

Soit la fonction Minimum qui a pour paramètres, une liste L, un entier d (indice du début de la liste) et un entier f (indice de fin de la liste).
Début fonction Minimum (L, d, f) :
    Si d est égale à f alors :
        Retourner L[d]
    Sinon :
        Affecter à la variable m le quotient de (d+f) par 2.
        Affecter à x le Minimum(L,d,m) (minimum de la liste de gauche par récursivité)
        Affecter à y le Minimum(L,m+1,f) (minimum de la liste de droite par récursivité)
        Si x<y alors :
            Retourner x
        Sinon :
            Retourner y
Fin fonction Minimum
Coder cette fonction et sauver-sous Fusion.py
Tester avec la liste [28,30,42,56,7,15,23,11,14,55,13]
Capturer le résultat.

De la même manière, créer une fonction Maximum d'une liste.
Coder cette fonction et sauver-sous Fusion.py
Tester avec la liste [28,30,42,56,7,15,23,11,14,55,13]
Capturer le résultat.

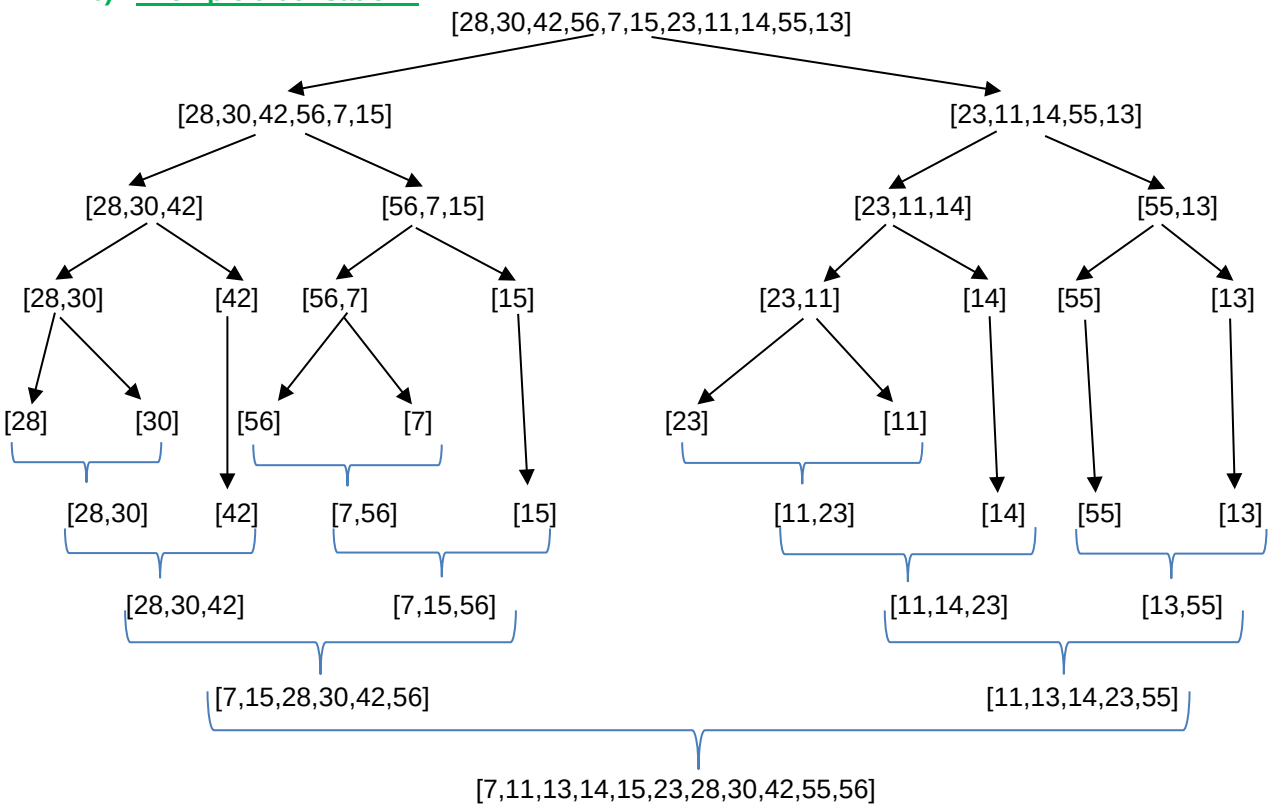
2. Tri par fusion :

a) Principe

Le tri par fusion utilise le principe "diviser pour régner". L'algorithme se décrit simplement de manière récursive :

- si la liste a 0 ou 1 élément, elle est déjà triée,
- si la liste a plus d'un élément, on la partage en deux listes et on applique le tri fusion sur chacune des deux listes.
- on fusionne les résultats.

b) Exemple d'utilisation :



c) Implémentation de la fusion :

Commençons par la fusion de deux tableaux trié (car s'il n'y a qu'un élément celui-ci est forcément trié).

Pour fusionner deux listes triées, on compare les éléments de chacune des deux listes et on copie le plus petit dans une nouvelle liste. Quand l'indice (i pour la liste1 et j pour la liste 2) est plus grand que la taille du tableau, on copie les éléments restants de la seconde liste.

Exemple :

liste1 = [7,15,28,30,42,56] et liste2=[11,13,14,23,55]

liste1	i	liste2	j	liste résultat
[7,15,28,30,42,56]	0	[11,13,14,23,55]	0	[]
[7,15,28,30,42,56]	1	[11,13,14,23,55]	0	[7]
[7,15,28,30,42,56]	1	[11,13,14,23,55]	1	[7,11]
[7,15,28,30,42,56]	1	[11,13,14,23,55]	2	[7,11,13]
[7,15,28,30,42,56]	1	[11,13,14,23,55]	3	[7,11,13,14]
[7,15,28,30,42,56]	2	[11,13,14,23,55]	3	[7,11,13,14,15]
[7,15,28,30,42,56]	2	[11,13,14,23,55]	4	[7,11,13,14,15,23]
[7,15,28,30,42,56]	3	[11,13,14,23,55]	4	[7,11,13,14,15,23,28]
[7,15,28,30,42,56]	4	[11,13,14,23,55]	4	[7,11,13,14,15,23,28,30]
[7,15,28,30,42,56]	5	[11,13,14,23,55]	4	[7,11,13,14,15,23,28,30,42]
[7,15,28,30,42,56]	5	[11,13,14,23,55]	5	[7,11,13,14,15,23,28,30,42,55]
[7,15,28,30,42,56]	6	[11,13,14,23,55]	5	[7,11,13,14,15,23,28,30,42,55,56]

L'algorithme est le suivant :
Début fonction fusion(liste1,liste2) :
    Créer une variable liste vide.
    Affecter à i et j la valeur 0
    Tant que i est inférieur à la longueur de la liste1 et j inférieure à la longueur de la liste2 :
        Si liste1[i] est inférieur ou égal à liste2[j] alors:
            Ajouter à liste, l'élément i de liste1
            Incrémenter i
        Sinon :
            Ajouter à liste, l'élément j de liste2
            Incrémenter j
    Tant que i est inférieur à la longueur de la liste1 : #la liste 1 n'a pas été copiée totalement
        Ajouter à liste, l'élément i de liste1
        Incrémenter i de 1.
    Tant que j est inférieur à la longueur de la liste2 : #la liste 2 n'a pas été copiée totalement
        Ajouter à liste, l'élément j de liste2
        Incrémenter j de 1.
    Retourner la liste.
Fin de la fonction

Coder cette fonction et sauver-sous Fusion.py
Tester avec liste1 = [7,15,28,30,42,56] et liste2=[11,13,14,23,55]
Capturer le résultat.

d) Implémentation du tri par fusion :

L'algorithme est le suivant :
Début de la fonction tri_fusion(liste)
    Si la liste a 0 ou 1 élément (longueur de liste <2) alors :
        Renvoyer la liste (car elle est déjà triée)
    Sinon :
        Calculer le milieu=quotient de la longueur de la liste avec la valeur 2.
        #on la partage en deux listes
        liste1= tri_fusion(liste[:milieu]) # on applique le tri fusion la liste du début au milieu-1.
        liste2= tri_fusion(liste[milieu :]) # on applique le tri fusion la liste du milieu à la fin.
        Renvoyer la fusion de la liste1 et la liste2 #on fusionne les résultats.
Fin tri_fusion

Coder cette fonction et sauver-sous Fusion.py
Tester avec la liste [28,30,42,56,7,15,23,11,14,55,13]
Capturer le résultat.

III) Complexité des tris :

a) Rappel tri par sélection du minimum

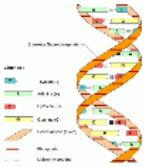
On souhaite trier par ordre croissant un tableau par sélection du minimum, pour cela visualiser la vidéo : Tri_selection_minimum.mp4
Donner un algorithme pour la fonction tri_minimum(t)
Coder cette fonction et sauver sous Fusion.py.
Tester votre fonction avec tab= [28,30,42,56,7,15,23,11,14,55,13] et afficher le résultat.

b) Rappel tri par insertion

On souhaite trier par ordre croissant un tableau par insertion, pour cela visualiser la vidéo : Tri_insertion.mp4
Donner un algorithme pour la fonction tri_insertion(t)
Coder cette fonction et sauver sous Fusion.py.
Tester votre fonction avec tab= [28,30,42,56,7,15,23,11,14,55,13] et afficher le résultat.

c) Rappel tri à bulles

On souhaite trier par ordre croissant un tableau par tri à bulle, pour cela visualiser la vidéo : Tri_bulle.mp4
Algorithme :
Début tri_bulle(t)
    Affecter à permutation la valeur booléenne Vrai
    Affecter à passage la valeur 0



```
    Affecter à nb la taille du tableau
    Tant que permutation est Vrai faire :
            Affecter à permutation la valeur booléenne Faux
            Incrémenter passage de 1
            Pour i variant de 0 à nb-passage faire :
                    Si t[i]>t[i+1] :
                            Affecter à permutation la valeur booléenne Vrai
                            Echanger t[i] et t[i+1]
                    Fin Si
            Fin Pour
    Fin tant que
    Renvoyer t
```

Coder cette fonction et sauver sous Fusion.py.
Tester votre fonction avec tab= [28,30,42,56,7,15,23,11,14,55,13] et afficher le résultat.

d) Mesure de la complexité de chaque tri

Le but de cette partie va être de faire un graphe pour chaque type de tri. En abscisse nous aurons le nombre d'éléments dans un tableau à trier et en ordonné le nombre de comparaisons faites pour trier le tableau.
Commençons par modifier les fonctions afin de récupérer le nombre de comparaisons à l'aide de variables globales cpt…

Rajouter sous les fonctions les variables compteurs dont la valeur est 0.

```
cpt_fusion=0
cpt_minimum=0
cpt_insertion=0
cpt_bulle=0
```

Dans chaque fonction déclarer la bonne variable en global et incrémenter celle-ci à chaque test.

```
def fusion(liste1,liste2):
    liste=[]
    i,j=0,0
    global cpt_fusion
    while i<len(liste1)and j<len(liste2):
        cpt_fusion+=1
        if liste1[i]<=liste2[j]:
            liste.append(liste1[i])
            i+=1
        else:
            liste.append(liste2[j])
            j+=1
    while i<len(liste1):
        cpt_fusion+=1
        liste.append(liste1[i])
        i+=1
    while j<len(liste2):
        cpt_fusion+=1
        liste.append(liste2[j])
        j+=1
    return liste
```

```
def tri_minimum(t):
    global cpt_minimum
    nb = len(t)
    for i in range (nb):
        min=t[i]#au départ on dit que le min est le premier élément
        posmin=i#au départ la poisiton du min est le premier élément
        for j in range(i+1,nb):
            cpt_minimum+=1
            if t[j]<min:
                min=t[j]#on récupère la valeur du min
                posmin=j#on récupère la poisition du min
        #on echange i et j
        t[posmin]=t[i]
        t[i]=min
    return t
```

```
def tri_insertion(t):
    global cpt_insertion
    nb = len(t)
    for i in range (1,nb):
        aux=t[i]
        j=i-1
        while(j>=0 and aux<t[j]):
            cpt_insertion+=1
            t[j+1]=t[j]
            j=j-1
        t[j+1]=aux
    return t
```

```
def tri_bulle(t):
    global cpt_bulle
    permutation = True
    passage = 0
    nb=len(t)
    while permutation == True:
        permutation = False
        passage = passage + 1
        for i in range(0, nb - passage):
            cpt_bulle+=1
            if t[i] > t[i + 1]:
                permutation = True
                # On echange les deux elements
                temp=t[i]
                t[i]=t[i+1]
                t[i+1]=temp
    return t
```

Sauver sous Fusion.py.

Afficher le compteur pour chaque tri du tab= [28,30,42,56,7,15,23,11,14,55,13] et capturer le résultat.

Attention, il faut après chaque tri remettre tab à sa valeur initiale, car une fois la fonction appelée, le tableau est déjà trié pour le tri suivant.

```
cpt_fusion=0
tri_fusion(L)
print("Compteur tri par fusion:",cpt_fusion)
L=[28,30,42,56,7,15,23,11,14,55,13]
cpt_minimum=0
tri_minimum(L)
print("Compteur tri par sélection du minimum :",cpt_minimum)
L=[28,30,42,56,7,15,23,11,14,55,13]
cpt_insertion=0
tri_insertion(L)
print("Compteur tri par insertion :",cpt_insertion)
L=[28,30,42,56,7,15,23,11,14,55,13]
cpt_bulle=0
tri_bulle(L)
print("Compteur tri bulle :",cpt_bulle)
```

Quel tri fait le moins de comparaisons donc le plus rapide ?

Est-ce toujours le cas ? Pour répondre à cette question, nous allons le faire avec des tableaux de tailles de plus en plus grande.

Commençons par créer tous les tableaux dont on aura besoin :

```
Tab_fusion=[] #tab à trier pour fusion
Tab_minimum=[] #tab à trier pour minimum
Tab_insertion=[] #tab à trier pour insertion
Tab_bulle=[] #tab à trier pour bulle
```

```
Tab_cpt_fusion=[]#Stockage de cpt_fusion pour chaque taille des tab triés
Tab_cpt_minimum=[]#Stockage de cpt_minimum pour chaque taille des tab triés
Tab_cpt_insertion=[]#Stockage de cpt_insertion pour chaque taille des tab triés
Tab_cpt_bulle=[]#Stockage de cpt_bulle pour chaque taille des tab triés

Tab_tailles=[]#Stockage de la taille des différents tableaux triés (Axe des x)
```

Nous allons demander ensuite à l'utilisateur la taille maxi du tableau et le pas (car si on lui demande 100 000 avec un pas de 1 cela va prendre trop de temps).

```
t_max=int(input("Donner le nombre maximum d'élément dans le tableau a trier : "))
pas=int(input("Entrer le pas de chaque tableau : "))
```

Commençons par créer ces tableaux de façons aléatoire et les copier dans chaque variable, afin que l'on tri bien chaque fois le même tableau avec chaque type de tri.
N'oublier pas d'importer la bibliothèque random

```
from random import *
```

```
for i in range (1,t_max+pas,pas):
    #On vide les tableaux à trier
    Tab_fusion.clear()#on vide le tableau fusion
    Tab_minimum.clear()#on vide le tableau minimum
    Tab_insertion.clear()#on vide le tableau insertion
    Tab_bulle.clear()#on vide le tableau bulle
    #on affecte 0 à tous les compteurs globaux
    cpt_fusion=0
    cpt_minimum=0
    cpt_insertion=0
    cpt_bulle=0
    #Création du tableau à trier
    for j in range (0,i):
        Tab_fusion.append(randint(0,9999))#on créer un tab aléatoire de i éléments
    Tab_minimum=Tab_fusion.copy()#copie de ce tableau dans minimum
    Tab_insertion=Tab_fusion.copy()#copie de ce tableau dans insertion
    Tab_bulle=Tab_fusion.copy()#copie de ce tableau dans bulle
```

Maintenant, il faut trier et stocker les résultats pour chaque tri

```
    #On fait les 4 tris
    tri_fusion(Tab_fusion)#tri par fusion
    tri_minimum(Tab_minimum)#tri par sélection du minimum
    tri_insertion(Tab_insertion)#tri par insertion
    tri_bulle(Tab_bulle)#tri bulle
    #On stocke les compteurs et la taille tu tableau
    Tab_cpt_fusion.append(cpt_fusion)#stockage de la valeur du compteur fusion
    Tab_cpt_minimum.append(cpt_minimum)#stockage de la valeur du compteur minimum
    Tab_cpt_insertion.append(cpt_insertion)#stockage de la valeur du compteur insertion
    Tab_cpt_bulle.append(cpt_bulle)#stockage de la valeur du compteur bulles
    Tab_tailles.append(j)#Stockage de la taille du tableau trié
```

Afficher tous les tableaux de compteur et la taille.
Tester votre programme avec une taille maxi de 100 éléments et un pas de 10.

```
Donner le nombre maximum d'élément dans le tableau a trier : 100
Entrer le pas de chaque tableau : 10
Compteur fusion: [0, 39, 94, 154, 223, 293, 363, 440, 520, 600, 680]
Compteur minimum: [0, 55, 210, 465, 820, 1275, 1830, 2485, 3240, 4095, 5050]
Compteur insertion: [0, 27, 113, 232, 375, 662, 906, 1256, 1498, 1778, 2412]
Compteur bulle: [0, 55, 195, 459, 784, 1275, 1802, 2457, 3174, 4040, 5022]
Tailles des tableaux triés: [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

Les tableaux ne sont pas très lisibles surtout si l'on teste avec 100 000 valeurs. Faisons cela sous forme de graphique.

Nous allons utiliser la librairie matplotlib : `from matplotlib import pyplot`
Il faut commencer par configurer la fenêtre graphique :

```
figure = pyplot.figure(figsize = (8, 8)) #taille de la fenêtre graphique en inches
pyplot.gcf().subplots_adjust(left = 0.11, bottom = 0.1,right = 0.9, top = 0.95, wspace = 0.41, hspace = 0.6)
```

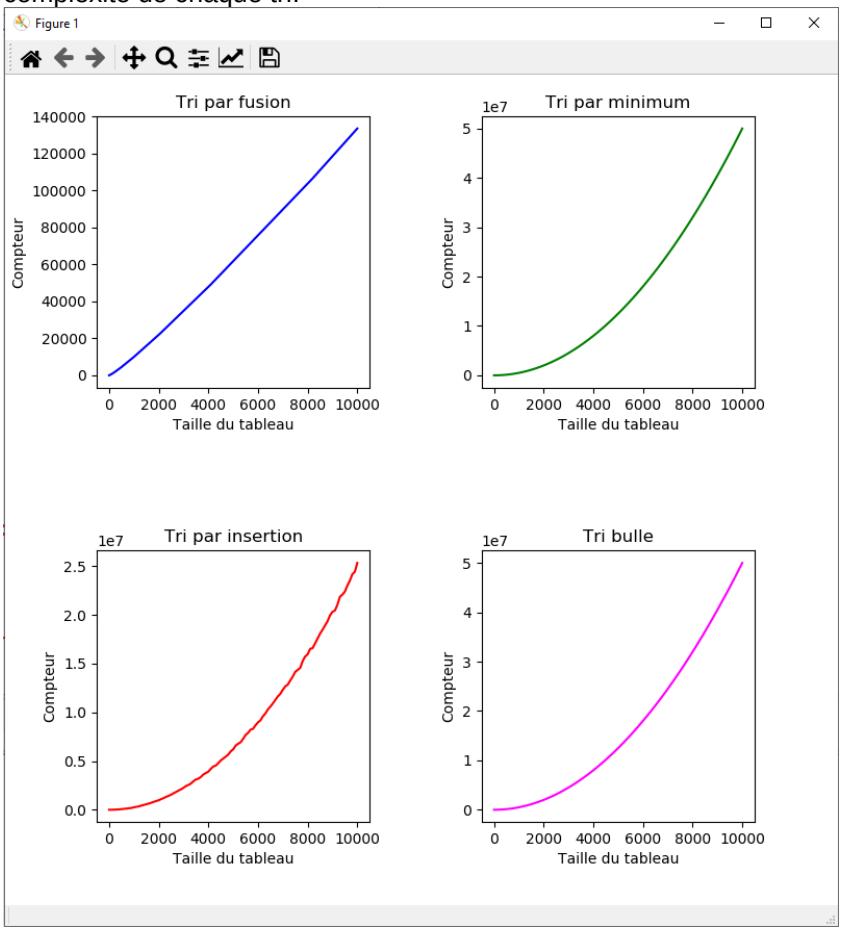
Puis reste à afficher chaque courbe :

```
axes = figure.add_subplot(2, 2, 1) # dans la fenêtre il y aura 2 graphes par colonne et 2 par ligne. On configure le premier
axes.set_xlabel('Taille du tableau')#Légende axe des X
axes.set_ylabel('Compteur')#Légende axe des Y
axes.set_title('Tri par fusion')#Titre du graphe
pyplot.plot(Tab_tailles,Tab_cpt_fusion, color = 'blue')#tableau X, y et couleur de la courbe bleu
```

Faire de même pour les 3 autres graphes :
- Tri par minimum avec une courbe en vert (green)
- Tri par insertion avec une courbe en rouge (red)
- Tri bulle avec une courbe en magenta.

Finir par l'affichage du graphe : `pyplot.show()`
Sauver-sous Fusion.py et tester avec un tableau max de 1000 éléments et un pas de 10 pour les autres.
Capturer le résultat.

On constate que le tri par fusion est le plus efficace. Cela est confirmé par le tableau de complexité de chaque tri.



Type de tri	Complexité
Tri Minimum	$O(n^2/2)$
Tri Insertion	$O(n^2/4)$
Tri bulle	$O(n^2/2)$
Tri fusion	O(n log (n))