Algorithmie
Epreuve pratique
Tri fusion

1 Spécifications

1.1 Spécifications du programme Python

Ecrire en langage Python une fonction nommée **tri_ fusion(T)** qui prend en argument une liste Python et renvoie une nouvelle liste triée, en utilisant la méthode du tri fusion.

Le tri fusion repose sur le principe de **Diviser pour régner** ; Il faudra bien mettre en évidence les 3 principales étapes :

- DIVISER : le problème d'origine est divisé en un certain nombre de sous-problèmes
- RÉGNER : on résout les sous-problèmes (les sous-problèmes sont plus faciles à résoudre que le problème d'origine)
- COMBINER : les solutions des sous-problèmes sont combinées afin d'obtenir la solution du problème d'origine.

2 Résolution

2.1 Des algorithmes de tri-fusion

2.1.1 Algorithme permettant la fusion

```
1 FONCTION FUSION_ ITERATIF(T1 : tableau trié,T2 :tableau trié)
2 |  T = [;
3 |  ]
4 |  TANT QUE T1 ≠ [] and T2 ≠ [] FAIRE
5 |  |  SI T1[O] < T2[0] ALORS
6 |  |  |  Ajouter T1[0] À T supprimer T1[0] À T1
7 |  |  SINON
8 |  |  |  Ajouter T2[0] À T supprimer T2[0] À T2
9 |  RENVOYER T + T1 + T2 # Concaténation de T,T1 et T2
```

2.1.2 Algoritme de tri fusion

```
1  VARIABLE
2  T : liste d'entiers
3  long(T) : longueur de T
4  DEBUT
5  Fonction TRI_FUSION(T)
6  |  SI long(T) <= 1 ALORS
7  |  |  Retourner T
8  |  milieu = long(T) // 2
9  |  T1 = T[0 :milieu] #slicing
10 |  T2 = T[milieu :] #slicing
11 |  Retourner fusion_iteratif (tri_fusion(T1),tri_fusion(T2))
12 end
```

⚠ Le slicing, bien que très pratique, n'est pas explicitement dans les programmes de NSI. Il peut être du coup très intéressant de trouver une solution alternative aux lignes 10 et 11.

```
1  VARIABLE
2  T : liste d'entiers
3  long(T) : longueur de T
4  DEBUT
5  Fonction tri_fusion(T)
6  |  SI long(T) <= 1 ALORS
7  |  |  Retourner T
8  |  n = len(T)
9  |  milieu = long(T) // 2
10 |  q = n - milieu
11 |  T1 = [0]*milieu
12 |  T2 = [0]*q
13 |  POUR i DE 0 À (milieu-1) FAIRE
14 |  |  T1[i] ← T[i]
15 |  POUR i DE 0 À (q-1) FAIRE
16 |  |  T2[i] ← T[i+q]
17 |  Retourner fusion_iteratif (tri_fusion(T1),tri_fusion(T2))
18 end
```

ou bien encore :

```
1  VARIABLE
2  T : liste d'entiers
3  long(T) : longueur de T
4  DEBUT
5  Fonction tri_fusion(T)
6  |  SI long(T) <= 1 ALORS
7  |  |  Retourner T
8  |  n = len(T)
9  |  milieu = long(T) // 2
10 |  T1 = [ ]
11 |  T2 = [ ]
12 |  POUR i DE 0 À n-1 FAIRE
13 |  |  SI i < milieu ALORS
14 |  |  |  T1.append(T[i])
15 |  |  SINON
16 |  |  |  T2.append(T[i])
17 |  Retourner fusion_iteratif (tri_fusion(T1),tri_fusion(T2))
18 end
```

2.2 Implémentation en Python

2.2.1 Une implémentation de la fusion

```
def fusion_iteratif(l1,l2):
    for i in range(len(l1)-1):
        assert l1[i] < l1[i + 1]
    for i in range(len(l2)-1):
        assert l2[i] < l2[i + 1]
    l = []
    while l1 != [] and l2 != []:
        if l1[0] < l2[0]:
            l.append(l1[0])
            l1.pop(0)
        else :
            l.append(l2[0])
            l2.pop(0)
    return l + l1 + l2
```

2.2.2 Une implémentation du tri fusion

```
def tri_fusion(T):
    if len(T) <= 1:
        Return T
    n = len(T)
    milieu = len(T) // 2
    T1 = []
    T2 = []
    for i in range(0,n):
        if i < milieu :
```

```
            T1.append(T[i])
        else:
            T2.append(T[i])
    return fusion_iteratif(tri_fusion(T1),tri_fusion(T2))
```

2.3 Complexité

Propriété 0.1

La complexité de l'algorithme de tri fusion est en $\mathcal{O}(nlog_2(n))$.

3 Questions possibles

☛ Créer une fonction fusion_ recursif qui produit le même résultat que fusion_ iteratif, mais en utilisant la récursivité.
☛ Bien savoir que le slicing n'est pas au programme de NSI, et savoir trouver une solution pour contourner ce problème.