Partie 1

Etude du tri fusion sur la liste `L = [1,10,8,4,3,6]`

Le tri fusion présente l'avantage d'utiliser la méthode *Diviser pour Regner.*

Les étapes 1 et 2 de la méthode *Diviser pour Regner* consistent à diviser la liste en 2 sous-listes, de manière recursive, jusqu'à obtenir des listes de 1 élément. Lorsque la liste contient un nombre impairs d'éléments, la division va créer une sous liste *gauche* contenant un élément de moins que la sous-liste *droite.*

> Question 1 : Compléter la séquence avec les sous-listes créées, jusqu'à ce que tout l'arbre soit "divisée" (sur le diagramme en annexe).

L'algorithme du tri-fusion est naturellement décrit de façon récursive (dans toute la suite, on supposera pour simplifier que le nombre d'éléments de la liste est une puissance de 2) :

- si la liste n'a qu'un ou aucun élément, elle est déjà triée,
- sinon :
  - on sépare la liste initiale en deux listes de même taille,
  - on applique récursivement l'algorithme sur ces deux listes,
  - on fusionne les deux listes triées obtenues en une seule liste triée.

> Question 2 : Compléter le script de la fonction **`fusion`** qui trie une liste par fusion de manière recursive.

```python
def fusion(L):
    if len(L) <= ... :
        return L
    m = len(L)//2
    gauche = ...
    droite = ...
    return interclassement(gauche,droite)
```

La **remontée** dans la pile d'appels commence lorsque l'on arrive à une sous-liste d'un seul élément pour *droite.*

La **fonction `interclassement`** prend deux listes en paramètres, L1 et L2, et retourne une seule liste avec les éléments de L1 et L2, mais classés.

> Question 3 : Décrire les étapes de la remontée (interclassement), jusqu'à ce que la liste L soit triée, en complétant le diagramme en annexe.

La **complexité** de la fonction `interclassement` est O(n), où n est égal à la taille de chaque sous-liste.

> Question 4 : Expliquer pourquoi, lors de la fusion, la complexité au rang n est donnée par la formule de recurence :

$$T(n) = 2 \times T(\tfrac{n}{2}) + n$$

> Question 5 : En déduire la complexité du tri fusion.

Question 6 : Donner les étapes successives du tri par insertion, puis du tri par selection pour cette même liste `L = [1,10,8,4,3,6]`. Quelle est la complexité algorithmique, dans le pire des cas, pour chacun de ces algorithmes de tri ?

Annexe

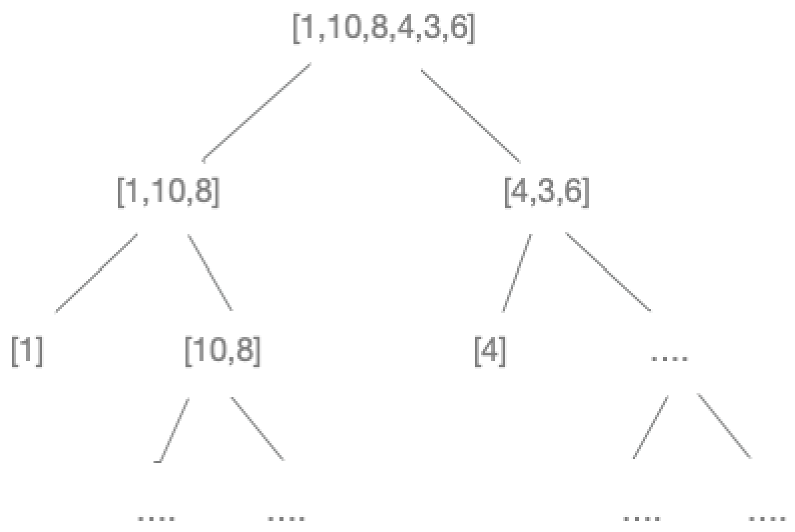

FIGURE 1 – les étapes du tri fusion

Partie 2

(extrait du sujet) : Bac 2023 Centres etrangers J4

L'exercice porte sur le tri de la fonction `sommes = [140, 78, 115, 94, 46, 108, 55, 53]`

On trouve ci-dessous un exemple de déroulement de cet algorithme pour la liste `[140,78,115,94]` de 4 éléments :

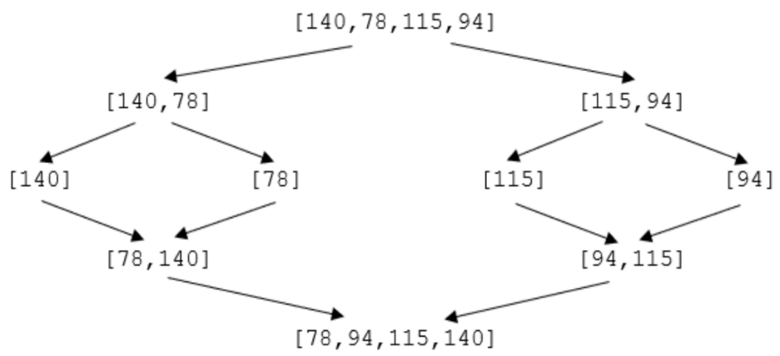

FIGURE 2 – division et tri fusion

a. Appliquer sur votre copie, de la même façon que dans l'exercice précédent, le déroulement de l'algorithme pour la liste suivante : `[46, 108, 55, 53]`

On rappelle que l'appel `L.append(x)` ajoute l'élément x à la fin de la liste L et que `len(L)` renvoie la taille de la liste L.

b. La fonction `fusion` ci-dessous renvoie une liste de valeurs triées à partir des deux listes `liste1` et `liste2` préalablement triées. Ecrire sur la copie l'instruction à placer à la ligne 4 du code de la fonction fusion :

```
1  def fusion(liste1, liste2):
2      liste_finale = []
3      i1, i2 = 0, 0
4      à compléter
5          if liste1[i1] <= liste2[i2]:
6              liste_finale.append(liste1[i1])
7              i1 =i1 + 1
8          else:
9              liste_finale.append(liste2[i2])
10             i2 = i2 + 1
11     while i1 < len(liste1):
12         liste_finale.append(liste1[i1])
13         i1 = i1 + 1
14     while i2 < len(liste2):
15         liste_finale.append(liste2[i2])
16         i2 = i2 + 1
17     return liste_finale
```

FIGURE 3 – script de la fonction fusion

Choisir parmi les propositions :

```python
while i1 < len(liste1) and i2 < len(liste2):
while i1 < len(liste1) or i2 < len(liste2):
```

Justifiez votre choix.

c. Recopier et compléter sur la copie la fonction récursive `tri_fusion` suivante, qui prend en paramètre une liste non triée et la renvoie sous la forme d'une nouvelle liste triée.

On utilisera la fonction `fusion` de la question précédente.

Exemple : `tri_fusion([140,115,78,94])` doit renvoyer `[78,94,115,140]`

Aide Python : si L est une liste, l'instruction `L[i:j]` renvoie la liste constituée des éléments de `L` indexés de `i` à `j-1`. Par exemple, si `L=[5,4,8,7,3]`, `L[2:4]` vaut `[8,7]`

```
def tri_fusion(liste):
  if len(liste)<=1:
    return liste
  # a completer
```

Partie 3

Diviser pour regner :l'algorithme d'exponentiation

$$y = x^n$$

L'exponentiation consiste à trouver une méthode pour calculer x à la puissance n, SANS utiliser l'opérateur *puissance*. L'idée est de se rappocher de l'algorithme utilisé par le processeur d'un ordinateur, qui n'utilise que les 3 opérateurs de base pour effectuer les calculs (+,-,*).

3.1 Programme récursif

```python
def exp1(n,x):
    """
    n : entier
    x : reel
    exp1 : reel
    """
    if n==0 : return 1
    else : return exp1(n-1,x)*x
```

Question : montrer que la complexité est en O(n).

3.2 Exponentiation rapide : application de la méthode diviser pour regner

Comme de nombreux algorithmes utilisant cette méthode, celui-ci fait des appels recursifs. Mais à la différence du précédent, l'appel recursif se fait avec un paramètre que l'on divise par 2 (le paramètre n). C'est ce qui fait que le nombre d'appels récursifs est plus réduit.

On retrouve l'étape 3 évoquée en introduction (la combinaison des sous problèmes) lorsque l'on réalise l'opération : `return y*y` ou bien `return x*y*y`.

```python
def exp2(n,x):
    """
    programme plus efficace que le precedent car
    le nombre d'operations est log2(n)
    """
    if n== 0 : return 1
    else :
        y = exp2(n//2,x)  # on prend la valeur inferieure de n/2
        if n%2==0:
            return y*y
        else : return x*y*y
```

Prenons pour exemple n = 8 :

- Dans la phase de descente : exp2(8,x) appelle exp2(4,x) appelle exp2(2,x) qui appelle exp2(1,x) puis exp2(0,x).
- Puis dans la phase de remontée :
  - exp2(0,x) retourne 1
  - exp2(1,x) retourne ...

- exp2(2,x) retourne ...
- exp2(4,x) retourne ...
- exp2(8,x) retourne ...

Questions :

a. Lorsque l'on execute `exp2(8,x)`, dans la phase de descente, combien y-a-t-il d'appels recursifs ?

b. Completer les fonctions de x pour les étapes proposées ci-dessus.

c. Comment calculer 12^8 à l'aide de cette fonction `exp2` ?

d. Quelle est la complexité temporelle de l'algorithme de l'exponentiation rapide ?