Stratégies "Diviser pour régner"
Tri fusion

I) Rappels : Fonctions récursives

- Cas de base
- Cas récursifs

Terminaison

Pour toute entrée, ~~après~~ ~~un nom~~ après un nombre fini d'appels récursifs, on tombe sur un cas de base.

Correction

Par récurrence

II) "Diviser pour régner"

A) Principe

- Cas de base
- Cas récursif :
  1) "Diviser" les problème en sous-problèmes
  2) "Régner" : Re
  3) "Rassembler" :

11

par u

B) exemple, exponentiation rapide

```
def puissance (x, n):
    # Cas de base (à écrire en 2°)
    if n == 0:
        return 1
    if n == 1:
        return x

    # Cas récursif
    # Diviser : n = 2k + r
    k = n // 2
    r = n % 2
    # régner : ~~y = x^k et z = x^r~~
    y = puissance (x, k)
    z = puissance (x, r)
    # Rassembler : x^n = x^k * x^k * x^r
    return y * y * z
```

3/

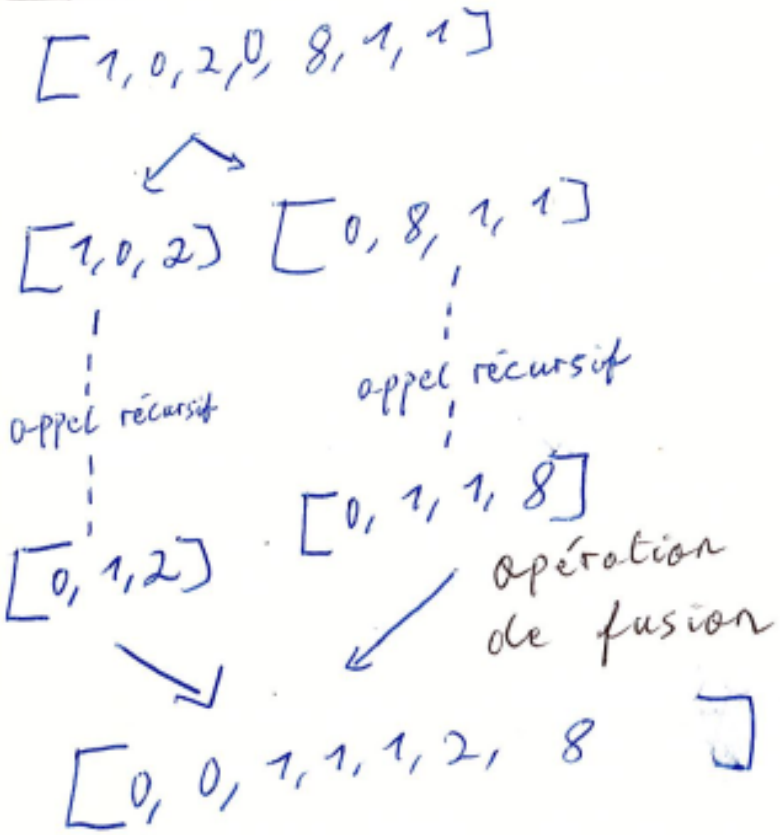
III) Tri Fusion

A) Code

```
def trifusion (l):
    n = len (l)
    ~~m = n // 2~~
    # Cas de base (à remplir après cas récursif)
    if n <= 1:
        return l

    # Diviser
    debut = l[ : m]
    fin = l[m : ]

    # Régner
    debut_trie = trifusion (debut)
    fin_triee = trifusion (fin)

    # Rassembler
    return fusion (debut_trie, fin_triee)
```

4/ Next = 134

B) Exemple



C) Fusion

```
def fusion(l1, l2):
    '''
    Entrée
        l1 et l2 listes de nombres, triées
    Sortie
        Permutation triée de l1+l2
    '''
    ~~n = len(l1)~~
    resultat = []
    i1 = 0
    i2 = 0
    for ires in range(~~len(l1)+len(l2)~~):
        if i1 == len(l1):
            resultat.append(l2[i2])
            i2 = i2+1
        elif i2 == len(l2):
            resultat.append(l1[i1])
            i1 = i1+1
        elif l1[i1] < l2[i2]:
            resultat.append(l1[i1])
            i1 = i1+1
        elif l2[i2] <= l1[i1]:
            resultat.append(l2[i2])
            i2 = i2+1
    return resultat
```

Suite p. 138

E) Stabilité (question + temps réflexion d'abord)

Le tri fusion est stable
ssi la fusion, en cas d'égalité, privilégie l'élément de gauche

D) Validité indices pour fusion
(En parallèle de C))

La comparaison entre
$l_1[i_1]$ et $l_2[i_2]$ ne peut se faire que si

- $i_1 < n_1$
- $i_2 < n_2$

- Si $i_1 \geq n_1$, on a déjà pris tous les éléments de l_1, il n'y a plus qu'à finir de remplir résultat avec l_2.
- (Symétriquement pour $i_2 \geq n_2$)

Remarque
En python, quand on évalue c_1 or c_2, si c_1 s'évalue vers True.

7/