Partie 6 : Tris récursifs de tableau

- Diviser pour régner.
- Tri fusion
- Tri rapide

Diviser pour régner

Trois étapes :

1. Diviser le problème de taille n en plusieurs sous-problèmes de tailles plus petites.

2. Résoudre les sous-problèmes (généralement de façon récursive).

3. Combiner les solutions des sous-problèmes pour obtenir une solution du problème initial.

Tri Fusion

Supposons que l'on dispose d'un algorithme qui construit un tableau trié à partir de deux tableaux triés.

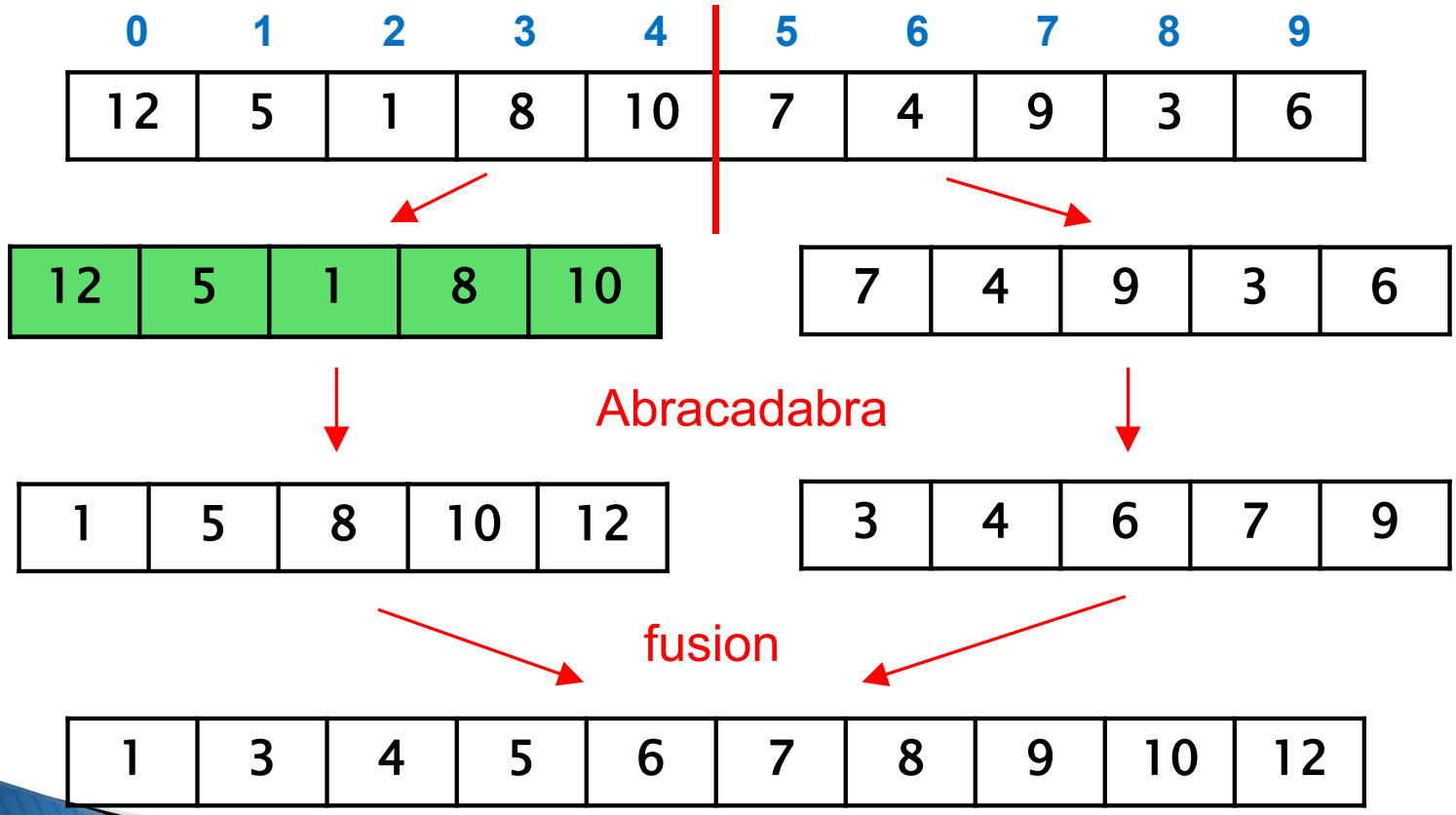
Une solution appliquant l'approche diviser pour régner est la suivante :

1. Diviser le tableau en deux tableaux de tailles identiques.

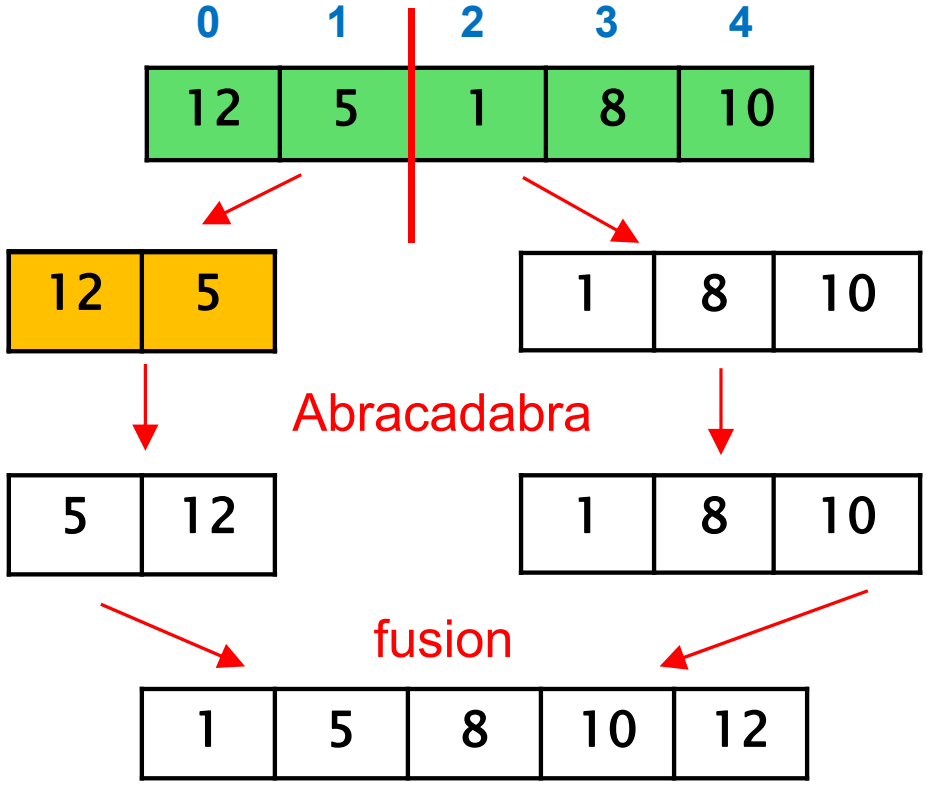
2. Trier récursivement les deux sous-tableaux.

3. Combiner les deux sous-tableaux triés en un seul tableau trié.

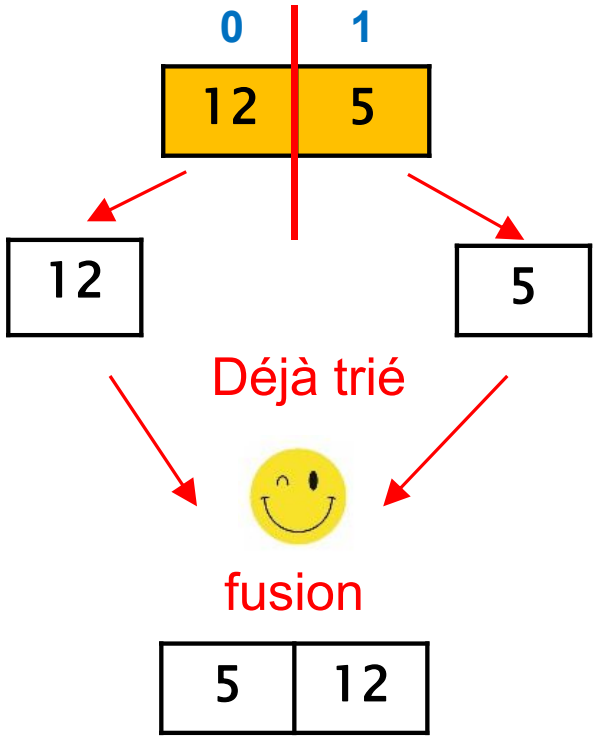
Tri fusion


0 1 2 3 4 5 6 7 8 9
12 5 1 8 10 7 4 9 3 6
12 5 1 8 10
7 4 9 3 6
Abracadabra
1 5 8 10 12
3 4 6 7 9
fusion
1 3 4 5 6 7 8 9 10 12

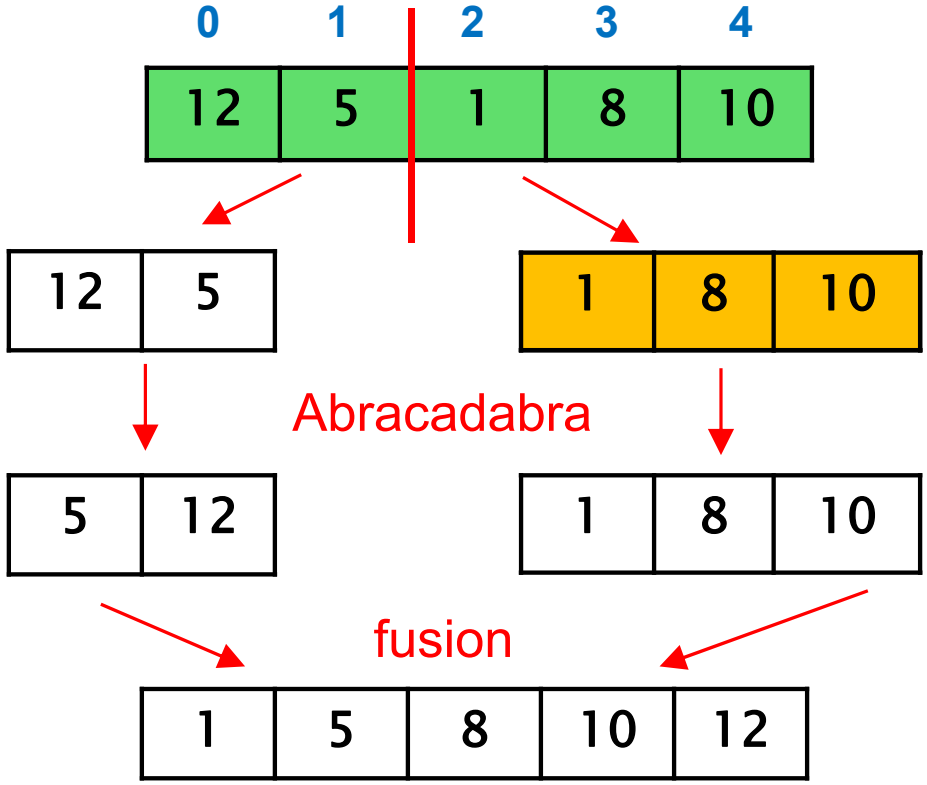
Abracadabra ?



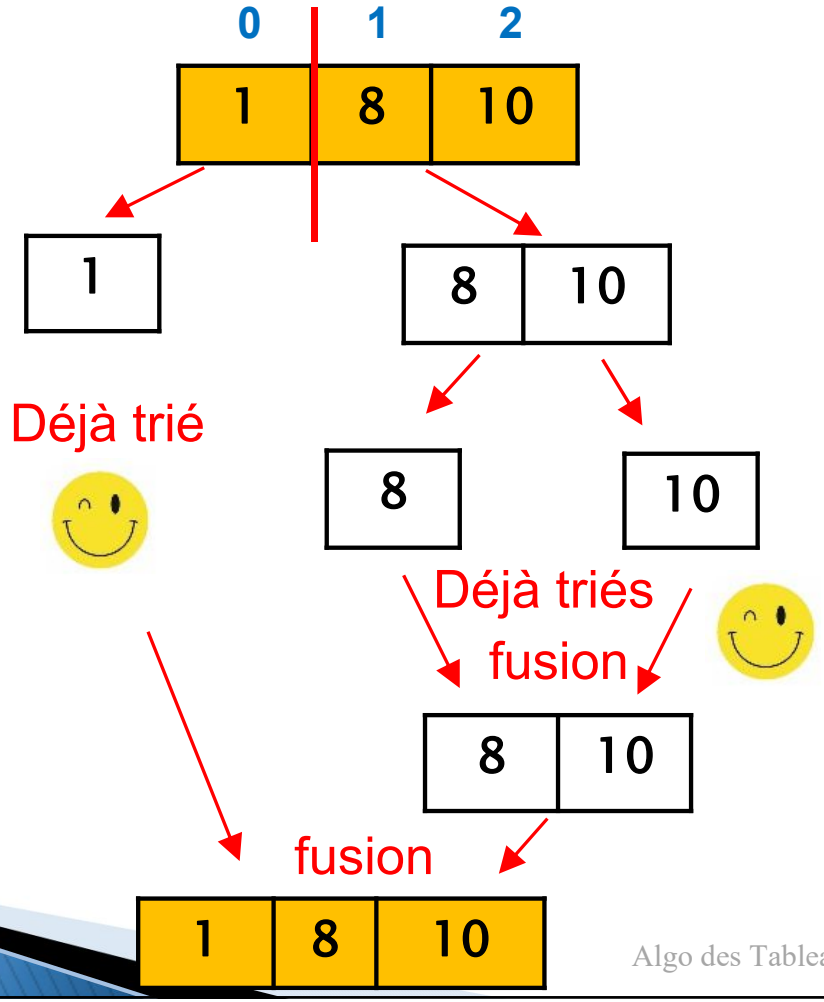
Abracadabra ?



Abracadabra ?



Abracadabra ?



Tri fusion

Exemple :
https://www.youtube.com/user/AlgoRythmics/videos

Fusion de 2 tableaux triés

```
def fusion(t, d, f):
  #tableau supplémentaire pour stocker la fusion
  r=creerTableau(f-d)
  m=(d+f)//2
  # t est trié entre d et m-1
  # t est trié entre m et f-1
   i1=d
   i2=m
   k=0

   ...
```

Fusion de 2 tableaux triés

```
def fusion(t, d, f):
    r=creerTableau(f-d)
    m=(d+f)//2
    i1=d
    i2=m
    k=0
    while (i1<m and i2 <f):
        if (t[i1] <= t[i2]):
            r[k] = t[i1]
            i1=i1+1
        else :
            r[k] = t[i2]
            i2=i2+1
        k=k+1
    while (i1 <m):
        r[k] = t[i1]
        i1=i1+1
        k=k+1
    while (i2 <f):
        r[k] = t[i2]
        i2=i2+1
        k=k+1
    for k in range(f-d):
        t[d+k]=r[k]
```

Complexité en temps : Θ(f-d+1)

Complexité en mémoire : Θ(f-d+1)

Tri fusion

```
def triFusion (t,n,d,f):
    if (d<f-1):
        m=(d+f)//2
        triFusion(t,n,d,m)
        triFusion(t,n,m,f)
        fusion(t,n,d,f)
```

1er Appel :

```
triFusion(t,n,0,n)
```

Complexité en temps :
$\Theta(n\log_2(n))$

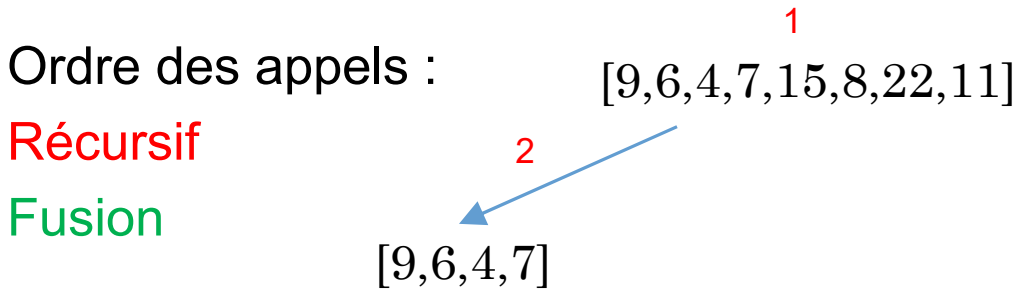
Complexité en espace :
$\Theta(\log_2(n))$ sur la pile
$\Theta(n)$ en mémoire

Stabilité : oui car
<= dans fusion

Ordre des appels :

Récursif

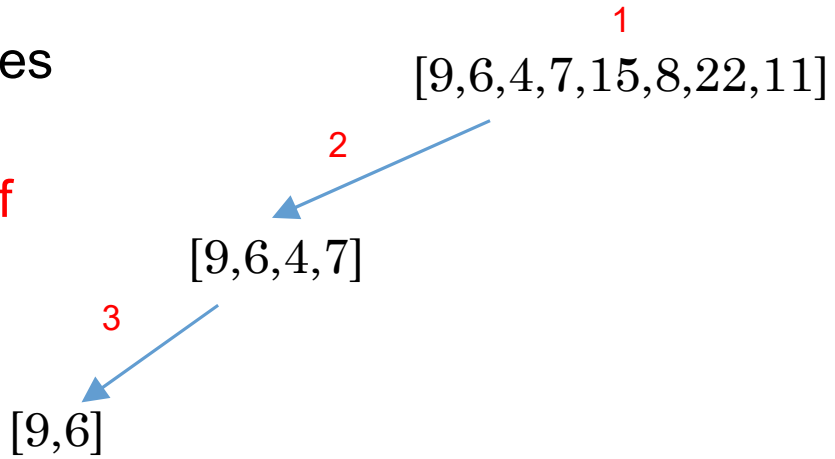
Fusion



Ordre des appels :

Récursif

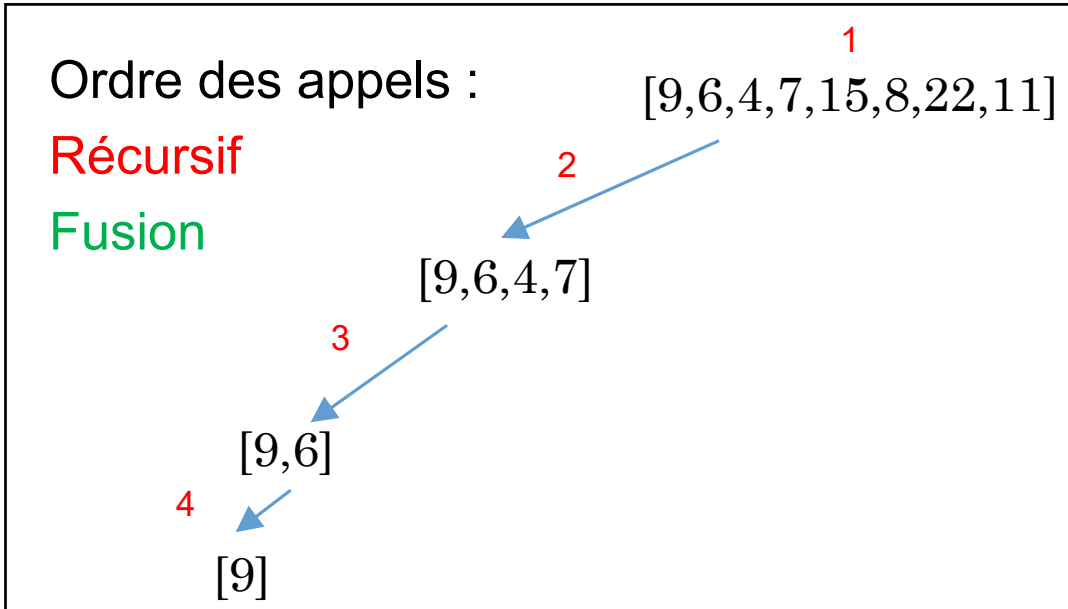
Fusion



Ordre des appels :

Récursif

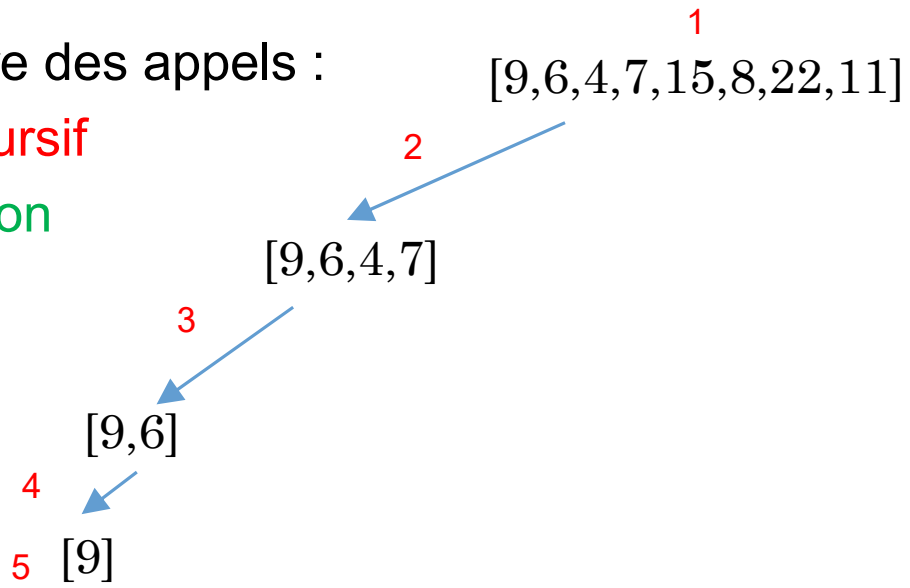
Fusion



Ordre des appels :

Récursif

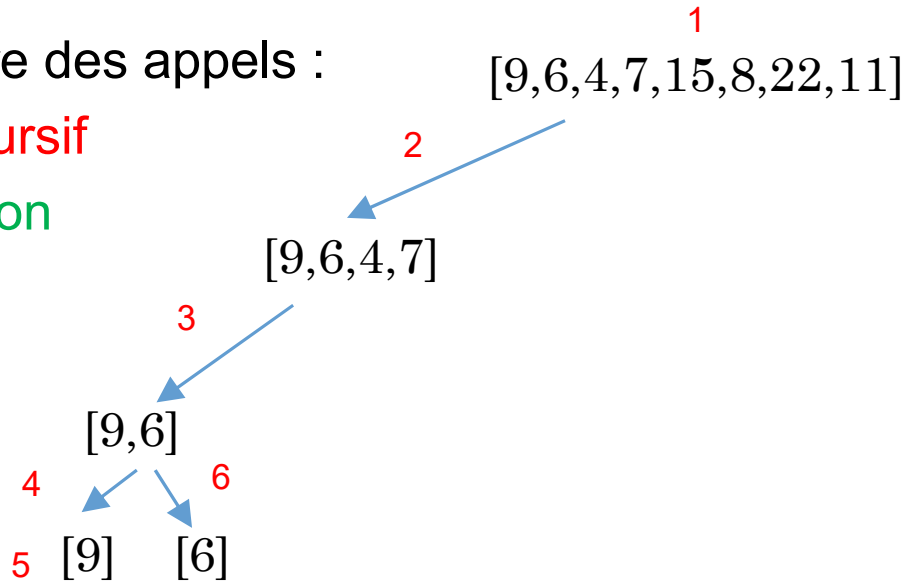
Fusion



Ordre des appels :

Récursif

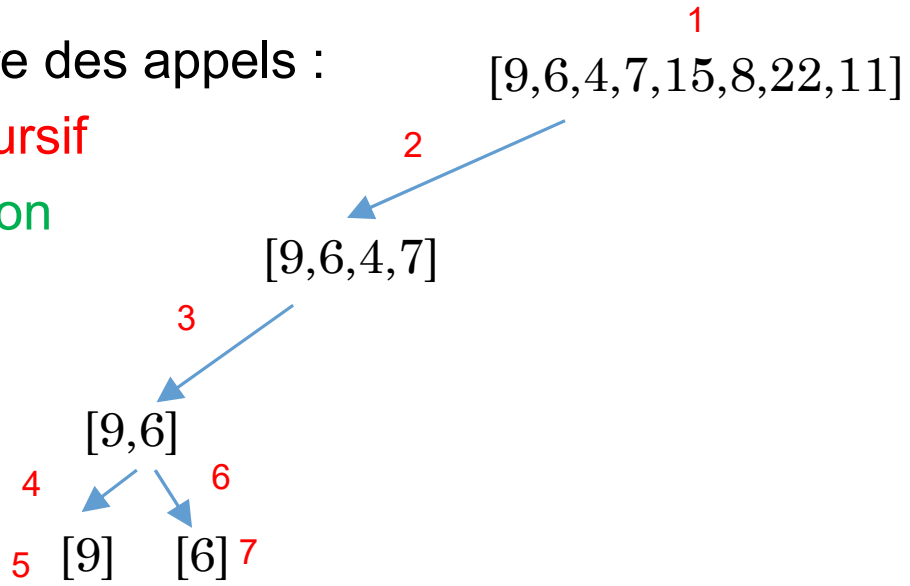
Fusion



Ordre des appels :

Récursif

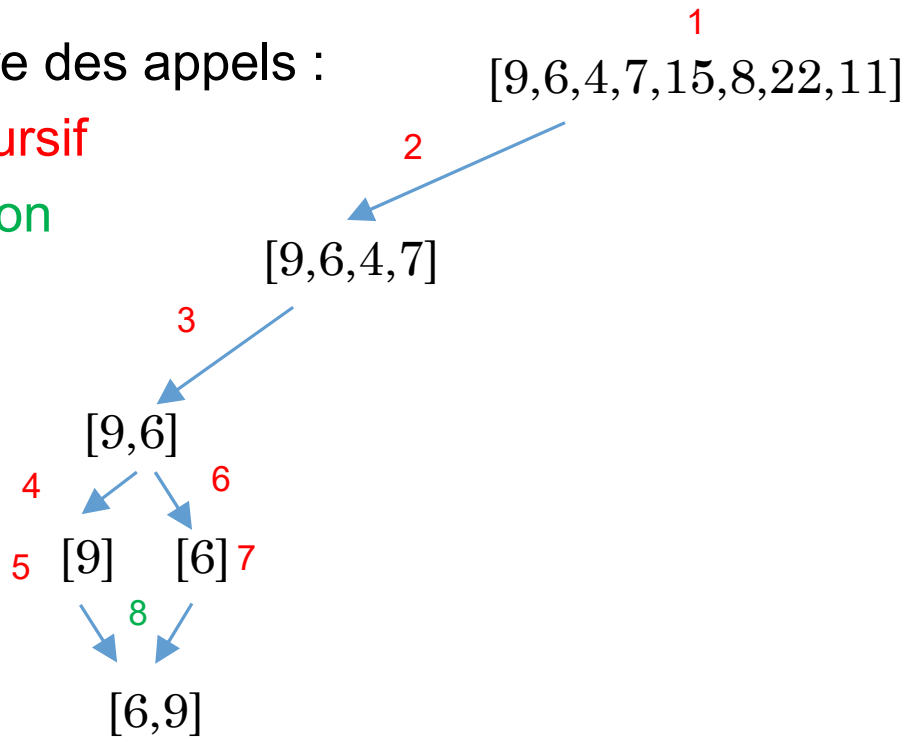
Fusion



Ordre des appels :

Récursif

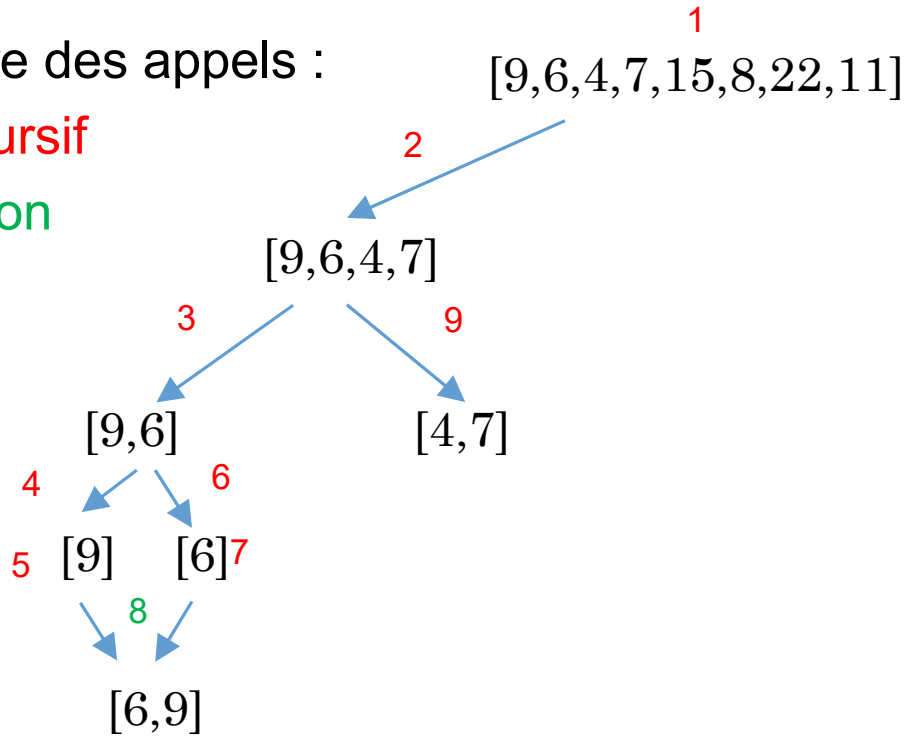
Fusion



Ordre des appels :

Récursif

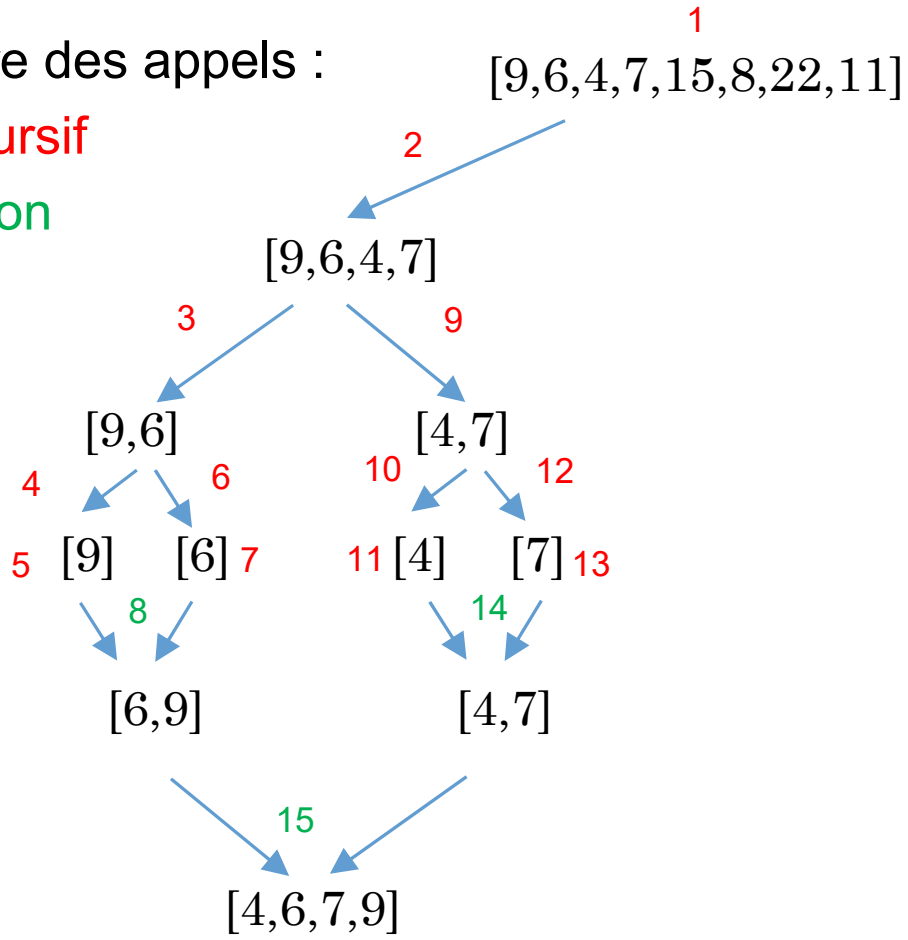
Fusion



Ordre des appels :

Récursif

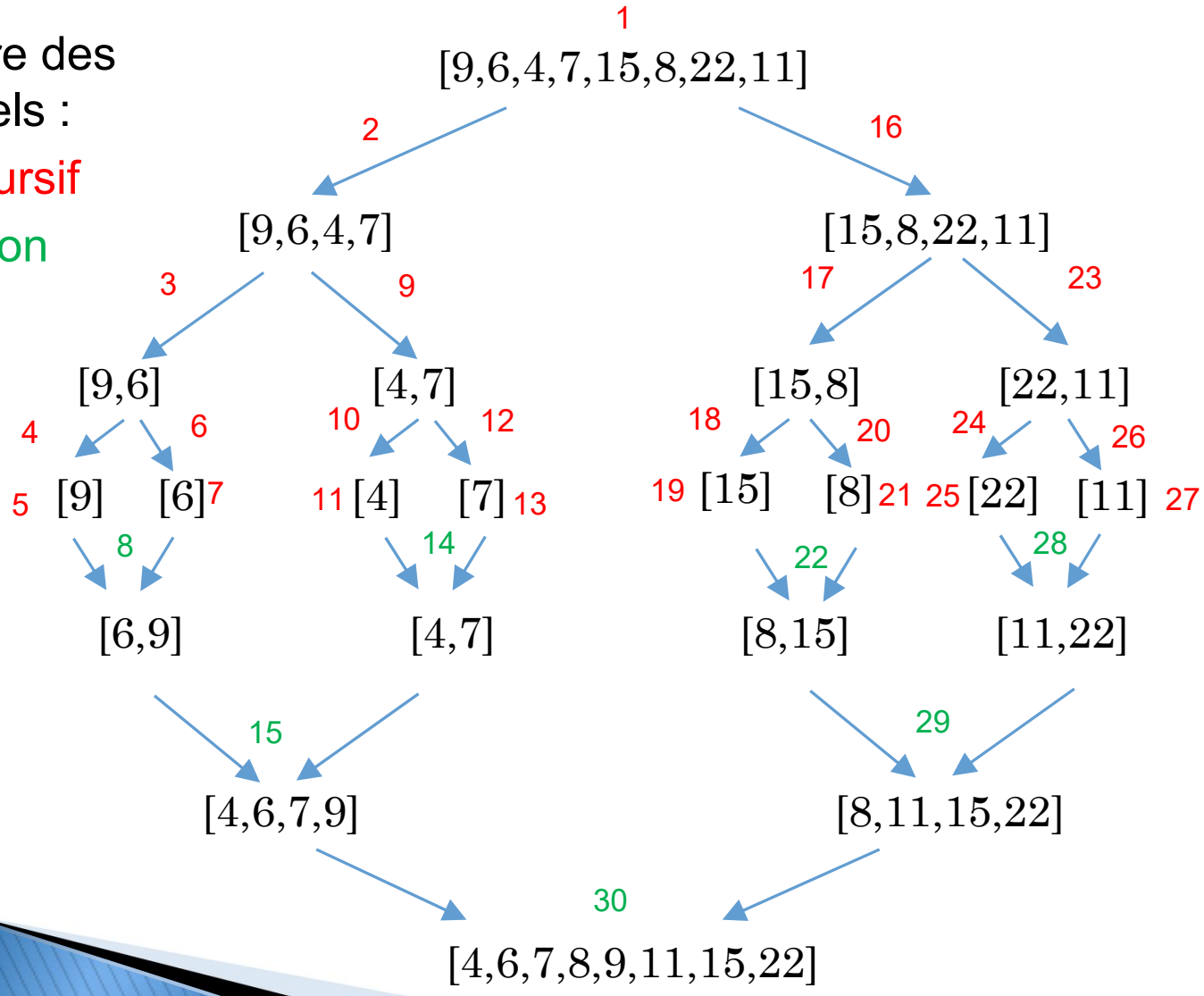
Fusion



Ordre des appels :

Récursif

Fusion



Fonction mystère

```
def mystere (t,n):
  i=0
  j= n-1
  while(i < j):
    if (t[i] == 0):
      i= i+1
    else :
      echanger(t,i,j)
      j= j-1
```

Que fait-elle en supposant que le tableau T ne contient que des 0 et des 1