Partie 10 : 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.

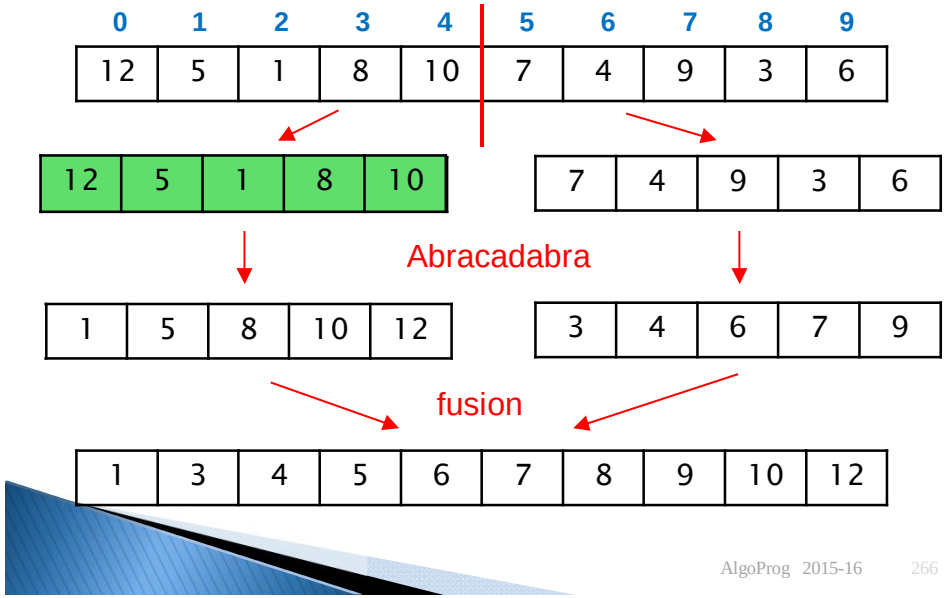
Une solution appliquant l'approche diviser pour régner est la suivante :

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

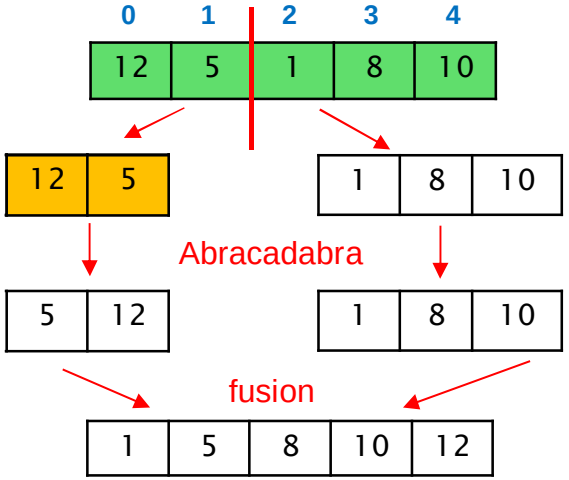
2. Trier récursivement les deux sous-tableaux.

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

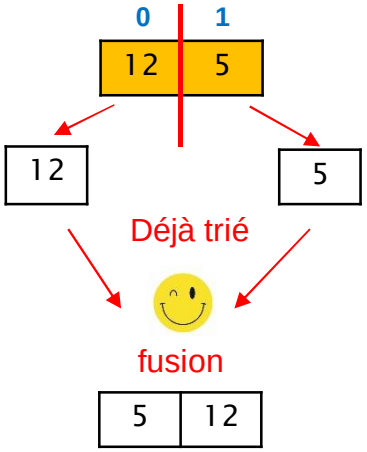
Tri fusion


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

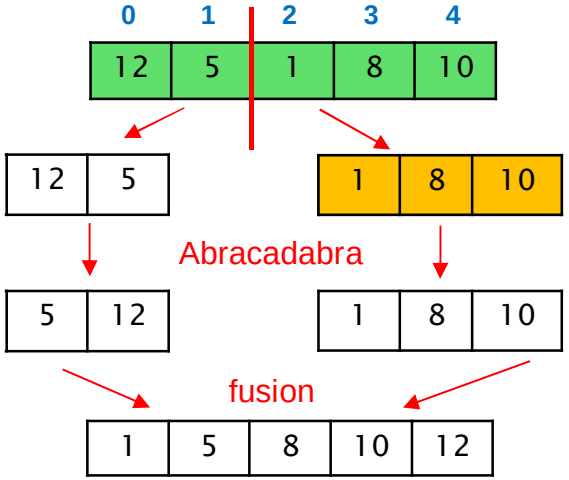
Abracadabra ?



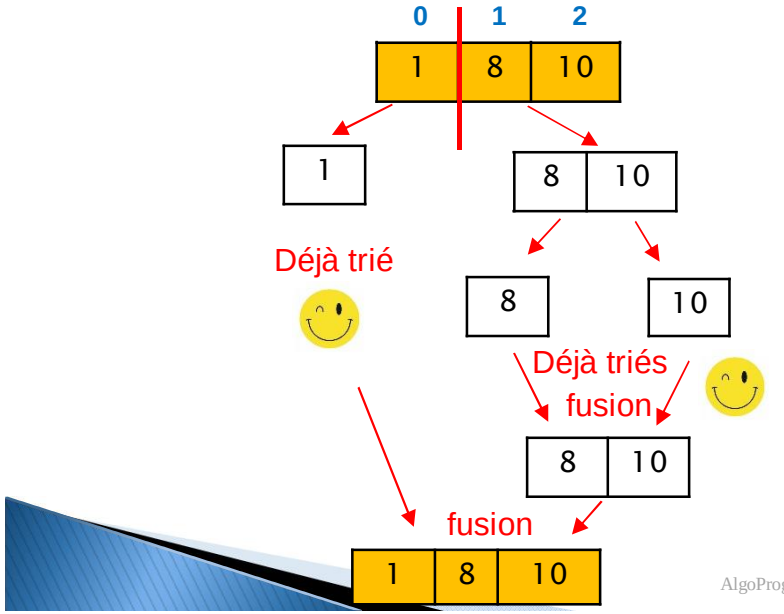
Abracadabra ?



Abracadabra ?



Abracadabra ?



Tri fusion

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

Fusion de 2 tableaux triés

```
void fusion(int t[], int n, int d, int f){
  /*tableau supplémentaire pour stocker la fusion */
  int r[f-d+1];
  int m=(d+f)/2;
  /* t est trié entre d et m*/
  /* t est trié entre m+1 et f*/
   int i1=d, i2=m+1, k=0;

   ...
```

Fusion de 2 tableaux triés

```
void fusion(int t[], int
n, int d, int f){
  int r[f-d+1];
  int m=(d+f)/2;
  int i1=d, i2=m+1, k=0;
  while (i1<=m && i2 <=f){
    if (t[i1] < t[i2]){
      r[k] = t[i1];
      i1++;}
    else {
      r[k] = t[i2];
      i2++;
    }
  k++;
  }
```

```
  while (i1 <=m){
     r[k] = t[i1];
     i1++;
     k++;
   }
   while (i2 <=f){
     r[k] = t[i2];
     i2++;
     k++;
   }
   for(k=0; k<=f-d; k++)
     t[d+k]=r[k];
 }
```

Complexité en temps : $\Theta(f-d+1)$

Complexité en mémoire : $\Theta(f-d+1)$

Tri fusion

```
void triFusion (int t[],int n, int d,
                int f){
if (d<f){
  int m=(d+f)/2;
  triFusion(t,n,d,m);
  triFusion(t,n,m+1,f);
  fusion(t,n,d,f);
}
```

1er Appel :

```
triFusion(t,n,0,n-1);
```

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

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

Stablité : oui car
< dans fusion

Fonction mystère

```
void mystere (int t[],int n){
  int i=0;
  int 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