Algorithmique: méthode diviser pour régner:

1. **Position du problème:**
   Le diviser pour régner est une méthode algorithmique basée sur le principe suivant:

   On prend un problème (généralement complexe à résoudre), on divise ce problème en une multitude de petits problèmes, l'idée étant que les 'petits problèmes' seront plus simples à résoudre que le problème original. Une fois les petits problèmes résolus, on recombine les "petits problèmes résolus" afin d'obtenir la solution du problème de départ.

   Le paradigme 'diviser pour régner' repose donc sur 3 étapes:

   - **Diviser:** le problème d'origine est divisé en un certain nombre de sous-problèmes,
   - **Rêgner:** 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.

   Les algorithmes basés sur le paradigme 'diviser pour régner' sont très souvent des algorithmes récursifs. Nous allons maintenant étudier un de ces algorithmes basés sur le principe diviser pour régner: le tri-fusion

2. **Application: le tri fusion:**
   Nous avons déjà étudié des algorithmes de tri: le tri par insertion et le tri par sélection. Nous allons maintenant étudier une nouvelle méthode de tri, le tri-fusion. Comme pour les algorithmes déjà étudiés, cet algorithme de tri fusion prend en entrée un tableau non trié et donne en sortie, le même tableau, mais trié.

   Exercice 1: On s'interesse à l'algorithme de tri fusion. Cet algorithme est divisé en deux parties:

```
TRI-FUSION(A, p, r)
1  si p < r
2     alors q ← ⌊(p + r)/2⌋
3           TRI-FUSION(A, p, q)
4           TRI-FUSION(A, q + 1, r)
5           FUSION(A, p, q, r)
```

   NB: la notation $\lceil x \rceil$ désigne le plus petit entier supérieur ou égal à x alors que $\lfloor x \rfloor$ désigne le plus grand entier inférieur ou égal à x.

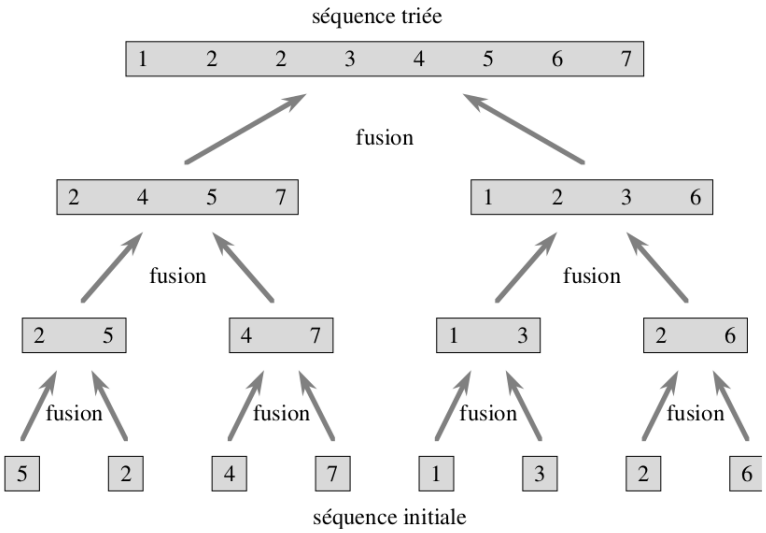
```
FUSION(A, p, q, r)
 1  n1 ← q − p + 1
 2  n2 ← r − q
 3  créer tableaux L[1 . . n1 + 1] et R[1 . . n2 + 1]
 4  pour i ← 1 à n1
 5      faire L[i] ← A[p + i − 1]
 6  pour j ← 1 à n2
 7      faire R[j] ← A[q + j]
 8  L[n1 + 1] ← ∞
 9  R[n2 + 1] ← ∞
10  i ← 1
11  j ← 1
12  pour k ← p à r
13      faire si L[i] ⩽ R[j]
14                alors A[k] ← L[i]
15                      i ← i + 1
16                sinon A[k] ← R[j]
17                      j ← j + 1
```

2.1 Cet algorithme est-il récursif?

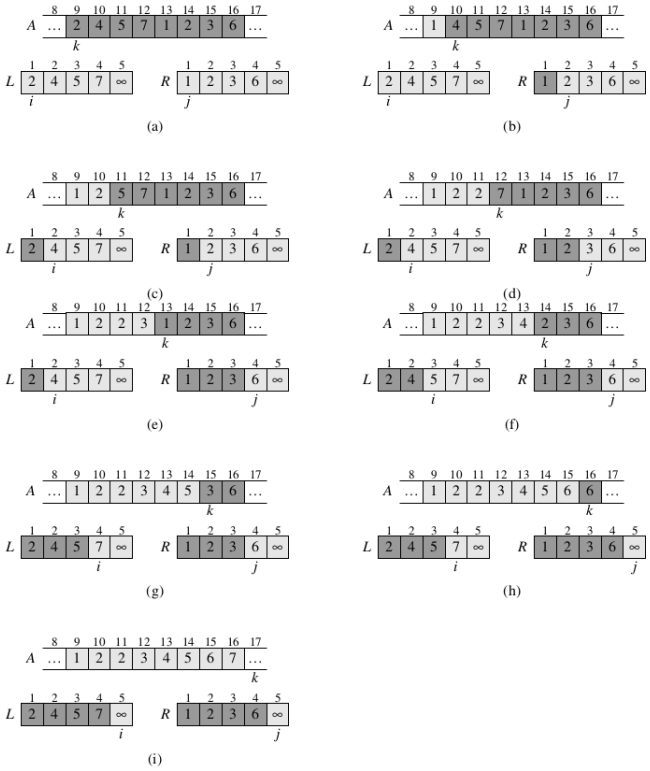
2.2 En analysant les fonctions FUSION et TRI-FUSION, expliquer ce qu'elles permettent de faire.

2.3 Lier chacune de ces fonctions aux trois étapes du paradigme 'diviser pour régner'.

Exercice 2: On schématise l'action des fonctions TRI-FUSION et FUSION sur le tableau $[5, 2, 4, 7, 1, 3, 2, 6]$:



TRI-FUSION



FUSION

Expliquer en détail comment est réalisé le tri du tableau $[5, 2, 4, 7, 1, 3, 2, 6]$.

Exercice 3:

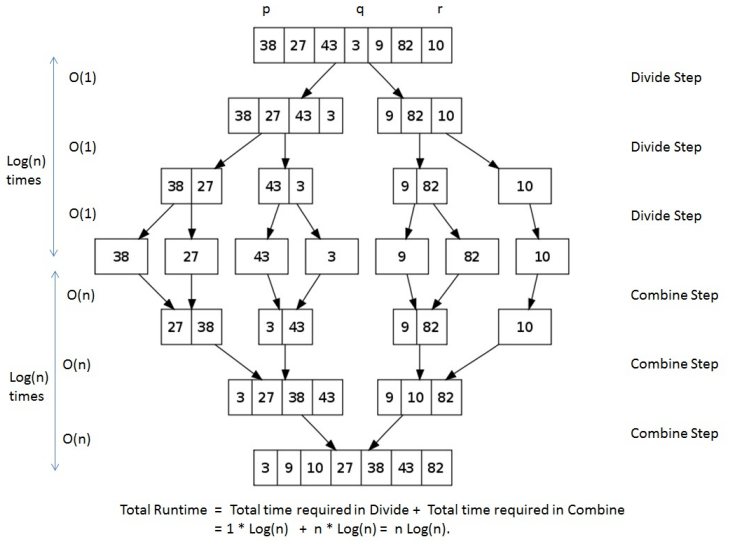
2.1 Reprenez tout le raisonnement qui vient d'être fait sur le tableau T = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]. Vous n'hésiterez pas à faire un schéma et à expliquer la fusion de 2 tableaux triés.

2.2 En déduire un lien avec la compléxité de cet algorithme.

Nous avons vu que le tri par insertion et tri par sélection ont tous les deux une complexité $O(n^2)$. Qu'en est-il pour le tri-fusion?

Le calcul rigoureux de la complexité de cet algorithme sort du cadre de ce cours. Mais, en remarquant que la première phase (DIVISER) consiste à "couper" les tableaux en deux plusieurs fois de suite, intuitivement, on peut dire qu'un logarithme base 2 doit intervenir. La deuxième phase consiste à faire des comparaisons entre les premiers éléments de chaque tableau à fusionner, on peut donc supposer que pour un tableau de n éléments, on aura n comparaisons. En combinant ces 2 constations on peut donc dire que la complexité du tri-fusion est en $O(n.log(n))$ (encore une fois la "démonstration" proposée ici n'a rien de rigoureux).

En voici une illustration:



Graphiquement, la comparaison des courbes de la fonction n^2 (en rouge) et $n.log(n)$ (en bleu) nous montre que l'algorithme de tri-fusion est plus "efficace" que l'algorithme de tri par insertion ou que l'algorithme de tri par sélection.