On souhaiterait trier un tableau en C et on va utiliser les idées du tri fusion pour ceci.

1) Le tri fusion suit une structure algorithmique assez répandue, laquelle ?

2) Rappeler les deux composantes fondamentales d'un tri fusion.

3) On stockera un "tableau" en C dans la structure suivante de façon à avoir accès facilement à sa taille :

```c
struct tableau {
  int *contenu;
  int taille;
};
typedef struct tableau tableau;
```

Écrire les fonctions `tableau *allouer_tableau(int taille)` et `void liberer_tableau(tableau *T)`.

4) Étant donné une telle structure A, on peut, sans allouer de nouvelle mémoire pour son contenu, prendre une *tranche* du tableau de l'indice i à l'indice j (exclu) correspondant à $[T[i], T[i+1], \ldots, T[j-1]]$ et en faire un nouveau tableau B.
Quelle serait la valeur du pointeur `B->contenu` en fonction de `A->contenu` ?
Quelle serait la taille de cette tranche B ?
Écrire la fonction `tableau *tranche(tableau *A, int i, int j)`.

5) Quelle est le grand danger de cette nouvelle fonction ? Par exemple que se produit-il dans l'exemple suivant :

```c
void test_tranche(void) {
  tableau *A = allouer_tableau(10);
  for(int i = 0; i < A->taille; i++) {
    A->contenu[i] = i;
  }
  tableau *B = tranche(A, 2, 5);
  printf("taille de la tranche : %d.\n", B->taille);
  printf("Contenu de B à l'indice 0 : %d.\n", B->contenu[0]);
  B->contenu[0] = 1000;
  printf("Contenu de A à l'indice 2 : %d.\n", A->contenu[2]);
}
```

6) Écrire une fonction `tableau *fusion(tableau *A, tableau *B)` qui crée un nouveau tableau et le remplit avec la fusion des tableaux triés A et B.

7) Écrire une fonction `void copier(tableau *A, tableau *B)` qui copie le contenu de A vers le contenu de B (à partir de l'indice 0). Ne pas oublier de vérifier si cela est possible.

8) Finalement implémenter votre tri fusion à l'aide des fonctions auxiliaires que vous venez d'écrire.

9) Analyser la complexité de votre algorithme en temps et en espace.