Exercice 1 : **Tri fusion**

1. Écrire une fonction `partition : 'a list -> 'a list * 'a list` qui prend en argument une liste `l` et renvoie un couple de listes `(l1, l2)`, de mêmes tailles à 1 près, qui contiennent les mêmes éléments que `l`. L'ordre des éléments n'importe pas.
   Par exemple, `partition [1; 2; 3; 4; 5]` peut renvoyer `([5; 3; 1], [4; 2])`, mais ce n'est pas la seule façon de faire la partition.
2. Écrire une fonction `fusion : 'a list * 'a list -> 'a list` qui étant donné un couples de listes `l1` et `l2` toutes deux triées par ordre croissant, calcule une liste `l` contenant les mêmes éléments que `l1` et `l2`, également triée par ordre croissant.
3. En déduire une fonction `tri_fusion : 'a list -> 'a list` qui calcule une liste triée contenant les mêmes éléments que son argument.
4. (a) Écrire, selon un algorithme similaire mais en utilisant des effets de bord, une fonction `tri_fusion : 'a array -> unit` qui trie un tableau.
   On utilisera pour cela une fonction auxiliaire récursive `tri_fusion_aux : 'a array -> int -> int -> unit` qui prend en argument supplémentaire deux indices `debut` et `fin` et ne trie que la portion entre les indices `debut` et `fin` -1.
   (b) Réécrire `tri_fusion` de façon à n'utiliser qu'un seul tableau auxiliaire au cours de son exécution.
   Pour cela on utilisera une fonction auxiliaire avec des arguments supplémentaires : les indices de début et de fin comme ci-dessus, le tableau auxiliaire (en plus du tableau à trier), et un booléen indiquant si le résultat doit être placer dans le tableau à trier ou le tableau auxiliaire.
5. (a) Justifier que, quelle que soit l'implémentation, les étapes Diviser et Rassembler ont un coût linéaire en la taille de la portion à trier. En déduire que la complexité du tri fusion vérifie $C(n) \leqslant C\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + C\left(\left\lceil \frac{n}{2} \right\rceil\right) + Kn$, avec K une constante.
   (b) En déduire un K' tel quel $C(n) \leqslant K' n \log_2(n)$ pour $n \geqslant 2$. *On trouvera K' en écrivant la preuve par récurrence, de façon à faire marcher la preuve.*
   (c) En déduire $C(n) = \mathcal{O}(n \log(n))$.

Exercice 2 : **Tri rapide (ou tri pivot)**

1. Écrire une fonction `echange : 'a array -> int -> int -> unit` telle que si `t` est un tableau de taille n, et $0 \leqslant i, j < n$, `echange t i j` échange les valeurs de `t.(i)` et `t.(j)`
2. Écrire une fonction `diviser : 'a array -> int -> int -> int` telle que si `t` est un tableau, $0 \leqslant$ `debut` $<$ `fin` $\leqslant n$ sont des indices, en posant $x =$ `t.(debut)`, `diviser t debut fin` échange les valeurs de `t` entre les indices `debut` et `fin` -1 de façon à placer à gauche les valeurs strictement inférieures à x et à droite les valeurs strictement supérieures à x (avec au moins un x entre ces deux groupes). La fonction renvoie l'indice de ce x placé entre les deux groupes.
3. En déduire une fonction `tri_rapide_aux : 'a array -> int -> int -> unit` qui trie le tableau entre les indices `debut` et `fin` -1, en utilisant une approche Diviser pour Régner avec la fonction ci-dessus.
4. En déduire une fonction `tri_rapide : 'a array -> unit` qui trie le tableau selon ce principe.
5. Que manque-t-il pour faire une analyse de complexité similaire à celle du tri fusion ?