Tris

*Pour simplifier on restreint ici le problème du tri (trier = **to sort** en anglais) au problème qui consiste à trier dans l'ordre croissant les éléments d'une liste d'entiers positifs sans doublons (`[4, 6, 1, 3, 2, 5]` par exemple).*

1 Tri par insertion

Principe : à chaque étape, on insère à sa place dans la partie triée le premier élément de la partie non triée.

Écrire une fonction récursive `insert(val, li)` qui prend en argument une liste *triée* et une valeur et qui retourne une liste dans laquelle la valeur est à sa place dans la liste triée. Quelle est la complexité de cette fonction ?

En déduire une fonction récursive `insertionsort(li)` de tri par insertion. Quelle est sa complexité ?

2 Tri par sélection

Principe : à chaque étape, on sélectionne le minimum des éléments restants dans la liste de départ, on le supprime de la liste de départ et on l'ajoute à la fin de la liste qu'on retourne.

Écrire une fonction récursive `min(li)` qui retourne le minimum de la liste `li`.

Écrire une fonction récursive `suppr(val, li)` qui prend en argument une liste `li` et une valeur `val` et qui retourne la liste `li` privée de `val`.

En déduire une fonction récursive `selectionsort(li)` de tri par sélection. Quelle est sa complexité ?

3 Diviser pour régner et tri fusion

3.1 Visualisation du diviser pour régner

Écrire une fonction `dc` (divide and conquer) qui met en évidence le principe des algorithmes diviser pour régner en affichant pour chaque appel récursif la portion d'une liste qui est traitée. Par exemple, l'appel `dc([1, 2, 3, 4, 5])` devra afficher :

```
[1, 2, 3, 4, 5]
[1, 2]
[1]
[2]
[3, 4, 5]
[3]
[4, 5]
[4]
```

`[5]`

3.2 Application à la somme

Écrire une fonction récursive `sum(li)` qui retourne la somme des éléments d'une liste en appliquant le principe du diviser pour régner.

3.3 Tri fusion (mergesort)

Principe : on utilise le principe du diviser pour régner en appelant récursivement la fonction de tri sur les deux moitiés de la liste à trier, jusqu'à ce que le problème soit résolu (la liste est alors vide ou ne contient qu'un élément). A chaque étape on fusionne (`merge`) les deux sous-listes triées en une seule liste triée.

Écrire une fonction récursive `mergesort(li)` qui prend en argument une liste `li` et qui la trie selon le principe du tri fusion (penser à la fonction `sum` précédente, il suffit d'utiliser la fonction `merge` de fusion de deux listes triées vue auparavant dans le TP récursivité à la place de l'opérateur `+`).

Quelle est la complexité de la fonction `merge` ? Quelle est la complexité de ce tri ?

4 Tri rapide

Principe : Si la liste `li` à trier contient au moins 2 éléments (sinon elle est triée), on choisit un élément de `li` qui sera le *pivot*. On retourne alors la liste des éléments de `li` inférieurs au pivot sur laquelle on applique récursivement le tri, à laquelle on concatène une liste qui ne contient que le pivot, à laquelle on concatène la liste des éléments de `li` supérieurs au pivot sur laquelle on applique récursivement le tri.

Étant donné une valeur `p` (pivot) et une liste `li`, comment construire la liste des éléments qui lui sont supérieurs avec une liste en compréhension ? Quelle est la complexité de cette opération ?

En déduire une fonction `quicksort(li)` de tri rapide. Quelle est la complexité du tri rapide ?