1. **Rappels.** Si `L` est une liste de longueur n, la *médiane* de `L` est la valeur qui sépare `L` en deux sous-listes de longueur égale. Plus précisément :

- si n est impair, la médiane de `L` est le $\frac{n+1}{2}$-ème élément de la liste `L` triée (par ordre croissant ou décroissant) ;
- si n est pair, la médiane de `L` est la moyenne du $\frac{n}{2}$-ème et du $(\frac{n}{2}+1)$-ème éléments de la liste `L` triée (notez que, dans ce cas, la médiane de `L` n'est pas nécessairement un élément de `L`).

2. **Pour s'échauffer.**

(1) Écrire une fonction `medianetriche` qui prend en entrée une liste `L` non vide et renvoie la médiane de `L` à l'aide d'un appel à la fonction native Python `sorted`.

Rappels :
- `sorted(L)` renvoie une copie de `L` dont les éléments sont triés par ordre croissant ;
- l'opération `n % 2` renvoie `0` si `n` est pair et `1` sinon.

(2) Estimer la complexité de l'algorithme en fonction de `n` (la fonction `sorted` est une implémentation de l'algorithme Timsort dont la complexité est $O(n \log n)$)

3. **Médiane approchée récursive.** On propose d'écrire une fonction renvoyant une valeur approchée de la médiane de `L` à l'aide de l'algorithme suivant :

- si `L` contient moins de trois éléments, renvoyer sa vraie médiane ;
- sinon, découper `L` en trois tranches de longueurs approximativement égales, calculer récursivement la médiane approchée de chaque tranche, puis renvoyer la médiane de ces trois résultats.

(3) Écrire une fonction `med3` qui prend en entrée trois nombres et renvoie leur médiane exacte.

(4) Écrire une fonction `medianeapprox` qui implémente l'algorithme décrit ci-dessus.

(5) Estimer la complexité de l'algorithme en fonction de `n` (faire le calcul dans le cas où `n` est une puissance de 3).

(6) Pourrait-on modifier l'algorithme en découpant la liste en un plus grand nombre de tranches ? Discuter.

4. **Recherche du p-ème plus petit élément.** Dans cette partie, on suppose que la liste `L` est de longueur n impaire. On va écrire une fonction récursive `pinimum` qui prend en entrée une liste `L` et un entier `p` $\leq$ `n` et renvoie le `p`-ème plus petit élément de `L`.

(7) Écrire *en une seule ligne* une fonction `medianerapide` qui prend en entrée une liste de longueur impaire et renvoie sa médiane à l'aide de la fonction `pinimum` (NB : on n'a pas encore écrit la fonction `pinimum` donc il est logique que cette fonction ne compile pas pour l'instant...).

L'algorithme sur lequel repose `pinimum` est inspiré du tri rapide : on choisit [1] un pivot `m` dans `L`, puis on répartit les autres éléments de `L` dans deux listes `L1` et `L2` suivant s'ils sont inférieurs ou supérieurs à `L`.

(8) Notons `n1` la longueur de `L1`. En séparant les cas suivant les valeurs relatives de `n1` et `p`, expliquez quel appel récursif permet de renvoyer le `p`-ème plus petit élément de `L` (indication : il est soit dans `L1`, soit dans `L2`, soit égal au pivot).

(9) Écrire la fonction `pinimum`.

(10) Évaluer la complexité des fonctions `pinimum` et `medianerapide` (faire le calcul dans le cas où `n` est une puissance de 2 et en supposant que les listes `L1` et `L2` sont toujours de tailles équilibrées).

1. par exemple à l'aide de `medianeapprox`