EA4 – Éléments d'algorithmique
TD n° 6 : tri rapide

Exercice 1 : déroulement du tri rapide

On considère le tableau `T` suivant :

7	1	5	6	2	3	10	8	0	9	4

Décrire le déroulement des variantes suivantes du tri rapide sur le tableau `T` :

1. variante avec mémoire auxiliaire, en prenant le premier élément comme pivot ;
2. *idem*, mais en choisissant toujours pour pivot l'élément médian.
3. variante « en place », le pivot étant toujours le premier élément.

Dans chaque cas, compter précisément le nombre de comparaisons effectuées.

Exercice 2 : complexité du tri rapide

1. Évaluer le nombre de comparaisons d'éléments effectuées par le tri rapide (avec pivot `T[0]`) dans les cas suivants :
   - `C[n] = [1, 2, ..., n-1, n]` ;
   - `D[n] = [n, n-1, ..., 2, 1]` ;
   - `M[k] = [2**k] + M[k-1] + [ i+2**k for i in M[k-1] ]` (avec `M[0] = [1]`).
2. Donner d'autres exemples de tableaux de taille 7 pour lesquels la complexité du tri rapide est la même que pour `C[7]`. Même question pour `M[2]`.
3. Quel est le nombre de sommets de l'arbre de récursion du tri rapide pour un tableau de longueur n dont tous les éléments sont distincts ? Que peut-on en conclure concernant sa hauteur minimale ? et sa hauteur maximale ?
4. Donner un encadrement aussi fin que possible du nombre de comparaisons effectuées, en cumulé, pour les appels récursifs de profondeur p.
5. En déduire la complexité en temps du tri rapide dans le meilleur et dans le pire cas (en supposant toujours que tous les éléments sont distincts).

Exercice 3 : hauteur de pile du tri rapide

Comme tout algorithme récursif, le tri rapide est sujet à des débordements de la pile si le nombre d'appels récursifs devient trop grand.

1. On considère l'implémentation « naïve » du tri rapide avec copies de tableaux :

```
def triRapide(T) :
    if len(T) <= 1 : return T
    pivot = T[0]
    gauche = [elt for elt in T[1:] if elt <= pivot]
    droite = [elt for elt in T[1:] if elt > pivot]
    return triRapide(gauche) + [pivot] + triRapide(droite)
```

   Quelle hauteur atteint la pile dans le pire cas ? dans le meilleur cas ?
2. Le deuxième appel récursif est terminal [1] et peut donc être dérécursivé. Quelle hauteur atteint alors la pile sur l'entrée `C[n] = [1, ..., n]` ? Et sur l'entrée `D[n] = [n, ..., 1]` ?
3. Proposer une solution permettant de garantir une hauteur de pile maximale en $O(\log n)$ dans tous les cas (où n est la longueur du tableau à trier).
4. Quel est alors le meilleur cas en ce qui concerne la hauteur de pile ?

1. enfin... presque... faisons comme si c'était le cas !

Exercice 4 : tri drapeau

Le problème du *drapeau hollandais* est le suivant : le tableau à trier contient trois types d'éléments, les bleus, les blancs et les rouges et on souhaite les trier par couleur. Tous les éléments d'une même couleur ne sont pas identiques, il ne suffit donc pas de les compter.

1. On dit qu'un algorithme de tri est *stable* s'il préserve l'ordre du tableau initial pour les éléments de même couleur. Proposer un algorithme linéaire et stable (mais pas en place) pour résoudre ce problème.

2. Proposer un algorithme linéaire et en place (mais pas stable) dans le cas où il n'y a que deux couleurs (bleu et rouge).

3. Comment adapter l'algorithme précédent au cas où le tableau contient un unique élément blanc, dans sa première case ?

4. Généraliser l'algorithme de la question 2 au cas de trois couleurs. Pour cela, on maintiendra l'invariant suivant :

> *« le tableau est constitué de 4 segments consécutifs, un bleu, un blanc, un non exploré, et un rouge ».*

Utiliser trois curseurs pour représenter les frontières entre ces segments.

Remarque : il n'existe pas à notre connaissance d'algorithme à la fois linéaire, stable et en place pour ce problème ; on peut en revanche modifier légèrement l'algorithme de la question 4 pour obtenir un algorithme stable et en place (mais pas linéaire, donc).