Travaux dirigés n° 4
Diviser pour régner

La méthode *Divide and conquer* consiste à subdiviser l'entrée d'un algorithme, à résoudre les sous-problèmes engendrés puis à combiner les résultats partiels. Le tri-fusion en est un exemple fameux. La recherche dichotomique en est un autre, encore que dans son cas, on parlera de *Decrease and conquer*. En effet, l'appel récursif qui se produit ignore une moitié du tableau. Ces méthodes peuvent conduire à des algorithmes très efficaces comme nous allons le constater.

Exercice 4.1 — *National Dutch Flag*

On considère un tableau dont les n cases contiennent des éléments de l'ensemble $\{B, R, W\}$. Les lettres correspondent respectivement aux couleurs bleu, rouge et blanc.
Ecrivez le pseudo-code d'un algorithme itératif NDF() qui trie les éléments d'un tableau selon les couleurs rouge, blanc puis bleu en temps linéaire.

Exercice 4.2 — Tri-rapide ou *Quicksort*

Cet algorithme de tri utilise une sous-fonction Partition () qui place les éléments plus petits que l'élément d'indice *pivot* au début du tableau et ceux plus grands à la fin du tableau. Avant de terminer, elle replace le pivot à sa bonne place parmi eux et retourne le nouvel indice du pivot.

Voici le programme principal de cet algorithme :

```
   Quicksort (tableau T, entier i, entier j){
1    si (i < j) alors {
2      pivot ← i + ⌊(j − i)/2⌋
3      pivot ← Partition (T, i, j, pivot)
4      Quicksort (T, i, pivot−1)
5      Quicksort (T, pivot+1, j)
     }
   }
```

1. Écrivez le pseudo-code de la sous-fonction Partition () en déduisant sa signature du programme principal. Vous pouvez vous inspirer de l'algorithme du *National Dutch Flag* pour écrire cette fonction. Quelle est sa complexité ?
2. A présent, calculez la complexité du *tri rapide* et comparez-la à celles des autres tris connus.

Exercice 4.3 — Pic

On considère un tableau T de n entiers distincts. Un élément de T est un *pic* s'il est supérieur à son ou ses voisins. Il a deux voisins sauf s'il est sur un bord auquel cas il en a un seul. Par exemple, les éléments 9, 8 et 1 sont les seuls pics du tableau suivant :

6	7	9	2	8	4	3	0	1

1. Ecrivez un premier algorithme itératif `pic(T)` retournant un élément qui est un pic quelconque dans le tableau T. Sa complexité sera en $O(n)$ donc linéaire en le nombre de cases du tableau.

2. On se propose à présent d'utiliser le principe *Divide and conquer* afin d'améliorer la complexité de l'algorithme précédent. Trouvez un algorithme récursif `pic_DC(T)` qui retourne un pic quelconque du tableau d'entiers T passé en paramètre avec une bien meilleure complexité que l'algorithme `pic(T)` précédent.