Exercice : Algorithmique : Diviser pour régner

Exercice 1

On cherche à écrire une fonction qui effectue la rotation d'une image de 90 degrés en utilisant le principe de **diviser pour régner**. Pour manipuler une image en Python, on utilise la bibliothèque **PIL** et plus précisément son module **Image**. Avec les quatre lignes suivantes :

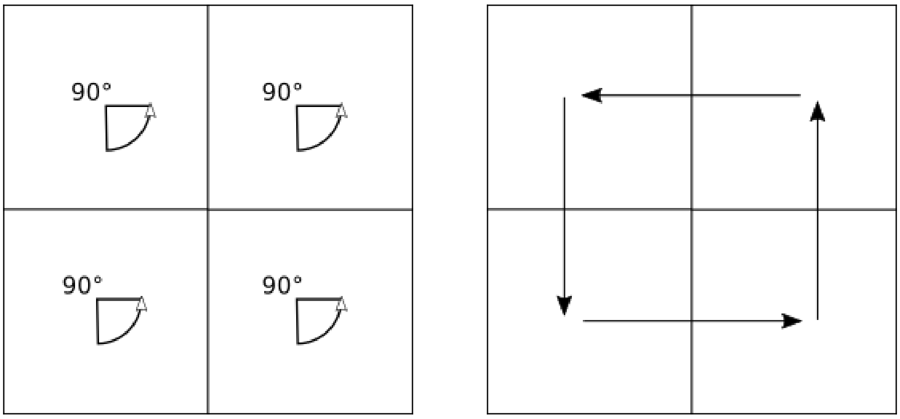
```python
from PIL import Image
im = Image.open("mon_image.png")
largeur,hauteur=im.size
px = im.load()
```

on charge l'image contenue dans le fichier **mon_image.png**, on obtient ses dimensions dans les variables **largeur** et **hauteur** et la variable **px** est la mtrice des pixels constituant l'image.

Pour $0 \leqslant x \leqslant$ largeur et $0 \leqslant y \leqslant$ hauteur, la couleur du pixel est donnée par px$[x, y]$. Une couleur est un triplet donnant les composantes rouge, vert et bleu, sous la forme d'entiers entre 0 et 255.

On peut modifier la couleur d'un pixel avec une affectation de la forme px$[x, y] = c$ où c est une couleur.

Dans cet exercice, on suppose que l'image est carrée et que sa dimension est une puissance de 2, par exemple 256×256. Le principe est de diviser l'image en quatre images, à effectuer la rotation des quatre morceaux, puis à les déplacer vers leur position finale. On illustre le procédé par la figure suivante :



Afin de procéder récursivement, on va définir une fonction **rotation_aux(px,x,y,t)** qui effectue la rotation de la portion carrée de l'image comprise entre les pixels (x,y) et (x+t,y+t).

Cette fonction ne renvoie rien. Elle modifie le tableau **px** pour effectuer la rotation de cette portion de l'image au même endroit. On suppose que **t** est une puissance de 2. Écrire le code de cette fonction.

En déduire une fonction **rotation(px,taille)** qui effectue une rotation de l'image toute entière, sa dimension étant donnée par le paramètre **taille**. Une fois la rotation achevée, on pourra enregistrer le résultat dans un autre fichier avec la commande **im.save("rotation.png")**.

Exercice 2

On propose d'appliquer le principe **diviser pour régner** pour multiplier deux nombres entiers, avec la méthode de **Karatsuba** (`https://fr.wikipedia.org/wiki/Algorithme_de_Karatsuba`). Le principe est le suivant :

Supposons deux entiers x et y ayant chacun $2n$ chiffres en base 2. On peut les écrire sous la forme $x = a \times 2^n + b$ et $y = c \times 2^n + d$ avec $0 \leqslant a, b, c, d < 2^n$, c'est à dire avec quatre entiers qui s'écrivent chacun avec n chiffres en base 2. Dès lors, on peut calculer le produit de x et y de la façon suivante :

$$\begin{aligned} xy &= (a2^n + b)(c2^n + d) \\ &= ac2^{2n} + (ad + bc)2^n + bd \\ &= ac2^{2n} + (ac + bd - (a - c)(c - d))2^n + bd \end{aligned}$$

Cette dernière forme, d'apparence inutile et compliquée, fait apparaître seulement 3 produits, à savoir ac, bd et $(a-b)(c-d)$. Ainsi, on a ramené la multiplication de 2 entiers de $2n$ chiffres à 3 multiplication d'entiers de n chiffres. Pour faire chacune de ces trois multiplications, on peut appliquer le même principe, et ainsi de suite jusqu'à obtenir de petits entiers dont la multiplication en temps proportionnel à $n^{1{,}58}$ (environ) au lieu de n^2, ce qui est un gain significatif lorsque le nombre de chiffres n est grand.

1) Écrire une fonction **taille(x)** qui renvoie le nombre de chiffres de l'entier x lorsqu'il est écrit en base 2.

2) Écrire une fonction **karatsuba(x,y,n)** qui calcule le produit de x pa y par la méthode Karatsuba, en supposant que x et y s'écrivent sur n chiffres en base 2 (*indication* : on peut calculer 2^n en Python avec l'expression $1 << n$. On peut décomposer x sous la forme $a2^n + b$ avec $a, b = x >> n, x\%(1 << n)$).

3) En déduire une fonction **mult(x,y)** qui calcule le produit de x et y.

4) Tester cette fonction avec plusieurs exemples.