1. Version récursive de l'élévation à la puissance par multiplications successives.

```
Puissance_rec (a, b) {
Si b = 0 Alors
   retourner 1;
retourner (Puissance_rec (a, b-1) * a);
}
```

2. Calcul de $n!$ (version itérative puis récursive).

```
Factorielle (n) {
fact <- 1;
Pour i <- 2 a n Faire
   fact <- fact * i;
retourner fact;
}
```

3. Version récursive du calcul du pgcd.

```
Pgcd_rec (a, b) {
Si a = b Alors
   retourner a;
Si a > b Alors
   retourner Pgcd_rec (a-b, b);
Sinon
   retourner Pgcd_rec (a, b-a);
}
```

Remarque : problème si un des deux nombres est nul.

4. Calcul du terme u_n de la suite de Fibonacci, définie par :

$$\begin{aligned} u_0 &= 0 \\ u_1 &= 1 \\ u_{n+1} &= u_n + u_{n-1} \end{aligned}$$

Donner une version itérative et une version récursive.

```
Fibo (n) {
Si n = 0 alors retourner 0;
u_i-1 <- 0;   //u_0
u_i <- 1;     //u_1

Pour i de 2 a n Faire { //Calcul du ie terme de la suite
   terme_suiv <- u_i + u_i-1;
   u_i-1 <- u_i;
   u_i <- terme_suiv;
}
retourner u_i;
}
```

```
Fibo_rec (n) {
Si n = 0 Alors retourner 0;
Si n = 1 Alors retourner 1;
retourner (Fibo_rec (n-1) + Fibo_rec (n-2));
}
```

5. Soit l'algorithme du tri rapide donné ci-dessous.

```
def partition (t,g,d):
    m = g + 1
    while m <= d:
        if t[m] < t[g]:
            m = m + 1
        else:
            echange(t,m,d)
            d = d - 1
    echange(t,g,d)
    return d

def tri_rapide (t,g,d):
    if g < d:
        m = partition(t,g,d)
        tri_rapide(t,g,m-1)
        tri_rapide(t,m+1,d)

def tri (t):
    tri_rapide(t,0,len(t)-1)
```

Remarque : `partition` est similaire à Dikstra avec 2 couleurs

(a) Faire tourner sur un exemple cet algorithme.

(b) Expliquer son principe.

(c) Donner un exemple pour lequel cet algorithme n'est pas efficace.

(d) Donnez la complexité de cet algorithme.