1 Une fonction récursive

```
int TailRecursion(int m, int n) {
    if (m == 0 || n == 0) return 1;
    else return 2 * TailRecursion(m-1, n+1);
}
```

1. Préciser ce que calcule cette fonction.
2. Écrire une version itérative équivalente pour cette fonction.

2 Deux autres

Soit la fonction f définie par $f(0) = 1$, $f(n) = nf(\frac{n}{2})$ si n est pair et $f(n) = 1 + f(\frac{n-1}{2})$ sinon.

1. Écrire une version récursive pour f.
2. Écrire ensuite une version itérative qui, à l'aide d'une pile, simule le comportement de la version récursive.

Soit la fonction g définie par $g(0) = 1$, $g(n) = ng(\frac{n}{2})$ si n est pair et $g(n) = g(\frac{n-1}{2})$ sinon.

3. Écrire une version itérative sans pile pour g.
4. Expliquer pourquoi c'est possible avec cette fonction, et non avec la fonction f ?

3 Tri rapide

Écrire une fonction `int partition(int[] t, int x, int p, int r)` qui réarrange le sous-tableau de `t` compris entre les indices `p` et `r` de manière à ce que les éléments strictement inférieurs à `x` soient regroupés au début (et donc ceux supérieurs ou égaux à `x` regroupés à la fin). La fonction doit renvoyer en résultat l'indice de la première case où elle aura mis un élément supérieur ou égal à `x`.
On peut utiliser cette fonction pour trier un tableau :
– on choisit un pivot, par exemple `t[0]` ;
– à l'aide de `partition`, on regroupe au début du tableau les éléments plus petits que le pivot, et à la fin ceux qui lui sont supérieurs ou égaux ;
– on continue de même sur chacun des deux sous-tableaux.
Écrire une version récursive de cet algorithme, puis une version itérative. Pour cette dernière, il faudra maintenir une pile contenant la liste des sous-tableaux en attente de traitement.

4 Chemin dans un graphe orienté

Un graphe orienté est un système de nœuds reliés par des flèches. On peut représenter un graphe orienté par un tableau de booléens à deux dimensions : les n nœuds du graphe sont numérotés de 0 à $n-1$ et la case (i, j) du tableau contient `true` si et seulement si il y a une flèche reliant le nœud d'indice i au nœud d'indice j. On s'intéresse au problème consistant à chercher s'il existe un chemin dans un graphe entre deux nœuds i et j donnés.
On peut utiliser l'algorithme dit de « parcours en profondeur » :
– on se donne un tableau auxiliaire `connus` de booléens, dont toutes les cases sont initialement à `false`, et qui servira à se souvenir de l'ensemble des sommets que l'on a rencontrés ;
– on part du nœud i :
  – on met `connu[i]` à `true` ;
  – on examine l'extrémité k de chacune des flèches sortant de i : si la case `connu[k]` contient `true` on ne fait rien, sinon on fait un appel récursif sur k.

On explore ainsi tous les sommets du graphe que l'on peut atteindre à partir de i. Si l'on rencontre j au passage, alors on a gagné.

1. Écrire une méthode récursive implémentant cet algorithme.
2. Écrire une méthode itérative qui fait la même chose. Pour ce faire, il faut maintenir dans une pile le chemin que l'on a parcouru entre le sommet de départ et le sommet courant.