NOM : PRENOM : GROUPE :

IUT de Montpellier **Partiel**
M3103 Algorithmique avancée **Durée : 2h**

Consignes :

- **commencez par mettre votre nom, prénom, groupe, en haut de *toutes* les pages du sujet**
- **répondez directement sur le sujet**
- **sauf pour l'exercice de complexité, il est interdit d'utiliser des boucles**

Exercice 1. Exercice sur les tableaux

Le but de cet exercice est d'écrire une méthode EchangePaires qui échange dans un tableau de longueur paire toutes les paires d'éléments consécutifs dont le premier élément est d'indice pair dans le tableau. Par exemple, cette méthode transforme le tableau [1,2,3,4,5,6] en [2,1,4,3,6,5].

Question 1.1.

Ecrire récursivement la méthode ci-dessous.

```
public static void EchangePairesAux (int[] t, int i){
 /* pré-requis : t est de longueur paire, 0 <= i <= t.length et i est pair
    action : échange dans t[i..t.length-1] toutes les paires d'éléments
    consécutifs dont le premier élément est d'indice pair dans t.
    Par exemple, si t = [1,2,3,4,5,6] et i = 2 alors t est
    transformé en [1,2,4,3,6,5].
*/
```

Question 1.2.
En déduire la méthode (non récursive) ci-dessous.

```
public static void EchangePaires (int[] t){
/* pré-requis : t est de longueur paire (éventuellement nulle)
    action : échange dans t toutes les paires d'éléments consécutifs
    dont le premier élément est d'indice pair dans t.
 */
```

Exercice 2. Exercice sur les listes

On utilise ici la même classe Liste que celle vue en cours. Il est interdit de modifier cette classe en y ajoutant vos attributs ou méthodes non demandées.

```
class Liste{
  private int val;
  private Liste suiv;

  public Liste(){//construit la liste vide
    this.suiv = null;
  }

  boolean estVide(){return this.suiv==null};
}
```

Question 2.1.

Ecrire la méthode suivante de la classe Liste :

```
boolean egale(Liste l2)
/* pré-requis : aucun
   résultat : retourne vrai si et seulement si les listes this et l2
     contiennent la même suite d'entiers (dans le même ordre) */
```

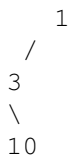
Question 2.2.
Ecrire la méthode suivante de la classe Liste :

```
boolean estSousListe (Liste l2)
/* pré-requis : aucun
   résultat : retourne vrai si et seulement si this est une sous-liste de l2,
     c'est-à-dire si la suite d'entiers contenue dans this est une sous-suite
     de celle contenue dans l2 (dans le même ordre, mais les éléments de this
      n'étant pas forcément consécutifs dans l2).
     Par exemple, la liste vide, (3), (2,4) et (1,2,3,4) sont des
     sous-listes de (1,2,3,4), mais pas (4,2). */
```

Exercice 3. Exercice sur les arbres : arbre croissant
On utilise ici la même classe Arbre que celle vue en cours. Il est interdit de modifier cette classe en y ajoutant vos attributs ou méthodes non demandées.

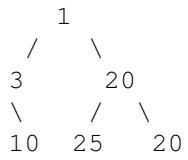
```
class Arbre{
  private int val;
  private Arbre filsG;
  private Arbre filsD;
  //invariant : filsG==null <=> filsD==null

  public Arbre(){//construit l'arbre vide
    this.filsG = null;
    this.filsD = null;
  }
  boolean estVide()
     return (this.filsG==null);
     //vu l'invariant, pas besoin d'ajouter "&&(this.filsD==null)";
}
```

Question 3.1.
Ecrire dans une méthode main de la classe Arbre une suite d'instructions permettant de construire l'arbre ci-dessous (encore une fois en utilisant seulement les attributs, le constructeur et la méthode estVide() donnés ci-dessus).



On dit qu'un arbre est **croissant** ssi pour tout chemin de la racine vers une feuille, la suites des entiers de ce chemin est croissante (pas forcement strictement)

Par exemple, l'arbre ci-dessous est croissant, mais il ne l'est plus si on remplace 25 par 15, ou 10 par 2, ou 3 par 0.



Question 3.2.

Ecrire la méthode suivante de la classe Arbre :

```
public boolean croissant(){
/* pré-requis : aucun
résultat : retourne vrai si et seulement si this est croissant.*/

}
```

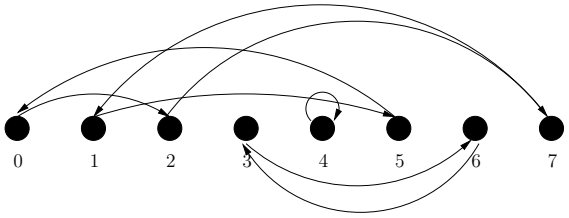
Exercice 4. Exercice de complexité : cycle max d'une permutation

Pour tout entier $n \geq 1$, on appelle permutation d'ordre n un tableau indicé de 0 à $n-1$ dont les valeurs sont exactement les éléments de l'intervalle $0..(n-1)$ dans un ordre arbitraire (chaque valeur est donc présente exactement une fois). Par exemple, le tableau $p = (2, 5, 7, 6, 4, 0, 3, 1)$ est une permutation d'ordre 8. On associe à une permutation p d'ordre n le graphe orienté $G(p) = (V, U)$, avec $V = 0..(n-1)$ et $U = \{(i, p[i]),$ pour i dans $0..(n-1)\}$. Autrement dit, pour chaque case i du tableau

— on associe un sommet i

— on ajoute l'arc $(i, p[i])$

Par exemple, pour $p = (2, 5, 7, 6, 4, 0, 3, 1)$ on obtient le graphe $G(p)$ représenté ci-dessous.



Comme p est une permutation, on peut remarquer que tous les sommets de $G(p)$ ont degré sortant 1 et degré entrant 1, et donc que $G(p)$ est une union disjointe de circuits (admis) appelés les cycles de la permutation p. Par exemple ici $G(p)$ est l'union disjointe des circuits $(0, 2, 7, 1, 5, 0)$ (de longueur 5), $(3, 6, 3)$ (aller-retour, de longueur 2) et $(4, 4)$ (boucle, de longueur 1).

On considère l'algorithme suivant :

```
public int longueurMaxCycle (int[] p) {
/* pré-requis : p est une permutation d'ordre p.length
   résultat : retourne la longueur maximum d'un cycle de p */

  int courant, long;
  int max = 1;
  for (int debut = 0; debut<p.length;debut++){
     courant = p[debut];
     long = 1;
     while (courant != debut) {
        courant = p[courant];
        long ++
     }
     if (long > max){
        max = long;
     }
  }
  return max;
```

Question 4.1.
Pour tout n, trouver une permutation p_n d'ordre n telle que $m \geq n^2$, où m est le nombre d'opérations de longueurMaxCycle(p_n). On admet que l'on peut en déduire que l'algorithme n'est pas linéaire (c'est à dire tel qu'il existe c tel que pour toute permutation d'ordre n, $m \leq cn$).

Question 4.2.
Déterminer la complexité de cet algorithme. C'est à dire, pour le plus petit a possible, prouvez qu'il existe une constante c_1 telle que pour toute permutation p d'ordre n, $m \leq c_1 n^a$ où m est le nombre d'opérations de longueurMaxCycle(p).

Question 4.3.
Bonus : Donner sans justification un algorithme longueurMaxCycle2 ayant les mêmes spécifications que longueurMaxCycle, mais de complexité linéaire ($m \leq cn$). Indication : on pourra ajouter un tableau de booléens comme variable locale.