Université de Technologie de Compiègne **Documents non autorisés**
Médian NF16, Printemps 2010 **Durée 2 heures**

Dans un souci d'équité, il ne sera pas répondu à des questions pendant la durée de l'épreuve (sauf aux étudiants étrangers ayant des difficultés pour comprendre le français qui seront aidés pour la compréhension du sujet). Si l'énoncé vous semble comporter une imprécision, faites un choix pour la lever que vous indiquerez clairement dans votre copie. Assurez-vous par une relecture attentive de l'énoncé que la réponse à votre question n'y figure pas. **Chaque partie est à rédiger sur une copie séparée, un point vous sera retranché sinon**.

Exercice I (copie numéro I : 5 points).

1- Montrer que la relation « est de l'ordre de », O(f(n)), est une relation réflexive, transitive, mais pas symétrique.

2- Calculer la complexité de l'algorithme suivant :

```
a := 1
Pour i := 1 à x faire
  k := 0
  Pour j := 1 à 2a faire k := k+1 finpour
  a := k
finpour
```

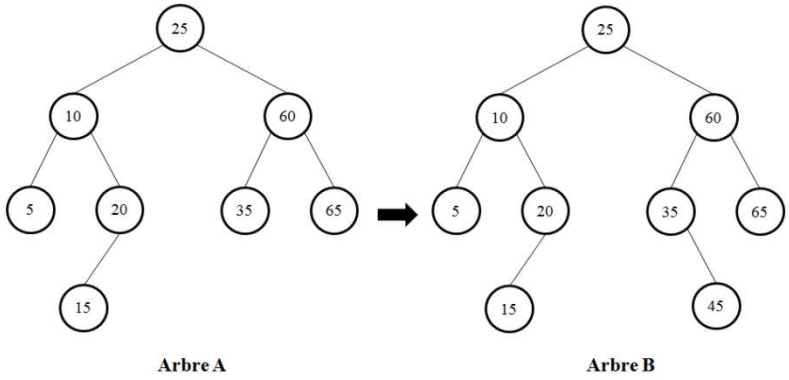
Exercice II (copie numéro II : 5 points).

Soit un arbre binaire de recherche (ABR). La représentation en langage C d'un nœud de l'arbre est comme suit :

```
typedef struct node
{
  int Valeur;
  struct node *gauche;
  struct node *droite;
}Node;
```

1- Ecrire en C la fonction **Node * insérerNoeud (int n, Node *root)** qui permet d'insérer la valeur n dans un ABR ayant comme racine root.

Exemple. Soit l'arbre A :



Après l'insertion de la valeur 45, on obtient l'arbre B.

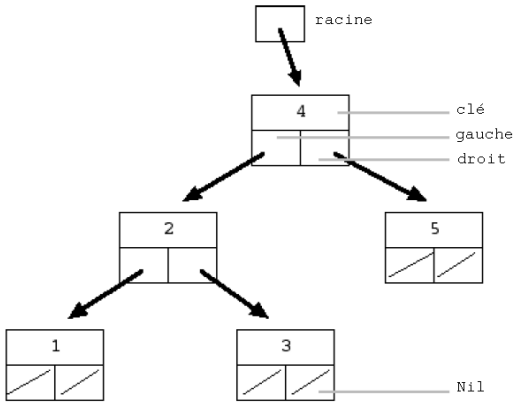
2- Ecrire en C la fonction **int hauteur (Node * root)** qui permet de retourner la hauteur de l'arbre ayant comme racine **root**. La hauteur des arbres A et B est 3.

Exercice III (copie numéro III : 10 points).

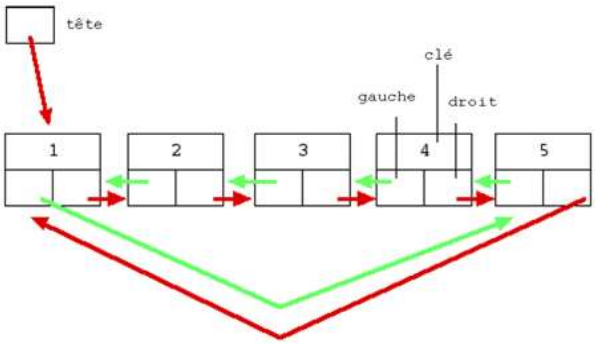
L'objectif de cet exercice est de transformer un arbre binaire de recherche (ABR) en une liste doublement chaînée circulaire de telle sorte que les nœuds apparaissent dans un ordre croissant.

Exemple.

Soit l'ABR ci-dessous :



La liste correspondante est :



1- Définir la structure Nœud d'un ABR.

2- Écrire une fonction C **void joindre (Nœud *x, Nœud *y)** qui permet à un noeud x d'avoir y comme successeur (utiliser le pointeur droit) et au noeud y d'avoir x comme prédécesseur (utiliser le pointeur gauche).

3- En déduire la fonction C **Nœud* fusionne (Nœud *x, Nœud *y)** qui renvoie une liste doublement chaînée circulaire composée des nœuds de la liste pointée par x suivis de ceux de la liste pointée par y. Préciser la complexité de cette fonction.

4- Écrire en C une fonction récursive **Nœud * arbreVersListe (Nœud * racine)** qui prend en argument un arbre binaire de recherche (pointé par sa racine) et retourne une liste (triée par ordre croissant) doublement chaînée circulaire **composée des noeuds de l'arbre**. Préciser la complexité de cette fonction.

NB: On utilisera la fonction fusionne.

5- Écrire une fonction **void afficheListe(Nœud * tete)** qui affiche les nœuds d'une liste doublement chaînée circulaire.

6- Écrire une fonction **main** permettant de créer l'arbre binaire de recherche **<4, 2, 1, 3, 5>**, le transformer en une liste doublement chaînée circulaire puis afficher le résultat.

NB: On dispose de la fonction **void insererDansAbr(Nœud *racine, int clé)** qui crée un nouveau nœud et l'insère dans un ABR.