Algorithmique et Structures de données
(Durée 02:00)

Les réponses doivent être claires, précises et concises.

Exercice 01: «Récursivité & Complexité » ..... (04 pts)

Soit les deux procédures ci-dessous.

```
Procédure Proc (n :entiers ; var Reslt :entiers) ;
 Var    j,S : Entiers ;
Début
        i :=n ;      S :=0 ;
        Tantque  i>0  faire
        Debut
                S :=  S+fon(i) ;
                i:=i div 4;
        Fin
        Reslt:=S;
Fin;
```

```
Fonction fon (n :entiers ) :entiers ;
 Début
        Si  n ≤0  alors
                fon:=n
        sinon
                fon :=  n+fon(n-1) ;
Fin;
```

$$S_n = \sum_{i=0}^{n} r^i = 1 + r + \dots + r^n = \begin{cases} n+1 & \text{si } r = 1 \\ \dfrac{1-r^{n+1}}{1-r} & \text{si } r \neq 1 \end{cases}$$

1. **Calculer** la complexité temporelle (en fonction de *nombre d'affectations*) de la fonction fon.
2. **Calculer** la complexité temporelle (en fonction de *nombre d'additions*) de la procédure proc.

Exercice 02: « traitement des exceptions » ..... (MI) (06 pts)

```
public class   CEnsemble {
  private  CEnsembleEntiers ensemble;
public void ajout(int x) throws Eplein {
            ..........
        } ;
public int supp( ) throws Evide {
            ..........
        } ;
public CEnsemble() {
            ..........
        } ;
}
```

Soit CEnsemble un type qui contient deux méthodes :

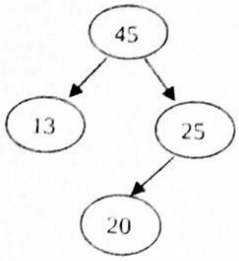
1. *ajout* : permet d'ajouter une valeur **x** si est seulement s'il y a une case vide dans **ensemble**, sinon l'exception Eplein sera déclenchée.
2. *supp* : si **ensemble** n'est pas vide, le premier élément de **ensemble** sera supprimé, sinon l'exception Evide sera déclenchée.

En utilisant la classe CEnsemble, **écrire** un programme qui nous permet de :

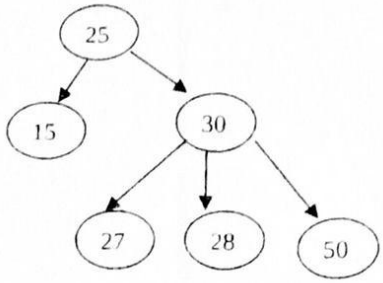
1. **Introduire** un ensemble d'entiers via le clavier . Il faut refaire la saisie en cas où la valeur introduite est d'un autre type.
2. **Afficher** cet ensemble sur écran.

Exercice 03: « Structures de données hiérarchiques » ..... (05 pts)

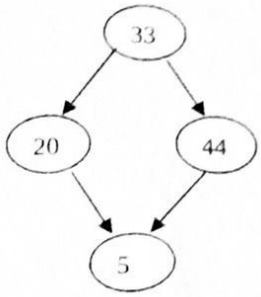
1. **Déterminer** parmi A_1,A_2 et A_3 les structures qui ne sont pas des arbres de recherche (justifier).



A_1

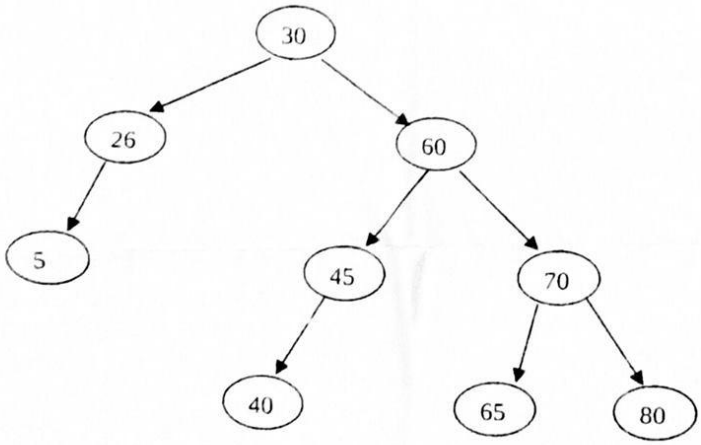


A_2



A_3

2. A_4 est un arbre de recherche, soit A_5 est l'arbre A_4 après la **suppression** du nœud 60 . **Donner** A_5.
3. Soit A_6 est l'arbre A_4 après **l'ajout** du nœud 75. **Donner** A_6. ..... . **(MI)**



A_4

Exercice 04: « Structures de données hiérarchiques » ..... **(05 pts)**

Compléter la classe **Arbre_3_aires** ci-dessous, où **Arbre_3_aires** est un type qui représente un arbre 3-aires et il contient :

- **existe** : pour un nœud donné x, cette méthode nous fournit vrai si x existe dans l'arbre sinon faux.
- **feuilles** : cette méthode nous fournit le nombre de feuilles dans l'arbre.

```
class  Arbre_3_aires {
        int              cle ;
        Object           info;
        Arbre_3_aires    fils1 ;
        Arbre_3_aires    fils2 ;
        Arbre_3_aires    fils3 ;
.....................
}
```