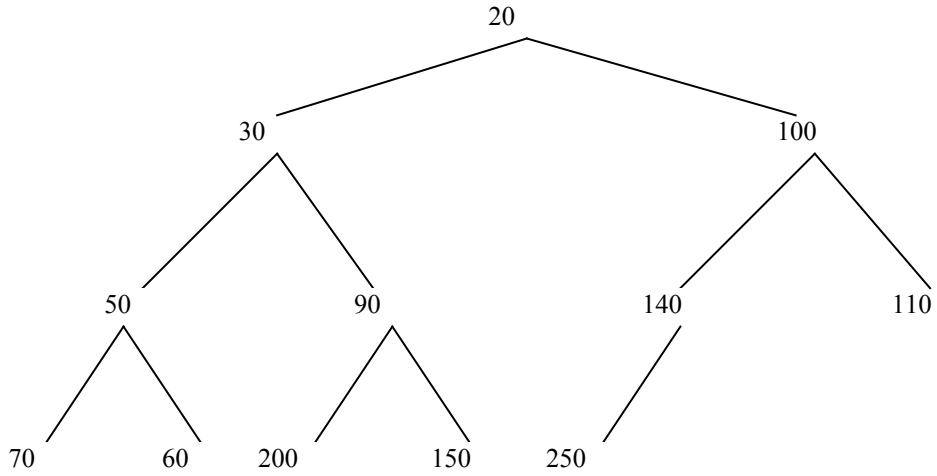
E.D. NFP 136 n°6

Thème : Arbres binaires et Tas

Exercice 1

Soit l'arbre binaire suivant (c'est un tas, et on suppose que les nombres sont des clés) :



Question 1

On utilise la classe suivante pour représenter les arbres binaires (tas ou non) :

```
class ArbreBinaire {
      int racine;
      ArbreBinaire gauche, droit;
...}
```

Soit la méthode récursive suivante applicable à un arbre binaire non vide :

```
static void lecture(ArbreBinaire a) {
      if(a.gauche != null) lecture(a.gauche);
      System.out.print(a.racine); //affiche la valeur de la racine
      if(a.droit != null) lecture(a.droit);
}
```

Donner la liste des valeurs obtenues en exécutant cette méthode sur le tas ci-dessus.

Question 2

On suppose à présent que `a, b, c, d, e` et `f` désignent des sous-arbres de l'arbre ci-dessus et sont tels que : `a.racine=30, b.racine=50, c.racine=90, d.racine=70, e.racine=60, et f.racine=200.` Compléter alors les valeurs ci-dessous :

`(i) b.gauche=?` `(ii) a.droit.racine=?`

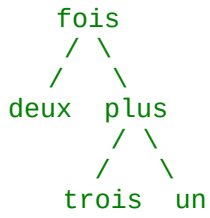
`(iii) f.droit=?` `(iv) a.gauche.droit.racine=?`

Exercice 2

On souhaite représenter une expression arithmétique par un arbre binaire d'opérateurs. Un opérateur est défini par l'une des huit valeurs suivantes : `plus`, `moins`, `fois`, `sur`, `zero`, `un`, `deux`, `trois`. Soit la classe `ArBinaireOp` implémentant les arbres binaires d'opérateurs :

```
enum Op {zero, un, deux, trois, plus, moins, fois, sur}

class ArBinaireOp {
       Op racine ;
       ArBinaireOp filsG, filsD;

       ArBinaireOp (Op r, ArBinaireOp gauche, ArBinaireOp droit) {
            racine=r ;
            filsG=gauche ;
            filsD=droit ;
       }

       boolean estFeuille() { return(filsG == null && filsD == null); }

       char caractereOperateur(Op o) {
            switch(o) {
               case Op.zero:  return '0' ;
               case Op.un:    return '1' ;
               case Op.deux:  return '2' ;
               case Op.trois: return '3' ;
               case Op.plus:  return '+' ;
               case Op.moins: return '-' ;
               case Op.fois:  return '*' ;
               case Op.sur:   return '/' ;
            }
       }
}
```

On associera donc à chaque expression un arbre binaire d'opérateurs. Par exemple, l'expression 2*(3+1) sera représentée par l'arbre binaire suivant :



Question 1

Déclarer cet arbre binaire précédent comme une instance de la classe `ArBinaireOp`.

Question 2

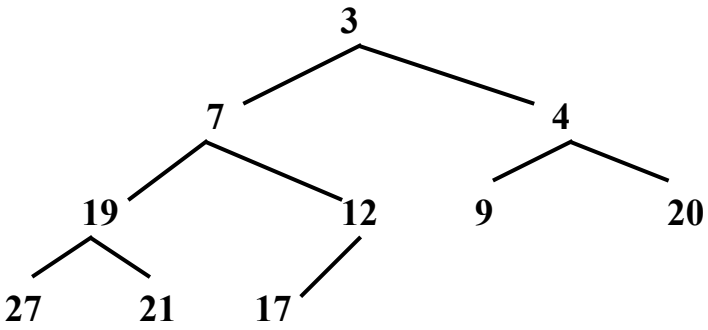
Ecrire une méthode récursive qui affiche un arbre binaire d'opérateurs sous la forme de l'expression qu'il représente (par exemple, « (2*(3+1)) » pour l'arbre précédent).

Question 3

Ecrire une méthode récursive qui calcule la valeur de l'expression représentée par un arbre binaire (le résultat sera un entier). Ainsi, la valeur sera 8 pour l'arbre précédent.

Exercice 3

Soit un ensemble de nombres rangés dans la structure suivante :



Question 1

Montrer que cette structure est bien un tas, et donner le tableau associé.

Question 2

On insère dans ce tas la valeur "5", selon la procédure vue en cours. Détailler les étapes de cette insertion, puis donner le tas obtenu.

Question 3

Même question si on supprime la valeur "3" dans l'arbre obtenu à l'issue de la question 2.

Exercice 4 Tri par tas

Question 1

Appliquer l'algorithme du tri par tas à la suite suivante :

16 - 10 - 8 - 11 - 5 - 6 - 9 - 1

On triera ces nombres par ordre croissant : illustrer le déroulement de l'algorithme, à la fois sur la représentation par arbre binaire, et sur la représentation par tableau.

Question 2

Quelle est la complexité de cet algorithme si les n entiers à trier sont triés par ordre décroissant ? Quelle est la complexité de sa première phase (construction du tas associé) si les n entiers à trier sont déjà triés par ordre croissant ?