1 Question (20 Pts.) - Tables de dispersement

Considérez une table de dispersement avec débordement progressif et sondage quadratique

$$H(clef) = clef \ \% \ N$$
$$HQ(clef, i) = ((H(clef) \ + \ i^2) \ \% \ N) \tag{1}$$

ou ($N \ = \ 13$) est la taille de la table et i est le nombre des sondages.

1.1 (5 Pts.) Insertion

Insérez l'élément 80 dans le tableau 1 suivant:

INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ENTREE	64		54	12			28						25			
ACTIVE	T		T	T			T						T			

Table 1: Table de dispersement

```
INS: 80

PROB NUM: 1 DELTA: 1
PROB NUM: 2 DELTA: 4
PROB NUM: 3 DELTA: 9
PHYS SIZE: 13

        POS: 0 VAL: 64 ACT: true
        POS: 2 VAL: 54 ACT: true
        POS: 3 VAL: 12 ACT: true
        POS: 6 VAL: 28 ACT: true
        POS: 11 VAL: 80 ACT: true
        POS: 12 VAL: 25 ACT: true
```

1.2 (5 Pts.) Insertion

Insérez l'élément 84 dans le tableau 2 suivant:

INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ENTREE			45				19	58		32	71					
ACTIVE			T				F	F		T	T					

Table 2: Table de dispersement

```
INS: 84

PROB NUM: 1 DELTA: 1
PROB NUM: 2 DELTA: 4
PROB NUM: 3 DELTA: 9
PROB NUM: 4 DELTA: 16
PROB NUM: 5 DELTA: 25
PHYS SIZE: 13

        POS: 2 VAL: 45 ACT: true
        POS: 5 VAL: 84 ACT: true
        POS: 6 VAL: 19 ACT: false
        POS: 7 VAL: 58 ACT: false
        POS: 9 VAL: 32 ACT: true
        POS: 10 VAL: 71 ACT: true
```

1.3 (5 Pts.) Élimination

Éliminez l'élément 0 dans le tableau suivant:

INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ENTREE	13	1			0			33	46	74						
ACTIVE	F	T			T			F	T	T						

Table 3: Table de dispersement

et REMPLISSEZ le tableau 4 avec les résultats de l'élimination:
ATTENTION: ce n'est pas nécessaire de recopier les valeurs qui n'ont pas changé.

INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ENTREE																
ACTIVE																

Table 4: Table de dispersement

```
DEL: 0

PROB NUM: 1 DELTA: 1
PROB NUM: 2 DELTA: 4
PHYS SIZE: 13

        POS: 0 VAL: 13 ACT: false
        POS: 1 VAL: 1 ACT: true
        POS: 4 VAL: 0 ACT: false
        POS: 7 VAL: 33 ACT: false
        POS: 8 VAL: 46 ACT: true
        POS: 9 VAL: 74 ACT: true
```

1.4 (5 Pts.) Démonstration

Pourquoi c'est TOUJOURS possible d'insérer un élément dans une table de dispersement à débordement progressif avec sondage quadratique, si elle contient au maximum un nombre d'élément correspondants à la moitié de sa taille et si sa taille est un nombre premier ?

SUGGESTION: Démontrez que pour deux nombre de sondage différents i et j, ($i \neq j$), c'est VRAI que $HQ(clef, i)$ est TOUJOURS différent de $HQ(clef, j)$.

Theorem 5.1.
If quadratic probing is used, and the table size is prime, then a new element can always be inserted if the table is at least half empty.

Proof.
Let the table size, *TableSize*, be an (odd) prime greater than 3. We show that the first $\lceil TableSize/2 \rceil$ alternative locations (including the initial location $h_0(x)$) are all distinct. Two of these locations are $h(x)+i^2$ (mod *TableSize*) and $h(x)+j^2$ (mod *TableSize*), where $0 \leq i, j \leq \lfloor TableSize/2 \rfloor$. Suppose, for the sake of contradiction, that these locations are the same, but $i \neq j$. Then

$$\begin{aligned} h(x) + i^2 &= h(x) + j^2 && \pmod{TableSize} \\ i^2 &= j^2 && \pmod{TableSize} \\ i^2 - j^2 &= 0 && \pmod{TableSize} \\ (i-j)(i+j) &= 0 && \pmod{TableSize} \end{aligned}$$

Since *TableSize* is prime, it follows that either $(i-j)$ or $(i+j)$ is equal to 0 (mod *TableSize*). Since i and j are distinct, the first option is not possible. Since $0 \leq i, j \leq \lfloor TableSize/2 \rfloor$, the second option is also impossible. Thus, the first $\lceil TableSize/2 \rceil$ alternative locations are distinct. If at most $\lfloor TableSize/2 \rfloor$ positions are taken, then an empty spot can always be found.

Figure 1: Demonstration

2 Question (15 Pts.) - Tri naifs

Considérez les tris naïfs suivants:

```
public static <AnyType extends Comparable<? super AnyType>>
    void algoSort_1(AnyType [] a) {

    int i, j;
    AnyType tmp;
    for (i = 0; i < a.length - 1; i++) {

        for (j = a.length - 1; j > i; j--) {

            if (a[j-1].compareTo(a[j]) > 0 ) {
                //Permutation des 2 éléments
                tmp = a[j-1];
                a[j-1] = a[j];
                a[j] = tmp;
            }
        }
    }
}

public static <AnyType extends Comparable<? super AnyType>>
    void algoSort_2(AnyType [] a) {

    int i, j;
    AnyType tmp;
    boolean change = true;

    i = 0;
    while ((change) && (i < a.length - 1)) {

        change = false;
        for (j = a.length - 1; j > i; j--) {

            if (a[j-1].compareTo(a[j]) > 0 ) {
                change = true;
                //Permutation des 2 éléments
                tmp = a[j-1];
                a[j-1] = a[j];
                a[j] = tmp;
            }
        }
        i = i + 1;
    }
}
```

2.1 (2 Pts.)

Quel est le nom de l'algorithme 1 (selon le livre et les slides du cours) ?

Tri en bulles ("BubbleSort")

2.2 (10 Pts.) Complexité asymptotique

Écrivez la formule de la complexité asymptotique des deux algorithmes dans le meilleur cas, le pire cas et le cas moyen, en REMPLISSANT le tableau 5.

Algorithme	Meilleur cas	Pire cas	Cas moyen
algoSort_1	$O(n^2)$	$O(n^2)$	$O(n^2)$
algoSort_2	$O(n)$	$O(n^2)$	$O(n^2)$

Table 5: Complexité asymptotique

2.3 (3 Pts.) Complexité asymptotique

JUSTIFIEZ brièvement vos réponses

L'algorithme algoSort_2 arrête s'il n'y a pas d'échanges de positions entre éléments. Si le vecteur est déjà trié (meilleur cas), cela se vérifie à la première itération de n éléments. Les autres cas demeurent égaux.

3 Question (20 Pts.) - Tri rapide (“Quicksort”)

3.1 (7 Pts.) - Tri rapide

Partez du vecteur à trier suivant :

INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CLEF	47	98	66	9	69	58	46	12	73	23	31	1	85	91	56	36

Table 6: Tri rapide

Indiquez la valeur du pivot choisi pour une seule récursion et remplissez le tableau suivant avec les partitions gauche et droite. (Note: le code de median3 est rapporté sans modifications en annexe pour votre référence).

PIVOT	

Table 7: Pivot (Tri rapide)

INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P. GAUCHE																
P. DROITE																

Table 8: Partitions (Tri rapide)

```
MEDIAN --->
        LEFT
                INDEX: 0
                        ELEMENT: 47
        RIGHT
                INDEX: 15
                        ELEMENT: 36
        CENTER
                INDEX: 7
                        ELEMENT: 12
        MEDIAN
                VAL: 36
<--- MEDIAN

        PIVOT VALUE:   36

        LEFT PARTITION --->
                INDEX: 0 ELEMENT: 12
                INDEX: 1 ELEMENT: 1
                INDEX: 2 ELEMENT: 31
                INDEX: 3 ELEMENT: 9
                INDEX: 4 ELEMENT: 23
        <--- LEFT PARTITION
        PIVOT --->
                INDEX: 5 ELEMENT: 36
        <--- PIVOT
        RIGHT PARTITION --->
                INDEX: 6 ELEMENT: 46
                INDEX: 7 ELEMENT: 56
                INDEX: 8 ELEMENT: 73
                INDEX: 9 ELEMENT: 69
                INDEX: 10 ELEMENT: 66
                INDEX: 11 ELEMENT: 98
                INDEX: 12 ELEMENT: 85
                INDEX: 13 ELEMENT: 91
                INDEX: 14 ELEMENT: 58
                INDEX: 15 ELEMENT: 47
        <--- RIGHT PARTITION
```

3.2 (7 Pts.) - Tri rapide

Partez du vecteur à trier suivant :

INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CLEF	61	27	10	75	91	90	56	15	18	63	82	6	97	74	64	23

Table 9: Tri rapide

Indiquez la valeur du pivot choisi pour une seule récursion et remplissez le tableau suivant avec les partitions gauche et droite. (Note: le code de median3 est rapporté sans modifications en annexe pour votre référence).

PIVOT	

Table 10: Pivot (Tri rapide)

INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P. GAUCHE																
P. DROITE																

Table 11: Partitions (Tri rapide)

```
MEDIAN --->
        LEFT
                INDEX: 0
                        ELEMENT: 61
        RIGHT
                INDEX: 15
                        ELEMENT: 23
        CENTER
                INDEX: 7
                        ELEMENT: 15
        MEDIAN
                VAL: 23
<--- MEDIAN

       PIVOT VALUE:   23

        LEFT PARTITION --->
                INDEX: 0 ELEMENT: 15
                INDEX: 1 ELEMENT: 6
                INDEX: 2 ELEMENT: 10
                INDEX: 3 ELEMENT: 18
        <--- LEFT PARTITION
        PIVOT --->
                INDEX: 4 ELEMENT: 23
        <--- PIVOT
        RIGHT PARTITION --->
                INDEX: 5 ELEMENT: 90
                INDEX: 6 ELEMENT: 56
                INDEX: 7 ELEMENT: 64
                INDEX: 8 ELEMENT: 75
                INDEX: 9 ELEMENT: 63
                INDEX: 10 ELEMENT: 82
                INDEX: 11 ELEMENT: 27
                INDEX: 12 ELEMENT: 97
                INDEX: 13 ELEMENT: 74
                INDEX: 14 ELEMENT: 91
                INDEX: 15 ELEMENT: 61
        <--- RIGHT PARTITION
```

3.3 (6 Pts.)

Considérez l'exemple de Tri-Fusion ("MergeSort") et ÉCRIVEZ l'expression des la récursion pour le Tri-Rapide ("QuickSort")

Algorithme	Générale	Meilleur cas
Tri-Fusion (exemple)	$T(N) = 2 \cdot T(\frac{N}{2}) + N$	$T(N) = 2 \cdot T(\frac{N}{2}) + N$
Tri Rapide	$T(N) = T(i) + T(N-i-1) + cN$	$T(N) = 2 \cdot T(\frac{N}{2}) + cN$

Table 12: Tri rapide (complexité)

Algorithme	Pire cas	Cas moyen
Tri-Fusion (exemple)	$T(N) = 2 \cdot T(\frac{N}{2}) + N$	$T(N) = 2 \cdot T(\frac{N}{2}) + N$
Tri Rapide	$T(N) = T(N-1) + cN$	$T(N) = \frac{2}{N}\left[\sum_{j=0}^{N-1} T(j)\right] + cN$

Table 13: Tri-Rapide (complexité)

4 Question (20 Pts.) - Parcours d'arbres

Considérez l'arbre binaire suivant:

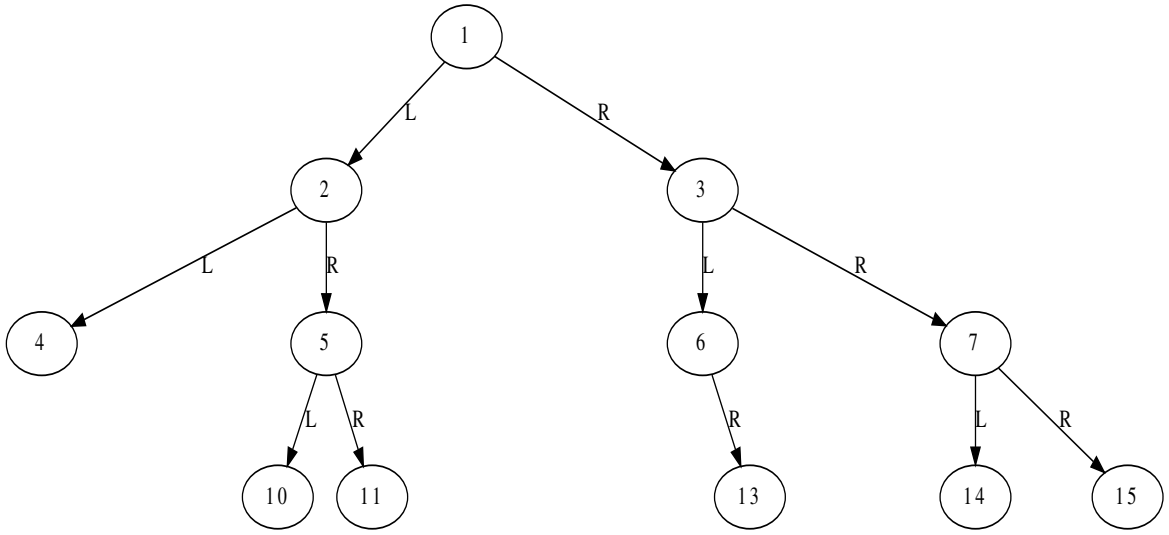

Figure 2: Arbre

4.1 (20 Pts.)

Parcourez l'arbre en traversant les enfants d'un nœud dans l'ordre numérique de leurs identificateurs et remplissez le tableau 14.

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Pré-ordre																
Ordre																
Post-ordre																
Par niveau																

Table 14: Parcours d'arbres

```
Pré-ordre: 1 2 4 5 10 11 3 6 13 7 14 15
Ordre: 4 2 10 5 11 1 6 13 3 14 7 15
Post-ordre: 4 10 11 5 2 13 6 14 15 7 3 1
Par niveau: 1 2 3 4 5 6 7 10 11 13 14 15
```

4.2 (4 Pts.) Insertion

Insérez l'élément 48 dans l'arbre binaire de recherche suivant:

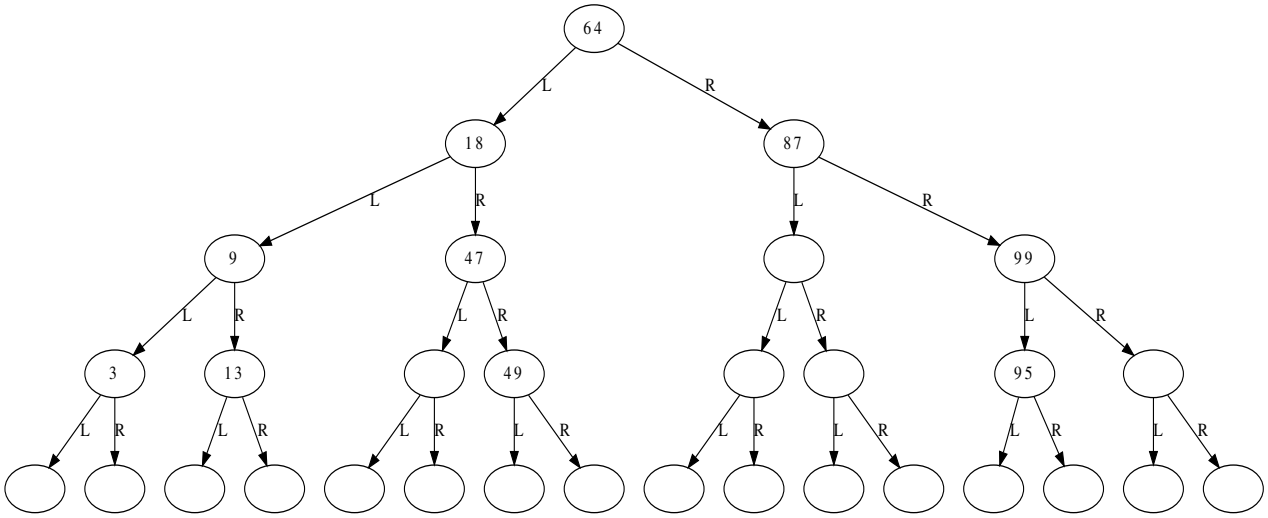

Figure 3: Arbre binaire de recherche

```
INS: 48

64      (1, ROOT)
  18    (2, LEFT)
    9   (3, LEFT)
      3 (4, LEFT)
      13        (4, RIGHT)
    47  (3, RIGHT)
      49        (4, RIGHT)
        48      (5, LEFT)
  87    (2, RIGHT)
    99  (3, RIGHT)
      95        (4, LEFT)
```

4.3 (4 Pts.) Insertion

Insérez l'élément 71 dans l'arbre binaire de recherche suivant:

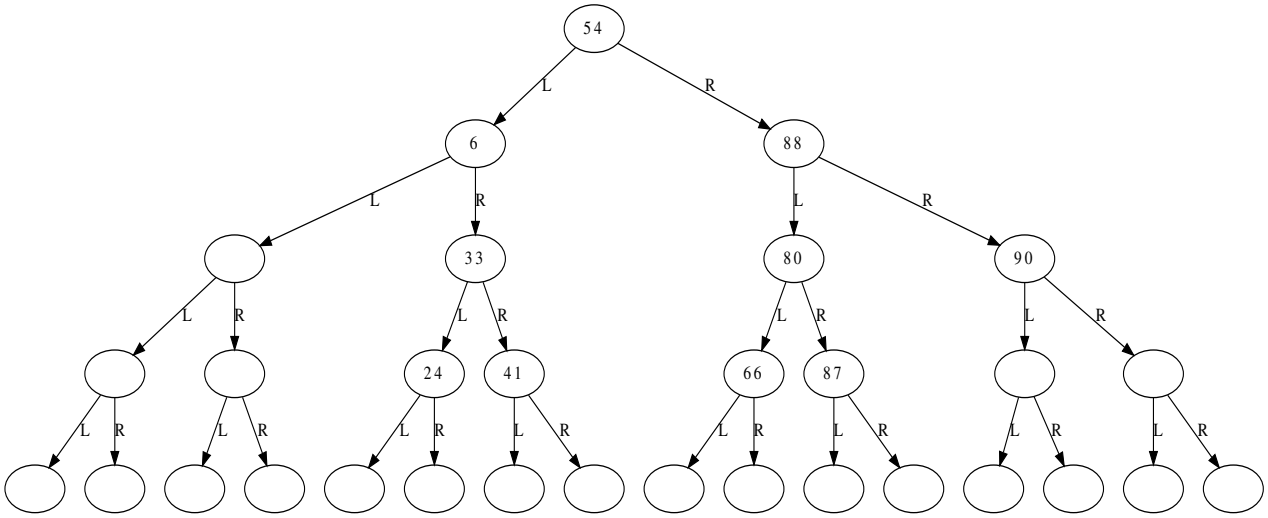

Figure 4: Arbre binaire de recherche

```
INS: 71

54      (1, ROOT)
  6     (2, LEFT)
    33  (3, RIGHT)
      24        (4, LEFT)
      41        (4, RIGHT)
  88    (2, RIGHT)
    80  (3, LEFT)
      66        (4, LEFT)
        71      (5, RIGHT)
      87        (4, RIGHT)
    90  (3, RIGHT)
```

4.4 (4 Pts.) Élimination

Éliminez l'élément 35 dans l'arbre binaire de recherche suivant, en REMPLISSANT la figure 6
ATTENTION: ce n'est pas nécessaire de recopier les valeurs qui n'ont pas change.

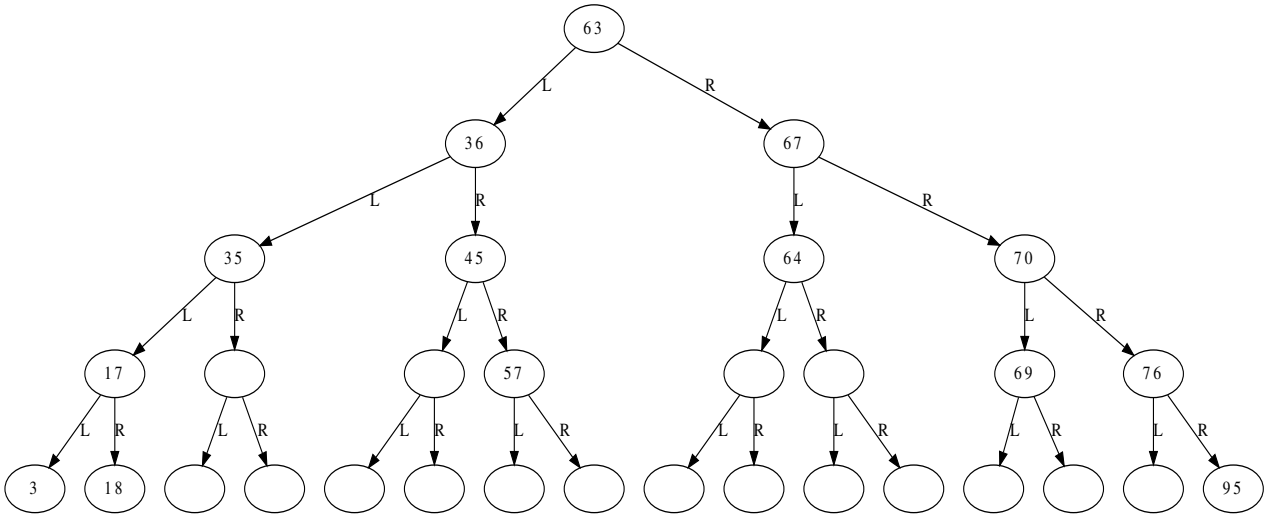

Figure 5: Arbre binaire de recherche

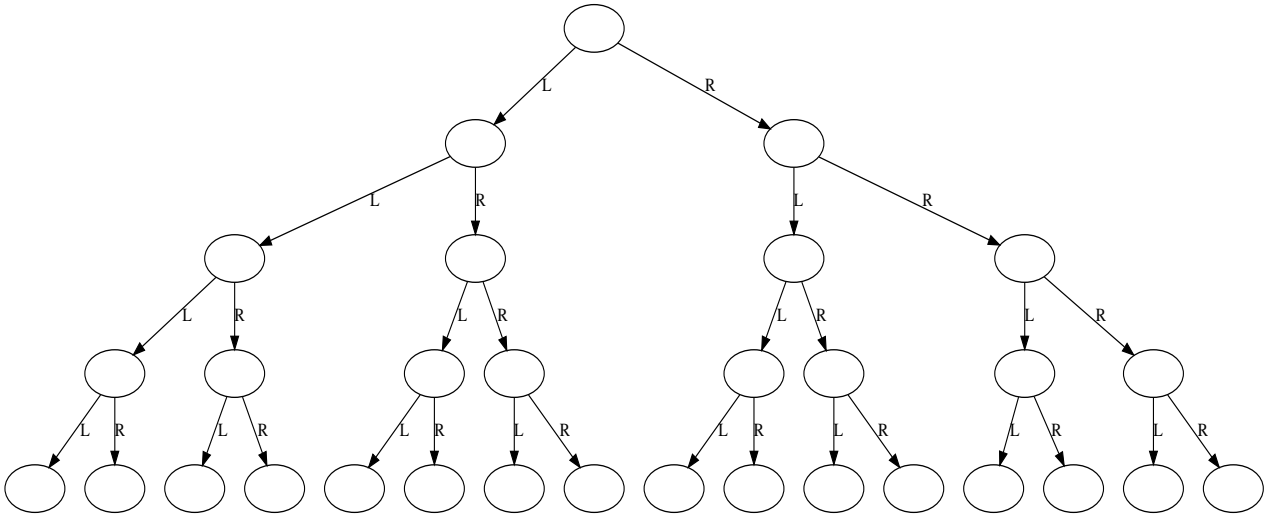

Figure 6: Arbre binaire de recherche

```
DEL: 35

63      (1, ROOT)
  36    (2, LEFT)
    17  (3, LEFT)
      3 (4, LEFT)
      18        (4, RIGHT)
    45  (3, RIGHT)
      57        (4, RIGHT)
  67    (2, RIGHT)
    64  (3, LEFT)
    70  (3, RIGHT)
      69        (4, LEFT)
      76        (4, RIGHT)
        95      (5, RIGHT)
```

4.5 (4 Pts.) Élimination

Éliminez l'élément 37 dans l'arbre binaire de recherche suivant, en REMPLISSANT la figure 8
ATTENTION: ce n'est pas nécessaire de recopier les valeurs qui n'ont pas change.

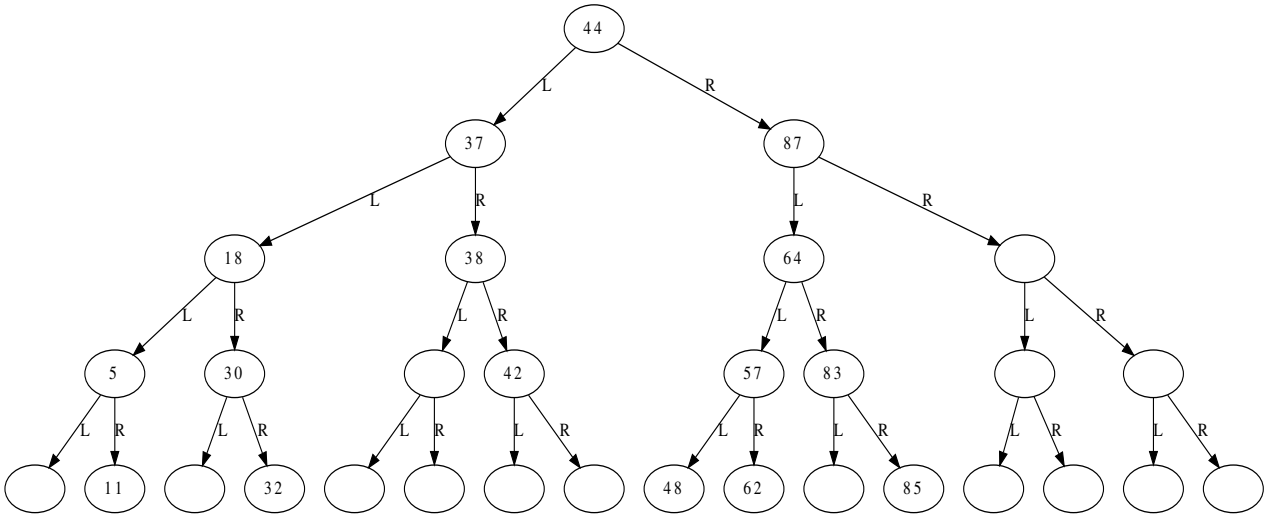

Figure 7: Arbre binaire de recherche

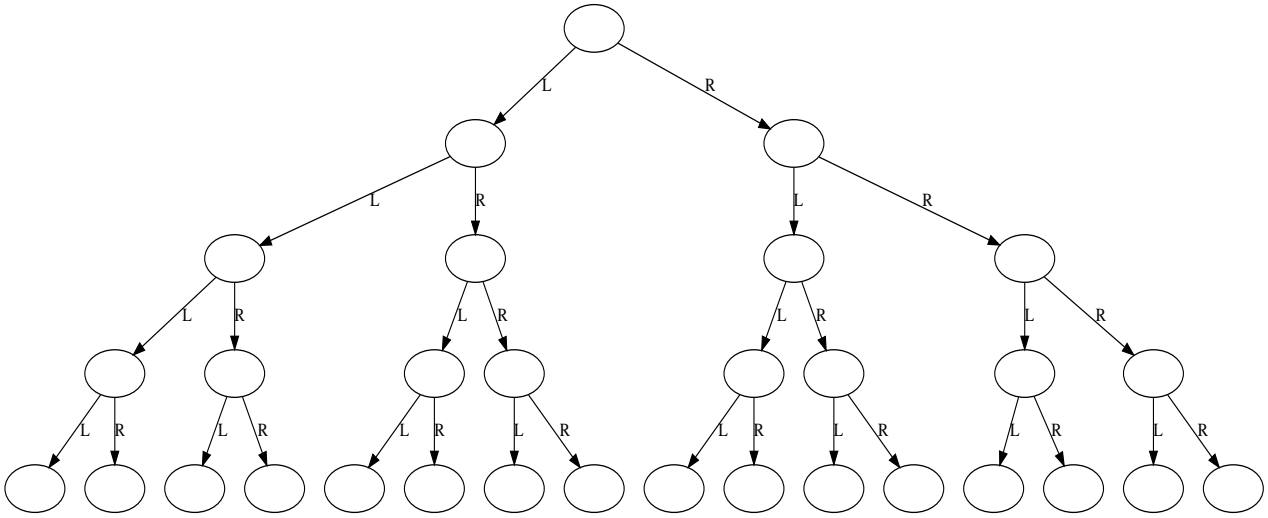

Figure 8: Arbre binaire de recherche

```
DEL: 37

44      (1, ROOT)
  38    (2, LEFT)
    18  (3, LEFT)
      5 (4, LEFT)
        11      (5, RIGHT)
      30        (4, RIGHT)
        32      (5, RIGHT)
    42  (3, RIGHT)
  87    (2, RIGHT)
    64  (3, LEFT)
      57        (4, LEFT)
        48      (5, LEFT)
        62      (5, RIGHT)
      83        (4, RIGHT)
        85      (5, RIGHT)
```

4.6 (5 Pts.) Traversement en pré-ordre d'un arbre binaire de recherche

Considérez la déclaration suivante d'un nœud d'un arbre binaire de recherche (tirée du livre du cours) :

```
public class binTree<AnyType extends Comparable<? super AnyType>>
{
    ...

    // Basic node stored in unbalanced binary search trees
    private static class BinaryNode<AnyType>
    {
        // Constructors
        BinaryNode( AnyType theElement )
        {
            this( theElement, null, null );
        }

        BinaryNode( AnyType theElement,
                    BinaryNode<AnyType> lt,
                    BinaryNode<AnyType> rt )
        {
            element  = theElement;
            left     = lt;
            right    = rt;
        }

        AnyType element;            // The data in the node
        BinaryNode<AnyType> left;   // Left child
        BinaryNode<AnyType> right;  // Right child
    }


    /** The tree root. */
    private BinaryNode<AnyType> root;

    public void traversement_pre_ordre(List<AnyType> resList) {

        ...

        return;
    }
```

Écrivez le code Java pour la méthode “traversement_pre_ordre”.

Cette méthode doit insérer dans une List<AnyType> le traversement en pré-ordre d’un arbre binaire de recherche en partant de sa racine et doit aussi imprimer en sortie, lors du traversement, le nœud en pré-ordre et son niveau dans l’arbre.

```
public void traversement_pre_ordre(List<AnyType> resList) {

    rec_traversement_pre_ordre(0,
                               root,
                               resList);
    return;
}

private void rec_traversement_pre_ordre(int level,
                                        BinaryNode<AnyType> t,
                                        List<AnyType> resultat)
{
    if( t != null )
    {

        System.out.println("NODE: " +
                           t.element +
                           " LEVEL: " +
                           level);

        resultat.add(t.element);

        rec_traversement_pre_ordre((level + 1),
                                   t.left,
                                   resultat);

        rec_traversement_pre_ordre((level + 1),
                                   t.right,
                                   resultat);
    }

    return;
}
```

```
public void traversement_pre_ordre_v2(List<AnyType> resList) {

    if (root != null) {
        rec_traversement_pre_ordre_v2(0,
                                      root,
                                      resList);
    }

    return;
}

private void rec_traversement_pre_ordre_v2(int level,
                                           BinaryNode<AnyType> t,
                                           List<AnyType> resultat)
{
    System.out.println("NODE: " +
                       t.element +
                       " LEVEL: " +
                       level);

    resultat.add(t.element);

    if (t.left != null) {
        rec_traversement_pre_ordre_v2((level + 1),
                                      t.left,
                                      resultat);
    }

    if (t.right != null) {
        rec_traversement_pre_ordre_v2((level + 1),
                                      t.right,
                                      resultat);
    }

    return;
}
```

4.7 (4 Pts.) Arbres arithmetiques

Considerez l'arbre arithmetique dans la figure 9 suivante:

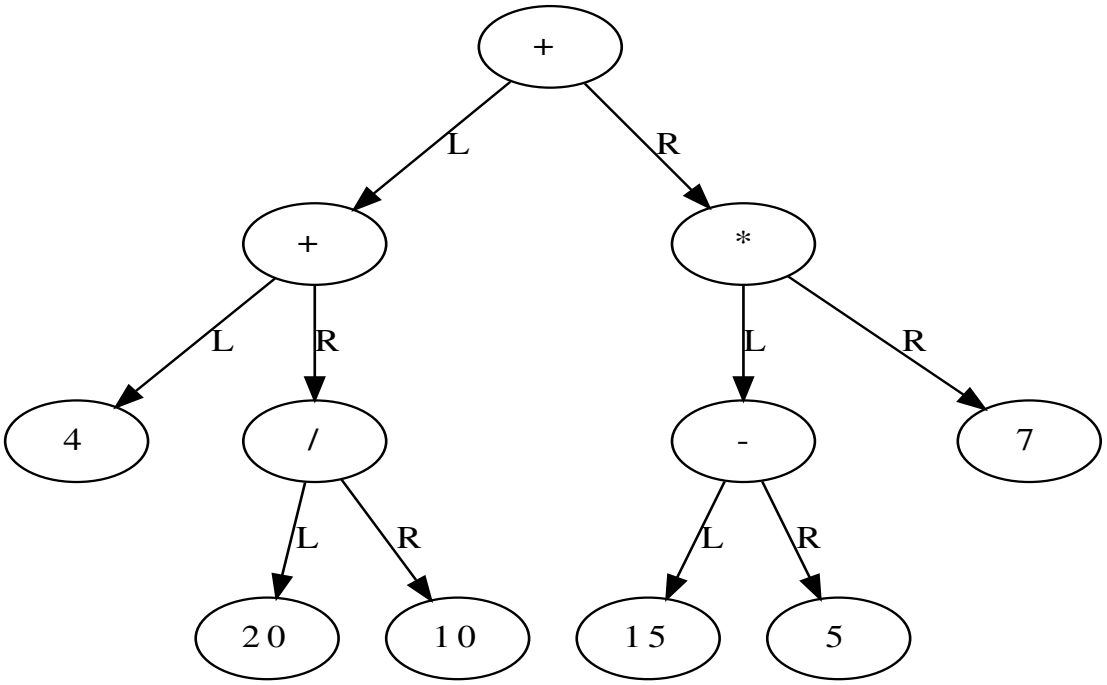

Figure 9: Arbre arithmetique

Expliquer en paroles comment modifier un algorithme de traversement d'arbres binaires afin d'évaluer le résultat de l'expression représentée dans un arbre arithmétique.

Il faut effecteur un traversement en post-ordre de l'arbre binaire. Au moment du traitement du nœud après ses enfants, si le nœud est une opération binaire, il faut retourner la valeur du résultat de l'opération appliquée à la valeur récursivement retournée par le sous-arbres gauche et par le sous-arbre droit. Si le nœud est un nombre, il faut retourner la valeur de ce nombre.

5 Annexe - 1

```
private static Comparable median3(Comparable [] a,
                                  int left,
                                  int right) {

    int center = ( left + right ) / 2;

    if( a[ center ].compareTo( a[ left ] ) < 0 )
        swapReferences( a, left, center );
    if( a[ right ].compareTo( a[ left ] ) < 0 )
        swapReferences( a, left, right );
    if( a[ right ].compareTo( a[ center ] ) < 0 )
        swapReferences( a, center, right );

    // Place pivot at position right - 1
    swapReferences( a, center, right - 1 );

    return a[ right - 1 ];
}
```